Best
Available
Copy

ADA022589

CULLER/HARRISON, INC.
150-A Aero Camino
Goleta, California 93017
(805) 968-1064

December 1975

APPENDICIES 1 THRU 9.
TO
SPEECH SIGNAL PROCESSING RESEARCH.

AD A020489

Final Technical Report.

Mar 73 - Sep 75

568p.

PRINCIPAL INVESTIGATOR: Dr. Glen J. Culler

PROJECT SCIENTISTS:
Dr. Michael McCammon
Dr. James F. McGill
Mr. Ray E. Bjorkman
Mr. Benjamin K. Lum
Mr. Dale E. Taylor
Mr. Jan M. Vanderford

DDC
RECEIVED
MAR 25 1976
A

This research was supported by the Defense Advanced Research Projects Agency under ARPA Order No. 2359
Contract No. DAHC15-73-C-0252
ARPA Order-2359

Distribution of this document is unlimited. It may be released to the Clearinghouse, Department of Commerce for sale to the general public.

The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Advanced Research Projects Agency or the U.S. Government.

409 276

CHI
CULLER-HARRISON INC.
150-A Aero Camino, Goleta, California 93017
Telephone (805) 968-1064

March 24, 1976

Defense Documentation Center
Cameron Station
Alexandria, Virginia 22314

Attn: R. Newsom

Gentlemen:

Per our telephone conversation today regarding our Final Technical Report on Speech Signal Processing Research, Contract #DAHC15 73 C 0252; the National Technical Information Service (NTIS) is authorized to reproduce and sell this report.

Sincerely yours,

Glen J. Culler

Glen J. Culler
President

GJC:st

APPENDIX 1

MP-32A Reference Manual

CHI
SIGNAL SYSTEMS

TMB2

TMB2

MP-32A MACROPROCESSOR REFERENCE MANUAL

CONTENTS

1. GENERAL INFORMATION
2. INSTALLATION
3. OPERATION
4. PRINCIPLES OF OPERATION
5. MAINTENANCE

ACCESSION for
NTIS White Section
DDC Buff Section
UNANNOUNCED
JUSTIFICATION
BY
DISTRIBUTION/AVAILABILITY CODES
Dist. AVAIL. and/or SPECIAL
A

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	1/23/74	pp. vi,1-31,1-32,1-35,1-36,2-1,2-2,3-3 thru 3-7, 4-2,4-3, 4-5,4-9,4-13,4-23,4-84,4-85
B	10/23/75	pp. ii,vi,1-5,1-13,1-16,1-17,1-18,1-19,1-20,1-21,1-23, 1-24,1-25,1-36,2-1,4-4,4-7,4-8,4-36,4-38,4-42,

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29101

© 1973
by Culler/Harrison, Inc.

TABLE OF CONTENTS

		Page
PREFACE		vi
I.	GENERAL INFORMATION	
1-1	Scope	1-1
1-3	Physical Description	1-1
1-4	Assemblies	1-1
1-25	Assembly Interconnection	1-18
1-27	Functional Description	1-18
1-28	General	1-18
1-34	Control and Timing Unit	1-18
1-40	Arithmetic Unit	1-23
1-45	Memory Unit	1-24
1-56	I/O Interface Unit	1-26
1-60	Instructions	1-30
1-71	System Configuration	1-36
1-72	General	1-36
1-74	AP-90 Array Processor	1-36
1-77	Model 114 Disk Drive	1-36
1-81	Model 100 User Station	1-39
1-83	Model 100 Keyboard	1-39
1-86	Tektronix Model 611 Display Unit	1-39
1-90	Leading Particulars	1-43
1-92	Capabilities and Limitiations	1-43
II.	INSTALLATION	
2-1	Scope	2-1
2-3	Installation Planning	2-1
2-6	Logistics	2-1
2-7	Receiving	2-1
2-9	Visual Inspection	2-1
2-11	Installation Procedures	2-1

TABLE OF CONTENTS (continued)

		Page
III.	OPERATION	
3-1	Scope	3-1
3-3	Controls and Indicators	3-1
3-4	Status Panel	3-1
3-6	Control Panel	3-1
3-12	Operating Instructions	3-7
3-13	Powering-Up	3-7
3-15	Dead Start	3-8
3-19	Powering-Down	3-8
IV.	PRINCIPLES OF OPERATION	
4-1	Scope	4-1
4-3	Functional System Description	4-1
4-4	Logic Design Representation	4-1
4-6	System Block Diagram	4-1
4-8	Subsystem Block Diagram	4-1
4-10	Logic Diagrams	4-1
4-12	Logic Elements	4-2
4-14	Logic Signals	4-2
4-16	Logic Names	4-2
4-24	Functional System Operation	4-3
4-25	Logic Circuits	4-3
4-27	Design Features	4-3
4-29	Control and Timing Unit	4-3
4-69	Arithmetic Unit	4-20
4-79	Memory Unit	4-36
4-95	I/O Interface Unit	4-41
4-134	Instruction Design & Implementation	4-64
4-136	Instruction Types	4-64
4-138	Machine Language Instruction Subsets	4-64
4-145	Input/output	4-68
V.	MAINTENANCE	
5-1	Scope	5-1
5-3	Routine Maintenance	5-1

TABLE OF CONTENTS (continued)

Page

FIGURES

1-1 The MP-32A Macroprocessor 1-2
1-2 Location of Assemblies 1-3
1-3 Status, Control & Maintenance Panels 1-4
1-4 Main Logic Plane Assembly 1-6
1-5 I/O Logic Plane Assembly 1-8
1-6 I/O Connector Panel Assembly 1-9
1-7 Memory Chassis Assembly 1-10
1-8 Lamp Power Supply Assembly 1-11
1-9 Input Power Panel Assembly 1-12
1-10 Top Cover Assembly 1-14
1-11 Power Supply Assembly 1-15
1-12A Analog Output Assembly 1-16
1-12B Analog Input Assembly 1-17
1-13 Component Block Diagram 1-19
1-14 Functional Block Diagram 1-20
1-15 Instruction Readout 1-22
1-16 N/A
1-17 CHI Signal/II System 1-37
1-18 Disk Storage Drive-Model 114 1-38
1-19 Keyboard-Model 100 1-40
1-20 Keycode & Color 1-41
1-21 Display Unit-Tektronix Model 611 1-42
2-1 MP-32A Installation Clearances 2-4
3-1 Status Panel 3-2
3-2 Control Panel 3-3
4-1 System Timing Diagram 4-8
4-2 Memory Cycle Timing 4-37
4-3 Device Input Bus Mechanization 4-45
4-4 Manual Load El-Register 4-48
4-5 Keyboard Controller Functional Block Diagram 4-50
4-6 Functional Block Diagram Unblank Display Command 4-52
4-7 Flow & Timing Diagrams Modes 0 & 1, OpCode 15 4-72

TABLE OF CONTENTS (continued)

Page

FIGURES (continued)

4-8 Flow Diagram Scan Instruction Mode 2, OPC 15 4-74

4-9 Contents of PA V_x System Clocks in Scan Instr 4-75

4-10 Scan Test - Condition Met - $Bit(5)_8 = 1$ 4-76

4-11 Scan Test - Condition Not Met - All bits in Selected Register=0 OPC-15 4-77

4-12 Link Jump Flow Diagram 4-80

TABLES

1-1 Instruction Subsets 1-33

1-2 Leading Particulars 1-43

1-3 Capabilities and Limitations 1-43

2-1 Physical Planning Notes - Model 32A Macroprocessor 2-2

3-1 Control Panel Controls 3-4

3-2 Control Panel Indicators 3-6

4-1 Design Features 4-4

4-2 Document Listing - Logic Design of Control and Timing Unit . 4-6

4-3 Instruction Address Register (IA) 4-12

4-4 Instruction Pad Write 4-13

4-5 Instruction Buffer Register 4-15

4-6 Halt Register 4-19

4-7 Adder Input Controls Chart 4-23

4-8 Adder Destination Controls Chart 4-25

PREFACE

This manual gives a complete description of the general physical and functional characteristics, installation, operation, theory of operation and maintenance of the CHI MP-32A Macroprocessor.

Other technical Manuals which may be used in conjunction with this manual are:

TMA2	User Manual
TMA3	MP-32 Macro-programming Manual
TMA4	MP-32 Micro-programming Manual
TMA8	AP-90 Programming Manual
TMB4	MP-32A Maintenance Manual
TMB8	AP-90 Reference Manual

The logical design principles embodied in the MP-32A are protected under U.S. Patent No. 3,771,141.

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No.29101
© 1973
by Culler/Harrison, Inc.

I. GENERAL INFORMATION

1-1. SCOPE

1-2. This section describes the physical and functional characteristics of the MACROPROCESSOR-32A(MP-32A). The MP-32A is described first as a single component and then as a part of the Signal System. A summary of the major characteristics of the MP-32A is also given.

1-3. PHYSICAL DESCRIPTION

1-4. ASSEMBLIES

1-5. The MP-32A is comprised of fourteen major assemblies, mounted in a custom made electronic equipment cabinet measuring 61.5 inches high, 24.0 inches wide, and 28.5 inches deep, as shown in Figure 1-1. The location of each assembly within the cabinet is shown in Figure 1-2. The assemblies with reference drawing numbers are listed below:

Assembly	Drawing No.
Status Panel	82001-100-1
Control Panel	81002-100-1
Maintenance Panel	82023-100-1
Main Logic Plane	82004-100-1, 82004-100-2
I/O Logic Plane	82005-100-1
I/O Connector Panel	82006-100-1, 82006-100-2
Memory Chassis	82007-100-1
Lamp Power Supply	82008-100-1
Input Power Panel	82009-100-1
Top Cover	82010-100-1
Power Supply	22011-100-1
Analog Assemblies(4)	82012-100-1, 82012,100-2, 82013-100-1, 81010-100-1

The assemblies are interconnected either through standard type connectors and cables or directly wired to form an integrated system.

1-6. STATUS PANEL. The status panel assembly, Figure 1-3, incorporates a system power switch, +5VDC, +19.7VDC and +23.2VDC on-off indicating lights, a disk status indicating light (operative if on - inoperative if off), and an elapsed time meter.

1-7. CONTROL PANEL. The control panel, Figure 1-3, serves as a mounting for controls and indicators which provide an operator the capability of data entry, program debugging, fault isolation and maintenance.

1-8. MAINTENANCE PANEL. The maintenance panel, Figure 1-3, serves as a mounting for two 3 position switches which give the maintenance engineer two functional options for disk and clock as follows:

Figure 1-1. (Photograph of MP-32A)

DATE	BY	REVISION RECORD	APPR.	DR.	CK.

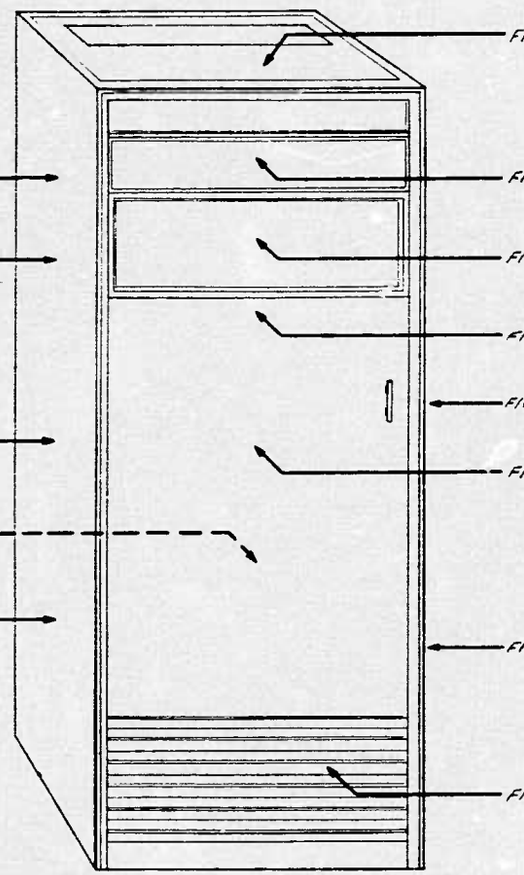


FIG. 1-10 TOP COVER ASSY.

FIG. 1-7 MEM CHASSIS

FIG. 1-3 STATUS PANEL

FIG. 1-5 I/O LOGIC PLN,
a. INTRNL. I/O CONN PNL.
b. ANALOG ASS'YS.

FIG. 1-3 CONTROL PANEL

FIG. 1-3 MAINTENANCE PANEL

FIG. 1-8 LAMP POWER SUPPLY

FIG. 1-12 ANALOG ASS'YS,
a. DISPLAY OUTPUT
b. ANALOG OUTPUT
c. ANALOG INPUT
d. ANALOG SWITCH

FIG. 1-4 MAIN LOGIC PLANE

FIG. 1-6 I/O CONN. PNL.

LOCATION OF ASSEMBLIES

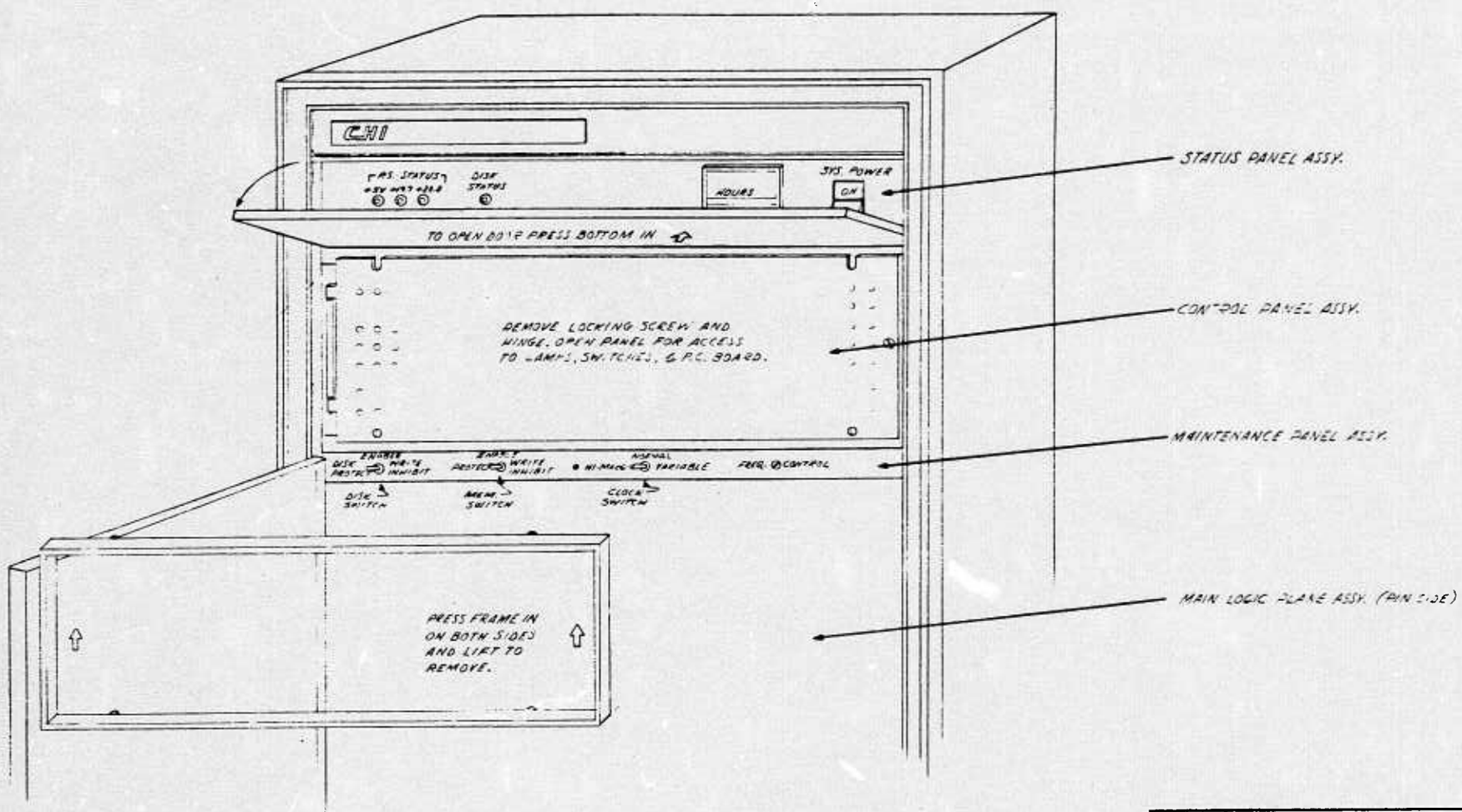
FIG. 1-11 PWR. SUPPLIES,
a. +19.7/23.2 VDC (MEM.)
b. ±15VDC (ANALOG)

FIG. 1-9 INPUT PWR. PNL.

FIG. 1-11 POWER SUPPLY, +5VDC (VCC)

TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL ±		SCALE NONE	DRAWN BY [illegible]
			APPROVED BY J. N. SCHROTH
FRACTIONAL ±	TITLE LOCATION OF ASSEMBLIES		
ANGULAR ±	DATE 12/12/72	DRAWING NUMBER FIGURE 1-2	

K+E



STATUS, CONTROL, & MAINTENANCE PANELS

TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL ±		SCALE NONE	DRAWN BY APPROVED BY
FRACTIONAL ±	TITLE STATUS, CONTROL, & MAINT. PNL.		
ANGULAR ±	DATE 12/12/72	DRAWING NUMBER FIGURE 1-3	

a. DISK SWITCH:

left position = prevents writing on cylinder zero of disk
center position = enables read/write of disk
right position = prevents writing on disk

b. CLOCK SWITCH:

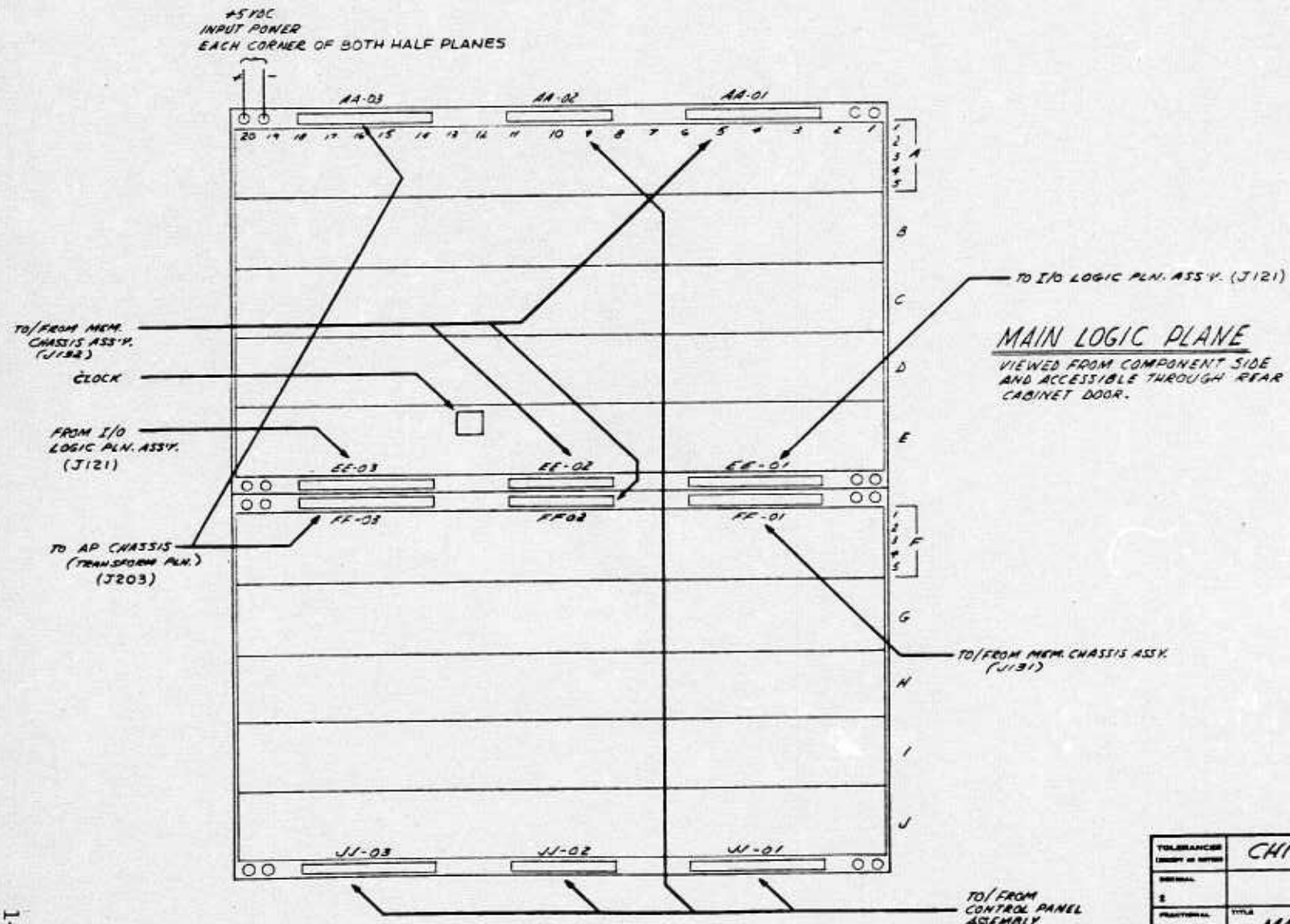
left position = runs clock at high frequency
center position = runs clock at 167 ns (crystal controlled)
right position = allows clock frequency to be varied by control screw

1-9. MAIN LOGIC PLANE. The main logic plane, Figure 1-4, is one of two master planes used in the computer. The main logic plane houses all of the integrated circuits used in the computer logic implementation except those used in the I/O interface logic plane. The main logic plane is actually composed of two planes connected by back plane wiring and physically located one above the other as shown in Figure 1-4. These two planes and the I/O logic plane, described in Section 1-14, are identical in size and construction.

1-10. The plane is comprised of a dual plane power distribution system, an array of sockets, and wire wrap interconnections. The dual plane configuration consists of two low resistivity planes separated by a thin sheet of dielectric material with a relative permittivity of approximately 7. One plane is used to distribute the 5VDC supply voltage to the individual sockets and the second plane serves as a ground plane. The composite, dual conducting planes separated by a dielectric, provides distributed capacitance for VCC decoupling. The wire wrap interconnections in conjunction with the power and ground planes form a high frequency signal transmission system. Logic signal transmission between individual circuits on the plane is via short lengths of 30 gauge solid wire in close proximity with a ground plane. Clock signals are distributed throughout the plane assembly in twisted pair terminated in 220 ohms. The power connection to sockets is made via clips that are soldered to the appropriate plane.

1-11. Individual circuit elements are plugged directly into sockets. This technique achieves elimination of possible circuit damage due to soldering when installing or removing parts, accessability for maintenance, and low cost complement of spares.

1-12. Eight 60-pin and four 44-pin connectors are mounted on the main logic plane for routing signals to and from the plane. Additional signal routing to and from the plane is accomplished via cables which are terminated in 16-pin plugs. The cables plug directly into designated sockets in the plane assembly.



MAIN LOGIC PLANE

VIEWED FROM COMPONENT SIDE AND ACCESSIBLE THROUGH REAR CABINET DOOR.

TOLERANCES	CHI CULLER-HARRISON INC.		
DECIMAL ±		SCALE NONE	DRAWN BY
FRACTIONAL ±	TITLE MAIN LOGIC PLANE		
ANGULAR ±	DATE 12/12/72	DRAWING NUMBER FIGURE I-4	

1-13. The plane is partitioned into bays to facilitate locating individual circuit elements. An x-y coordinate system is used to specify the location of sockets within a bay. The equipment is labeled as to bay, column and row as shown in Figure 1-4.

1-14. I/O LOGIC PLANE. The I/O logic plane, Figure 1-5, houses the complement of circuits which constitute the I/O interface. The construction employed in the I/O plane is identical to that of the main logic plane. Four 60 pin and two 44 pin connectors are mounted on the I/O plane for peripheral signal cables and analog modules. Signals between the I/O plane and the main logic plane are routed via wire wrap between the I/O plane and three 88 pin connectors mounted below the I/O plane and cables from these connectors to connectors on the main logic plane.

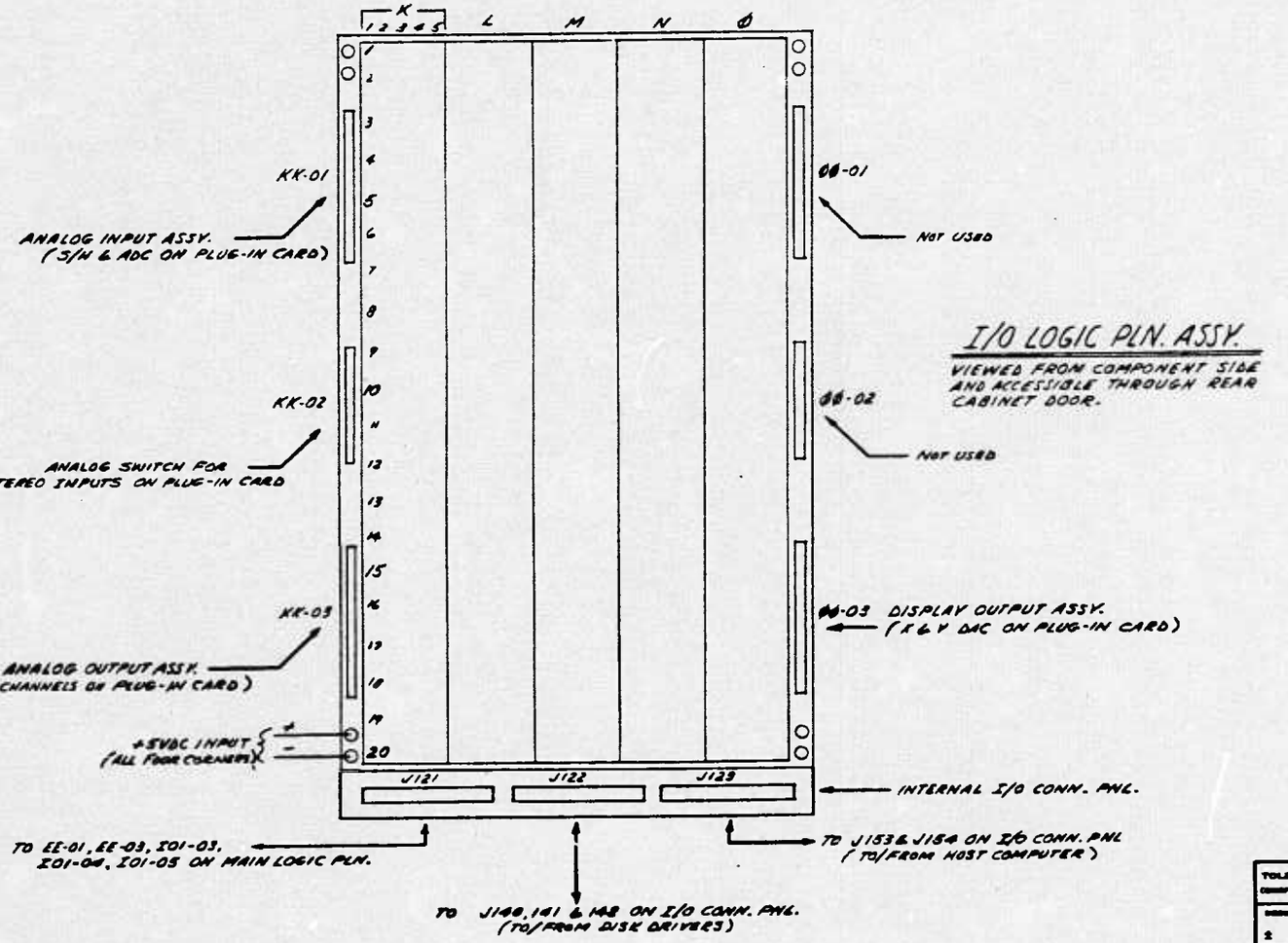
1-15. I/O CONNECTOR PANEL. The I/O Connector panel assembly, Figure 1-6, provides a mount for connectors which service the peripheral devices. There are locations for up to eight 20-pin and eight 50-pin peripheral connectors to be used for keyboard/display stations (up to four), analog I/O, host I/O, disk (up to eight), etc.

1-16. MEMORY CHASSIS. The memory chassis assembly, Figure 1-7, houses 25 plug-in memory cards. The components are mounted on a hinged chassis for accessibility. The assembly incorporates fifty 80-pin connectors. Each of the memory assembly cards requires two connectors. Signals between the memory cards and the main logic plane are first routed, via back-plane wiring, between the memory assembly connectors and two 88-pin connectors attached to the side of the memory chassis, and then by cables between these connectors and the connectors on the main logic plane. The power supply outputs are routed to the connectors by direct wiring. Inter-connector wiring is accomplished with wire wrap techniques. Two 117 cfm fans are located on the bottom of the chassis.

1-17. LAMP POWER SUPPLY. The lamp supply assembly, Figure 1-8, incorporates the components and interconnections which develop the 12VDC to drive the panel indicator lights. The AC line voltage drives a power transformer. The output of the transformer is full wave rectified and fed to a filter capacitor. The output of the filter capacitor is fed through a series resistor to a 12V Zener diode. The output is taken across the diode, and is used to drive the lights.

1-18. INPUT POWER PANEL. The input power panel, Figure 1-9, provides a mount for a Hubbel 3-prong male receptacle (117VAC input), 208/230VAC input, 208/230 VAC output, voltage sensitive relay, two time delay relays and a 208/230VAC DPDT relay.

1-19. The voltage sensitive relay, in combination with the "ON" time delay relay, provides power up sequencing by first applying 208/230VDC to the disk drives and then, after a 30-second delay, by automatically resetting the MP-32A and sending a controlled ground to the disk drives which permits the disk packs to rotate.

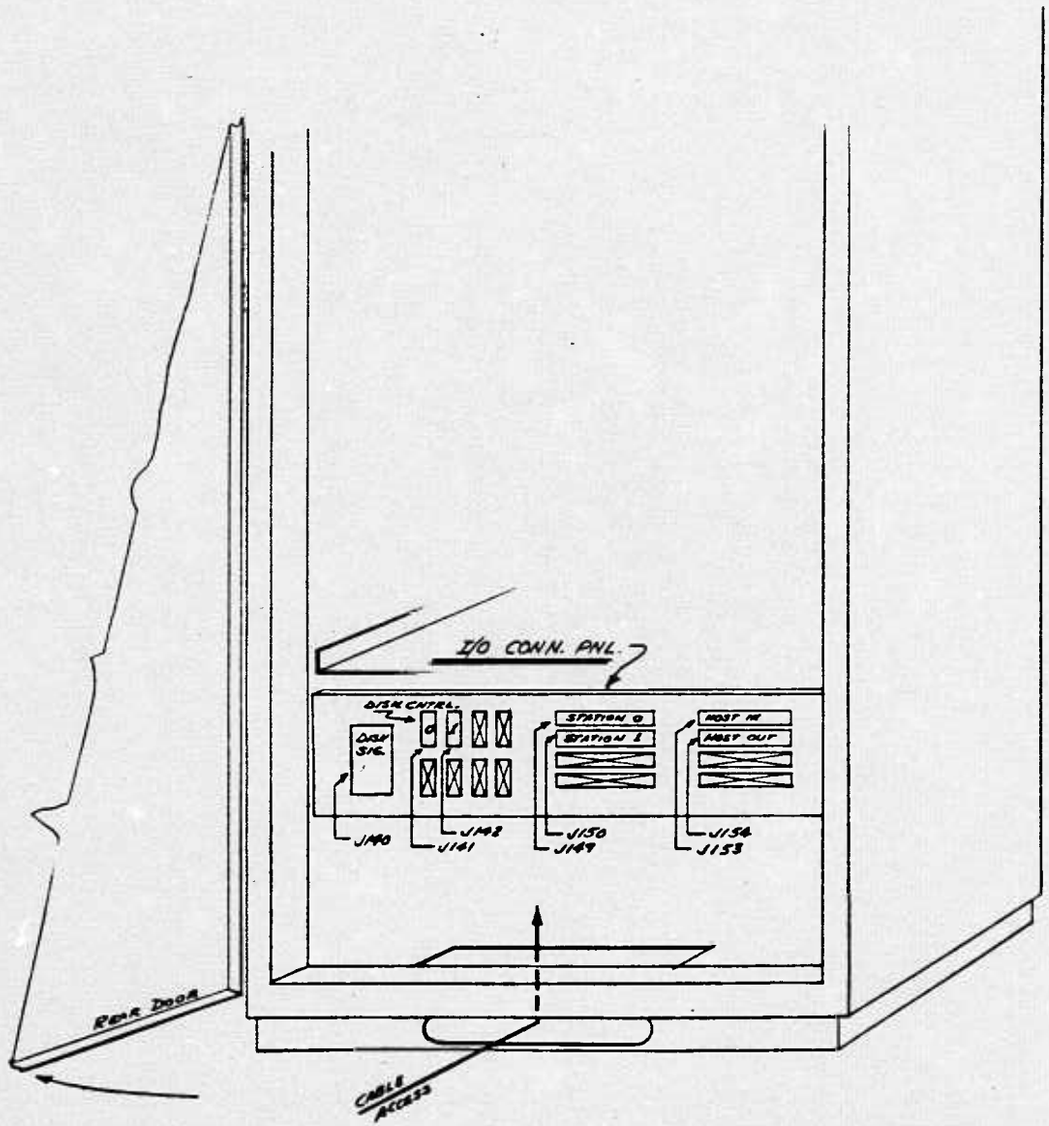


I/O LOGIC PLN. ASSY.

VIEWED FROM COMPONENT SIDE AND ACCESSIBLE THROUGH REAR CABINET DOOR.

DATE	OPR	REVISION RECORD	APPVD.	DR.	CK.

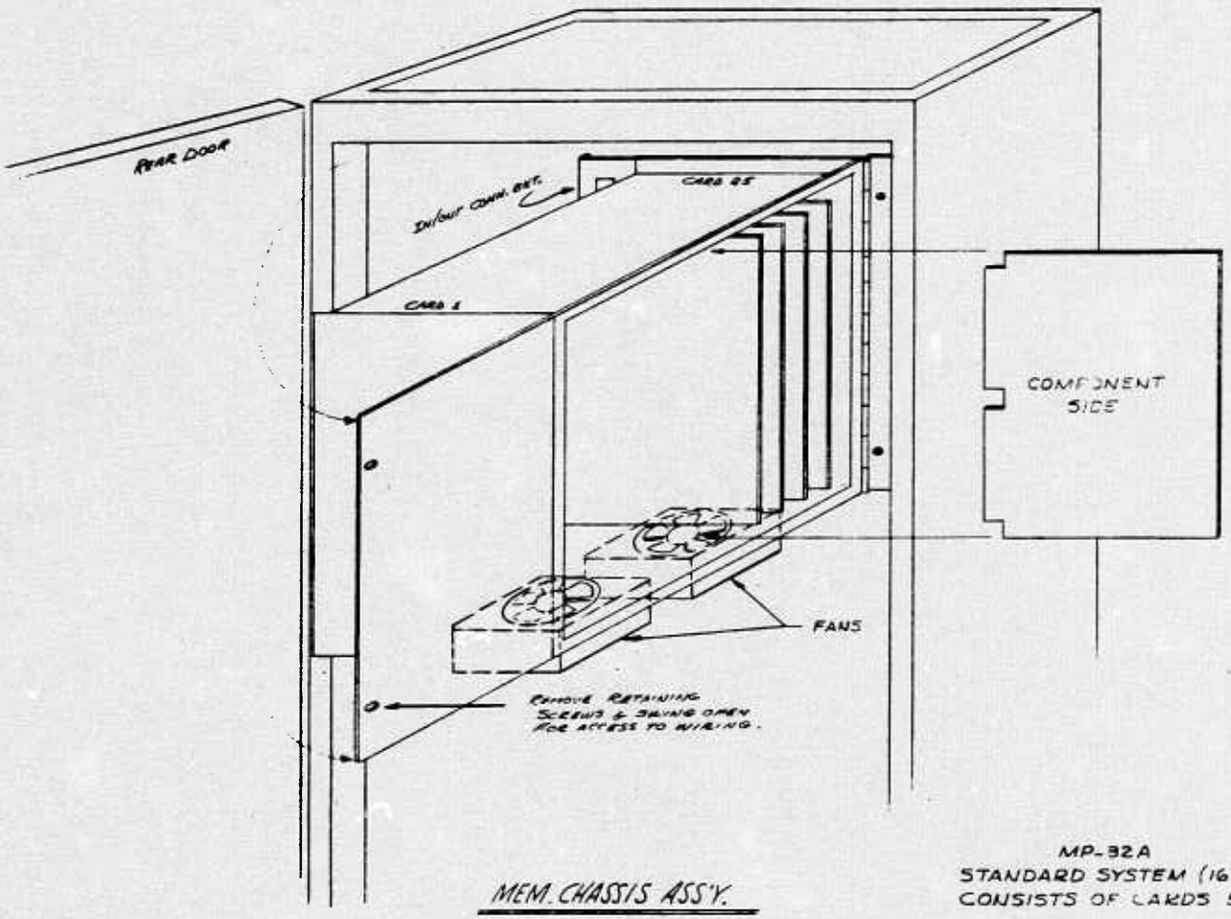
TOLERANCES UNLESS OTHERWISE SPECIFIED	CHI CULLER-HARRISON INC.		
DECIMAL ±		SCALE NONE	DRAWN BY CHI.
FRACTIONAL ±	TITLE I/O LOGIC PLANE ASSY.		
ANGULAR ±	DATE 12/18/72	DRAWING NUMBER FIGURE 1-5	



REAR VIEW OF MP CABINET

DATE	BY	REVISION RECORD	DR	CK

TOLERANCES (UNLESS OTHERWISE NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL ±		SCALE NONE	DRAWN BY AGH.
FRACTIONAL ±	TITLE I/O CONN. PANEL ASS'Y.		APPROVED BY DON SCHROTH
ANGULAR ±	DATE 12/13/72	DRAWING NUMBER FIGURE 1-6	



MEM. CHASSIS ASS'Y.

ACCESSIBLE FROM REAR

MP32A MEMORY (INTER LEAVED)

CARD 1	A	RIGHT
2	B	"
3	C	"
4	D	"
5	A	"
6	B	"
7	C	"
8	D	"
9	NOT USED	
10	A	LEFT
11	B	"
12	C	"
13	D	"
14	A	"
15	B	"
16	C	"
17	D	"
18	NOT USED	
19	OUTPUT GATING	
20	OUTPUT GATING	
21	ADDRESS REGISTER	
22	CONTROL CARD	
23	DATA REGISTER	
24	"	"
25	"	"

4K X 18 BITS EACH CARD

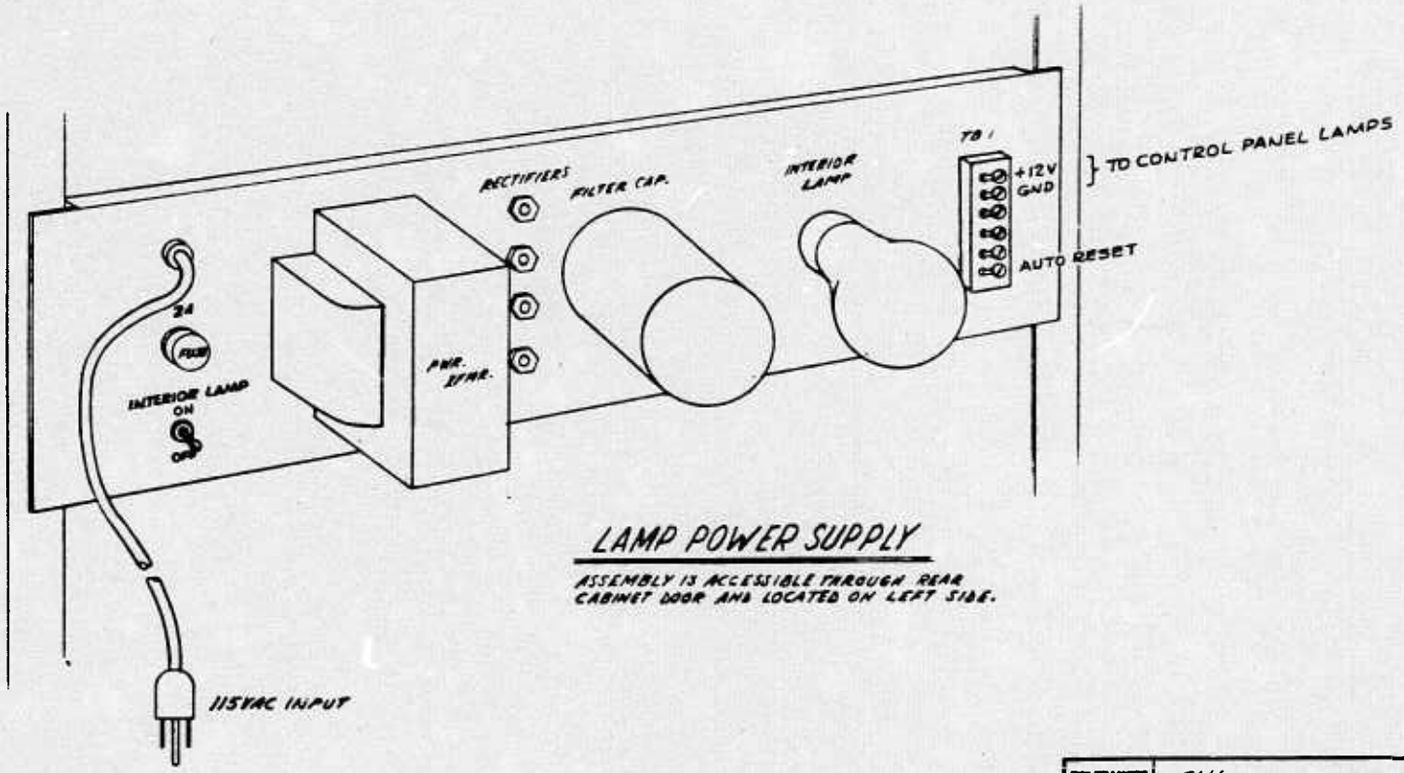
4K X 18 BITS EACH CARD

AP120 MEMORY (NON - INTER LEAVED)

CARD 1 - 20	NOT USED
21	DATA RIGHT STORE
22	DATA LEFT STORE
23	ADDRESS REGISTER
24	CONTROL
25	DATA REGISTER

MP-32A
STANDARD SYSTEM (16K 36-BIT WORDS)
CONSISTS OF CARDS 5-8 AND 14-25

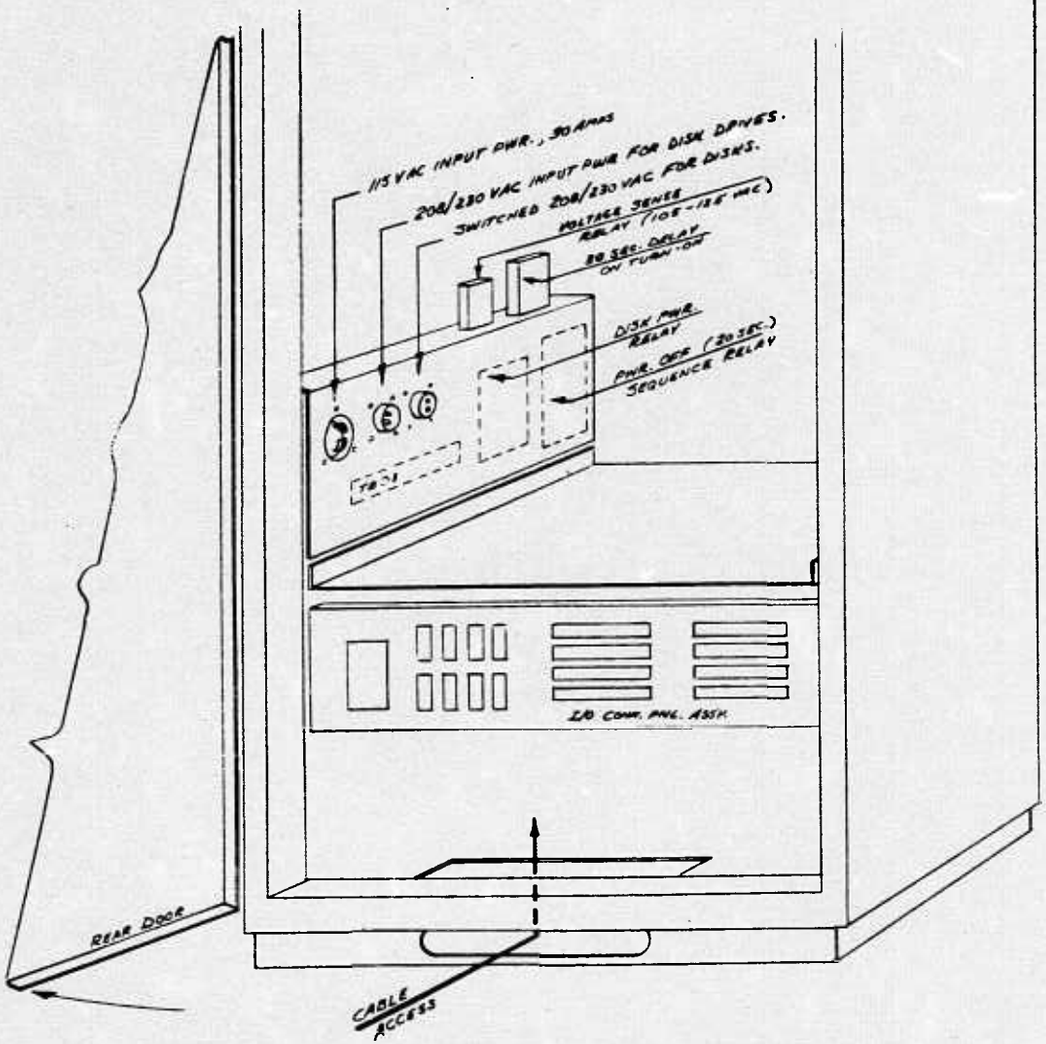
TOLERANCES (EXCEPT AS NOTED)	CHI CULLER-HARRISON INC.
DECIMAL ±	MP32A/AP120
SCALE	NONE
DRAWN BY	[illegible]
APPROVED BY	JON SCHROTH
FRACTIONAL ±	TITLE: MEM. CHASSIS ASS'Y.
ANGULAR ±	DATE: 5/29/73
DRAWING NUMBER	FIGURE 1-7



LAMP POWER SUPPLY

ASSEMBLY IS ACCESSIBLE THROUGH REAR CABINET DOOR AND LOCATED ON LEFT SIDE.

TOLERANCES UNLESS OTHERWISE NOTED	CHI CULLER-HARRISON INC.	
DECIMAL ±	SCALE NONE	DRAWN BY JWL APPROVED BY R.A. SCARRITT
FRACTIONAL ±	TITLE LAMP POWER SUPPLY	
ANGULAR ±	DATE 10/10/72	DRAWING NUMBER FIGURE 1-8



REAR VIEW OF MP CABINET

TOLERANCES (UNLESS OTHERWISE NOTED)	CHI CULLER-HARRISON INC.		
DECIMAL ±	MP32A	SCALE NONE	DRAWN BY DWL APPROVED BY DON CULLER
FRACTIONAL ±	TITLE INPUT PWR. PANEL ASS'Y.		
ANGULAR ±	DATE 5/20/13	DRAWING NUMBER FIGURE 1-9	

1-20. The voltage sensitive relay in combination with the "OFF" time delay relay and the 208/230VAC DPDT relay provides power down sequencing by turning the MP-32A power off and removing the 208/230VAC to the disk drives 40 seconds later. This time delay allows the packs to stop rotating before power is removed.

1-21. One eight pin amphenol connector is mounted on the assembly to distribute the AC line voltage to six 117 volt convenience outlets located on a strip along the side of the cabinet.

1-22. TOP COVER. The top cover, Figure 1-10, houses a 265 cpm fan which is used to cool the components on the memory chassis and exhaust air from the cabinet.

1-23. POWER SUPPLY. This assembly, Figure 1-11, incorporates one 117 cpm fan and three power supplies as follows:

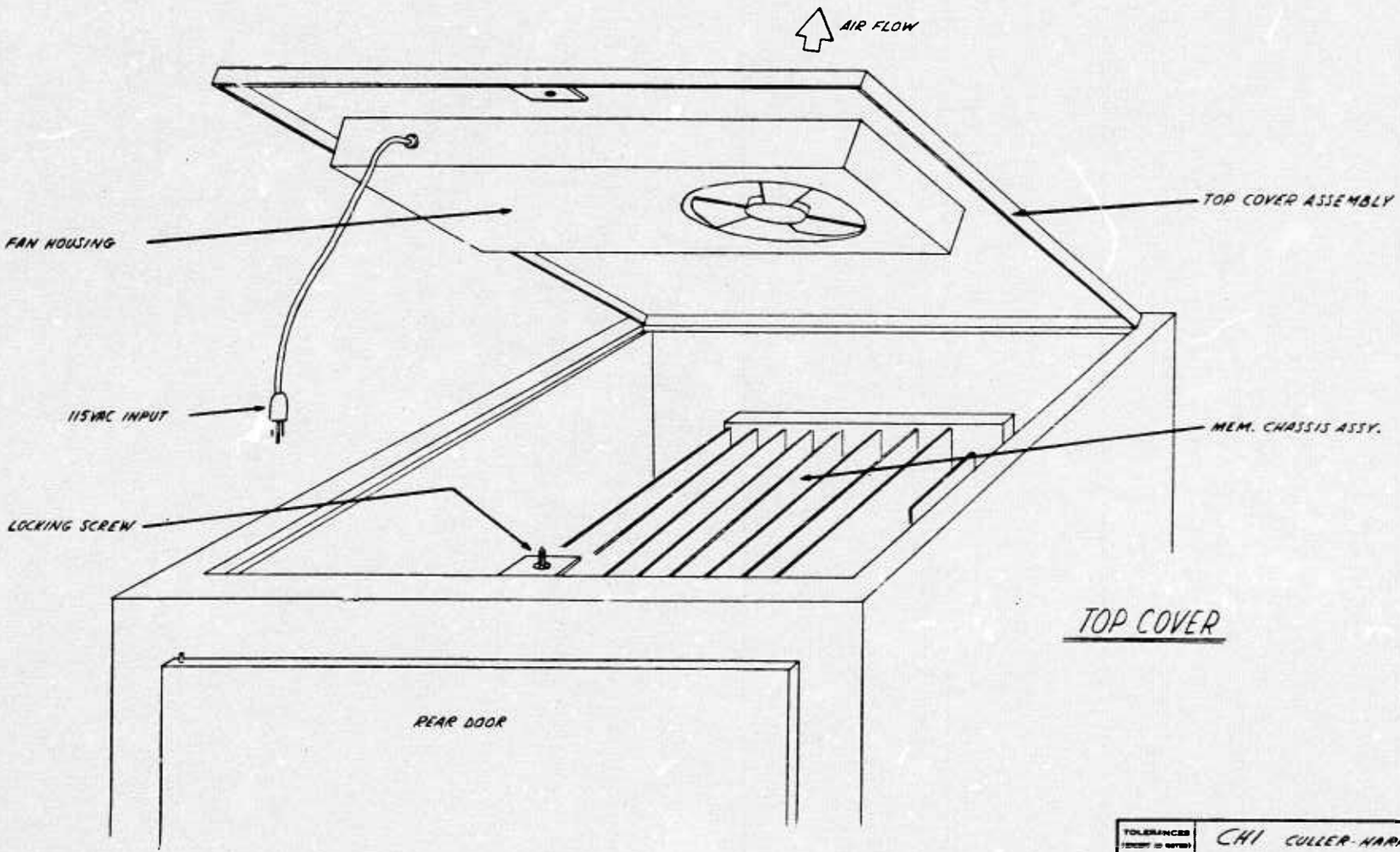
a. A +5VDC supply powers the main logic plane and the I/O logic plane. Two dual pairs of 12 guage wires bring the +5V output and its return to a pair of +5V and common buses. Each +5V bus is connected to opposite sides of the +5V power plane via 4-12 guage wires. The common buses are connected to the ground plane in an identical fashion. The +5VDC and its return are distributed to individual sockets via the power and ground planes. The return leads of all DC supplies (common) are tied to the cabinet (chassis ground).

b. A 20VDC power supply is modified to output +19.7VDC and +23.2VDC. The +23.2V supplies the VSX and the 19.7VDC supplies the VSS voltages necessary for the Memory planes. The supply outputs and common lead are wired to terminals on the memory chassis.

c. A $\pm$15VDC supply powers the analog assemblies.

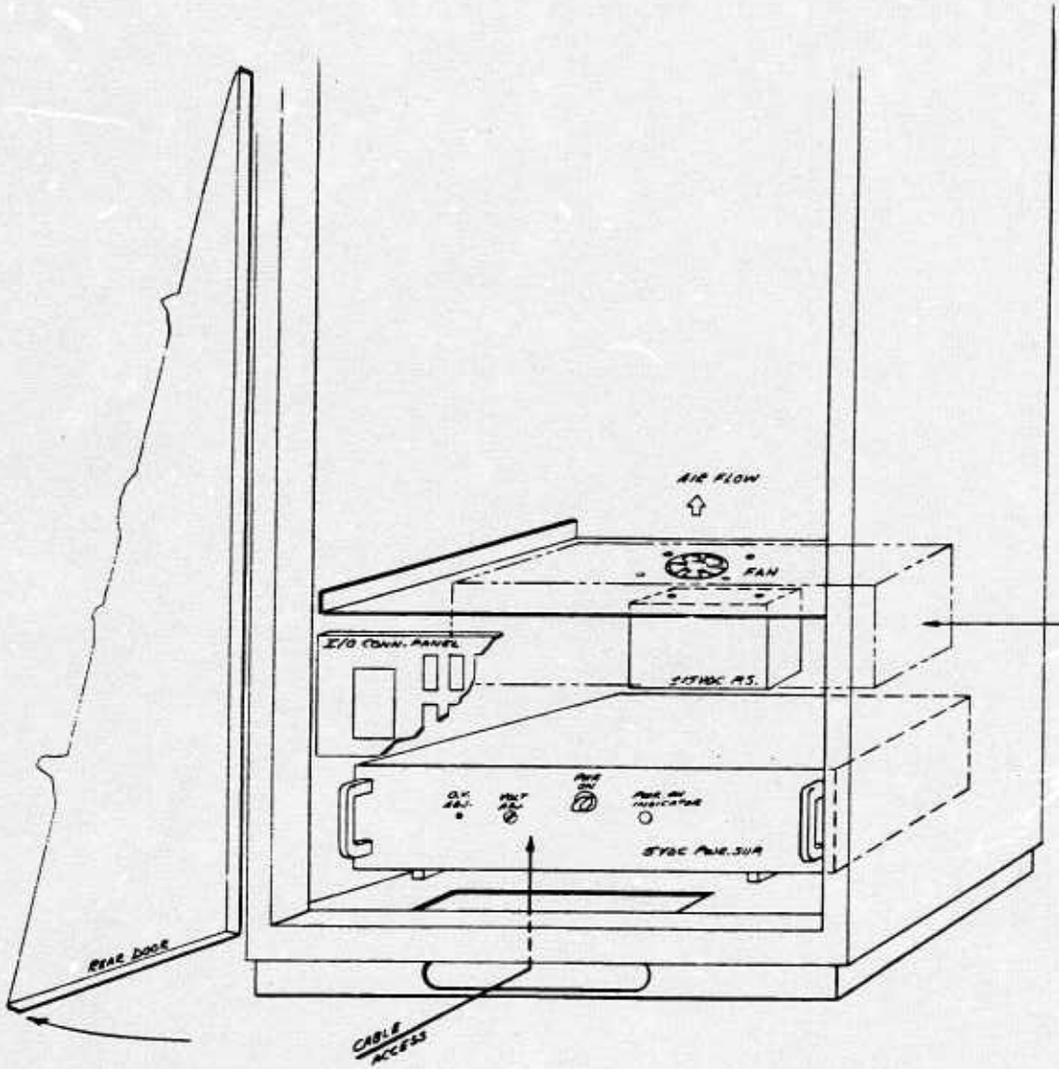
1-24. ANALOG ASSEMBLY. This assembly, shown in Figure 1-12, consists of the analog printed circuit board described below:

a. A Display Output p.c. board assembly plugs directly into connector ØØ-03 of the I/O plane. It contains two 12-bit DAC's with settling time $\leq$1µs. The outputs of these "X" and "Y" DAC's drive the display scopes of User Stations.



TOP COVER

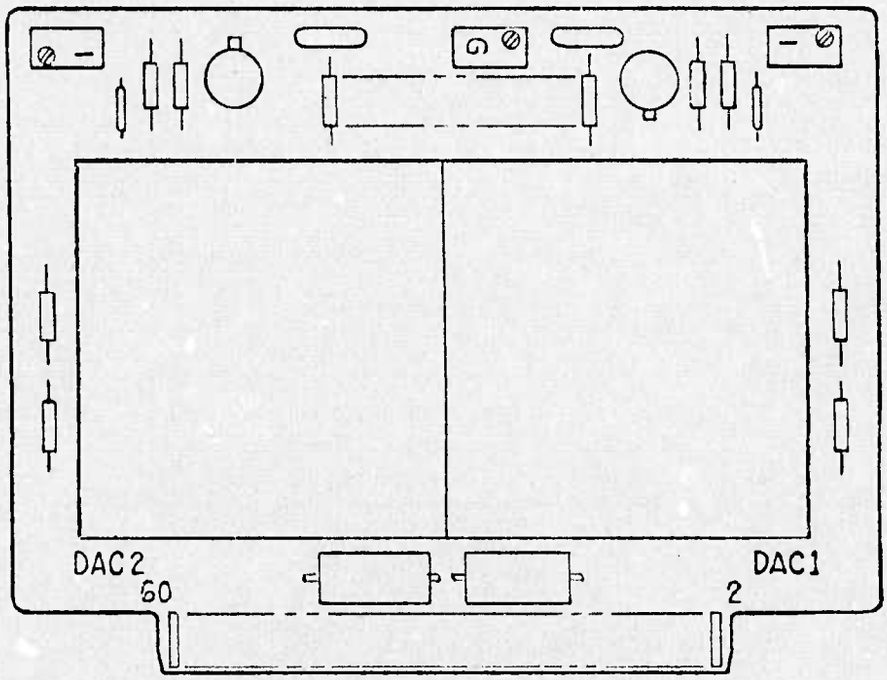
TOLERANCES (UNLESS OTHERWISE SPECIFIED)	CHI CULLER-HARRISON INC	
DECIMAL ±		SCALE NONE; DRAWN BY KHH; APPROVED BY C.H. SCHROTH
FRACTIONAL ±	TITLE TOP COVER ASS'Y.	
ANGULAR ±	DATE 12/2/72	DRAWING NUMBER FIGURE 1-10



REAR VIEW OF MP CABINET

IN SYSTEMS WHICH INCLUDE AN AP-120 ARRAY PROCESSOR, THIS POWER SUPPLY IS LOCATED IN THE AP-120.

TOLERANCES	CHI CULLER-HARRISON INC.		
DECIMAL	MP32A	SCALE	DRAWN
FRACTIONAL	TITLE POWER SUPPLY ASS'Y.		
ANGULAR	DATE 5/24/78	DRAWING NUMBER FIGURE 1-11	



DISPLAY OUTPUT ASSEMBLY

FIGURE 1-12A

1-25. ASSEMBLY INTERCONNECTION

1-26. The interconnecting paths of the machine assemblies are shown on the Internal Interconnection Diagram, drawing number 22002-100-1. This drawing, in combination with the drawings it references, completely specifies the inter-assembly electrical connections. This set of drawings, coupled with the wire lists, specify the machine interconnections in total.

1-27. FUNCTIONAL DESCRIPTION

1-28. GENERAL

1-29. The equipment complement of the MP-32A can be grouped functionally into four major units:

a. Control and Timing.

b. Arithmetic.

c. MOS Memory, and

d. I/O Interface.

The simplified block diagram, Figure 1-13 and the detailed block diagram, Figure 1-14, depict the major functional units and the principal data and control paths.

1-30. The program unit, as shown in Figure 1-13, controls the control unit. Though structurally not a part of the computer hardware, it has been included in the block diagram to show the primary source of control and to emphasize the fact that the internal structure of the MP-32A is to a large measure software dependent.

1-31. The MP-32A operates synchronously from an 6 MHz clock which defines the basic machine cycle time (MCT) of 167 ns. A single instruction which incorporates up to four operations, can be carried out in one machine cycle thus allowing an execution rate up to 24 million operations per second.

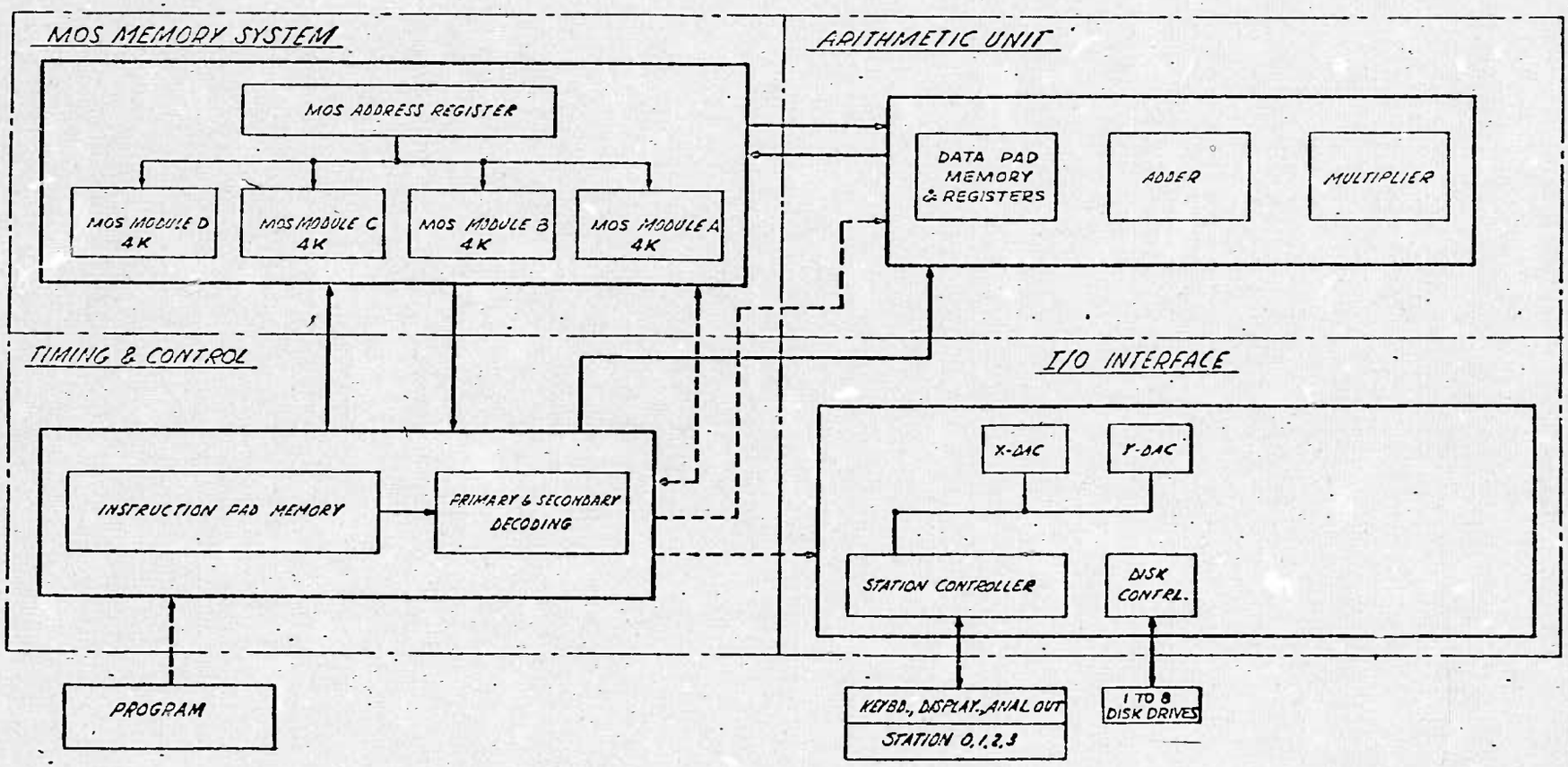
1-32. The MP-32A can be operated manually from the control panel or automatically from an instruction sequence stored in memory. The manual capability is provided primarily as a diagnostic tool. A set of switches and indicator lights are available on the control panel which allow an operator to exercise the machine and check its status.

1-33. Two dead start bootstrap routines are available to the user and are permanently stored in a 768 bit Read Only Memory. The bootstrap programs allow the user to automatically enter a load program from the disk or host.

1-34. CONTROL AND TIMING UNIT

1-35. The control and timing unit interprets each instruction and generates an exacting set of control and timing signals which determine inter-

PAGE BLANK-NOT FILMED

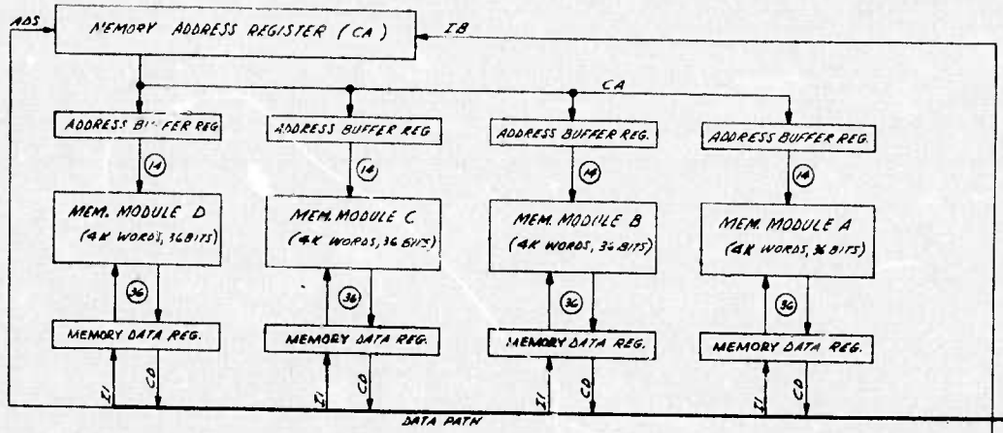


——— DENOTES DATA FLOW
– – – – DENOTES CONTROL SIGNALS

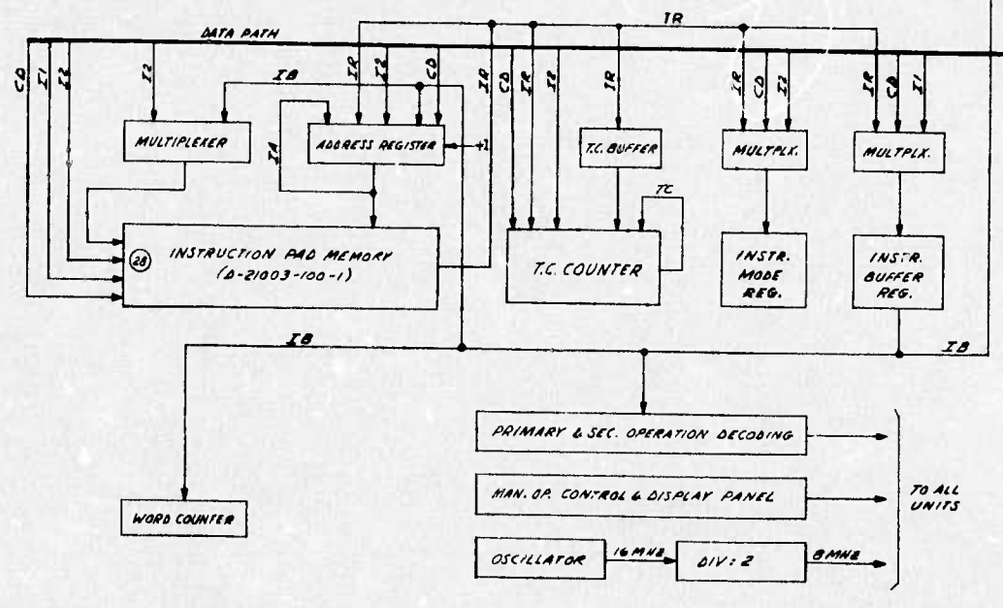
COMPONENT BLOCK DIAGRAM

FIGURE 1-13

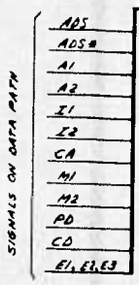
MEMORY SYSTEM D-21001-100-2



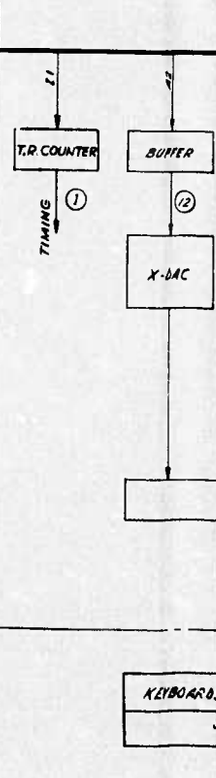
TIMING & CONTROL D-21001-100-3

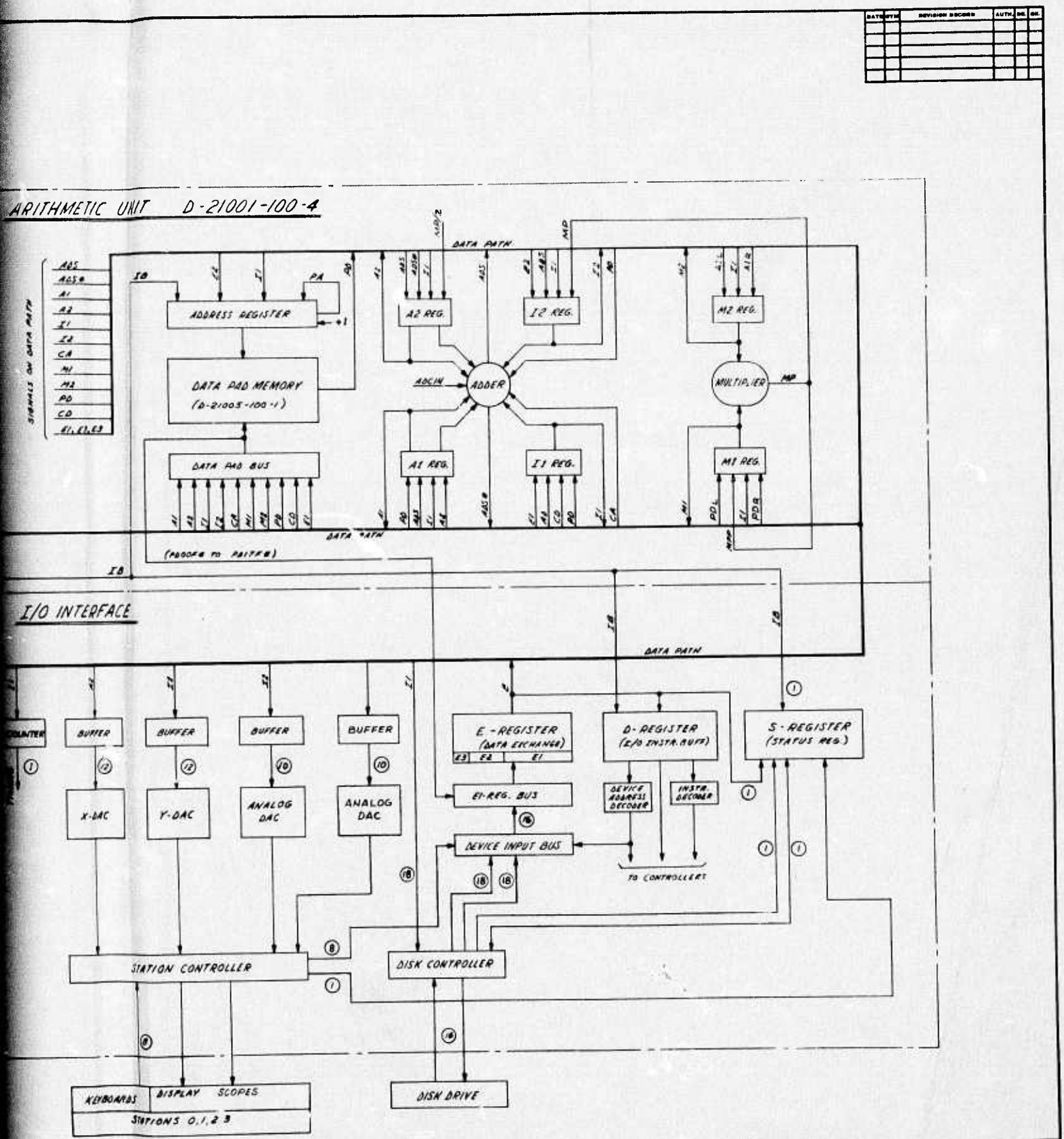


ARITHMETIC UNIT



I/O INTERFACE





TOLERANCES	CHI CULLER-HARRISON INC.	
MP32A	SCALE	DRAWN BY Bill
TITLE	MACRO-PROCESSOR BLOCK DIAG.	
DATE 5/30/73	DRAWING NUMBER D-21001-100-1	

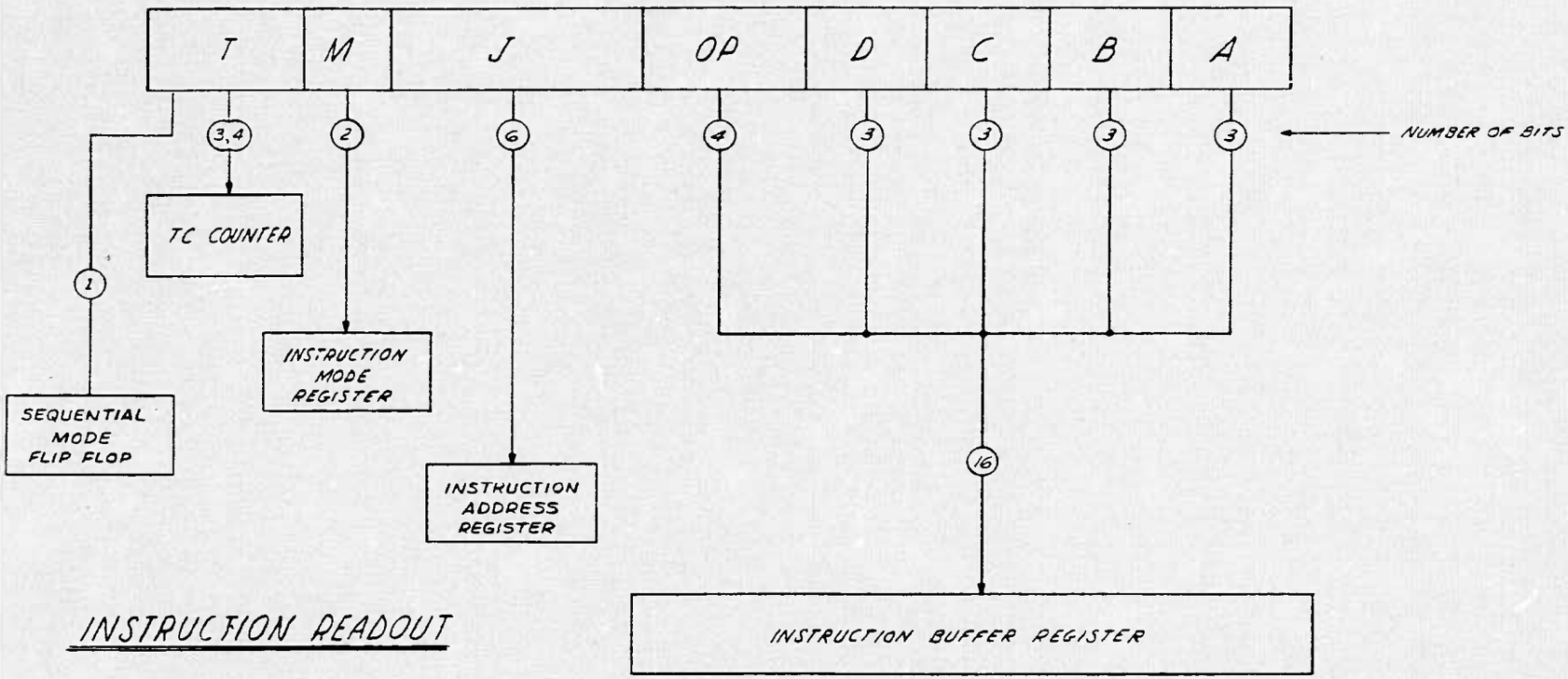
connecting paths for data transfer and logical or arithmetic operations to be performed.

1-36. INSTRUCTION PAD AND REGISTER. The unit incorporates a high speed 64 word, 28-bit per word, bipolar semiconductor memory called the Instruction Pad. In general, blocks of instructions are transferred from the main memory to the Instruction Pad to be executed in a loop mode. The control unit causes an instruction to be read out of the Instruction Pad into registers as shown in Figure 1-15. The instruction is decoded from the registers where it is stored during its time of execution. The control signals derived from the decode of the registers activate those components required for a particular instruction.

1-37. TIMING. The system timing is derived from a crystal controlled 12MHz oscillator. The timing system generates 6MHz clocks which define the basic machine cycle time of 167 ns. Two counters, the timing counter, TC, and the word counter, WC, provide extended timing sequences of up to 341μs with a time resolution of 167 ns. The TC is a four bit counter used to indicate the number of times an instruction is to be repeated and to generate other timing sequences of up to 16 clock times. The WC is an eight bit counter used in conjunction with the TC counter to generate timing sequences in excess of 8 clock times but not to exceed 341μs. The WC is used primarily in load type commands which effect transfers of blocks or words.

1-38. COMPONENTS AND THEIR FUNCTION. The principal components of the control and timing unit, together with a brief description of their primary function follows:

Component	Function
Instruction Pad Memory Capacity: 64 words Word Length: 28 bits Cycle Time: 167 ns	Holds sequence of up to 64 instructions.
Instruction Address Register (6 bits)	Holds the address of the word to be read from or written into instruction pad.
Instruction Mode Register (2 bits)	Specifies instruction mode.
Instruction Buffer Register (16 bits)	Holds current instruction op code, D,C,B, and A fields.
Clock Generator:	Defines basic machine cycle time, MCT.(167 ns)
TC Counter (3 or 4 bits):	Defines extended timing sequences of up to 16 clock times for current instruction.



INSTRUCTION READOUT

TOLERANCES (EXCEPT AS NOTED)			
DECIMAL ±		SCALE NONE	DRAWN BY [illegible] APPROVED BY D. H. SCHROEDER
FRACTIONAL ±	TITLE INSTRUCTION READOUT		
ANGULAR ±	DATE 3/3/71	DRAWING NUMBER FIGURE 1-15	

WC Counter (8 bits):	Employed in the generation of timing sequences in excess of 8 clock times with an upper limit of 341 μs.

1-39. CONTROL PANEL. The control panel provides a means of manual control of the central processor. An operator is provided four major functions:

a. Data entry.

b. Program debugging and alteration.

c. Fault diagnosis and isolation.

d. Maintenance.

1-40. ARITHMETIC UNIT

1-41. The arithmetic unit performs the basic arithmetic and logical operations and provides temporary storage for the results. Numbers are handled in both sign-magnitude and two's complement form.

1-42. ADDER. The adder is capable of handling two 16-bit numbers. Four-way gating at the adder inputs allows the adder inputs to be selected from one of four data sources, on either side of the adder.

1-43. MULTIPLIER. The multiplier is capable of multiplying two eight bit numbers. Longer numbers are multiplied by a distributed algorithm. The two registers M1 and M2 which feed the multiplier, incorporate four way gating at the input to allow the registers to be loaded from one of four data sources.

1-44. COMPONENTS AND THEIR FUNCTION. The principal components of the arithmetic unit together with a brief description of their primary function follows:

Component	Function
Data Pad Memory: Capacity: 64 words Word length: 16 bits Cycle Time: 167 nsec.	Buffer Storage
Data Pad Address Register (6 bits):	Holds the address of the word to be read from or written into Data Pad Memory
Adder (16 bits):	Performs logical and arithmetic operations

Multiplier (8 bits):	Forms a 16-bit product of two 8 bit signed numbers
Flag Register (16 bits):	Holds tag or sign bit associated with data in the individual registers.
A1,A2,I1,I2 Registers (16 bits):	Transient storage, outputs connected to various components as shown in Figure 1-14.

1-45. MOS MEMORY UNIT

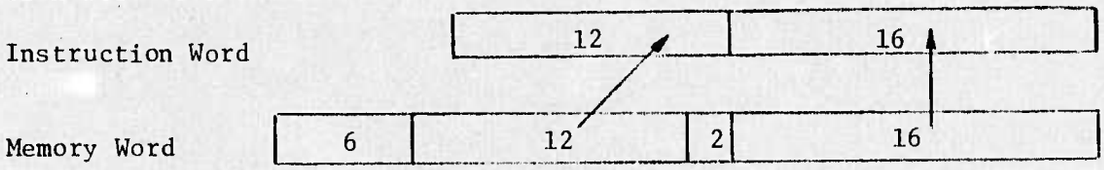
NOTE

The MOS Memory can be considered to be either an 18 bit memory of N words or a 36 bit memory of N/2 words. Throughout this section it will be thought of as N words of 18 bits.

1-46. SIZE. The standard MOS memory system is composed of 32,768 18-bit words and is used to store instructions and data. The memory can be expanded to 65,536 words.

1-47. ACCESS TIME. The access time of 18-bit words or 36-bit words in random mode is 500 ns. The access time to 18-bit words or 36-bit words in sequential mode is 167 ns. This reduction in access time is accomplished by four-way interleaving of the memory modules.

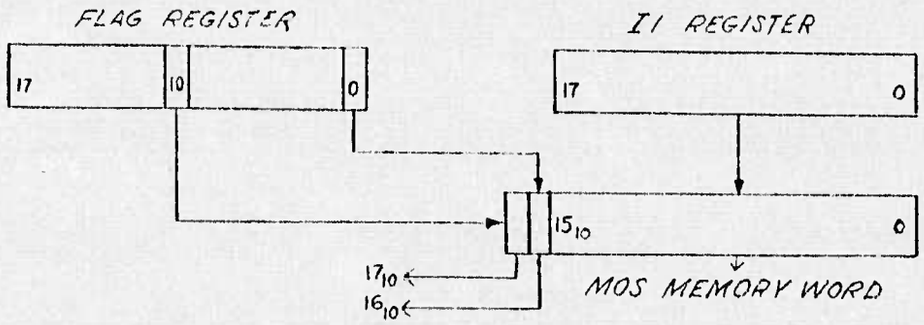
1-48. INSTRUCTION STORAGE. Since an instruction word is 28-bits in length, two adjacent 18-bit memory words are used to store a single instruction word as shown below:



NOTE

Throughout all the documentation of the Signal System it has been the convention to designate bits in MOS memory in decimal and bits in all other memories or registers in octal. Specifically, bit numbers without subscripts are in octal and those with subscript 10 are in decimal. The number of bits in a field, word or register, however, is always in decimal.

1-49. DATA STORAGE. Words of data up to 18-bits in length can be stored at a single memory location. The 18-bit memory word accommodates the typical 16-bit data word plus two flag bits. The bit assignment of words in memory is as shown below:



From a data transmission viewpoint, the composite of the 16-bit data words and the corresponding flag bits are handled simply as an 18-bit word.

1-50. TRANSFERS TO INSTRUCTION PAD. An instruction pad load, in sequential mode, of n instructions, where $1 \leq n \leq 64$, takes:

$$(0.167n+0.667)\mu s \text{ for 16K 36-bit memory or larger}$$

Thus, an instruction pad load of 64 instructions takes 11.3μs for 16K 36-bit memory or larger.

1-51. TRANSFERS TO DATA PAD. A data pad load, in sequential mode, of n double words, where $1 \leq n \leq 32$, takes:

$$(0.167n+0.667)\mu s \text{ for 16K 36-bit memory or larger}$$

Thus, a data pad load of 32 double words of data (32-bits each) takes 6μs for 16K 36-bit memories or larger.

1-52. EXECUTE FROM MEMORY. Execution of instructions directly from memory (without transfer to instruction pad) is accomplished in 667ns in random mode and 167ns when in sequential mode.

1-53. MOS TO AP-90 TRANSFERS. Information from the MOS memory may be obtained from or transferred to the AP-90's data pad unit at an 6 MHz rate, for 32-bit words. This will be true when the host data input of the AP is connected to the 32-bit memory input of the MP-32A and the MP-32A is in sequential mode. (See TMB8 for detailed information on the Array Processor.)

1-54. PHYSICAL COMPONENTS. The standard 16K 36-bit MOS memory system is composed of:

a. Eight 4Kx18-bit memory cards.

b. Three data register cards.

c. One control card.

d. One address register card.

e. Two output gating cards.

1-55. LOGICAL COMPONENTS AND THEIR FUNCTION. The principle logical components of the MOS memory system and their major functions are as follows:

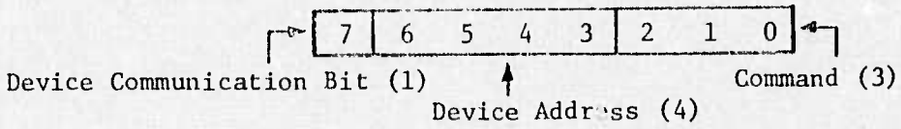
Component	Function
Memory module, 4K x 36-bits, expandable to eight modules	Storage for instructions and data
Address register (16-bits)	Samples the address presented at initiate time
Data input register (36-bits)	Samples the input data presented at initiate time
Data output register (36-bits)	Samples the stored output data at read enable time
TTL level controls	Input Controls: Enable left half, initiate, WRT/READ, data output strobe & refresh command. Output Controls: Refresh request.
Refresh logic	Supplies refresh request, refreshes automatically or upon command.

1-56. I/O INTERFACE UNIT

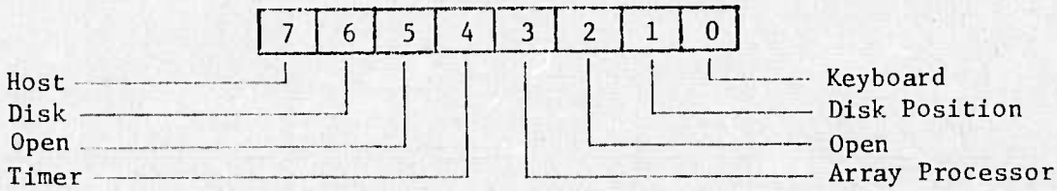
1-57. LOGICAL COMPONENTS. The principal components of this unit are shown in Figure 1-14. I/O operations are effected through the I/O instruction: Op-Code 14; D-field = 7; C, B and A-fields variable. Bits (0-7) of the instruction form a single I/O device address and variable command portion of the instruction which governs all transfer to and from peripheral devices. This portion of the I/O instruction is unique in the I/O device address portion only, bits (3-6). The unique address insures that only the selected device responds to signals from the MP-32A. The command portion, bits (0-2), can have the same or different meanings among the various devices.

1-58. REGISTERS AND BUSES. The unit contains the following five registers and buses:

a. The D Register contains 8 bits and holds bits (0-6) of the I/O instruction plus a device communication bit.



b. The S Register contains 8 bits and holds eight device service request bits:



c. The Device Input (DI) Bus contains 18 bits and provides a means of transmitting data from the various input devices to the E1 Register.

d. The E1 Register Bus contains 16 bits and provides a means of loading E1 from various sources (e.g. Device Input Bus and the ten possible outputs of Data Pad Bus).

e. The E1 Register contains 16 bits and is a data exchange register and holds data being transferred from input devices to the MP-32A.

1-59. PERIPHERAL CONTROLLERS. The MP-32A contains the following controllers as standard:

Controller	I/O Address(s)
User Stations 0,1.2.3	0,1,2,3
Display (Status)	4
Disk Storage Drive	10
Analog I/O	12
Host (Full Duplex)	13
Timer	14

User Station:

Command Code	Command
0	Read Keyboard entry onto DI Bus
1	Load A2 into X DAC, I2 into Y DAC and, after allowing a DAC settling time of 8 μsec, write Display for 24 μsec.
2	Erase Display

User Station (continued)

Command Code	Command
3	Set Display in Non-Store Mode
4	Clear Non-Store Mode
5	Light Keyboard STATUS REQUEST indicator (red light) to indicate user requests Analog I/O channel
6	Select Analog Channel. Clear red light and light STATUS LOCKOUT indicator (green light) to indicate user is connected to the Analog Channel.
7	Deselect Analog Channel, clear red and green lights

Display (Status):

Command Code	Command
0	Display Status→DI If DI(STN)=0, station display unbusy
1	A2→XDAC I2→YDAC In SLEW MODE allow 2 μs per output
2	SLEW MODE begins (SLEW MODE ends if no output for 32μs.

Disk Controller:

(1) Assembles serial data from the disk for parallel transfer to the E1 Register Bus

(2) Accepts parallel data from the I1 Register and converts it to serial form for recording on the disk

(3) Generates a check sum.

(4) Executes the following commands:

Command Code	Command
0	Read disk data onto DI Bus
0	Write contents of I1 onto disk
1	Read disk status word onto DI Bus and clear S(6) (with DI→E1 command)
2	Execute command in I1
3	End transfer
4	Reset head assembly to cylinder zero

Disk Controller (continued)

Command Code	Command
5	Read position status word onto DI Bus
6	Not used
7	Select disk drive number from I1 (bits 10,11,12)

NOTE
In write commands, bit (7) of the I/O instruction is a "1".

Analog Controller:

Command Code	Command
0	Not used
1	A/D-DATA→DI (next channel determined by I1)
2	Not used
3	Connect TIMER SYNC to A/D SYNC
4	Disconnect A/D and D/A SYNC
5	START A/D conversion (asynchronous)
6	Connect TIMER SYNC to D/A SYNC
7	I2→D/A (channel determined by I1)

Host Controller (Full Duplex):

Command Code	Command
0	HOST DATA→DI
0	I1→HOST DATA
2	I1→HOST COMMAND

Timer Controller

Command Code	Command
0	TIMERDATA→DI
1	START Synchronization
2	END Synchronization
3	I1→TIMER DATA

The formats for the status words referenced in several of the commands listed above are detailed in Section 4-160 and in TMA4.

1-60. INSTRUCTIONS

1-61. EXECUTION SEQUENCE

1-62. In general, blocks of instructions are transferred from the main MOS memory to the Instruction Pad Memory to be executed in a loop mode. An instruction is read from the Instruction Pad Memory into registers as shown in Figure 1-15. The instruction may be one of eight general types which may have different execution times. The Control Unit effects the execution of the instruction and, if the required time of execution is one machine cycle, reads the next instruction from the Instruction Pad Memory at the next clock time. This process continues in accordance with the instruction sequence stored in the Instruction Pad Memory.

1-63. FORMAT

1-64. The 28 bits used to specify an instruction are grouped into eight FIELDS as shown and described below:

33 32 31 30	27 26	25 24 23 22 21 20	17 16 15 14	13 12 11	10 7 6	5 4 3	2 1 0*
13 12 11 10	7 6	5 4 3 2 1 0	17 16 15 14	13 12 11	10 7 6	5 4 3	2 1 0**
T FIELD	MODE	J FIELD	OP CODE	D FIELD	C FIELD	B FIELD	A FIELD

NOTE:

* octal
** octal compatible with control panel

a. The T-FIELD is the repeat number and is decremented at each clocktime (125 ns) during execution until it is zero. In general, the instruction is performed one more time than shown in the Repeat Number. In certain instructions the most significant bit of the T field, T(13), is used to control the entering or terminating of sequential mode.

b. MODE is the instruction mode and specifies the overall meaning of the fields OP-CODE, D-FIELD, C-FIELD, B-FIELD, A-FIELD.

c. J-FIELD is the instruction address of normal successor instruction.

d. OP-CODE is the operation type. After the instruction MODE has been selected, the set of operations that can be performed in parallel is determined by the Operation Type or OP-CODE.

e. D,C,B,A are parallel operations. Each of these three-bit fields permit the selection of one out of eight possible operations as defined by MODE and OP CODE.

1-65. INSTRUCTION SET

1-66. TMA4 provides D,C,B and A field definitions for each OP-CODE in combination with the applicable MODE or MODES and incorporates a number of tables organized by instruction subset which relate specific field codes with unique hardware operations. These tables in combination with the accompanying text of TMA4 provide field coding information sufficient for formulating individual machine language instructions.

1-67. SUBSETS

1-68. The MP-32A instruction set is comprised of eight instruction subsets. An instruction subset corresponds to a number of allowable hardware options exhibiting a large measure of commonality. A list of the instruction subsets along with allowable operations per single instruction appear in Table 1-1.

1-69. FEATURES FOR PROGRAMMING

1-70. Certain features imbedded in the machine design which aid the programmer in the utilization of the equipment are as follows:

a. Useful complement of instructions

b. Versatile addressing capability

c. Registers incorporate 4-way input gating with parallel access to each bit position

d. Automatic adder overflow detection

e. Control panel which provides the facility for program debugging.

Table 1-1. Instruction Subsets....Page 1 of 3

Instr. Subset Name	Clocks		Mode	Op Code $_8$	Allowable Operations/Single Instruction
Single Bit Decode	Reg. Opr.	Spec. Mult.			
	1	1-4	0	0	Register Operations, Special Functions and Special Multiply
Succinct Adder	1		0	1-5	1 of 7 Adder Operations and Load of 3 registers, or special functions in lieu of loading registers.
	1		1	1-5	All mode-0 operations with adder output to memory address register (CA)
	1		2	1-4	1 of 7 adder operations, and load of 1 register, and set data pad address register (PA) = bits 0-5 of instruction
Full Adder	1		0	6-13	1 of 63 adder operations, and load of 2 registers or special functions in lieu of loading registers
	1		1	6-13	All mode-0 operations with adder output to memory address register (CA)
	1		2	6	1 of 63 adder operations and set data pad address register (PA)
	1		2	7,10,11,12	1 of 63 adder operations with the adder destination selected by op code and set pad address register (PA)
	1		2	13	1 of 63 adder operations, adder outputs to memory address register and set write mode flip-flop MOS memory control

*OP-CODE	ADDER DESTINATION
7	A1
10	A2
11	I1
12	I2

Table 1-1. Instruction Subsets....Page 2 of 3

Instr. Subset Name	T	Mode	Op Code 8	Allowable Operations/Single Instruction
I/O & Control	Variable	0	14	Load Data Pad, or MOS Memory, or Instruction Pad or E2 Register or D Register or Instruction Buffer Register, TC Counter, Instruction Pad Address Register and Instruction Mode Register, or generate variable timing sequences (wait time). In OPC 1404, 1414 & 1424 memory execute mode is entered.
	Variable	1	14	Same as mode-0 except for load of D Register where device is left enabled.
Bit Select Conditionals	Test F: 0 / Test P	0	15	Test a "0" in single bit of Flag Register, or A2 Register, or I1 Register or E1 Register. Condition met:**NIA = Jump Address = (B&A) Fields. Condition not met: NIA address in normal sequence (J-Field)
	0	1	15	Same as mode-0 except a test for "1" is performed. Condition not met: NIA (J-Field) Condition met: NIA = Jump Address = (B&A Fields)
	Variable	2	15	Sequential test for a "1" in each bit in Flag Register, or A2 Register, or I1 Register, or E1 Register, Condition met: NIA = Jump Address = (B&A) Fields. Condition not met: NIA = next in sequence (J-Field)

**NIA ≡ Next Instruction Address.

Table 1-1. Instruction Subsets....Page 3 of 3

Instr. Subset Name	T	Mode	Op Code 8	Allowable Operations/Single Instruction
Logical Operations	Variable	0	16	Logical Operations: (1) Single condition tests resulting in program jumps or a continuation of normal instruction sequence. (2) Logical actions (sets, clears, shifts, transfers & swaps). (3) Single condition tests in combination with logical actions.
	Variable	1	16	Same as mode-0, except in general under mode 1, tests for equality become tests for inequality and instructions which set a bit=1, set it=0.
	Variable	2	16	Sequential test for a "1" in each bit of the S Register. Condition met: NIA = Jump Address (B&A Fields). Condition not met: NIA (J-Field)
Memory Controls	0	3	NA	Load Memory Address Register with the contents of the Instruction Buffer Register
Linkage Operations	2	0	17	Writes return address specified by the D & C fields in the J Field of the instruction at location J. Executes J+1 as next instruction
	2	1	17	Same as mode 0 except that the next instruction to be executed is defined by the Jump Address (B&A Fields)
	2	2	17	Same as mode 0 except that in addition, PA is set = B&A Fields

1-71. SYSTEM CONFIGURATION

1-72. GENERAL

1-73. As a component of a CHI Signal System, the MP-32A may serve as the only processor, or in systems containing an Array Processor as pre-processor, post-processor and control processor. The Signal System, as shown in Figure 1-17, includes the following possible components: MP-32A, one to eight Model 114 Disk Drives and one to four Model 100 User Stations. Communication between the MP-32A and the various peripheral devices is accomplished under program control. The controllers described in Section 1-59 provide the electronic link between a peripheral device and the MP-32A. For a more detailed description of the Signal System see TMB1.

1-74. AP-90 ARRAY PROCESSOR

1-75. The AP-90 is housed in the same type of cabinet as the MP-32A and contains its own power supplies. In systems containing both units, one side panel is removed from each unit and the units are bolted together. Signals from one unit to the other are transmitted via cables connected between the units.

1-76. The AP-90 functions as a fixed point array processor and a floating point arithmetic unit. The MP-32A transmits data to and from the AP-90 and tells it which operation to perform. The operations are either fixed point, floating point or data transfer operations. The fixed point format is 16-bits, 2's complement, with complex words made up of two 16-bit words. The floating point format is 8-bit 2's complement scale and 24-bit 2's complement mantissa. For a detailed description of the AP-90 see TMB3.

1-77. MODEL 114 DISK DRIVE

1-78. Up to eight Model 114 Disk Drives may be attached to a MP-32A. These drives, as shown in Figure 1-18, are 40.25 inches high, 30 inches wide, 24 inches deep and weigh 350 poinds. These dimensions include all panels and top covers. For a detailed description of the Model 114 Disk Drive see TMB1.

1-79. The Model 114 Disk Drive is a random-access memory device for mass data storage. The Disk Packs for the disk are IBM 2316 or CHI approved equivalent. Each pack has 203 recording tracks per disk surface and 20 recording surfaces. The pack speed is 2400 rpm for a maximum rotational latency of 25 ms and an average of 12.5 ms. The track to track access time is 12 ms, the average access time 35 ms, and the full stroke access time 65 ms.

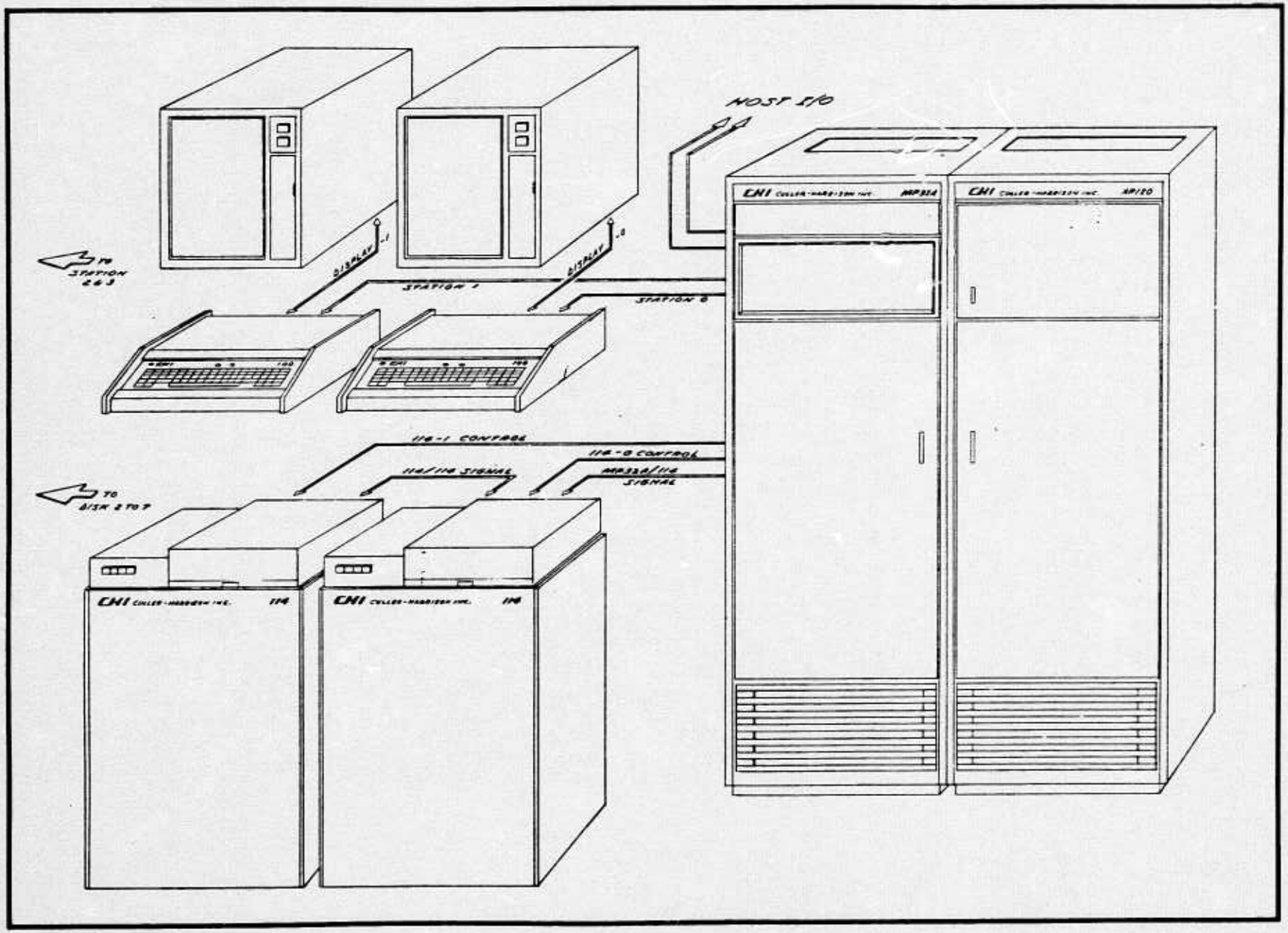
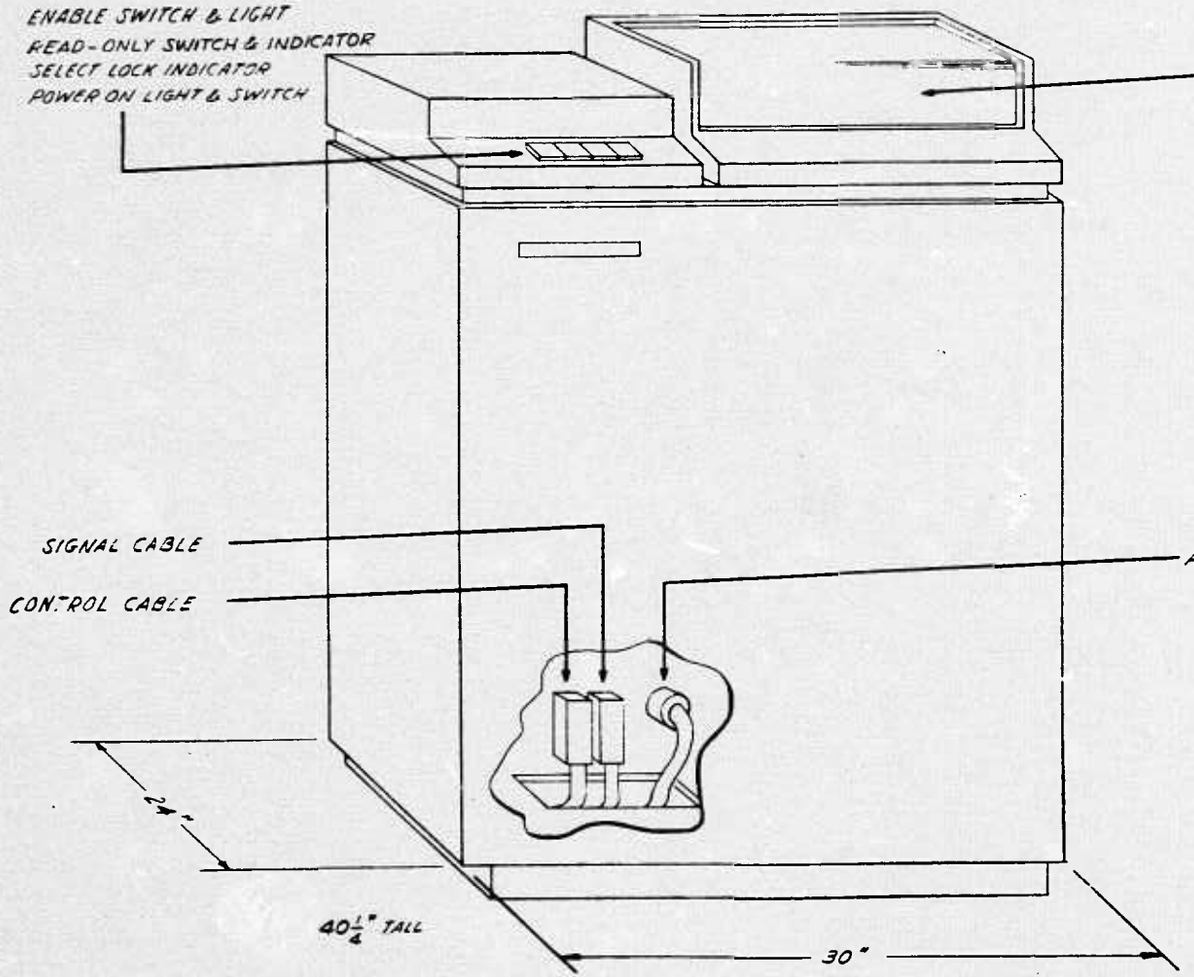


FIGURE 1-17 CHI SIGNAL/II SYSTEM



DISK STORAGE DRIVE - MODEL 114

DATE	BY	REVISION RECORD	APPR.	DR.	CK.

TOLERANCES (EXCEPT AS NOTED)			
DECIMAL ±		SCALE NONE	DRAWN BY Gill
FRACTIONAL ±	TITLE DISK STORAGE DRIVE - MODEL 114		
ANGULAR ±	DATE 3/31/71	DRAWING NUMBER FIGURE 1-18	

1-80. The disk write format is such that 3,200 18-bit words or 3,600 16-bit words may be written per track. This gives a pack capacity of 12,992,000 18-bit words or 14,616,000 16-bit words.

1-81. MODEL 100 USER STATION

1-82. The Model 100 User Station consists of a CHI Model 100 Keyboard and a Tektronix Model 611 Storage Display Unit.

1-83. MODEL 100 KEYBOARD. The Model 100 Keyboard is a modified Microswitch keyboard housed in a custom made cabinet. The approximate overall dimensions of the cabinet are 5 inches high, 18 inches deep, and 19 inches wide. The unit, as shown in Figure 1-19, incorporates the following controls and indicators: AC power switch, DC power indicator, request indicator, and lockout indicator. The unit provides signal and power connectors, which include an AC power outlet for providing power to the display unit.

1-84. The keyboard layout, codes and color are shown in Figure 1-20. A detailed description of key functions can be found in TMA2. The codes are the octal representation of the 8-bit binary code placed on the data lines to the MP-32A.

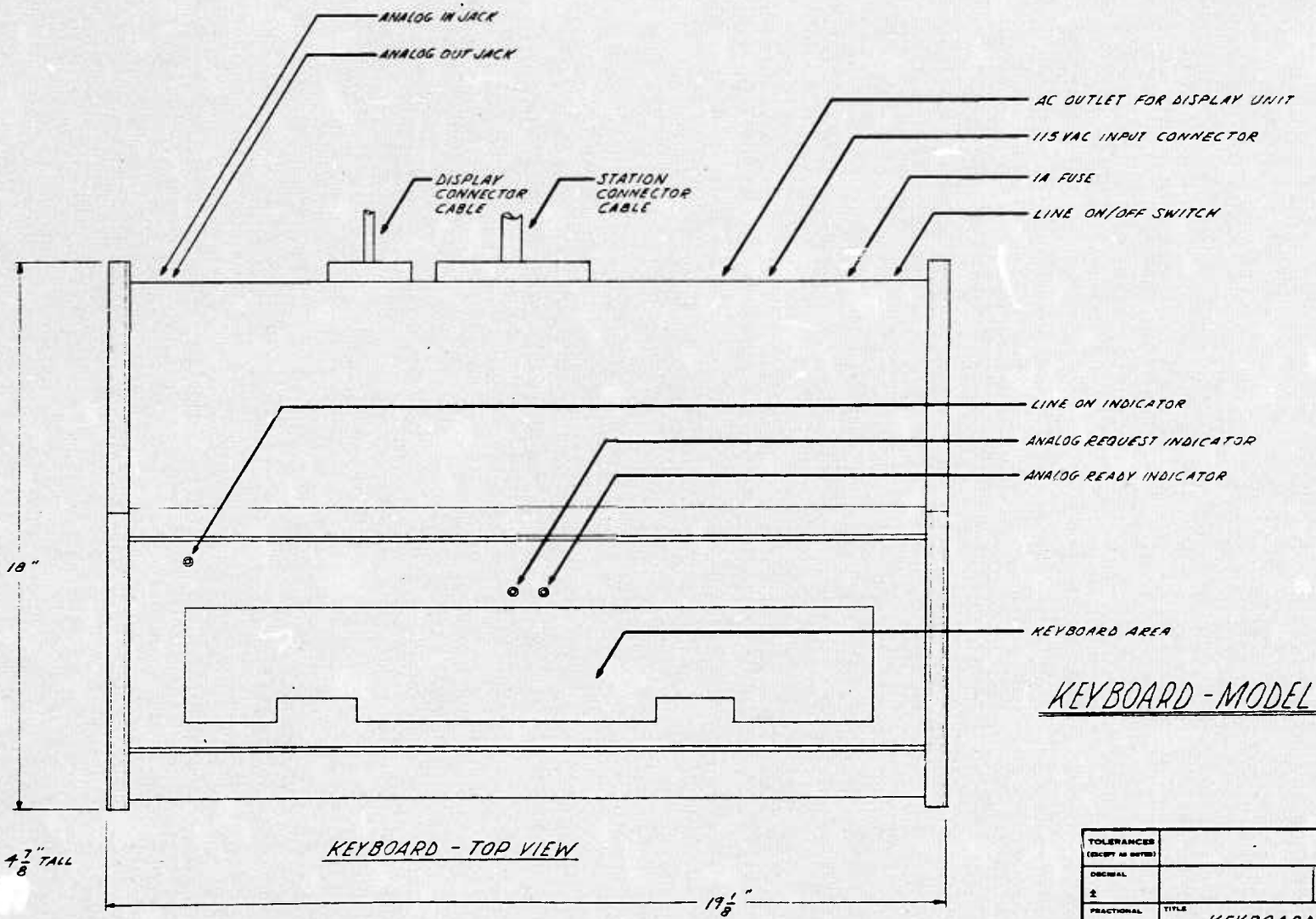
1-85. Up to four keyboard/display stations can be attached to the MP-32A. A particular keyboard is selected through a station address. The keyboard controller of the MP-32A accepts the key output data from the selected keyboard.

1-86. TEKTRONIX MODEL 611 STORAGE DISPLAY UNIT. The Tektronix Model 611 is a data storage and display instrument. The unit, as shown in Figure 1-21, is housed in a vinyl-coated aluminum cabinet with the following overall dimensions: 11-7/8 inches high, 11-5/8 inches wide and 22-3/8 inches long. The instrument weighs 50 pounds and incorporates VIEW and ERASE front-panel push button controls in addition to signal and power connectors.

1-87. The Tektronix's Model 611 is used in the MP Signal System to provide the user with an interactive display capability. The unit incorporates a high resolution, direct view storage tube. To perform the display function the unit must be supplied X and Y deflection signals and a Z axis control signal.

1-88. In multiple User Station systems a display unit must be continuously ready for use, but actual use is generally infrequent. The unit incorporates a holding-mode allowing the life expectancy of the tube to be extended without degrading the system performance.

1-89. The unit incorporates two front panel controls for local operation, ERASE and VIEW. The ERASE control provides the means of erasing any previously stored information. The VIEW control switches the instrument



KEYBOARD - TOP VIEW

KEYBOARD - MODEL 100

TOLERANCES (EXCEPT AS NOTED)			
DECIMAL ±		SCALE NONE	DRAWN BY RJD.
FRACTIONAL ±	TITLE KEYBOARD - MODEL 100		APPROVED BY J.F. SCHROTH
ANGULAR ±	DATE 3/31/71	DRAWING NUMBER FIGURE 1-19	

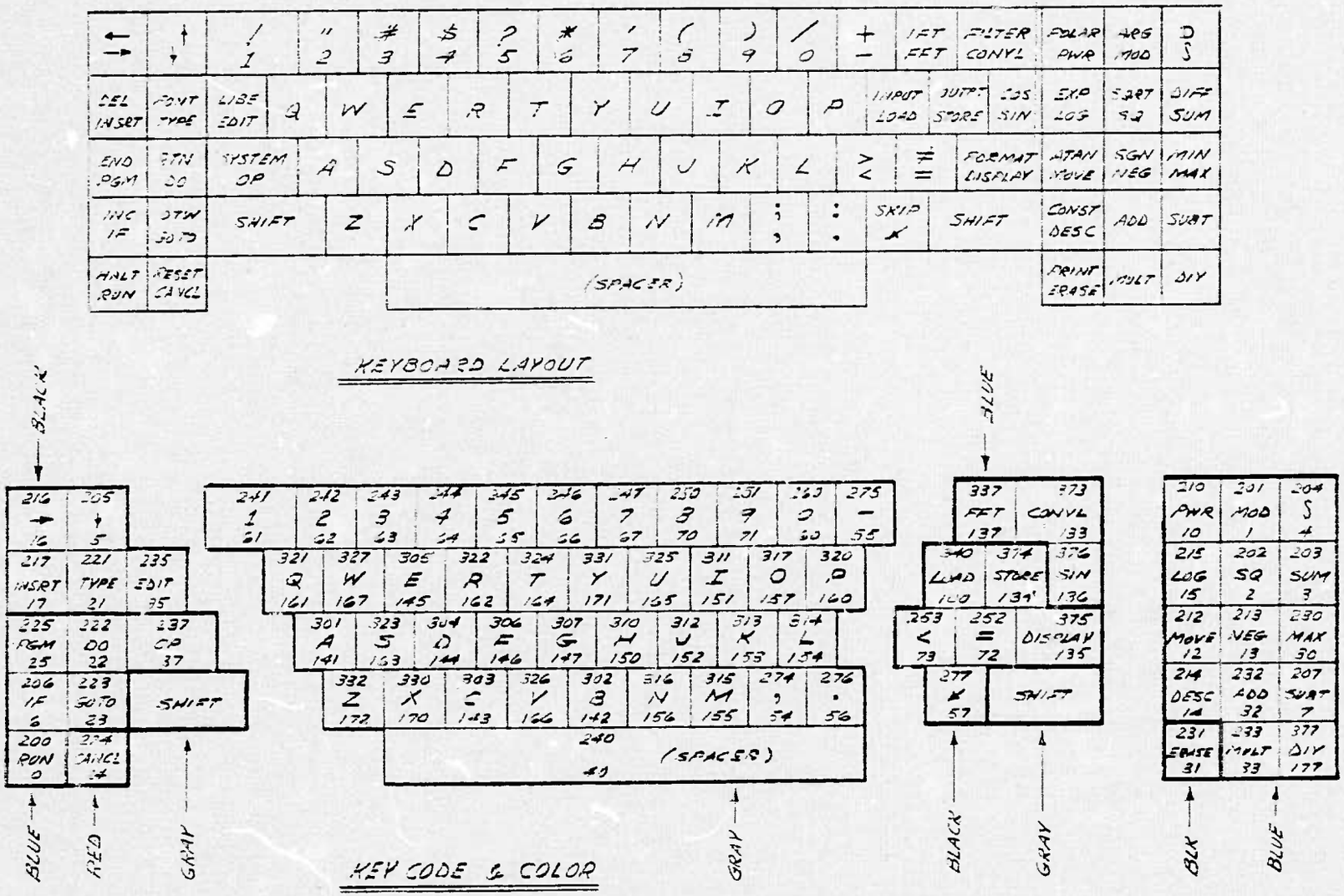
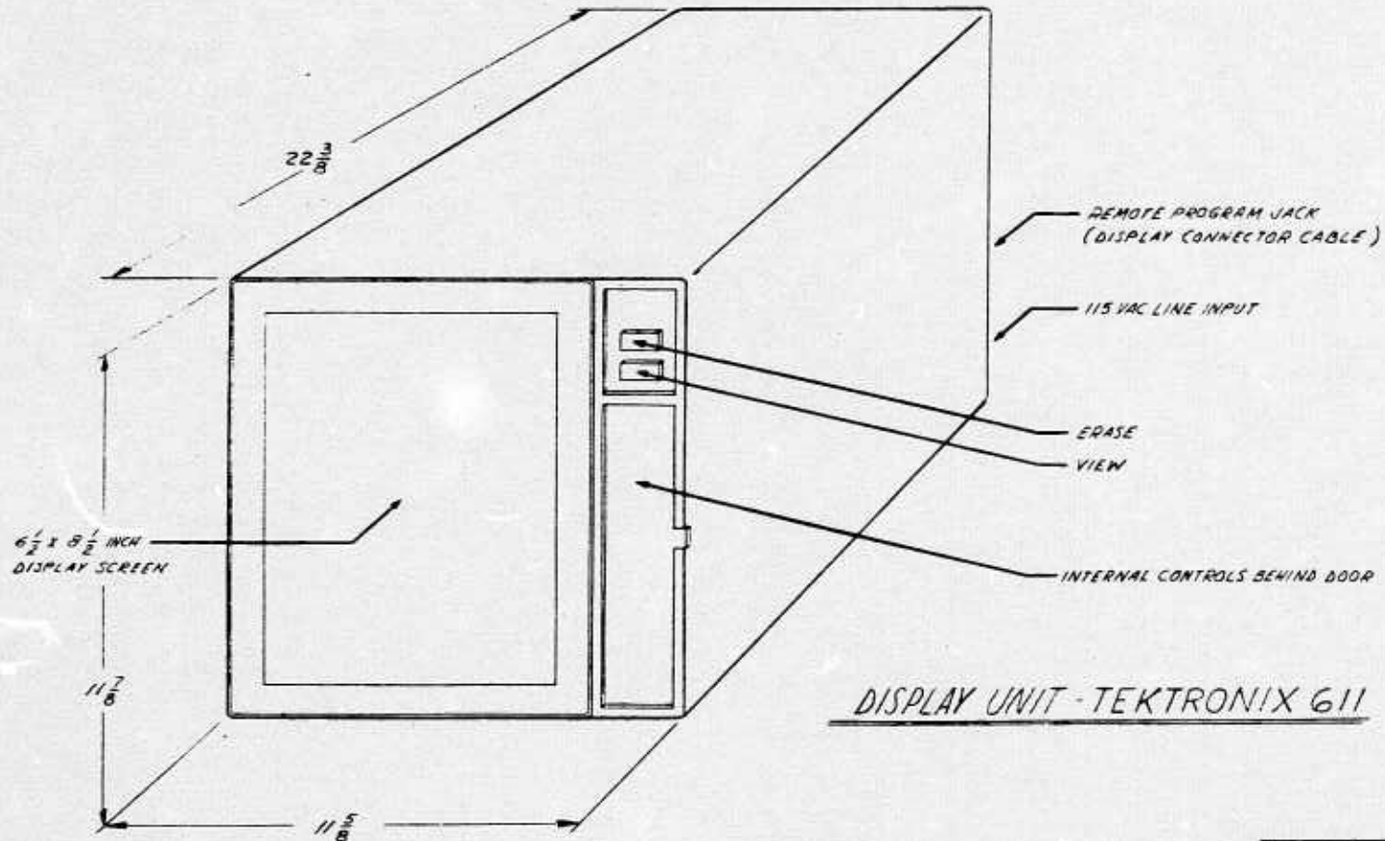

KEYBOARD LAYOUT
KEY CODE & COLOR
BLACK
BLUE
RED
GRAY
(SPACER)
SHIFT

FIGURE 1-20



DISPLAY UNIT - TEKTRONIX 611

TOLERANCES (UNLESS OTHERWISE SPECIFIED)			
DECIMAL ±		SCALE NONE	DRAWN BY
FRACTIONAL ±	TITLE DISPLAY		
ANGULAR ±	DATE 3/31/71	DRAWING NUMBER FIGURE 1-21	

from a holding-mode to a view mode. The VIEW switch is illuminated when in the holding-mode. The display switches from view to holding-mode about 1 minute after the last writing function or after a view mode has been initiated by the front panel VIEW switch. In a remote mode, the unit is under program control and the input signals are supplied on command from the MP-32A. The status of a display unit is determined programmatically. The input signals are supplied through four unique commands:

a. Write display
b. Erase
c. Set Non-store mode
d. Clear Non-store mode.

1-90. LEADING PARTICULARS

1-91. The logistic characteristics of the MP-32A are given in Table 1-2.

Table 1-2. Leading Particulars

Description	Characteristics
Power Requirements	105-132 Vac, 60Hz ± 5% 30 amperes at 115 Vac
Dimensions and Weight	Height:61.5", Width:24",Depth:28.5" Net Weight = ~350 pounds
Source Plugs	Bryant 3 prong #3330 receptacle or equivalent (@ 30 amps)
Ventilation	Intake Location: Front Exhaust Location: Top Airflow: 265 cfm exhaust
Service Accesses	Front door, rear door, control panel door, side panels
Cable	15' AC cable supplied with unit
Heat Dissipation	~9000 Btu/hour

1-92. CAPABILITIES AND LIMITATIONS

1-93. Capabilities and limitations of the MP-32A are given in Table 1-3.

Table 1-3. Capabilities and Limitations

Description	Characteristics
Start-Up Time	~40 seconds
Operating Temperature	0°C (32°F) to +50°C (122°F) ambient
Operating Humidity	10% to 80% with no condensation
Storage Conditions	Temperature=0°C(32°F) to 75°C(167°F) Humidity: 0% to 90% with no condensation

II. INSTALLATION

2-1. SCOPE

2-2. This section contains information pertinent to planning the installation of a MP-32A Macroprocessor and receipt, unpacking and installation of the processor.

2-3. INSTALLATION PLANNING

2-4. Table 2-1, Physical Planning Notes, gives the information necessary for planning the physical, environmental and power requirements of the site. A 15' AC power cable is supplied with the unit and requires a 3 prong, Bryant #3330 receptacle or equivalent service power outlet.

2-5. Figure 2-1 gives the installation clearances required for cables, air flow and servicing the processor.

2-6. LOGISTICS

2-7. RECEIVING & VISUAL INSPECTION

2-8. The MP-32A is shipped as a complete unit with a slip-over plastic cover. The unit is on 360° casters and can, therefore, be rolled into position.

2-9. The following is a list of checks which should be made subsequent to unpacking the equipment:

a. Insure that all packaging materials have been removed.

b. Inspect the processor including the memory modules for any signs of shipping damage.

c. Check the intra-cabinet cables to assure that all cable terminations, paddles and plugs, are properly seated.

d. Check the main logic and I/O logic planes to assure that all integrated circuit modules are properly seated in the mating socket and that no pins are shorting.

e. Check the complete unit for mechanical integrity.

f. Check the resistance between the VCC and ground planes and main logic and I/O logic planes with a volt-ohm-meter (Simpson 260 or equivalent) and then reverse the leads. Both readings should be greater than 0.5 ohms.

Table 2-1

INTRODUCTION

This document contains information helpful in ordering and planning for the installation of the MP-32A.

DOCUMENTS

The following documents contain additional MP-32A information:

Signal System Hardware Documentation Manuals:
- B1 - System Configuration
- B2 - Model 32A Macroprocessor
- B4 - Maintenance

PHYSICAL DIMENSIONS

Height	61.5"
Width	24"
Depth	28.5"
Weight	~350 pounds

(Dimensions include all panels, top covers and casters)

SERVICE CLEARANCES

Front to Rear	4'
Sides	4'
Top	2'

INSTALLATION CLEARANCES

Space between MP-32A and AP-120....None

HEAT DISSIPATION/AIR FLOW

Dissipation(max)	~9000 Btu/hr.
Air Flow	7500 cu.in/sec.

ENVIRONMENT (Operating)

Temperature	0°C to 50°C
Rate of Temp. Change	10°/hr.
Relative Humidity	

AC POWER

Input Power	105-132vac,30amps
Input Frequency	60Hz±5%
Operating Current	22amps
Starting Current	<30amps

DC POWER

All dc voltages are supplied by internal power supplies.

AC POWER CABLES

If no peripheral devices are ordered with the Macroprocessor, the only cable required is a 115 vac power cable. This cable, 15' in length, is supplied with the unit.

For systems including Model 114 Disk Drives, two 15' ac power cables are supplied to connect from ac source to Macroprocessor and from Macroprocessor to Disk 0. One 8' ac cable, which connects between drives, is supplied for each additional drive.

DC CABLES

For systems including Model 114 Disk Drives, one 15' dc cable, which connects between Macroprocessor and drive, is supplied for each drive.

SIGNAL CABLES

For each Model 100 User Station ordered with the unit, a 50' signal cable is supplied. Longer cables of up to 100' may be purchased from CHI.

For systems including Model 114 Disk Drives, one 15' signal cable, which connects between Macroprocessor and drive is supplied and one 8' signal cable, which connects between drives is supplied for each additional drive. A terminator must be plugged into the last unit.

SOURCE PLUGS

The 115 vac requires a 3 prong, Bryant #3300 recepticle or equivalent (@30 amps). The 208 vac 3Ø requires a 4-prong Bryant #7410 recepticle or equivalent (@ 20 amps).

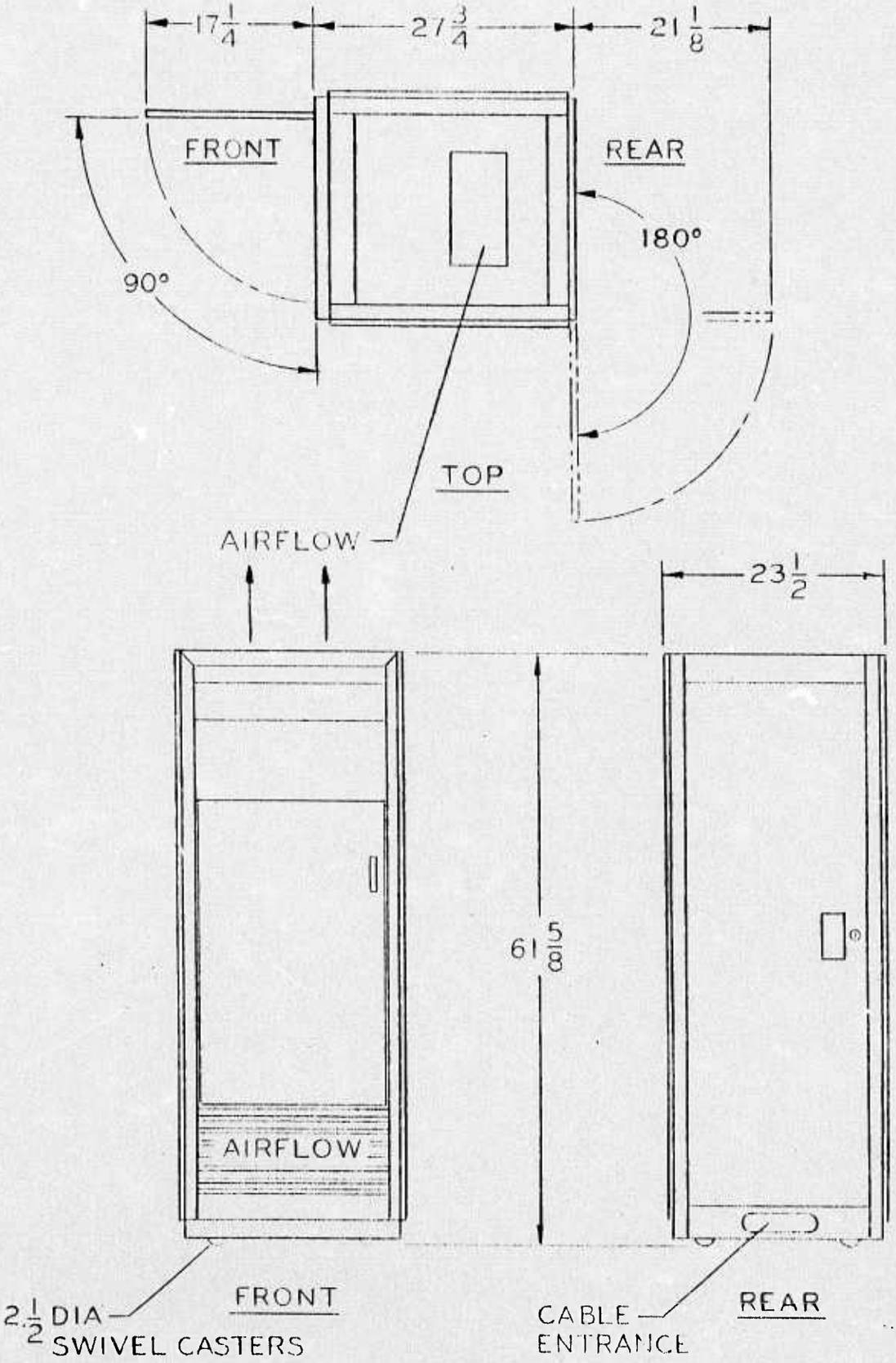


FIGURE 2-1 MP-32A INSTALLATION CLEARANCES

III. OPERATION

3-1. SCOPE

3-2. This section provides data in the form of text, illustrations and tables necessary to powering-up and powering-down the MP-32A. (System operation is described in detail in TMB1.) In this section the Controls and Indicators available are first described and then their use is explained for powering-up and powering-down the MP-32A.

3-3. CONTROLS AND INDICATORS

3-4. STATUS PANEL

3-5. The status panel, shown in Figure 3-1 is located directly above the control panel and in back of a hinged cover plate. The status panel incorporates SYSTEM POWER ON/OFF switch, +5VDC, & +19.7VDC and +23.2VDC POWER SUPPLY STATUS indicators (lit when supplies are on), DISK STATUS light (disk operative if lit--disk inoperative if not lit), and an Elapsed Time Meter. The status panel provides a gross indication as to equipment readiness subsequent to setting the system power switch in the "ON" position. Status panel checks need only be made following Power-Up or in the event of malfunctions. For this reason, the panel indicators have not been made readily viewable. Access to the system power switch and the equipment status indicators is accomplished by applying pressure to a point on the lower half of the cover plate.

3-6. CONTROL PANEL

3-7. Control Panel Implementation. The control panel incorporates a single 3-position toggle switch in combination with a number of pushbutton switches and indicator lights. The switches and indicator lights are logically arranged and identified with descriptive labels as shown in Figure 3-2.

3-8. Major Functions. The control panel provides an operator the capability of:

a. Manual load -- data or instructions

b. Loading the dead-start Disk Loader (manual to automacic operation)

c. Single instruction execution

d. Breakpoint operations

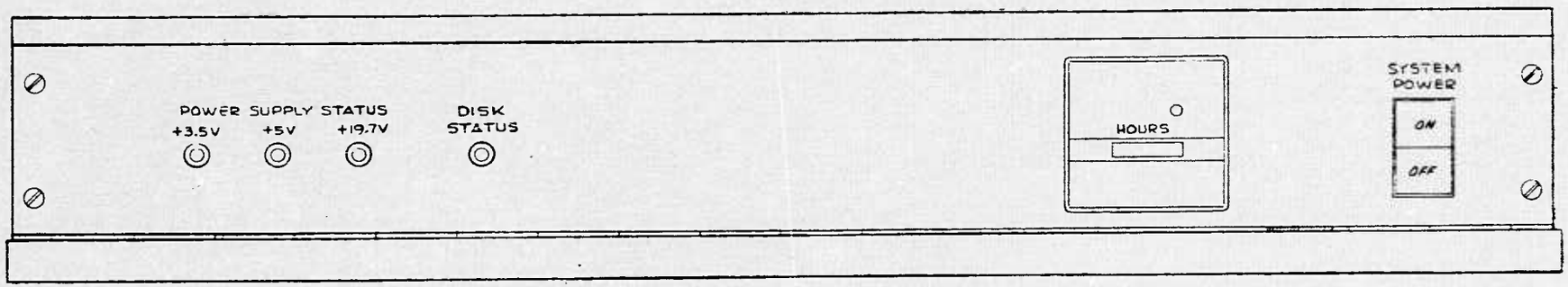
e. Displaying contents of all registers.

The panel controls and indicators, along with the function of each, are shown in Tables 3-1 and 3-2.

3-9. Control Panel Operation. Panel load operations require two steps:

a. Selectively loading the E-Register from a bank of switches, and

b. Selecting and actuating a particular switch.



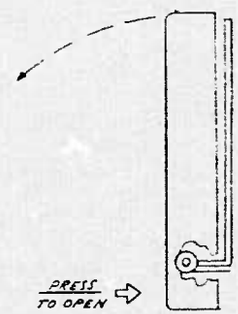
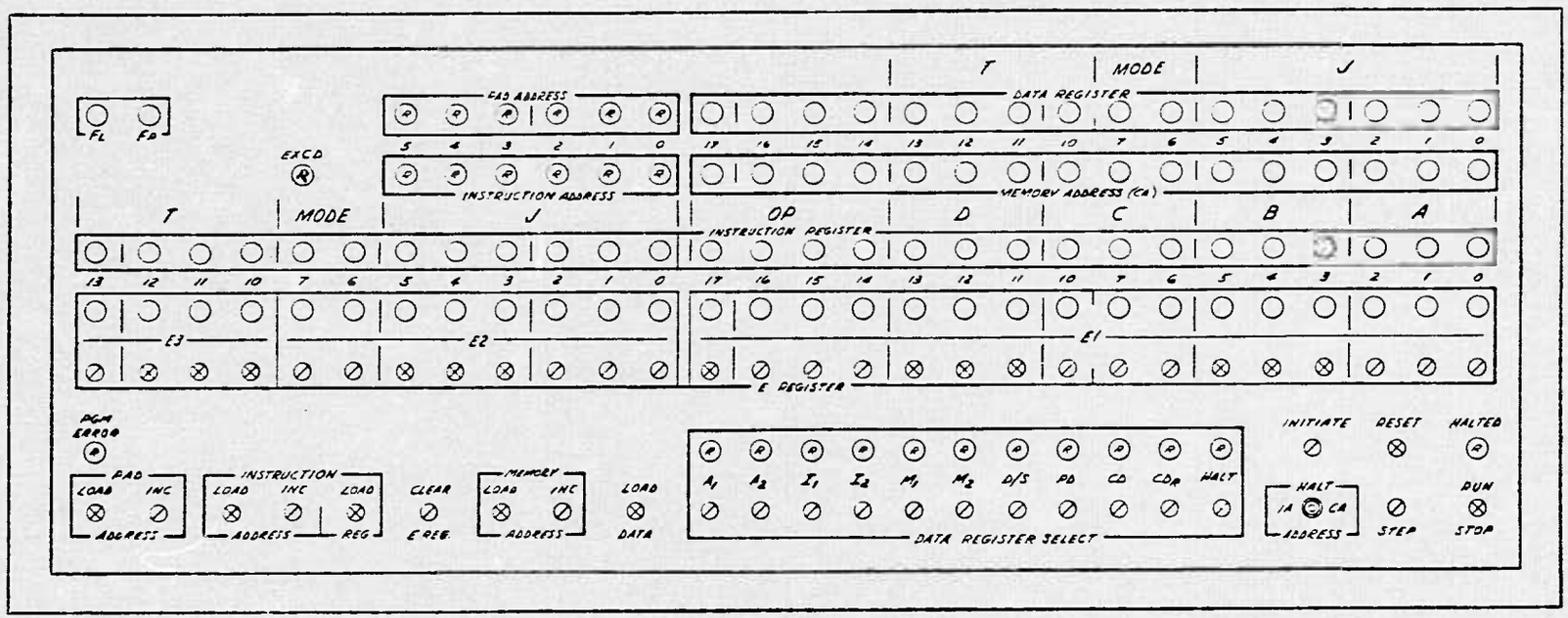


Figure 3-1. Status Panel



WHITE INDICATOR LAMP

RED INDICATOR LAMP

BLUE INDICATOR LAMP

BLACK PUSH BUTTON SW.

WHITE PUSH BUTTON SW.

3 POSITION TOGGLE SW.

Figure 3-2. Control Panel

Table 3-1. Control Panel Controls

Panel Designation	Switches	Function
PAD ADDRESS	LOAD	Transfer the contents of E2 to Pad Address Register and I2
	INC	Pad Address + 1 to Pad Address Register
INSTRUCTION ADDRESS	LOAD	Transfers the contents of E2 to Instruction Address Register & I2
	INC	Instruction Address + 1 to Instruction Address Register
INSTRUCTION REG	LOAD	Transfers contents of E to Instruction Register, I2 and I1
MEMORY ADDRESS	LOAD	Transfers the contents of E1 to MOS Address Register and I1
	INC	MOS Address + 1 to MOS Address Register
E REGISTER		Change state of corresponding E-bit
CLEAR E REG		Replace information in E with zeros
DATA REGISTER SELECT		Each switch (A1, A2, I1, I2, M1, M2, D/S, PD, CD, CDR, HALT) selects the corresponding register to be displayed in the Data Register.
LOAD DATA		Transfers the contents of the E1 Register to the register selected by the Data Register select controls; may also set I1
HALT ADDRESS Switch	IA Position	Breakpoint on Instruction Address: Computer stops on address set in Halt Register
	CA Position	Breakpoint on MOS Address: Computer stops in address set in Halt Register
	CENTER Position	Halt Register may be used for program SYNC.

Table 3-1. Continued

Panel Designation	Switches	Function
RUN-STOP		Starts and stops the computer
INITIATE		Sets bootstrap mode
RESET		Clears all registers in the computer except: (1) Halt, (2) Instruction, and (3) Data Select
STEP		Executes a single instruction each time the switch is depressed.

Table 3-2. Control Panel Indicators

Panel Designation	Function
PAD ADDRESS	Displays current address of Data Pad Memory
INSTRUCTION ADDRESS	Displays current address of Instruction Pad Memory
MEMORY ADDRESS (CA)	Displays current address of MOS Memory
INSTRUCTION REGISTER	Displays contents of Instruction Register
E REGISTER	Displays contents of E Register
DATA REGISTER	Displays the contents of the Register selected by the Data Register Select control
F_L	Displays contents of left most flag bit of the register selected by the Data Register Select control. When the data register selected is CD_R, F_L lights if the program is in sequential mode. When the data register selected in D/S, F_L lights if IØDRDY=1.
F_R	Displays the contents of the right most flag bit of the register selected by the Data Register Select control. When the data register selected is CDR, F_R lights if the MOS memory is performing a write operation. When the data register selected is D/S, F_R lights if APDRDY=1
PGM ERROR	Not presently used.
EXCD	Indicates that the program is executing out of the MOS memory.
HALTED	Indicates that the machine is stopped
DATA REGISTER SELECT	Indicates which of eleven registers has been selected to be displayed in the Data Register Indicators. (When D/S is selected, both registers are displayed -- D in the eight highest bits and S in the lowest eight bits.)

All other panel operations, exclusive of dead-start, involve only the selection and actuation of a particular switch. The dead-start operation involves the actuation of the RESET, INITIATE, and RUN switches in a particular sequence which is explained in paragraphs 3-15 thru 3-17.

3-10. A panel operation is executed in one panel cycle time. A panel cycle is generated by any one of the following controls, when the computer is stopped:

a. PAD ADDRESS LOAD

b. PAD ADDRESS INC

c. INSTRUCTION ADDRESS LOAD

d. INSTRUCTION ADDRESS INC

e. INSTRUCTION REG LOAD

f. MEMORY ADDRESS LOAD

g. MEMORY ADDRESS INC

h. RESET

i. RUN-STOP

j. STEP

k. INITIATE

Note that the computer may be stopped in two ways:

a. RUN/STOP switch

b. Breakpoint operation through the HALT logic.

3-11. Manual Control. The use of the various controls and indicators is described in Sections 4-49 through 4-67.

3-12. OPERATING INSTRUCTIONS

3-13. POWERING UP

3-14. Access to the SYSTEM POWER switch and STATUS indicators was described in Section 3-5. Verify that all interconnecting cables are properly installed. Position Power Switch to the "ON" position. After a delay of approximately 30 secs., the state of the indicators should be as follows:

a. POWER SUPPLY STATUS, and DISK STATUS indicators should be ON

The state of the indicators provide a gross indication of equipment readiness subsequent to the application of system power.

3-15. DEAD-START

3-16. Two independent dead-start bootstrap programs, Disk Loader and Host Loader, are available to the operator. The bootstrap programs are stored in a 768 bit bipolar Read-Only-Memory (ROM).

3-17. Disk Loader. The Disk Loader is entered into the machine as follows:

a. Press RESET

b. Press INITIATE

c. Press RUN.

3-18. Host Loader. This routine, which waits for an interrupt from the Host, is executed as follows:

a. Press RESET

b. Press INITIATE

c. Press INC IA

d. Press RUN.

3-19. POWERING-DOWN

3-20. To power-down the equipment, press the Power Switch on the status panel to "OFF". The system will automatically sequence down and all equipment attached to the system will have power removed.

IV. PRINCIPLES OF OPERATION

4-1. SCOPE

4-2. This section discusses the principles of operation of the MP-32A from the standpoint of a person having previous experience with integrated circuit computer hardware. The contents of this section, in combination with the MP-32A Maintenance Drawings (Manual TMB4) and the MP-32A Wire Lists, describe completely the logical design and implementation of the MP-32A.

4-3. FUNCTIONAL SYSTEM DESCRIPTION

4-4. LOGIC DESIGN REPRESENTATION

4-5. The logical design of the MP-32A is represented in logic diagram form.

4-6. SYSTEM BLOCK DIAGRAM

4-7. The system block diagram, Figure 1-14 (drawing number 21001-100-1), is the top drawing in the set of drawings which describe the logical design of the system. The system block diagram depicts the functional grouping of the system components together with the principal interconnecting paths. The functional groups are titled: Timing & Control, Arithmetic Unit, MOS Memory System, and I/O Interface. The system block diagram lists the drawing number of the subsystem block diagrams.

4-8. SUBSYSTEM BLOCK DIAGRAMS

4-9. The subsystem block diagrams represent the next level of machine logical detail. The subsystem block diagrams show: components which constitute a particular subsystem, data paths, primary control terms, and interface signals. Included on each major individual block of the subsystem block diagram is the logic diagram drawing number corresponding to the particular component which the block represents. The logic diagram drawing number is designated with a single decimal digit, 1 through 4, in combination with a single letter or in combination with a letter followed by a single decimal digital. The leading decimal digit associates the drawing with a subsystem. The four digits are allocated as follows: 1 - Control and Timing, 2 - Arithmetic, 3 - MOS Memory, and 4 - I/O Interface. For example, the logic diagram drawing number for the Instruction Buffer Register is 1E.

4-10. LOGIC DIAGRAMS

4-11. Each subsystem is comprised of a number of logic networks. The logic networks are grouped by function. A logic diagram, in general, represents the logic implementation of a particular function or groups of related functions such as individual registers, adder input controls, etc. The logic diagram illustrates the interconnection of individual logic elements and the element type either by symbol or part number along with the physical location of each element. For example, the set of characters F20-1 on a logic symbol, indicates that the element is located in bay F, column 20, row 1. (For an example, see Figure 1-4).

4-12. LOGIC ELEMENTS

4-13. Individual elements represent the lowest level of functional detail in the machine.

4-14. LOGIC SIGNALS

4-15. A tabulation of all signals in the machine including power and ground appear in both the logic wire list and the connector wire list. The logic wire list groups the signals by name and shows all of the points to which an individual signal is connected. The connector wire list groups the individual signals and part numbers by connector thus giving the physical location of individual signals. The wire lists also incorporate the drive capability and DC loading of individual signal sources, and the logic diagram drawing number on which individual signals appear. The wire lists show in column 6 a number which is 10 times the number of required standard loads (one standard load is 0.8 ma) and in column 7 a number which is 10 times the number of required standard drives (one standard drive is 8 ma). At the end of each column for the same logic name, the loads (or drives) are added and divided by 10 to give the total number of standard loads (or drives).

4-16. LOGIC NAMES

4-17. Component dimensions are specified in decimal notation. For example, a register comprised of sixteen storage elements is said to be 16-bits in length. The rank or element position within a register is specified in octal notation. Rank specification of a typical 16-bit component, in octal notation, is illustrated in the diagram shown below where MSB stands for the most significant bit and LSB the least significant bit:

17	16	15	14	13	12	11	10	07	06	05	04	03	02	01	00

MSB LSB

4-18. Names assigned to elements and signals are, in general, descriptive of the element or signal use. Registers composed of J-K flip flops best illustrate the rationale of name assignment.

4-19. Register elements have a five character designation. The mnemonic name of the register is taken as the first three characters and the last two characters, octal digits starting with 00 for the LSB, represent the element position in the register. Inputs and outputs are identified by the element designation plus an added letter or an added letter plus an asterisk. For example, IBR05, is the sixth bit in the Instruction Buffer Register having inputs IBR05J and IBR05K and outputs IBR05Q and IBR05Q*. The outputs IBR05Q and IBR05Q* represent the true and complement outputs respectfully.

4-20. Names associated with registers comprised of MSI elements follow the above convention in terms of the output signal only. The typical input to the register is through a four-wide AND-OR-INVERT gate. The input signal to the register element, "gate output", is designated with a six character name followed by an asterisk. For example, A1R10F*, is the input to the ninth bit of the A1 Register with the active level at ground.

4-21. In general, signals associated with components comprised of elements exhibiting a storage capability plus the adder and multiplier are assigned names in accordance with the following rule. The logic name is comprised of six characters which can be interpreted as follows: the first group of three characters is the mnemonic name of the component, the second group of two characters designate bit position or rank, and the sixth character in combination with an asterisk specify the logic level characteristic.

4-22. The logic name assigned signals which have gates as sources is less regular. However the assigned name is to some measure descriptive of the signal use.

4-23. The following provides the general rule for relating logic structures with corresponding logic functions:

a. Input terms with signal names followed by an asterisk, must be Low to be True and High to be False.

b. Input terms with signal names not followed by an asterisk must be High to be True and Low to be False.

c. Output terms with signal names followed by an asterisk are active Low.

d. Output terms with signal names not followed by an asterisk are active High.

4-24. FUNCTIONAL SYSTEM OPERATION

4-25. LOGIC CIRCUITS

4-26. The machine design incorporates state-of-the-art devices and the logic is implemented with a high speed TTL circuit family. Extensive use is made of MSI circuits. The MSI level of integrated circuit provides advantages over logic implemented with less complex functions such as fewer external nodes and corresponding interconnections, and higher effective speed of operation.

4-27. DESIGN FEATURES

4-28. A summary of the design features incorporated in the logic implementation is given in Table 4-1.

4-29. CONTROL AND TIMING UNIT

4-30. DOCUMENTS. The documents which represent the logic design of the control and timing unit are listed by title and drawing number in Table 4-2 .

Table 4-1. Design Features

5V Power Supply & Ground Distribution

Configuration: Dual Plane-Two, low resistivity planes, (power & ground) separated by a thin sheet of dielectric material (permittivity ≈ 7)

Decoupling: The composite provides distributed capacitance for VCC decoupling in addition to power & ground distribution.

VCC Power Supply Characteristics

Nominal Voltage Level: 5.0VDC

Regulation

Line: 0.05% + 4 mv for line variations 105-132 or 132-105 volts AC

Load (no load → full load): 0.03% +3 mv, no load to full load or full load to no load

Ripple: 1 mv rms

Overload Protection

Voltage Protection Point: 5.7V (+5, -0%)

Current Protection Point: <1.45 of 40°C current rating (80 Amp)

Clock Transmission

Frequency: 6MHz

Distribution: Twisted pair terminated at the receiving end.

Signal Transmission

Interconnections: Wire-wrap

Length ≤21 in.: Single 30 gauge wire in close proximity to a ground plane.

Length >21 in.: Twisted pair terminated at the receiver - ground returns carried through from transmitting end to receiving end.

Table 4-1. Continued

Typical Characteristics - Logic Circuits

Supply Voltage: 5.0VDC

High Logic Output Level: $\geq$2.4VDC

Low Logic Output Level: $\leq$0.4VDC

Noise Immunity: -1.0VDC

Unused Inputs: Tied to output of Hex Inverter the input of which is tied to ground.

Table 4-2

Title	Dwg. No.
Timing & Control Diagram	21001-100-3a
Instruction Pad Memory System	21001-100-3b
Instruction Register, Bits 0-17	1A1
Instruction Register, Bits 20-33	1A2
Instruction Address Register	1C
IBR, IA & IRWRT Controls	1D
Instruction Buffer Register	1E
Register Decoders	1H
Panel Logic, Sheet 1 of 2	1I1
Panel Logic, Sheet 2 of 2	1I2
Light Drivers	1K
Clock System	1L
TC Counter & Controls	1M
WC Counter & Controls	1N
Modified Jump Logic	1P
Read-Only-Memory	1R

4-31. SYSTEM TIMING. The nominal clock frequency is 6MHz. The clock period, equal to 167 ns, defines a machine cycle. A machine cycle starts at a clock time and is terminated at the next clock time. The majority of the machine instructions are executed in a single machine cycle. The basic system clocks specify the timing for instructions of this type. Extended timing sequences of more than one machine cycle are required for certain operations, namely: multiple executions of the same instruction, load type instructions, and jump type instructions when the jump condition is satisfied. Two additional timing components, the TC Counter and the WC Counter, are used to generate and control the extended timing sequences required by the first two of the above mentioned operations. The third type of operation, jump instructions with the test conditions met, requires two machine cycles. Logic term JMPTP* (jump test passed active ground) disables loading the next instruction address from either control pad (IALA) or memory (IALC) and enables IALD which loads IAR (instruction address register) from the low order 6 bits of IBR (instruction buffer register). OPCENQ (Op code enable) is cleared inhibiting execution of the next instruction. At the next clock time the new instruction is loaded, OPCENQ set and normal execution resumes. The above logic is represented on logic drawings 1P, 1D and 3A2.

4-32. CLOCK GENERATION & DISTRIBUTION. All of the clocks in the processor are derived from one of two sources: a 12 MHz crystal oscillator or a voltage controlled oscillator. One of these sources is selected via a switch on the maintenance panel and applied to the clock input of CL8MHZQ, a 74S112 JK flip flop with both inputs held true. The false output of this flip flop (CL8MHZQ*) is buffered and applied to a delay line. Various taps on this line are used to produce the two system clocks, CLOKO and CLOKS, and all other timing functions. CLOKO is a narrow negative pulse approximately 12 ns wide and CLOKS is a square wave. The clock system is described on logic drawing 1L and System Timing is shown in Figure 4-1.

4-33. The clocks are distributed to the various bays of the machine by twisted pair and terminated with a 220 Ω resistor. All clocks are then buffered before being used locally.

4-34. Figure 4-1 depicts the clock timing along with the semiconductor memory write timing signal, WRTQ, which is derived from the basic clock system.

4-35. TC COUNTER. The TC Counter is a four bit synchronous counter used to generate and control extended timing sequences. In most instructions, when the next address from IR(20-25) is loaded into IA, the T-field, IR(20-33), is loaded into the TC Counter and the TC Buffer Register. It is then decremented at each clock time until it reaches zero. When an OP-CODE 14 (load command) is executed, the TC Counter is not decremented at the first clock time. At this time the number of words to be loaded is entered into the Word Counter. The TC Counter is now decremented at each clock time. When the TC Counter reaches zero, the load flip-flop , LOADQ, is on and, if the Word Counter is non-zero, the original value of the TC Counter is reloaded from the TC Buffer Register.

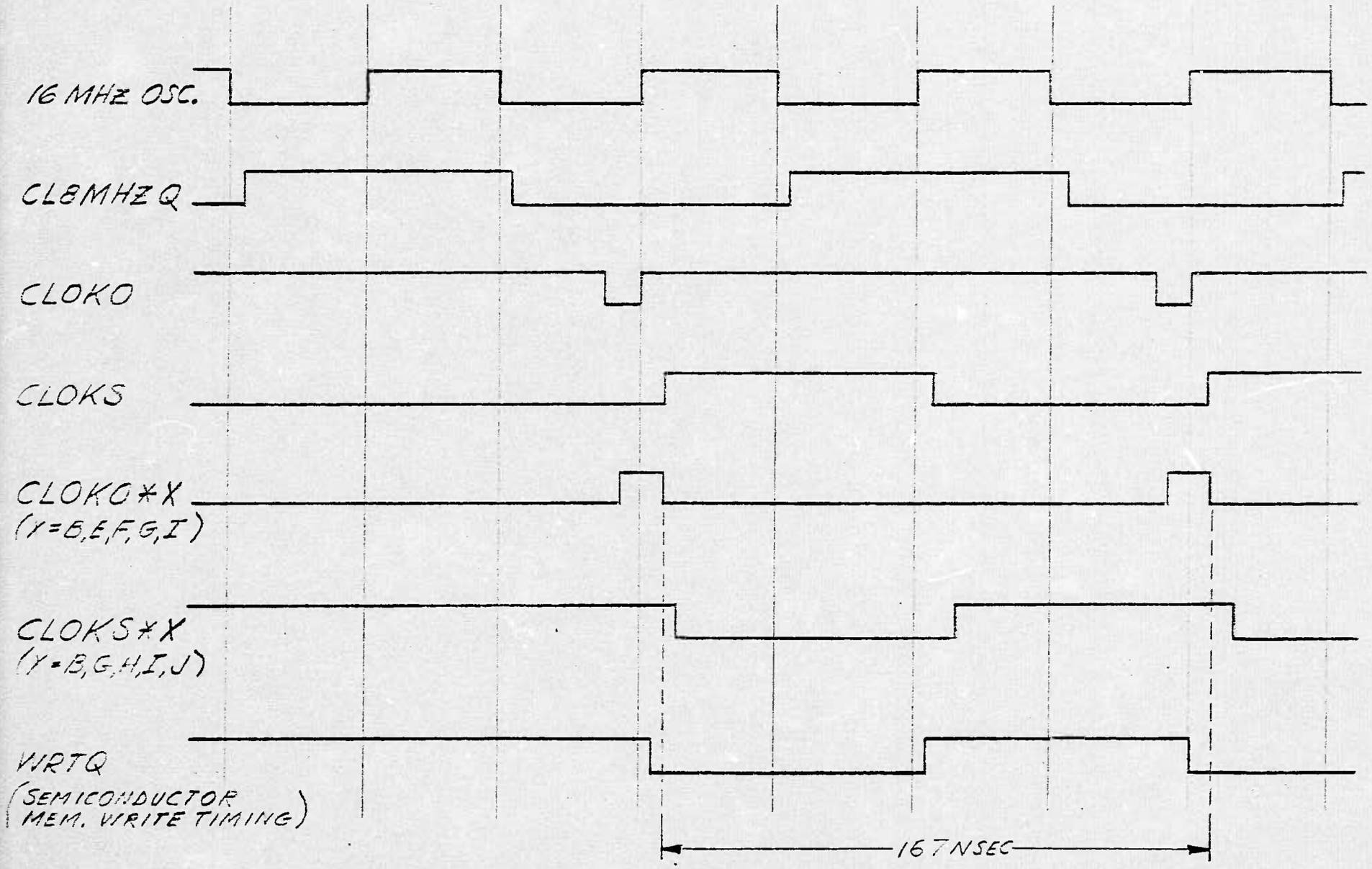


FIGURE 4-1 SYSTEM TIMING DIAGRAM

The process of counting down the TC Counter continues until the Word Counter reaches zero, at which time a new instruction is loaded into IBR and a new T value is loaded. For load types 1404 and 1424, when the Word Counter and the TC Counter reach zero, the last instruction loaded is executed if the mode field is 0 or 2. The contents of I2(10-13) is loaded into the TC Counter, bits (6) and (7) are loaded into the Mode Register, and bits (0-5) are loaded into IA. The contents of I1 is loaded into the Instruction Buffer Register in the Load 1424 command. In the Load 1414 command, TC, IM, IA & IB are loaded directly from memory (CD) when TC and WC equal zero.If the mode field is 1 the next instruction in memory is loaded into TC, IM, IA and IB at the clock following the completion of the load command, if the instruction in memory is ready. If the instruction is not ready, the memory controller causes a pause until it is ready, which is defined by INSTRDY being true, before loading the instruction.

4-36. The storage elements of the TC Counter are 74S114 J-K flip-flops, operating as D type flip-flops. The input selection is done using And-Or-Invert gates which allow four ways in: IR(30-33), the output of the TC decrement adder, I2(10-13) or CD(30-33) and the TC Buffer Register. The content of the TC Counter is fed into a 7483N four stage full adder, where it can be decremented by adding all one's, or left unchanged by adding all zeros. The TC Buffer Register is a 4-bit shift register, part no. 8271B. The TC buffer load control is IBRKEN, the term used to load the Instruction Buffer Register.

4-37. WC COUNTER. The Word Counter is used to count the number of words (or double words) to be loaded during an OP-CODE 14, load command. The Counter, WC, is 8-bits in length. Its storage elements are 74163 4-bit counter chips. Since this counter only counts up it is loaded with the 1's complement of IBR(0-7) at the first clock in all load commands except 1460 and 1470, which require no word count. WC counts only when TC is zero and WC is non zero. WC = zero occurs when all of the outputs of the word counter, WC00Q* - WC07Q* are true.

4-38. AUTOMATIC CONTROL. The operation of the MP-32A is controlled automatically from a stored program. In general, instructions are made available to the machine from either Instruction Pad Memory, when not in memory execute mode, or from MOS memory (CD) when in memory execute mode, signified by CEXQ=1. In load instructions with D and C-field octal code combinations of 14, 24, and 44 through 74 ,instructions are made available to the machine from the following additional sources:

a. Data Pad Memory, for an octal code combination equal to 24.

b. MOS Memory, for an octal code combination equal to 14 or 74.

c. I2 and I1 for an octal code combination of 44.

d. Instruction pad, (IR20-33) and I1(00-17) for an octal code combination of 54.

e. I2(00-13) and IR(00-17) for an octal code combination of 64.

4-39. LOADING INSTRUCTIONS. At clock time the 28-bit instruction word is loaded as follows:

a. Bits (0-17) into the Instruction Buffer Register

b. Bits (20-25) into the Instruction Address Register

c. Bits (26 & 27) into the Instruction Mode Register

d. Bits (30-33) into the TC Counter.

The control terms which effect the loading of the various portions of an instruction are described under separate headings, i.e. TC Counter, Instruction Address Register, and Instruction Buffer Register. The Instruction Mode Register is loaded from the same sources and by the same control terms as the Instruction Buffer Register.

4-40. INSTRUCTION DECODING. Each instruction is broken down into eight fields containing two to six bits each. Each field performs a related function. The various fields can be categorized as fixed and variable. The T field contains a 4-bit repeat number for op codes 15-17 and a 3-bit repeat number for all other op codes. In these instructions the most significant bit of the T field, TC13, is used to control the entering or terminating of sequential mode in certain memory instructions. In the mode 3 instruction, the T field is not used as a repeat number but instead has the following meaning:

TC10 = 1 Initiate memory
TC11 = 1 Set memory write control
TC12 = 1 Enter memory execute mode
= 0 Terminate memory execute mode
TC13 = 1 Enter sequential mode
= 0 Terminate sequential mode

The OP CODE-field along with the D,C,B and A-fields are variable, i.e. have different meanings in different instructions. The interpretation of all variable fields for each instruction subset or single instruction is given in TM A4.

4-41. The M-field code selects one of four pages of instructions. The OP-CODE specifies a unique line within a page. The D,C,B and A-field codes specify the operations or operation which occur along the individual lines of a page. The decoding of the M-field in combination with the OP-CODE field is delineated on logic diagram drawing numbers 1F1 and 1F2. The D, C, B and A-field decoding, as a function of the various OP-CODES, appears on logic diagram drawing number 1H.

4-42. Some additional control terms are taken from the outputs of individual flip-flops. These control terms are described under the operations to which they apply.

4-43. INSTRUCTION PAD MEMORY SYSTEM. The Instruction Pad Memory is a self-contained 64 word by 28-bit semiconductor memory system which is comprised of:

a. Storage array consisting of 28 bipolar memory chips (Intel 3101A or equivalent),

b. Six-bit Instruction Address Register,

c. Write timing control,

d. A set of 28 output lines (IR00Q-IR33Q).

4-44. The Instruction Address Register output, bits 4 and 5, (IA04Q) and (IA05Q), are decoded to select one of four groups of 16 words. Bits 0 thru 3, (IA00Q, IA01Q, IA02Q and IA03Q), are routed to the address inputs of all memory chips. The 1/16 on-chip decoding then selects one of the 16 addresses to effect reading or writing at a unique memory location. Table 4-3 summarizes the logic of this register.

4-45. The memory controls are implemented to allow selective write-in of portions of a word or a complete word. Data is written into the memory in accordance with the information in Table 4-4. The write control terms incorporate the write timing. The write timing pulse WRTQ is derived from the system timing and occurs during the last half of each machine cycle. The logic gates, shown on drawing 1D which develops IRWRTR* and IRWRTL*, have WRTQ as one of the input terms.

4-46. In the normal mode of operation a new instruction is brought into IBR at each clock time. The exceptions to this are:

a. When an instruction takes more than one clock time to execute. The control signal TCWO is the "and" of the Word Counter being zero and the TC Counter being zero. If this is not the case, the present instruction is not completely executed and TCWO is used to inhibit IBREN*.

b. When the machine is being stopped, the next instruction on the Instruction Register Output lines is not loaded into IBR to be executed. STOP D*, the term used to set the stop flip-flop, is used to indicate that the machine will stop at the next clock time, and inhibits the transfer into IBR.

c. Once the machine has stopped the transfer is inhibited by STOPD*.

d. For most Op-Code 14 load instructions, more than one clock time is used to execute the instruction. LOADEN is used to set the load flip-flop and it is also used to inhibit the transfer during the clock time before it sets.

e. For Op-codes 1404, 1414, 1424 the instruction pad is loaded. If the instruction is mode 0, when the last instruction is loaded into pad it is also loaded into IBR and executes. These conditions enable CDIB* which enables the transfer from I1 or CD and inhibits the transfer from IR. Op-codes 1444-1474 are execute commands causing IBR to be loaded directly from the following sources:

Op-code 1444, 1454 - I1
Op-code 1464 - IR
Op-code 1474 - CD

Table 4-3. Instruction Address Register (IA)

Function:	Temporary storage register, 6 bits
Use:	Hold address of current cell being used in Instruction Pad
Mechanization:	The IA register is implemented with J-K flip-flop storage elements (74S114) with And-Or-Invert gates (74S64) for data input selection. The incremented address is obtained using 7483A adder chips.
Mode of Operation:	The register can be cleared, loaded in parallel and incremented.
Data:	The data is stored in the register in a non-inverted form, i.e. a high level represents a logical "one", a low level represents a logical "zero."
Parallel Data Entry:	To effect parallel data entry, one of three inputs is selected and the register hold control, IALB, is disabled.

Data Input Sources	Data Input Selection Controls
Instruction Register bits 20-25	IALA
Incremented IA bits 0-5 or hold	IALB
I2 register bits 0-5 or CD bits 20-25	IALC
Instruction Buffer Register bits 0-5	IALD

Data Input Selection Controls: IALA is enabled whenever IBR is loaded from the Instruction Register (see IBR controls) and also with Op-Code 1454. IALB is enabled when neither IALA, IALC or IALD are active to hold the current address in IAR. In addition, IACIN is enabled whenever IAR is to be incremented. This occurs for Op-Codes 1404, 1414 and 1424 (IRLOAD); for panel increment (IAPNINC*); for adder test passed (ADMJTP*), modified jump test passed (MJMPTP*) and mode 0

Table 4-3. (continued)

	or mode 2 link jumps (OP17MO2*) IALC is enabled by the panel (IALC1*), by op-codes 1404-1474 (OPC14x4), except for Op-Code 1454, and when in memory execute mode (CEXD1), when the instruction is ready. IALD is enabled when a jump test is passed (JMPTP*) and by mode 1 link jump (OP17M1*).

Table 4-4. Instruction Pad Write

Data Input Sources	Instruction Register	Write Control Term
I1 Register, bits 0-17 or CD, bits 0-17	Bits 0-17	IRWRTR*
I2 Register, bits 0-7 or CD, bits 16-21	Bits 20-27	IRWRTL*
I2 Register, bits 10-13 or CD, bits 22-25	Bits 30-33	IRWRTR*
Instruction Buffer Register bits, 0-5	Bits 20-25	IRWRTL*(Op-Code 17)

f. When a test is passed, the jump address (BA-field of the instruction) must be loaded into IA. During the clock time that this is done, the execution of IBR is inhibited by OPCENQ*.

g. If the test is a modified jump and it is passed, the jump address must be incremented. During the clock time that this requires, the execution of IBR is inhibited by OPCENQ*.

4-47. When any transfer into IBR is required, the k-inputs must be enabled (IBRKEN*). IBREN* provides anIBRKEN*. IBRKEN* is also enabled by STOP D* in order to clear IBR when the machine stops. A buffer gate is used to provide the necessary drive.

4-48. The Instruction Buffer Register is summarized in Table 4-5 .

4-49. MANUAL CONTROL. The operational characteristics of the control panel were described in Sections 2-6 thru 2-16. The following paragraphs provide a literal description of the more fundamental aspects of the logic design and implementation. The logic design is represented on the logic diagrams, drawing numbers 1I1 and 1I2.

4-50. PANEL TIMING. Panel operations, exclusive of Dead-Start, are executed during a panel cycle time 3. A panel cycle is normally of three clock times duration, designated panel times: 1, 2 and 3. The duration of panel time 3 is terminated when the condition for stopping the machine is met. A panel cycle is generated by actuating any one of the following controls when the machine is stopped:

a. PAD ADDRESS LOAD
b. PAD ADDRESS INC
c. INSTRUCTION ADDRESS LOAD
d. INSTRUCTION ADDRESS INC
e. INSTRUCTION REG LOAD
f. MEMORY ADDRESS LOAD
g. MEMORY ADDRESS INC
h. LOAD DATA
i. INITIATE
j. RESET
k. RUN/STOP
l. STEP

4-51. STOPPING THE PROCESSOR. The processor is stopped in two ways:

a. the STOP button,
b. the Halt logic.

4-52. The processor can only be stopped when both the TC counter and the WC counter are equal to zero (TCWO) and no pause condition exists (PAUSE*). When these conditions are met, logic signal TCWOP is a logical true. In addition, one of the following conditions must be satisfied:

a. Op-Code 1460 or 1470
b. Op-Code 1400 - 1450 and LOAD (LOAD=LOADQ·OPCENZ)
c. Not Op-code 14.

Table 4-5. Instruction Buffer Register

Function:	Storage register, 16-bits
Use:	Holds OP and D, C, B, A of the current instruction
Mechanization:	The IB register is implemented with J-K flip-flops(74S114 and N8H21) and 2 input multiplexers (74S158) for data input selection.
Modes of Operation:	The register can be cleared and loaded in parallel.
Data:	Data is stored in the register in a non-inverted form, i.e., a high level on the data input lines represents a logical "one", a low level represents a logical "zero".
Parallel Data Entry:	To effect parallel data entry into the register, one of two inputs is selected and the load control term IBRKEN is enabled.

Data Input Sources	Data Input Controls
Instruction Register	CDIB*·IBREN*
I1 Register or CD	CDIB·IBREN*

The stop logic is shown on logic drawing 1I2.

4-53. DEAD START PROVISIONS. Two independent Dead-Start bootstrap routines are available to the user. These are permanently stored in 256-bit, bipolar Read Only Memory chips or ROM. These chips are organized as a 24-bit, 32 word memory. Since the system requires 28-bits for an instruction, zeros are inserted in bits 22 and 25-27.

4-54. If the Disk Dead-Start routine is desired, the panel procedure is to press RESET, INITIATE and RUN, which initiates the following action:

a. RESET. All registers are reset to zero (including IA which addresses ROM).

b. INITIATE. This de-selects the 3101-A RAM chips and allows the bootstrap ROM to control IR and be transferred to IB, IA, IM and TC.

c. RUN. This causes the Disk Bootstrap to be executed which selects the first enabled disk drive, issues a RESTORE command and reads the operating system located on track 0, into memory, putting the system on the air.

4-55. If RESET, INITIATE, INC IA and RUN are pressed, the Host Bootstrap is executed which waits for an Interrupt from Host.

4-56. STARTING THE PROCESSOR. The processor can be started in the run mode by the RUN-STOP button, or a single instruction can be executed with the STEP button. If the processor is stopped and the RUN-STOP or the STEP button is pushed, a panel cycle is initiated. At the start of panel time 3, the STOP flipflop is cleared and the instruction at the current IA or CA location is loaded into TC, IM, IA and IBR. During panel time 3 the instruction is executed. As soon as the conditions for stopping are met, the stop flip-flop is set and the panel time 3 flip-flop cleared.

4-57. DATA SELECT REGISTER. Any register not permanently displayed on the panel can be displayed with the Data Select Register. To display a register, the desired register is selected by the DATA SELECT REGISTER buttons. The button is a single pole double throw momentary switch with the common grounded. The normally closed contact goes to the D input on a quad-latch chip (7475N) making the inputs normal ground. The normal open contacts of all the buttons are tied to a pulse generating one shot. When a button is pushed,the normally closed contact is opened so the D input goes high, and then the normally open contact is pulled to ground,generating a clock pulse. The high on the selected D input is then clocked into the latch , while all other latches are cleared. Thus,the desired register is selected.

4-58. LOADING THE SELECTED REGISTER. All registers selected by one of the DATA SELECT REGISTER controls are loaded in much the same manner. The machine must be stopped. A unique register must be selected with a DATA SELECT REGISTER control. The LOAD DATA control must be actuated to initiate a panel cycle as follows:

a. LOAD: A1, A2, I1, I2, M1, and M2

Panel Time 1: E1R17 - E1R00 transfers to I1R17 - I1R00
E3R13 and E3R12 transfers to I1 flag bits

Panel Time 2: I1 transfers to selected register

When I1 is selected, no transfer occurs at panel time 2.

b. LOAD: D/S and HALT

Panel Time 1: E to I1 transfer inhibited

Panel Time 2: E transfers to selected register

c. LOAD CD and LOAD CD_R

Panel Time 1: Same as in a.

Panel Time 2: The term, CDLDI*, sets the write control flip-flop and after a half clock time delay causes a memory initiate signal to be issued, (if the memory is not busy), which writes the contents of I1 into memory). After the write cycle is completed a read cycle is initiated.

d. LOAD PAD:

Panel Time 1 and 2: Same as in a., except flag bits are not transferred to Data Pad.

4-59. INSTRUCTION REGISTER LOAD FROM PANEL. The latch for the IR LOAD button is IRLDL. After the button is pushed, E1R (00-17) is loaded into I1 and E2R(00-07) & E3R(10-13) are loaded into I2 at panel time 1. At panel time 2, I1 and I2 are loaded into IR.

4-60. INSTRUCTION ADDRESS LOAD FROM THE PANEL. The latch for the IA LOAD button is IALDL. After the button is pushed, I2 is loaded from E2R(00-07) and E3R(10-13) at panel time 1. At panel time 2, IA is loaded with I2 (0-5).

4-61. INSTRUCTION ADDRESS INCREMENT FROM PANEL. The latch for the CONTROL PAD ADDRESS INCREMENT button is IAINCL. After the button is pushed, IACIN, the IA increment control, is enabled, and IA+1 is loaded into IA at panel time 2.

4-62. MEMORY ADDRESS LOAD FROM THE PANEL. The latch for the MEMORY ADDRESS LOAD button is CARLDL. After the button is pushed, E1R (00-17) is loaded into I1 at panel time 1. At panel time 2, I1 is gated to the adder. At the end of panel time 2 I1+0 is loaded into CAR.

4-63. MEMORY ADDRESS INCREMENT FROM PANEL. The latch for the MEMORY ADDRESS INCREMENT button is CARINCL. After the button is pushed, CAR is gated to the X side of the adder and zero to the Y side at panel time 2. A carry-in is generated, and the sum (CAR+1) is loaded into CAR.

4-64. PAD ADDRESS LOAD FROM THE PANEL. The latch for the PAD ADDRESS LOAD button is PALDL. After the button is pushed, E2 is loaded into PA at panel time 2.

4-65. PAD ADDRESS INCREMENT FROM THE PANEL. The latch for the PAD ADDRESS INCREMENT button is PAINCL. After the button is pushed PA is incremented and loaded into PA at panel time 2.

4-66. RESET*. The machine can be reset by the RESET button, a system reset if the processor is to be used in a larger system where a common reset line is provided or by Power-On reset which resets the machine when the power is turned on. All registers that can be direct cleared, are cleared by RES*. Some registers, however, can only be cleared by loading them with all zeros. Therefore, a panel cycle is generated by any reset, and used to load the necessary registers with zeros.

NOTE

The Halt Register and the Data Select register are not cleared by RESET.

4-67. HALT LOGIC. As with all panel switches, the HALT switch is used to set a latch to eliminate any contact bounce. When the HALT switch is set to either CA or IA, the halt flip-flop is set. Then, when the appropriate address compares with the address of the Halt Register and a condition for stopping as given above is satisfied, the stop flip-flop is set. Table 4-6 describes the Halt Register.

4-68. PROGRAM ERROR. Not yet defined.

* Only manual reset incorporated presently.

Table 4-6 . Halt Register

Function:	Temporary storage register, 16-bits.
Use:	Holds the address at which the processor will stop when placed in the halt mode.
Mechanization:	The Halt Register is implemented with shift register storage elements (8270A).
Modes of Operation:	The register is loaded only in parallel from E1R(17 -00)
Data:	Data is stored in the register in a non-inverted form, i.e. a high level represents a logical "one", a low level a logical "zero."
Parallel Data Entry:	To effect parallel data entry, the Halt Register must be selected by the Data Select Register. The LOAD DATA button is then pressed and E1R(17-00) is loaded into the Halt Register at panel time 2. The load control is HLTLA.

4-69. ARITHMETIC UNIT

4-70. DOCUMENTS. The documents which represent the logic design of the arithmetic unit are listed below by title and drawing number.

Title	Dwg. No.
Arithmetic Unit Block Diagram	D-21001-100-4a
Data Pad Memory Block Diagram	D-21001-100-4b
Adder	2A1
Adder	2A2
Adder Input Controls	2B
Adder Destination Controls	2C
Multiplier	2D
M1 & M2 Registers	2E
A1 Register	2F
A2 Register	2G
I1 Register	2H
I2 Register	2I
Register Shift Controls	2J
Register Tests, OPC 15	2K
Register Tests, OPC 16	2L
Flag Register	2M
Flag Register Controls	2N
Data Pad Storage Array	2P
Data Pad Address Register	2Q
Data Pad Address Controls	2R
Data Pad Input Bus 1 of 3	2S1
Data Pad Input Bus 2 of 3	2S2
Data Pad Input Bus 3 of 3	2S3
Data Pad Input Controls	2T

4-71. ADDER. The Adder is capable of adding two 16-bit numbers in approximately 42 ns. Each side of the Adder can be fed from four sources. The Adder can perform several logical operations; sum, coincidence, and AND. Two's complement, subtraction, OR, and exclusive-OR can also be performed by using these three functions and varying the inputs. If the inverse data is fed into one side of the Adder and a carry introduced, subtraction is obtained. Two's complement can be obtained in the same way as subtraction but with one side of the Adder zero. The Adder is constructed with four 8260 arithmetic logic elements and an 8261A fast carry extender used in a carry look ahead configuration.

4-72. Three control terms effect the functions performed by the adder: ADEINH, ADCINH, and ADCINI*. The truth table for these functions is as follows:

ADCINH	ADEINH	ADCINI*	FUNCTION
0	0	0	ADX + ADY
0	0	1	ADX + ADY + ADCIN
1	0	X	Coincidence X_n 0 Y_n
1	1	X	And X_n Y_n

ADCINH is enabled with Op Codes 6-13 and a C-field decode of 1 or 5.

ADEINH is enabled with Op Codes 6-13 and a C-field decode of 1.

4-73. Adder Inputs. The inputs to the Adder are multiplexed using And-Or-Invert gates. The Adder accepts trues as inputs, so the inputs to the And-Or-Invert gates are inverted. The following tables describes the Adder Input logic:

Adder Inputs	Control Term	Source
X Side		
A1 Register	*ADLA	A1 Register inverse
A1 Register inverse	ADLB	A1 Register
I1 Register	ADLC	I1 Register inverse
Memory Address Register	ADLD	Memory Address Register Inverse
Y Side		
A2 Register	**ADLE	A2 Register inverse
A2 Register inverse	ADLF	A2 Register
I2 Register	ADLG	I2 Register inverse
Data Pad Data	ADLH	Data Pad Data inverse

ADLAI can override ADLA and inhibit the A1 Register input
**ADLEI* can override ADLF and inhibit the A2 Register input

If ADLA and ADLB are both enabled at the same time, the X input to the Adder is zero since the merge of A1 and A1* is all lows in the And-Or-Invert gates. If ADLE and ADLF are both enabled at the same time, the Y input to the Adder is zero.

The gates for generating the Adder input controls are shown on the Adder Input Controls Drawing. The controls enabled by the various Adder commands are shown on the Adder Input Controls Chart, Table 4-7 .

4-74. Adder Overflow. Adder Overflow (ADOVF) is detected using the algorithm that if the carry into the last bit is not equal to the carry out of the last bit an overflow has occurred. Whenever the Adder sum is transferred into a register, the fact that an overflow has or has not occurred is saved in the overflow flip-flop.

4-75. Adder Output. The controls for selecting the destination of the Adder are shown on the Adder Destination Control Chart, Table 4-8. The D and C-fields are decoded to enable the proper register control gates. The terms enabled for each Adder operation are shown on the chart. The normal destinations, as shown on the chart, can be defeated by calling for the Adder output to be loaded into another register in the register decode fields of the command. Mode 1 as well as certain register decodes call for the sum or sum* to be transferred to another destination. These are A1D2*, A2D1*, A2D2*, I1D2*, and I2D2*. Any one of these terms will disable the Adder destination controls, A1LB2*, A2LA2*, A2LB2*, I1LB2*, I2LB2* and CALA2*, thus defeating the normal Adder destination called for in the D & C-fields.

4-76. MULTIPLIER. The Multiplier is an 8 x 8 bit multiply with a 16-bit product. It uses 7 stages of 8-bit adders. The adders themselves are made up of 7483N 4-bit full adder chips. The adders are arranged to do a standard shift-and-add type multiply as shown below:

```
                                                M1 · M2 bit 1
                                           M1 · M2 bit 2
                                      M1 · M2 bit 3
                                 M1 · M2 bit 4
                            M1 · M2 bit 5
                       M1 · M2 bit 6
                  M1 · M2 bit 7
             M1 · M2 bit 8
PRODUCT
```

As shown, M1 is anded with each bit of M2, shifted the appropriate number of bits, and added in the cascade of adders. The anding of M1 and the bits of M2 is done in 2 input nors. There are no controls going to the Multiplier. The Multiplier generates a product from the contents of M1 and M2 at all times. The product is used by loading it into another register. It takes a maximum of 375 ns for the product to become available from the time that either M1 or M2 is changed. The data in M1 and M2 is assumed to be in sign magnitude form with the sign bits in M1F1 and M2F1. The sign of the product is formed by performing the exclusive-OR of M1F1Q and M2F1Q,

Table 4-7

ADDER INPUT CONTROLS CHART

C-FIELD / D-FIELD	0	1	2	3	4	5	6	7
0	ADLA ADLE1*	ADLA ADLE1*	ADLA ADLB1* ADLE1*	ADLA ADLB1* ADLE1*	ADLA ADLAI1* ADLE1*	ADLA ADLE1* ADLE3*	ADLA ADLB1* ADLAI2* ADLE1*	ADLA ADLB1* ADLAI2* ADLAI4* ADLE1*
1	ADLA ADLF1* ADLF3* ADLF6*	ADLA ADLB23* ADLF1* ADLF3* ADLF4*	ADLA ADLB23* ADLF1* ADLF3*	ADLA ADLB23* ADLF1* ADLF3*	ADLA ADLAI1* ADLC1* ADLF4*	ADLA ADLE3* ADLF1* ADLF4*	ADLA ADLF1*	ADLA ADLA14* ADLF1*
2	ADLA ADLG2*	ADLA ADLG2*	ADLA ADLB1* ADLB23* ADLG2*	ADLA ADLAI1* ADLG2* ADLG3*	ADLA ADLAI1* ADLG2* ADLG3*	ADLA ADLB4* ADLG2*	ADLA ADLBI* ADLAI2* ADLB23* ADLG2*	ADLA ADLB1* ADLAI2* ADLAI4* ADLB23* ADLB4* ADLG2*
3	ADLA ADLHM	ADLA ADLH1*	ADLA ADLB23* ADLH1*	ADLA ADLB23* ADLH1*	ADLA ADLAI1* ADLH1*	ADLA ADLH1*	ADLA ADLA22* ADLB23* ADLH1*	ADLA ADLAI2* ADLAI4* ADLB23* ADLH1*
4	ADLA ADLE	ADLA ADLE	ADLA ADLF2* ADLE	ADLA ADLF2* ADLE	ADLA ADLEI2* ADLE	ADLA ADLE	ADLA ADLE	ADLA ADLAI4* ADLE
5	ADLB5* ADLF3* ADLF6* ADLE	ADLB23* ADLB5* ADLF3* ADLF4* ADLEI1*	ADLB23* ADLB5* ADLF2* ADLF3* ADLE	ADLB23* ADLB5* ADLF2* ADLF3* ADLE	ADLB5* ADLG1* ADLEI2* ADLE	ADLB5* ADLF4* ADLG1* ADLEI1* ADLE	ADLB5* ADLE	ADLAI4* ADLB5* ADLE
6	ADLC2* ADLE	ADLE ADLC2* ADLE	ADLC2* ADLF2* ADLE	ADLC2* ADLF2* ADLE	ADLC2* ADLEI2* ADLE ADLG3*	ADLC2* ADLE ADLE	ADLC2* ADLF5* ADLEI3* ADLE	ADLZI4* ADLC2* ADLE5* ADLEI3* ADLE

Table 4-7 (continued)

C-FIELD	0	1	2	3	4	5	6	7
D-FIELD 7	ADLD2* ADLEI* ADLE	ADLD2* ADLE1* ADLE	ADLD2* ADLF2* ADLF1*	ADLD2* ADLF2* ADLE1* ADLE	ADLD2* ADLE1* ADLEI2* ADLE	ADLD2* ADLE1* ADLE	ADLD2* ADLE5* ADLE1* ADLEI3*	ADLAI4* ADLD2* ADLF5* ADLE1*

Table 4-8

ADDER DESTINATION CONTROLS CHART

C Field / D Field	Normal Add	Mod Add 0	Mod Add 1	Mod Add 2	Mod Add 3	Mod Add 4	Mod Add 5	Mod Add 6	Mod Add 7
0			A2LAC (A2LA2*)	CALAC (CALA2*)	A2LAC (A2LA2*)	A2LAC (A2LA2*)			
1	if A2F1Q*1 A2LAC A2LAD (A2LA2*)	if A2F1Q=1 A2LAC (A2LA2*)	CALAC (CALA2*)	A2LAC (A2LA2*)	if A2R16A A2LAC (A2LA2*)	A2LAC (A2LA2*)		A2LAC (A2LA2*)	A2LAC (A2LA2*)
2	I2LBC I2LBD (I2LB2*)	I2LBC (I2LB2*)	I2LBC (I2LB2*)	I1LBA (I1LB2*)	I2LBC (I2LB2*)	I2LBC (I2LB2*)			
3	A1LBA A1LBC (A1LB2*)	A1LBA (A1LB2*)	A1LBA (A1LB2*)	CALAC (CALA2*)	A1LBA (A1LB2*)	A1LBA (A1LB2*)			
4	A2LAC A2LAD (A2LA2*)	A2LAC (A2LA2*)	A1LBA (A1LB2*)	A2LAC (A2LA2*)	A1LBA (A1LB2*)	A1LBA (A1LB2*)		A1LBA (A1LB2*)	I1LBA (I1LB2*)
5	if A1F1Q=1 A1LBA A1LBC (A1LB2*)	if A1F1Q=1 A1LBA (A1LB2*)	A2LBA (A2LB2*)	A1LBA A1LBA (A1LB2*)	if A1R16Q=1 (A1LB2*)	A1LBA (A1LB2*)	A2LBA (A2LB2*)	A1LBA (A1LB2*)	A2LAC (A2LA2*)
6	I1LBA I1LBC (I1LB2*)	I1LBA (I1LB2*)	I1LBA (I1LB2*)	CALAC (MALA2*)	I1LBA (I1LB2*)	I1LBA (I1LB2*)			
7	CALAC (CALA2*)	CALAC (CALA2*)	CALAC (CALA2*)	I1LBA (I1LB2*)	CALAC (CALA2*)	CALAC (CALA2*)			

All Mode Adds Generator

A1LBC
A2LAD
I1LBC
I2LBD
MALAD

the sign bits of the multiplicand and the multiplier. The product sign is transferred into A2F1 when MPP is transferred to A2 (1-15), 0→A2 (16,17) The sign is not used when MPP is transferred to I2.

4-77. DATA PAD MEMORY. The Data Pad is a 64 word,16-bit high speed random access NDRO memory. The memory is used as a scratch pad (intermediate storage) and effectively buffers the high speed arithmetic unit from the large capacity MOS memory (32K to 64K 18-bit words). The memory consists basically of a Data Pad Address Register and a Semiconductor Storage Array as summarized below. The timing is derived from the system timing. The system and timing diagrams are shown on D-21001-100-4b.

a. Pad Address Register.

Function:	Temporary storage register, 6 bits
Use:	Holds the address of the cell currently accessable in data pad.
Mechanization:	The PA register is implemented with J-K flip-flop storage elements (74S14) and And-Or-Invert gates (74H54N) for data input selection. The incremented or decremented address is obtained using adder chips(7483A).
Modes of Operation:	The register can be cleared, loaded in parallel, incremented and decremented by either 1 or 10_8, and incremented by 2
Data:	The data is stored in non-inverted form, i.e. a high level represents a logical "one", a low level represents a logical "zero".
Parallel Data Entry:	To effect parallel data entry, one of four inputs is selected, fed into the adder where it is incremented, decremented or left unchanged before it is loaded to PA.

Data Input Sources	Data Input Selection Controls
PA register bits 5-0	PALA
IB register bits 5-0	PALB
E2 register bits 5-0	PALC
I1 register bits 5-0	PALD

Data Input Selection Controls:

PALA is enabled at all times except when PALB is enabled or when PALD is enabled. The PALA control is also overriden by PALC. PALB is enabled by mode two ,except for OP CODE 5. PALC is enabled from the panel or by Op Codes 1, 6, or 7 and a B-field decode of 4 or 5. DPALD is enabled by Op Codes 1, 6 or 7 and a B-field decode of 3.

Incrementing:

The controls for incrementing and decrementation are as follows:

Control	Function
PAC0	INC selected input by 1
PA23 and no PADEC0, PADEC1 or PADEC2	INC selected input by 10_8
PA23 and PADEC4	DEC selected input by 10_8
PADEC0, PADEC1, PADEC2, PADEC4, & PA23	DEC selected input by 1

These controls are enabled by:

(1) Op Codes 1, 6 or 7 and the appropriate B-field decode or

(2) By the panel or

(3) Mode 2 and Op Code 15 or 16 or by

(4) Op Code 0 and bit 10.

b. Semiconductor Storage Array. The array contains 16 bipolar MSI, Intel 3101A, memory packages (16 words x 4 bits).

4-78. REGISTERS. There are seven additional registers associated with the Arithmetic Unit as follows:

a. A1 Register

Function: Temporary storage register, 16-bits

Use: Input to Adder, M2 register, and Data Pad

Mechanization: The A1 regiser is implemented with shift register storage elements (8270A) and And-Or-Invert gates (8848A) for data input selection.

Modes of Operation: The register can be cleared, loaded in parallel or serial fashion and is capable of being left shifted.

Data: Data is stored in the register in an inverted form, i.e. a high level on the data input lines represents a logical "zero", a low level represents a logical "one."

Parallel Data Entry: To effect parallel data entry into the register, one of four inputs is selected and the load control term A1LO is enabled.

Data Input Sources	Data Input Selection Controls
Pad	A1LA
Adder Sum	A1LB
I1 Register	A1LC
A2 Register	A1LD

Data Input Selection Controls: Each of the controls are enabled from several sources: (1) decode of A-field for Op Codes 3, 7, 10, 11, (2) adder controls, and (3) the control panel. The Pad input bit 17 can be inhibited with A1LA I16* and the pad input bits 16-10 can be inhibited with A1D6*. Both of these controls are enabled from source (1) above.

Load Control: The load control, A1LO, is enabled: (1) by any one of the input controls, (2) when the register is cleared, i.e. reset, and (3) for Op Code 0, bits 0 or 3.

Left Shift: The control term is A1LE which is enabled by Op Code 1640 or 1641 with a decode in the B-field of 1, 2, or 3.

Serial Data Entry:

Data Input Sources	Serial Input Sources	Data Input Selection Control	Shift Control
A1F1Q	A1R01E*	OP CODE = 1641 B-FIELD DECODE = 1	A1LE
A1R17Q	A1R01E*	OP CODE = 1640 B-FIELD DECODE = 2	A1LE
I1R17Q	A1R01E*	OP CODE = 1640 B-FIELD DECODE = 3	A1LE

The magnitude of the shift is determined by the code in the T-field of the Op-Code, where a code of 0 produces a 1-bit shift.

b. A2 Register

Function:	Temporary storage register, 16 bits
Use:	Input to adder, A1 register, Data Pad and X DAC.
Mechanization:	The A2 register is implemented with shift register storage elements (8270A) and And-Or-Invert gates (8848A) for data input selection.
Modes of Operation:	The register can be cleared, loaded in parallel or serial fashion and is capable of being right shifted.
Data:	Data is stored in the register in an inverted form, i.e. a high level on the data input lines represents a logical "zero", a low level represents a logical "one".
Parallel Data Entry:	To effect parallel data entry into the register, one of four inputs is selected and the load control term A2L0 is enabled.

Data Input Sources	Data Input Selection Controls
Adder Sum	A2LA
Inverse Adder Sum	A2LB
I1 Register	A2LC
Multiply Product bits 0-16, 0→A2(17)	A2LD

Data Input Selection Controls:	Each of the controls are enabled from several sources: (1) decode of B-field for Op-Codes 2, 10, 12 (2) adder controls, and (3) the control panel.
Load Control:	The load control, A2L0, is enabled: (1) by any of the input controls, (2) when the register is cleared, i.e. reset, and (3) for Op-Code 0 bits 1 or 3.

Right shift: The control term is A2LE which is enabled by Op-Code 1640 or 1641 with a decode in the B-field of 4,5,6 or 7.

Serial Data Entry:

Data Input Sources	Serial Input Sources	Data Input Selection Control	Shift Control
Carry Flag	A2R17E*	OP CODE = 1641 B-FIELD DECODE=4	A2LE
A2R17Q	A2R17E*	OP CODE = 1640 B-FIELD DECODE=5	A2LE
A2R00	A2R17E*	OP CODE = 1640 B-FIELD DECODE=6	A2LE
I2R00Q	A2R17E*	OP CODE = 1640 A-FIELD AND B-FIELD DECODE=7	A2LE

The magnitude of the sift is determined by the code in the T-field of the Op-Code, where a code of 0 produces a 1-bit shift.

c. I1 Register

Function: Temporary storage register, 16 bits

Use: Input to Adder, A1 register, A2 register, I2 register, M1 register, M2 register, Instruction register, MOS Memory, and I/O controllers.

Mechanization: The I1 register is implemented with shift register storage elements (8270A) and And-Or-Invert gates (8848A) for data input selection.

Modes of Operation: The register can be cleared, loaded in parallel or serial fashion and is capable of being left shifted.

Data: Data is stored in the register in an inverted form, i.e. a high level on the data input lines represents a logical "zero", a low level represents a logical "one".

Parallel Data Entry: To effect parallel data entry into the register, one of four inputs is selected and the load control term I1LO is enabled.

Data Input Sources	Data Input Selection Controls
E1 register (0-17)	I1LA
Adder Sum	I1LB
MOS Memory	I1LC
Pad bits (0-7)→I1 (0-7)	I1LDR
Pad bits (10-17)→I1 (10-17)	I1LDL

Data Input Selection Controls: Each of the controls is enabled from several sources: (1) decode of B-field for Op-Codes 5, 11, 13, (2) adder controls and, (3) the control panel. The input of the E1 register bits 10-17 can be inhibited with I1LAIL*. The input of the E1 register bits 0-7 can be inhibited with I1LAIR*. Both of these controls are enabled from source (1) above, and from a decode of 6 or 7 in the B-field.

Load Control: The load control, I1LO, is enabled: (1) by any of the input controls, (2) when the register is cleared, ie. reset, and (3) for Op-Code, 0 bits 2 or 6.

Left Shift: The control term is I1LE which is enabled by Op-Code 1640 or 1641 with a decode of the A-field of 1, 2, or 3.

Serial Data Entry:

Data Input Sources	Serial Input Sources	Data Input Selection Control	Shift Control
I1F1Q	I1R01E*	OP CODE = 1641 A-FIELD DECODE=1	I1LE
HRF2Q	I1R01E*	OP CODE = 1641 A-FIELD DECODE=2	I1LE
I1R17Q	I1R01E*	OP CODE = 1640 A-FIELD DECODE = 2	I1LE
A1R17Q	I1R01E*	OP CODE = 1640 A-FIELD AND B-FIELD DECODE = 7	I1LE

The magnitude of the shift is determined by the code in the T-field of the Op-Code, where a code of 0 produces a 1-bit shift.

d. I2 Register

Function: Temporary storage register, 16-bits.

Use: Input to Adder, Data Pad, Analog DAC, and Y DAC.

Mechanization: The I2 register is implemented with shift register storage elements (8270A) and And-Or-Invert gates (8848A) for data input selection.

Modes of Operation: The register can be cleared, loaded in parallel or serial fashion and is capable of being right shifted.

Data: Data is stored in the register in an inverted form, i.e., a high level on the data input lines represents a logical "zero", a low level represents a logical "one".

Parallel Data Entry: To effect parallel data entry into the register, one of four inputs is selected and the load control term I2LO is enabled.

Data Input Sources	Data Input Selection Controls
E3 bits (10-13), E2 bits (0-7)	I2LA
Adder Sum	I2LB
I1 Register	I2LC
Multiply Product	I2LD

Data Input Selection Controls: Each of the controls is enabled from several sources: (1) decode of A-field for Op-Codes 1,4,12,13, (2) adder controls, and (3) the control panel.

Load Control: The load control, I2LO is enabled: (1) by any of the input controls,

(2) when the register is cleared, i.e., reset, and (3) for Op-Code 0 bits 10 or 11.

Right Shift: The control term is I2LE which is enabled by Op-Code 1640 or 1641 with a decode in the A-field of 4,5,6 or 7.

Serial Data Entry:

Data Input Sources	Serial Input Sources	Data Input Selection Control	Shift Control
I2F1Q	I2R17E*	OP CODE = 1641 A-Field DECODE = 4	†
I2R17Q	I2R17E*	OP CODE = 1640 A-Field DECODE = 5	†
I2R00Q	I2R17E*	OP CODE = 1640 A-Field DECODE = 6	†
A2R00Q	I2R17E*	OP CODE = 1640 A-Field DECODE = 7	†

† The magnitude of the shift is determined by the code in the T-field of the Op-Code, where a code of 0 produces a 1-bit shift.

e. M1 Register

Function: Temporary storage register, 8-bits.

Use: Input to the Multiplier and Data Pad Bus.

Mechanization: The M1 register is implemented with J-K flip-flop storage elements (8H21A) and And-Or-Invert gates (8848A) for data input selection.

Modes of Operation: The register can be cleared and loaded in parallel.

Data: Data is stored in the register in a non inverted form, i.e. a high level on the data input lines represents a logical "one", a low level represents a logical "zero".

Parallel Data Entry: To effect parallel data entry into the register, one of four inputs is selected and the load control term M1LO is enabled.

Data Input Sources	Data Input Selection Controls
Data Pad Bits (10-17)	M1LA
Multiply Product Bits (10-17)	M1LB
I1 Register Bits (0-7)	M1LC
Data Pad Bits (0-7)	M1LD

Data Input Selection Controls: Each of the controls is enabled from several sources: (1) decode of C-field for Op-Codes 3 or 4, and (3) the control panel. The Data Pad input bit (17) can be inhibited with a C-field decode of 7.

Load Control: The load control, M1LO, is enabled: (1) by any of the input controls, (2) when the register is cleared, i.e. an Op-Code of 3 or 4 with a C-field decode of 5.

f. M2 Register

Function: Temporary storage register, 8-bits.

Use: Input to the Multiplier and the Data Pad Bus.

Mechanization: The M2 register is implemented with J-K flip-flop storage elements (8H21A) and And-Or-Invert gates (8848A) for data input selection.

Modes of Operation: The register can be cleared and loaded in parallel.

Data: Data is stored in the register in a non inverted form, i.e. a high level on the data input lines represents a logical "one", a low level represents a logical "zero".

Parallel Data Entry: To effect parallel data entry into the register, one of three inputs is selected and the load control term M2LO is enabled.

Data Input Sources	Data Input Selection Controls
A1 Register bits (10-17)	M2LA
E1 Register bits (0-7, reverse ordered)	M2LB
I1 Regiser bits (0-7)	M2LC
A1 Register bits (0-7)	M2LD

Data Input Selection Controls: Each of the controls are enabled from the decode of B-field for Op-Codes 3 or 4, and the control panel.

Load Control: The load control, M2LO is enabled by any of the input controls or when the register is cleared, i.e. an Op-Code of 3 or 4 with a B-field decode of 5.

g. Flag Register. Each register, A1, A2, I1, I2, M1, M2, PD, HALT has 2 flag bits associated with it. All of these bits together comprise the flag register. This register uses 8270A parallel entry shift register for storage elements and And-Or-Invert gates for input selection. The And-Or-Invert gates have been expanded for the M1, M2, I1, A1, A2 register inputs. The bits in the flag register are used as sign magnitude sign bits, or for program control whenever extra bits are required. Each flag can be loaded from several sources. In mode 0 and Op-Code 1647 the flag picked by the decode of the A & B-fields is set. In mode 1 and Op-Code 1647 the flag picked by the decode of the A & B-fields is cleared. The flags are numbered as follows:

00	I1F1	10	I1F2
01	A1F1	11	I1F2
02	A2F1	12	A2F2
03	I2F1	13	I2F2
04	M1F1	14	M1F2
05	M2F1	15	M2F2
06	PDF1	16	PDF2
07	HF1	17	HF2

The flags hold their data by feeding the output back into one of the inputs in the And-Or-Invert gates. This input is enabled at all times except when other inputs are used. When a register is loaded from the panel, its flags are also loaded. The load path is from E3 to I1F1 & I1F2 to the appropriate register. The other transfers into the flags will be discussed with the instructions that cause them. The flag register can also be shifted with Op-Code 1641.

4-79. MEMORY UNIT

4-80. OVERVIEW. The standard main memory contains 16K, 36-bit words and is implemented on 8 MOS memory storage cards, each with a capacity of 4K by 18 bits. The memory is divided into four modules -- A, B, C & D each with independent controls. Each module contains an address register, a data input register, output gating and two MOS storage cards. The memory system can be expanded to 32K, 36-bit words by inserting an additional eight MOS storage cards (two per module).

4-81. The memory storage element is an AMS6002 high speed 1K by 1 bit, P-channel MOS dynamic random-access storage device. Stored information can be non-destructively read but since it is dynamic it must be refreshed periodically. Refreshing of the total memory requires 32 cycles every 2 ms or one cycle every 60 µsec. Refresh control is handled automatically by the memory controller. The system access and cycle time is 667 ns/36-bit word in a random access mode. This same time applies for instructions that are executed out of MOS.

4-82. In the load commands such as, load instruction pad, load data pad and store data pad, the memory addressing , by interleaving the individual memory modules, is always incremental and four words are fetched within a single memory cycle. Except for an initial set up time of 500 ns, 36-bit words are loaded at a clock time rate of 167 ns. The 64, 28-bit words of instruction pad can be loaded in 11.3 µsec, and the 64, 16-bit words of data pad can be loaded in 6 µsec, since they are loaded in pairs. The store pad command for a store of 64, 16-bit words requires 11.3 µseconds.

4-83. DOCUMENTS.

Memory Controls	3A1, 3A2	Block Diagrams 21001-100
Memory Address Register	3B	-2a,2b,2c,2d
Memory Data	3C	Logic Diagrams 23100 thru 23104

4-84. MEMORY CONTROLLER. Memory control is implemented using a 4 channel controller. The channels are designated 0 through 3 with priority assigned in ascending order, i.e. channel 0 has the highest priority. Channel assignment is:

Channel 0 -- Refresh logic
Channel 1 -- MP
Channel 2 -- DMA #1
Channel 3 -- DMA #2

4-85. TIMING. Whenever a cycle is requested the appropriate cycle request flip-flop (CDCROQ - CDCR03Q) is set and remains on until the cycle is awarded. This is signified by a cycle acknowledge signal (CDCA0* - CDCA3*) being generated at the first clock in the 4 clock memory cycle. A memory cycle is defined by CDINAQ followed by CDCAQ, CDCBQ & CDCQ. Memory busy (CDBSYQ) sets with CDCAQ and is on for 3 clocks. CDCQ remains on until the next initiate signal, Figure 4-2 . CDINAQ is used to gate the appropriate set of address and data lines onto the address bus (CA01*-CA15*) and the memory data bus (CDI00*-CDI35*) which go to all memory modules.

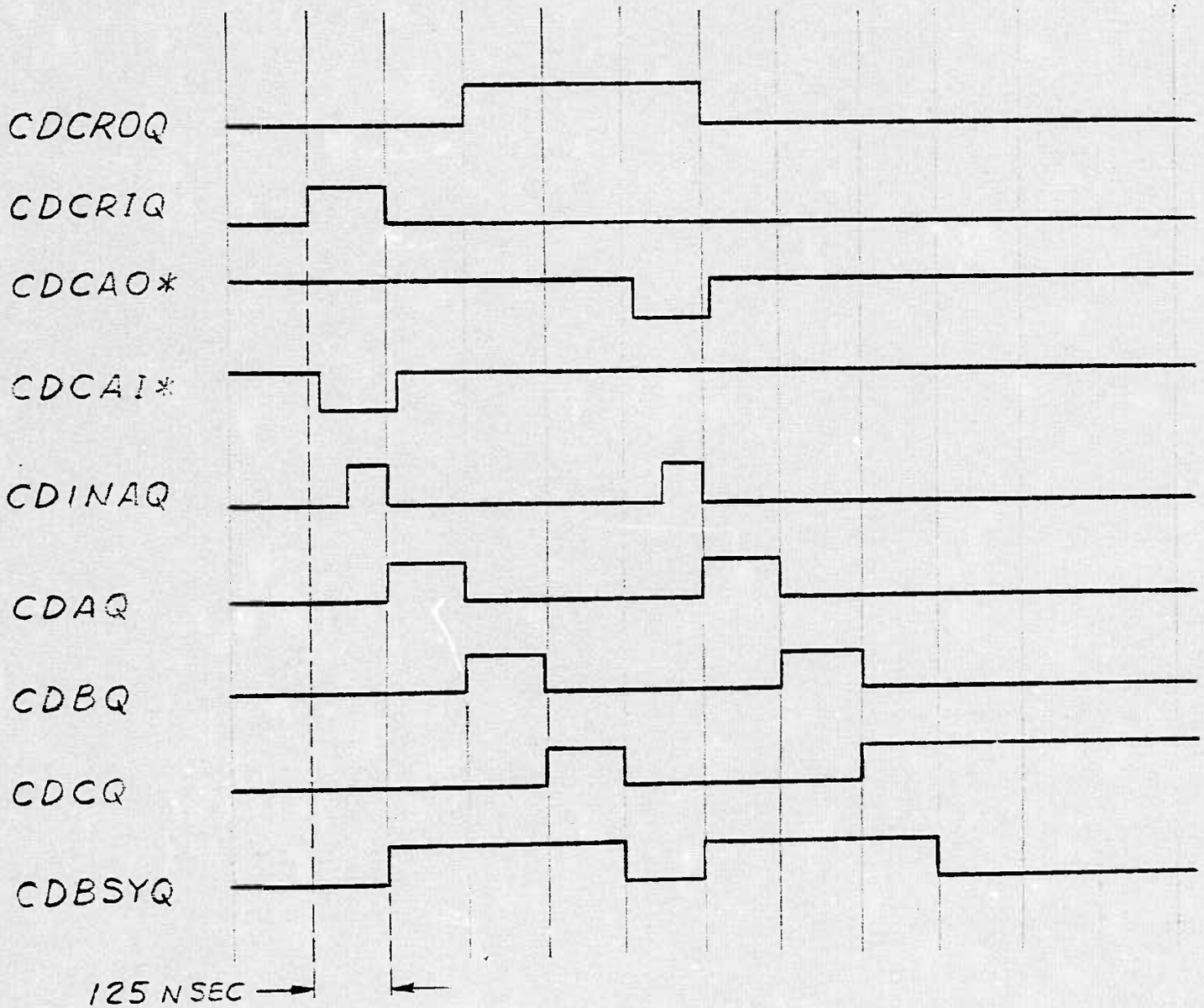


FIGURE 4-2 MEMORY CYCLE TIMING

4-86. The four states of CA01* and CA02* are decoded to provide the correct initiate signal (CDINAA*, CDINAB*, CDINAC*, or CDINAD*) which occurs at about mid-clock during the cycle acknowledge pulse. The leading edge of the initiate pulse is used to sample the address bus and input data register of the proper module. All of the internal memory module control signals such as reset, clock, chip select read and write enables are derived using one-shot multivibrators and gate delays from the initiate signal. If the MP is continuously requesting memory cycles, CDINAQ will set every 667ns. If the MP is in sequential mode, CDINAQ will set at mid-clock every 167 ns and remain on for about 50 ns.

4-87. REFRESH CONTROL. Whenever a refresh cycle is required, CDREFQ* becomes active setting cycle request 0 (CDCROQ). If the memory is not busy, cycle acknwoeldge 0 (CDCYACKO*) is sent to the memory which initiates a refresh cycle. Figure 4-2 illustrates a memory cycle being awarded to the MP followed by a refresh cycle.

4-88. MEMORY WRITE. Whenever a write command is executed, CDWRTQ sets and causes a memory write cycle at the next initiate pulse. If the command were a half word write, CDSNGLQ would be set. The least significant bit in the memory address register, CAROOQ, is used as a left/right control, when the memory is in a half word (18 bit) mode. CAROOQ=1 indicates right half.

4-89. MEMORY READ. During a read cycle data from the appropriate memory storage module is gated onto the memory data output bus (CDOO*-CD35*) under control of CRO1Q & CRO2Q. CRO1Q & CRO2Q are a copy of memory address register bits 1 and 2 except in sequential mode.

4-90. SEQUENTIAL MODE. Sequential mode is entered by calling for a memory access with TCB13Q=1. This sets count control, CDCCQ, permitting CDCCO and CDCC1 to count. While CDCCQ is on, the memory address register (CA) increments by 2's and four sequential memory locations are initiated. After 500 ns all memory modules have a word in their output data register which can be strobed every 125 ns. While in sequential mode, CA points to the last word that has been initiated while CRO1 & CRO2 (memory read control) are four double words behind.

4-91. MEMORY ADDRESS REGISTER

Function:	Temporary Storage
Use:	Accepts memory address data from the adder sum lines or the Instruction Buffer Register. Feeds memory address bus $(CA01^*-CA15^*)_{10}$ which loads the address register of the selected memory module. Enable term-CDCA1*. Transfer control terms: CDINAA*, CDINAB*, CDINAC* or CDINAD*. Can be used as an auxiliary arithmetic register (it is the only register that can be loaded with a 16-bit constant from the instruction.

Mechanization:

The memory address register is implemented with counter storage elements (SN7461N), D-type flip-flop (SN74S74N), 2 input Multiplexers (SN74S158) for input selection and tri-state bus drivers (DM8096) for memory address bus.

Mode of Operation:

The register can be loaded in parallel, incremented by 2 and cleared.

Data:

The data is stored in a non-inverted form, i.e. a high level represents a "one" and a low-level represents a "zero".

Parallel Data Entry:

To effect parallel data entry the load control term, CALA*, is enabled. The input to the register is always the adder sum except in mode 3, in which case it is IBR.

Data Input Load Control:

The load control, CALA*, is enabled by TC=0 and one of the following:

a. Mode 1, Op-Codes 1-13
b. Adder functions to CAR (CALA2*)
c. Loaded from panel (CARPNL2*)
d. Mode 2, Op-Code 13 (OPM13*)
e. Mode 3 (MODE3*)

Increment by 2:

Memory address register CAR(01-17)Q are composed of synchronous counter chips that increment by one when both the P&T inputs are a logical "one." P is the count control and T is the carry input. The least significant bit (CAR00Q) is a D-type flip flop and doesn't count. Increment by one must be done using the adder and the parallel input.

Increment Control:

The increment control, CAINC2, is enabled by TC=0 and one of the following:

a. CDCCQ (enabled when entering sequential mode)
b. LOAD and one of the following op-codes: 1410, 1414, 1420, 1430, 1440 or 1474
c. CEXD1·IBREN (active when in memory execute mode)
d. Mode 1 and adder code of 73.
e. JMPQ·IBRKENQ*

4-92. MEMORY DATA

4-93. Memory Data Bus (CD00Q-CD21Q)

Function: Multiplexes left and right half words from MOS memory.

Use: Allows 36 bit memory word to be treated as two 18 bit data words. Select left half control - CAR00Q*1.

Mechanization: The memory data bus is implemented with 2-input multiplexer chips (SN74S158).

Data: The data is stored in a non-inverted form, i.e. a high level on the output represents a "one" and a low level represents a"zero".

4-94. Memory Input Data Bus (CDI00*-CDI35*)

Function: Multiplexes the I1 register (I1R00Q*-I1R17Q*) and the Array Processor's accumulator (AC00*-AC31*).

Use: Allows the contents of the I1 register to be written into either the left or the right half of memory, or allows the contents of the array processor's (AP) accumulator to be written into memory. Select AC control is AC2CD*. Disable control is CDCA203A. If this control is "high" the bus is disabled and can be used by DMA1 or DMA2.

Mechanization: The memory data input bus is implemented with 2-input tri-state multiplexer chips (SN74S257N).

Data: The data is stored in an inverted form, i.e. a low level on the output represents a "one" and a high level represents a "zero". When the chip is disabled it is in the high impedance state and the output is floating (unless controlled by another DMA channel).

4-95. I/O INTERFACE UNIT

4-96. DOCUMENTS. The documents which represent the logic design of the I/O Interface Unit are listed below by title and drawing number:

Title	Dwg. No.
E1 Register, Bits 00-17	4A1
E2, E3 Register, Bits 20-33	4A2
S Register & Controls	4B
D Register & Controls	4C
D Register Decoders & I1 Buffers	SP5A
External Bus	SP5B
Keyboard Control Logic	SP5C
Station Select and Scope Control	SP5D
Analog Input Device (AID) - Control & Timer	SP5E
DAC - Registers & DAC Boards	SP5F
Host Interface Logic	SP5G
Disk Controller	
Sequence Logic	SP5H
Error Logic	SP5I
Checksum Logic	SP5J
Write Logic & Clock	SP5K
Reg. - Input, Shift & Output	SP5L
Disk Commd. & Proc. Hand Share	SP5M
Disk Data Demodulator & Select	SP5N
Disk Control, Record Format, Data Cell & Command Seq.	SP5N1
Write Block Diagram	SP5N2
Write Command Sequence	SP5N3
Write Mode Data Transfer & Write Termination Sequence	SP5N4
Composite One Word Write	SP5N5
Read Block Diagram	SP5N6
Read Command Sequence	SP5N7
Read Mode, Data Transfer, Read Term, Data Recovery	SP5N8
Composite One Word Read	SP5N9A
Composite One Word Read	SP5N9B
Seek Block Diagram	SP5N10
Composite Seek	SP5N11
Composite Restore & Command Sequence	SP5N12
Read/Write Control	SP5Ø

4-97. GENERAL CHARACTERISTICS

4-98. All I/O transfers are controlled from the MP*. The I/O Interface Unit is complete with controllers for handling the following devices:

a. Eight Model 114 Disk Storage Drives

*Macroprocessor

b. Analog Input/Output (Sample & Hold, A/D and D/A Converters)

c. Four Model 100 User Stations.

4-99. Additional peripheral devices, including a host computer, can be accommodated by adding the appropriate controllers. The basic I/O instruction incorporates a direct address capability for 16 devices. The instruction can specify up to eight functions. The I/O capability can be easily expanded, in terms of the allowable number of peripheral devices, through the indirect address capability. Bits (0-7) of the I/O instruction provide the means for transferring the contents of the 16-bit I1 register to peripheral device controllers and can specify the contents of I1 as follows: (1) address, (2) single address instruction, and (3) data. Case (1) provides a 2^{16} device address capability. Case (2) provides both device address and function information. The number of allowable device addresses and functions depend on the bit allocation. Case (3) allows data to be transferred to the selected I/O device. Decoding by the controller is required for both case (1) and case (2). I/O devices serviced through the indirect address capability share a common service request bit and address code.

4-100. STANDARD PERIPHERAL DEVICES

4-101. Standard peripherals are as follows:

a. Two to eight Disks: Model 114
Storage capacity = 14.6 million 16-bit words
or 13 million 18-bit words
Data Rate = 2.3 megabits/sec

b. One to Four Model 100 User Stations:

Input: Model 100 Keyboard

Output: Model 611 Storage Display Unit
(2 D/A Converters)

c. AP-90 Array Processor

4-102. I/O STRUCTURE

4-103. The MP communicates with an I/O device through the D and E1 registers. Input data from an I/O device is transferred via the Device Input Bus onto the E_1 Register Bus and then loaded into the E_1 register on command

from the MP. I/O devices request service by setting their service request bit in the S register. These bits (0-7), are tested by the software and the appropriate macros called to process the requests. Each I/O device has a dedicated service request bit (see S register bit assignment, Section 1-58). The command structure in the 8-bit D register is as follows:

a. Four bits of address giving 16 possible I/O devices

b. Three command bits which are expandable by making one command an execute I_1 as an instruction. These commands may have different meanings for each I/O device.

c. One bit, D(7), is used as a device communication bit which indicates to the I/O device that the address and command portion are stable. This bit is first cleared and then set one clock time after the rest of the D register is loaded, to give the lines a chance to settle. The addressed device turns off this bit to indicate command accept.

4-104. REGISTER & BUSES

a. D Register (8 bits)

Use:	The D register is used to hold the device command, I/O device address, and device communication bit.
Loading:	The D register is loaded by means of an Op-Code 14 command. IBR bits (0-6) are transferred to D bits (0-6). One clock time later, bit (7) of the D register is unconditionally set to a one. If IMR bit (0) is a zero, bit (7) of the D register will be cleared when the TC Counter is zero. This allows bit 7 to generate a pulse up to 0.625 μs. If IMR bit (0) is a one, bit (7) of the D register is set and remains on. In this case, bit (7) can be cleared by the addressed device to indicate command accept. The D register can also be loaded from the control panel.

b. S Register (8-bits)

Use:	Service request bits for I/O devices.
Loading:	Each I/O device has its own dedicated service request bit in the S register which it can set by supplying a J to

the J-K flip-flop. The register can also be loaded in parallel from the control panel, or by instruction.

Clearing: The processor can clear a device's service request bit by reading status from that device. (Address 10-17 only), or by command.

c. Device Input Bus (16-bits)

Function: 16 pole, Multi-position Electronic switch.

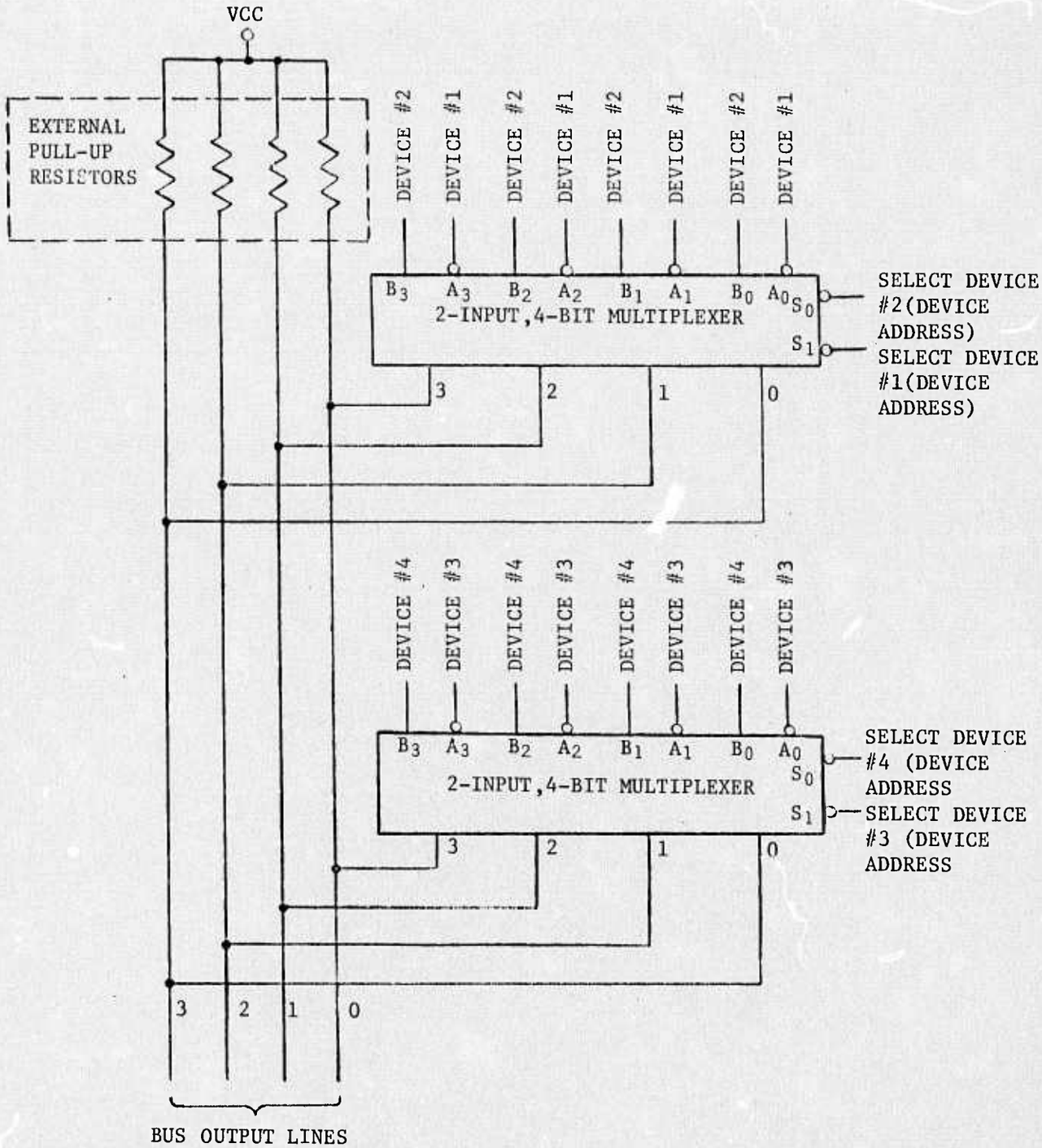
Use: Selective transfer of parallel information from I/O device controllers to E1 register Input-Bus.

Mechanization: The bus is implemented with a plurality of 2-input, 4-bit digital multiplexers 8267B Bare Collector. The Truth Table for the 8267B is shown below with the mechanization shown in Figure 4-3.

Truth Table - Multiplexer, Type 8267B

Select Lines		Output
S_0	S_1	fn (0,1,2,3)
0	0	B_n
0	1	B_n
1	0	$\overline{A}_n$
1	1	1

Bus Inputs	Control Terms Device Selection
Keyboard-0, Data	DADR0*
Keyboard-1, Data	DADR1*
Keyboard-2, Data	DADR2*
Keyboard-3, Data	DADR3*
Display Units (4), Status	DSPLEN*
Disk, Data	DSKDATA*
Disk, Status	ADDSTAT*
Disk Cylinder Address & Disk Drive No.	POSTAT*
A/D Converter, Digital Output Code	DADR12*
Host Computer	DADR13*
Real Time Counter	DADR14*
Slow I/Ø	DADR15*
Array Processor	DADR17*



NOTES: 1. FOUR 4-BIT MULTIPLEXERS ARE USED IN THE HORIZONTAL DIMENSION TO PROVIDE A 16-BIT DATA PATH.
2. DEVICE INPUTS TO THE BUS REPRESENT DATA OR DEVICE STATUS

Figure 4-3. Device Input Bus Mechanization
4 Devices 4-Bit Data Path

Device Selection Controls: The information on the device lines, data or status, is transferred to the bus output lines by enabling a particular control term. A unique control term is enabled when the device is addressed through the enable I/O instruction (14XXX). The control terms are decodes of the address field when data is being transferred. The control terms are derived from the address and command fields to transfer status.

d. E1 Register Bus. The E1 register bus is one of three independent sections of the 28-bit E-Register Bus. The E1 Bus provides a means of selectively transferring data from multiple sources to the E1-Register. The mechanization of the E Register Buses is similar to the mechanization of the Device Input Bus. The E1 Register Bus controls are described in the section, E-Register Load Controls.

e. E Register (28 bits). The E register, from a hardware viewpoint, forms a single register 28-bits in length. The 28-bit register has been portioned functionally into three independent registers designated as follows: E3, bits (30-33); E2, bits (20-27); E1, bits (0-17). E3 and E2 are not strictly part of the I/O Interface Unit but are discussed here for convenience.

Use: The E1 register is used primarily for temporary storage when transferring data into the machine in both manual and automatic modes. E2 is used as an address save for PA & E3 used for breakpoints.

Loading: E1 can be loaded from: the control panel, external bus, and pad input bus. [E2 can be loaded from: the control panel, I2 register, Pad Address Register, and Instruction Buffer Register. E3 is loaded from the panel]

D-type flops, SN74S74N, are used as storage elements. Input data is multiplexed onto the E input bus with open collector multiplexers, 8267B, organized as three independent bus structures. A hold feature is incorporated by feeding the output of the E registers back to the corresponding multiplexer input. This input is enabled whenever a transfer is not taking place. It is possible to change the state of any bit in the E register from the panel. When the processor is stopped, the inverse of E is multiplexed into the E input bus. The clock input to the 74S74's is also multiplexed between the normal 8 MHz clock in the run mode, and the E button latches in the stop mode. When the processor is stopped, the transition of any one of the E button latches will clock the 74S74's and cause it to toggle.

4-105. PANEL LOAD (Manual Mode). E-Register bits can be selectively changed from the Control Panel. The Q* output of each E Register flip-flop is fed back to its corresponding input via a multiplexer (input bus). The control term, STP1Q* enables the complement of the current state of E onto the bus when the machine is stopped. The clock input to the E-Register flip-flops is multiplexed between the normal 8 MHz clock in the Run mode and the E-Button latches in the stopped mode. In the stopped mode, when an E-Register switch is actuated, the corresponding latch changes state producing a clock pulse. The clock pulse causes the storage flip-flop to change state since the complement of its current state appears at its input. Figure 4-4 describes the Manual Mode logic.

4-106. E-REGISTER LOAD CONTROLS. Transfers into E1, E2, and E3 are all independent, and each section has its own hold control. When a transfer takes place, the appropriate hold term must be disabled. Each command , that changes the state of a particular section of the E-Register, disables the hold control to that section. Op-Code 16, D & C fields of 6 and 4 respectively, clears E1 and E2. This command disables the E1 and E2 hold contr ıs. The following table gives control names and the transfer performed for transfer to E1, E2, and E3.

Control Name	Transfer to E1
EDPD1R*	Pad Data, bits (0-7) to E1 (0-7)
EDPD1L*	Pad Data, bits (10-17) to E1 (10-17)
EEXTOQ*	I/O Bus to E1 (00-17)
ESWAP*	E1 (0-7) to E1 (10-17) E1 (10-17) to E1 (0-7)
STP1Q* - (Manual load)	E1*, E2*, E3* to E1, E2 & E3
E1HLDR*	Hold Control, Bits (0-7)
E1HLDM*	Hold Control, Bits (10-17)

Control Name	Transfer to E2
Op-Code = 14, D=6, C=0	IB (0-7) to E2 (0-7)
PAE*	PA (0-5) to E2 (0-5), 0→E2 (6-7)
E1D1*	I2 (0-7) to E2 (0-7)
E1HLDL*	Hold Control, Bits 0-7)
STP1Q*	Manual Load

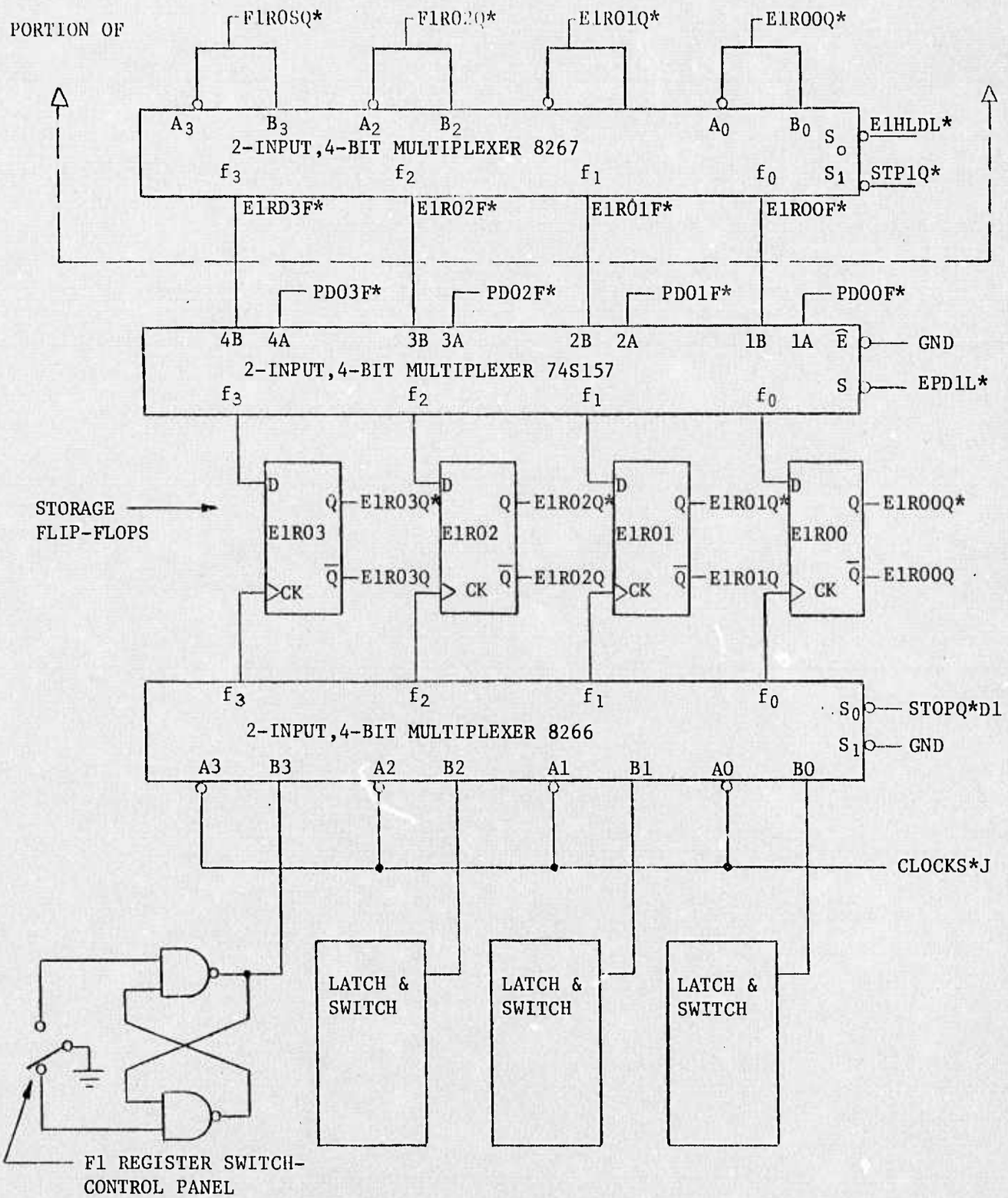


Figure 4-4. Manual Load E1-Register

Transfers to E3 are from control panel latches only and are enabled in stop or run mode.

Most of the E-Bus controls are enabled through C-field decodes for Op-Codes 1,2,5 and mode 2, OP-CODE 1. The function performed for each decode is given in the register operation tables section of TMA5.

4-107 PERIPHERAL CONTROLLERS. Controllers provide the electronic links between the MP and peripheral devices. The standard controllers of the MP-32A are the Station Controller and the Disk Controller.

4-108 Station Controller. The Station Controller can handle up to four User-Stations. Each User Station incorporates a keyboard unit and a display unit. A single stereo analog channel, comprised of a switched Sample and Hold module and a 10-bit A/D converter (input) and two 10-bit D/A converters (output) is timed shared by multiple stations.

(a) Keyboard Controller. The Keyboard Controller loads the key data into an 8-bit storage register and signals to the processor that a key has been depressed by turning on the keyboard interrupt bit. The keyboard output consists of 8 data signals and a strobe signal. The strobe signal is used to derive the basic timing and control signals within the controller. The equipment within the controller per keyboard, shown in Figure 4-5 , consists of an 8-bit storage register, a D-Type flip-flop which is set asynchronously from the strobe signal, one stage of synchronous storage, plus control and timing logic. A keyboard output is transmitted to the controller on 8 data lines and a strobe line. The data lines are connected to the inputs of the 8-bit storage register. The strobe line output drives a pulse forming network. The output of the pulse forming networks supplies a clock signal to the strobe flip-flop and enables the load control to the 8-bit storage register. During the positive level of the output of the pulse forming network, key data is entered into the storage register. When the pulse goes negative, the strobe flip-flop is set. The output of the pulse forming network is inverted prior to being fed to the clock input of the D-type flip-flop. The Q* output of the strobe flip-flop is transferred to the assigned bit of a four-bit register at system clock time. The outputs of the four bit register, one per keyboard, are mixed in an OR gate and used to set the interrupt flip-flop. When the MP loads the contents of the storage register into E1 (D I to E1 transfer), the strobe and interrupt flip-flops are reset. The controller is then ready to accept a new key code.

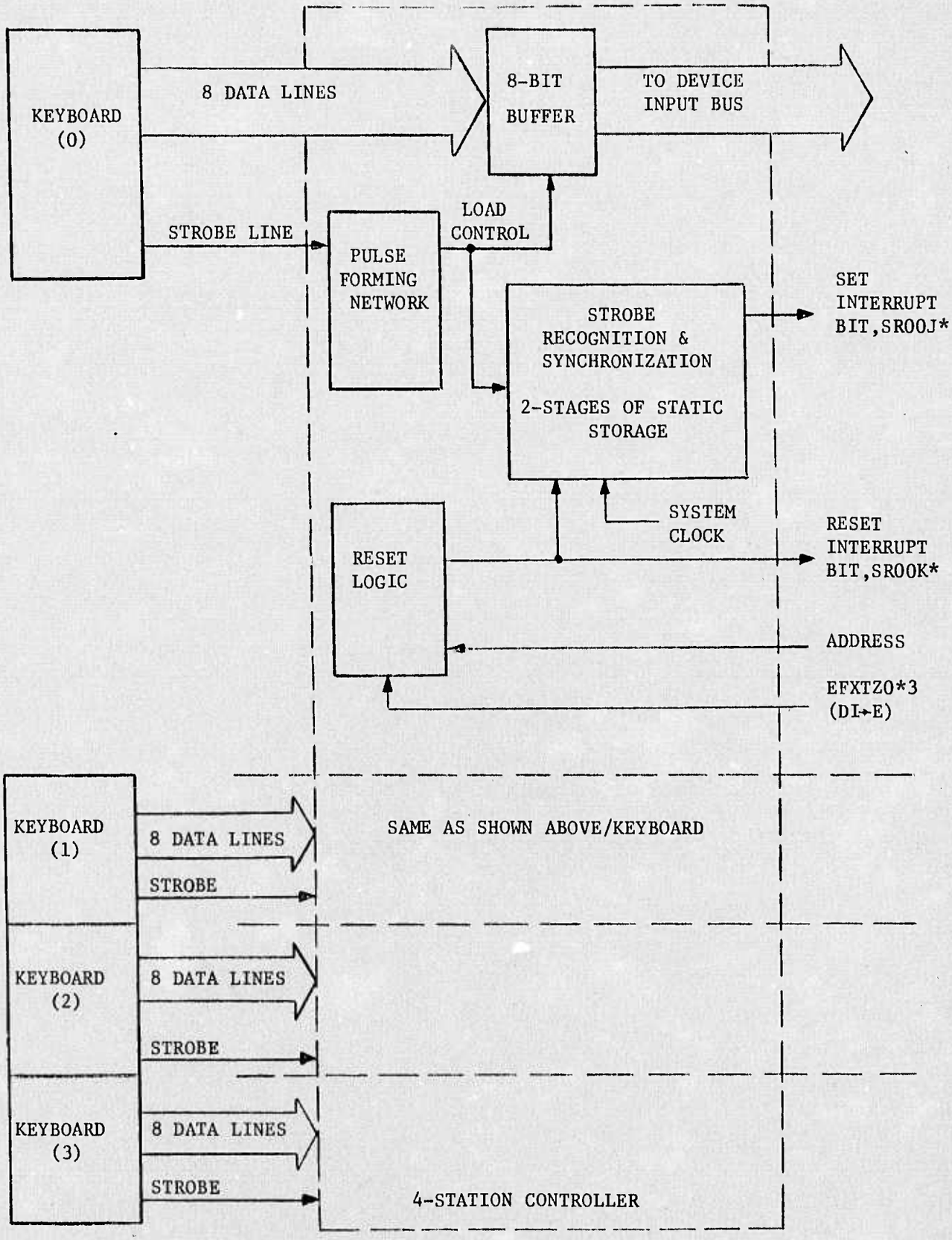


Figure 4-5. Keyboard Controller Functional Block Diagram

(b) Display Controller. The Display Controller effects the execution of all display functions. The unblank Display instruction, command decode of 1, provides a means for writing information on the storage target. The instruction is executed in two basic time periods: beam positioning and write. At the onset of the instruction, the contents of A2 and I2 are transferred to the X and Y D/A converter registers. Following a delay of 6-8 μs beam positioning time, the beam is turned on for a period of 24 μs. The gross instruction timing intervals are derived from a 4-bit binary counter which is driven with a 500KC square-wave from the Timer. The counter is gated on at the fourth clock of the instruction. When the counter reaches the count of 4, approximately 8 μs, the writing beam is turn on. The beam remains on through the count of 16, write time of 24 μs. If the display controller had previously been placed in the slew mode by execution of a 147042 command, the writing beam is turned on immediately, and remains on. Slew mode automatically terminates 32 μsec after the last unblank display command. When a different display address has been selected or the beam is to be moved more than 1 centimeter additional beam settling time must be allowed (5 μsec + 3.5 sec/cm). This is done with a load X,Y, DAC command 147041 followed by a programmatic wait of the appropriate amount. A particular display unit, one of four, is selected through the two low order address bits, DR04 and DR03. The contents of DR04 and DR03 are transferred to the display address register. The address bits are mixed with the DCOM2 signal in a binary to octal decoder. The DCOM2 signal effectively partitions the eight outputs into two groups of four Write control signals and four Erase control signals. The Write signals are selected for a command decode of one and the Erase signals for a command decode of two. A functional block diagram depicting the implementation and gross timing of the Write Display instruction is shown in Fig. 4-6 . The Erase Display instruction, command decode of 2, employs much of the same equipment as the Write instruction: address register, timing counter, control flip-flops, address and function decoder. For the Erase Display instruction, the DCOM2 signal sets the DCOM2 flipflop. The Q* output of the DCOM2 flip-flop is then at a low logic level and the erase function is selected. The address bits specify which of the four display units receive the erase pulse from the controller & triggers a 0.5 μs one shot, the duration of the Erase cycle, in the selected unit. The active unit sends a busy signal to the computer via its status line during the Erase interval.

4-109. Disk Controller

4-110. Physical Description

The disk control unit (DCU) is located on bays M and N of the I/O plane. The following three connections serve to tie the DCU to the external world.

Connector J122 ties the DCU to the Model 114 Disk Drive

Connector J121 ties the command/(write data)lines from the MP to the DCU

Connector J121 ties the DCU status/(read data) lines to the MP.

4-111. Functional Description.

The DCU is the electronic link between the Macroprocessor and the Model 114 Disk Drive as shown in Dwg.SP5N. The function of the DCU is to

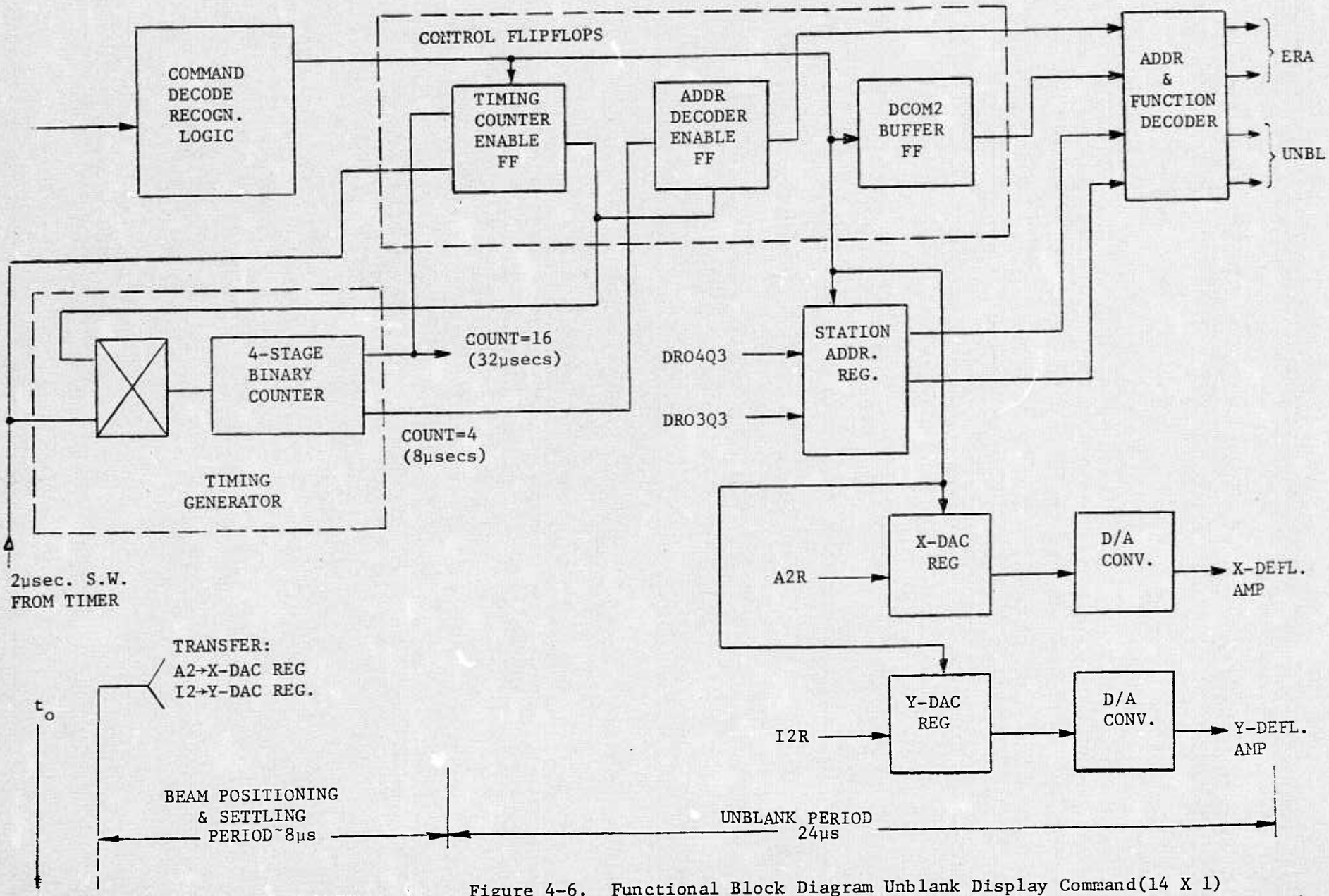


Figure 4-6. Functional Block Diagram Unblank Display Command(14 X 1)

enable the MP to read or write information on one of the 20 recording surfaces of the disk thus providing the MP with a storage capacity of approximately 29 million bytes.

4-112. Basic Registers - The basic registers used in the disk control unit are listed and defined below:

(a) IØ Register (4 bits): Bit number four is used for synchronizing the command and related lines coming from the MP. Bits 3 and 4 are used to generate a 200 ns pulse which loads the input buffer with the contents of the I1 register. Bits 1 and 2 are both 200 ns pulses meaning data accept and command accept respectively.

(b) Input Buffer Register (16 bits): For read and seek commands, the input buffer register holds the original 16 bit command. When the actual write data transfer begins however, the data is transferred into the IB register thereby clearing the original command. For this reason we have a command buffer register.

(c) Command Buffer Register (2 bits): This register buffers the coded form of the command issued. The register contents are not cleared at the termination of the command.

(d) Dynamic Shift Register (18 bits): In write mode the 16 or 18 data word is transferred in parallel to the DSR and shifted serially out to be written on disk. The DSR in read mode accepts data serially from the disk. When one 16 or 18 bit data word has been loaded, it is transferred in parallel to the output buffer register.

(e) Output Buffer Register (18 bits): This register is used for data transfer from the disk controller to the Macroprocessor in read mode.

(f) Checksum Register (18 bits): The Checksum register is used only in read or write mode. It is initially set to all ones. For each data transfer mode the checksum register is updated. In write mode, the last word written onto disk is the current checksum word. For this case, the 16 or 18 bits in the checksum register are serially shifted out onto disk.

4-113. Communication Signals - The signals connecting the disk control unit to the Macroprocessor are listed and defined below.

(a) DCOMO is a data tranfer command signal for read or write mode. When the controller receives this signal, the input lines are interpreted as data, when IODRDY="1".

(b) DCOM1 is a request from the computer that status be multiplexed on the output lines.

(c) DCOM2 is the "command here" line meaning the input lines can be interpreted as a command. This line may initiate a command sequence.

(d) DCOM3 assumes that either a read or write command has been previously issued. This signal signifies end of block and hence terminates the read or write. The timing relationships for terminating are shown in Dwgs.SP5N4 & SP5N8.

(e) DCOM4 causes a restore command to be issued to the disk drive, resetting the head assembly to cylinder zero.

(f) DCOM5 is a request from the computer that position status be multiplexed on the output lines.

(g) DCOM7 is a command to select a specific disk drive from bits (10-12) of I1.

(h) DADR10 signifies that the disk is addressed. The disk controller recognizes its signals only when addressed.

(i) IØDRDYQ3 has two meanings depending on whether a read or write data transfer is currently being executed. In the case of read mode, IØDRDYQ3 signifies that the computer may read the data on the controller output lines. In write mode, it indicates (to the disk controller) a data word is available. Note that unless the disk is being addressed, simultaneously, this line is meaningless.

(j) RES*3 signifies that the reset button on the computer control panel was pushed. The disk controller is initialized.

(k) IØDRDYJ2*D. This 200 ns pulse is generated only when the disk is addressed, the disk is in read data mode and the output buffer has first been filled with a data word. Its function is to set the IØDRDY bit in the computer.

(l) IØDRDYK2*D is a 200 ns pulse generated when the disk is addressed, the disk is in write data mode and a data word has been accepted into the input buffer. Its function is to clear the IØDRDY bit in the computer.

(m) S1R01J2 is a 200 ns interrupt pulse which sets bit 1 of the S register in the computer to indicate that a seek operation has been completed.

(n) S1R06J2 is a 200 ns interrupt pulse which sets bit 6 of the S register in the computer. The generation of this bit is attributed to various conditions occurring within the interface. This is further discussed in Section

(o) DR07K2*D is a 200 ns pulse meaning that the command issued by the computer has been accepted by the disk controller for execution.

(p) I1R(00-21)Q3 are the 18 input data lines feeding the input buffer. These lines are used for commands and write data issued to the disk controller.

(q) E1X(00-21)D* are the 18 output data lines feeding the computer. These lines are used by the disk controller for transmitting output read data and status to the computer.

4-114. Operation

This Section describes the control features on the MP control panel relating specifically to the operation of the Model 114 Disk Drive. The instruction features are described in Sections 1-56 thru 1-59 and 4-152 thru 4-167.

4-115. Turn on Procedure - The 208/230 volt power source for the Model 114 Disk Drive has intentionally been made available at the MP. The MP's "system power" switch applies 115 volt power to itself as well as supplying 208/230 volt power to the disk drive. The Model 114 Disk Drive is under direct control of the MP, provided that the disks "AC power switch" and the white "power-on" pushbutton are both activated. Thus applying power to the MP also initiates the disk power-on sequence. Powering the disk down is simply a matter of placing the MP's "system power" switch in the off position. If any other source of 208/230 volt is used, the power on and off sequence for the disk must be made according to the Model 114 Interface Specification.

4-116. Disk Status Light - The disk status light, located behind the Macroprocessor door, signifies that the disk is in an online and not unsafe condition. When this light is on, the disk is ready to accept a command.

4-117. Cylinder Protect Switch - The cylinder protect switch in the on position indicates that if an attempt is made to write data on cylinder 0, the write command will be immediately aborted and an error flag posted. In this mode, cylinder 0 serves as read only memory. In the off position, cylinder 0 is no longer protected. This switch is located on the maintenance panel.

4-118. Reset Button - The system reset button on the Macroprocessor control panel is used to initialize the disk control unit logic. All status flags are reset along with several registers and flip flops as will be as discussed in Section 4-133.

4-119. Disk Control Unit Specifications

General Specifications

Word length: 18 bits
Word rate: 6.4 μsec
Error detection: Checksum
Interrupts: One (S1R06J2)
Location: Bay M and N, row 1 thru 20 on the I/O plane.

Disk Capacity Specification

Maximum # of data words: 18 bit format is 12,992,000
16 bit format is 14,616,000

This specification provides for the maximum number of words which can be written on a fixed specified track. The number of data words written must be such that the DCU is able to complete the write command, enter into a not busy state and prepare to execute another command on the next record. This specification allows one word time for the MP to recognize the last interrupt corresponding to a not busy condition and issue the next command. Thus the MP may do a full track write of 3200 18 bit or 3600 16 bit data words beginning at head 0 followed by another full track write at head 1 in two consecutive disk revolutions. Dwg.SP5N1 shows the record format and the overhead involved in reading or writing a record.

4-120. Write Mode Free Timing - Write Mode Free Time is defined during the write data transfer mode as the time interval the MP may leave the DCU

unattended without a write overflow error occurring. This time interval extends from that time at which the MP's disk interrupt bit is guaranteed to be set to that time at which the MP may still answer the data request without a write overflow error occurring. This is illustrated in Dwg. SP5N4 from which the write mode free time is found to be 5.875 µs.

4-121. Read Mode Free Time - Read Mode Free Time is defined during the read data transfer mode as the time interval the MP may leave the DCU unattended without a read overflow error occurring. This time interval extends from that time at which the MP's disk interrupt bit is guaranteed to be set to that time at which the MP may still answer the data request without a read overflow error occurring. The corresponding timing diagram is shown in Dwg. SP5N8. For this example, the read mode free time is found to be 5.675 µs.

4-122. Theory of Operation

4-123. Command Definitions - The Model 114 Disk Drive is capable of performing four types of operations as described in Sections 4-154 and 4-160.

4-124. Command Sequence - A command sequence (read, write, seek and select disk drive) is defined as a series of 200 ns pulses. These pulses initialize, load and accept the command issued by the MP. The following signals are necessary to initiate a command sequence.

DSKOK Disk is "on-line" and "not unsafe"

BUSYQ* DCU is not busy

DADR1O MP has disk address line active

DCOM2 MP has "command here" line active

The first step in the command sequence is the initialization of the DCU by the signal CLF* and the resultant loading of the input buffer with the 16 bit command word. The command accept signal (DR07K2*D) indicates that the command presently in the IB register is possible for the disk to execute. In order for a command to be accepted for execution, the command format must be identical to that in 4-160 with the additional restrictions that the head address be less than (24) for the read and write case and that the cylinder address be less than (315) for a seek. If this is not the case, the command will not be accepted (although it will be loaded into IB). The computer will remain in a spin condition with the incorrect command in the MP I1 register. The command sequence is shown in Dwg.SP5N1.

The read and write commands are similar insofar as they both have a head designation. The head digits specify which one of the 20 heads is to perform the actual reading or writing on disk. Notice that neither the read nor write command have a cylinder address available. To change cylinders, a seek command must be issued with the desired cylinder address as specified in Section 4-160.

4-125. Restore Sequence - The restore operation, being unique, has its own restore sequence which is almost identical to the command sequence. One exception is that the sequence is initiatied by DCOM4 only and doesn't require the use of I1. The other exception is that the restore operation does not require that an accept signal be sent to the computer. The Restore command sequence is shown in Dwg.SP5N12.

4-126. General Description of the Write commands - Dwg.SP5N2 shows the block diagram of a write command. Note that each block contains a square with a number inside. This number corresponds to the drawing number on which the actual logic is found. Dwg.SP5N3 shows the write command sequence.

A write command is issued according to the previous discussion and accepted. The computer is then free to leave the DCU provided that it returns to check for an interrupt condition periodically. The length of time the Macroprocessor is permitted to leave the DCU unattended is discussed in Section 4-160. The DCU meanwhile is waiting for the disk to arrive at the index point. The index pulse(INDEXQ) initiates the communication between the DCU and the disk drive. The four following instructions are issued to the disk: gated attention, clear head address register, set head address register, set write gate and erase gate. The disk sequencing is now complete and the actual data transfer can be initiated.

The written data as it is recorded on disk is shown in Dwg.SP5N5.

The first step in actually writing data out on disk is writing a header as Dwg.SP5N1 illustrates. Eighteen words of zero are written using the signal BC288Q. This is followed by writing a 16 bit code word which is multiplexed and parallel loaded into the DSR. The code word is then right shifted as each bit is recorded on disk in its appropriate data cell.

Each data word is initially buffered in the IB register. It is then multiplexed with DTALD and parallel loaded into the DSR. Note that as each data word is tranferred from the input buffer to the DSR, an interrupt is sent

signifying for the computer to load the input buffer with the next data word. After supplying the requested data word, the computer is free to leave the DCU unattended---returning soon after to check for the next data interrupt. The checksum register is updated at this time.

Assume that N data words are to be written on disk. The computer responds to the Nth interrupt pulse by making the Nth word available to the DCU. As soon as the DCU clears IØDRDY the end of block (EOB) command, DCOM3, may be issued. It must be issued before the next interrupt pulse. Note that the DCU must write the checksum word following the last data word. The N+1st interrupt pulse signifies the entire write command has been completed and that the DCU is ready to accept another command. This occurs when erasing is completed, approximately 50 μsec after the checksum word is written.

4-127. General Description of the Read Command - The block diagram for the read command is shown in Dwg.SP5N6, and in many ways is similar to the write command. Dwg.SP5N7 shows the read command sequence.

The command is issued according to Sec.4-123 and accepted. The DCU waits for the disk to arrive at the index point. When this occurs the disk communication begins. The four following instructions are issued to the disk: gated attention, clear head address register, set head address register and finally set read gate. At this point the read data is available on the read coax.

The raw read data consists of a stream of data cells. Each data cell is composed of a sync bit and a data bit. Dwgs.SP5N9A & 9B show the one-word read timing.

The first task for the DCU is stripping the data bit from its corresponding sync pulse. This is accomplished in the Read control Logic in such a way as to provide a 200 ns synchronous level LOAD BIT Q whose function it is to shift the data bit synchronously into the DSR and increment the bit counter.

The first data bits to enter the DCU are the 18 words of "0" as written in the header. When the bit counter equals 256 the scan for the code word is initiated and continues thru the word count of 512. When the 16 bit code word has been shifted into the DSR, the Code Word Logic recognizes the code word and immediately clears the bit counter to zero. This synchronizes the bit counter to the actual recorded data. Each time the bit counter increments 16 or 18 times (if 18 bit format), the signal LDWRD is generated enabling the data word load from the DSR to the output buffer. The 200 ns signal (LDCKSQ) is delayed 200 ns from LDWRD and updates the checksum register with the contents of the output buffer.

Assume that N words are to be read off disk. The MP responds to the Nth interrupt by sampling the Nth data word present on the output lines. After the last word is loaded into the E register the EOB command may be issued. It must be issued before the next interrupt is sent. The DCU continues receiving data until the write checksum word is loaded into the output buffer. Note that the write checksum word should be identical to the current contents of the checksum register so that the final updated contents of the checksum register is zero (if not zero---a checksum error flag is posted). As in the case of a write command, the N+1st interrupt pulse signifies the read command has been terminated and the disk interface is ready to perform

another command. At this time status may be checked to see if there has been a checksum error.

4-128. General Description of the Read and Write Header Command - These commands are similar to the read and write data command with the following exceptions: Reading or writing is started at the index point. In the data commands, reading or writing is started approximately 192 μs after index. This time is controlled by one shot multivibrator INDEXAQ and is long enough for the MP to read the header, test the track number and issue a read or write data command without missing a revolution. The header normally contains only the preamble and 3 words.

4-129. General Description of the Seek Command - The block diagram for the seek command is shown in Dwg.SP5N1Q. The corresponding timing diagram is shown in Dwg.SP5N11. The seek command is issued and accepted, provided the cylinder address to which the disk is to seek is not greater than 312. The communication with the disk drive begins immediately. The three commands issued to the disk are: gated attention, set cylinder address and set seek. At this time an interrupt is issued indicating that the DCU is free to accept any command for a different disk drive when a SEEKRDY signal is received from the disk indicating that the seek has been completed, a position interrupt pulse (SR01J3*) is sent to the Macroprocessor indicating termination.

The question arises as to what happens if the disk is told to seek to its present cylinder. This situation is like any other and an interrupt occurs at termination 6.6 μs after the seek command was issued.

4-130. Initialization - Two types of reset pulses are available in the disk controller: CLF and INPQ. The function of INPQ is to terminate the command in progress. CLF is used to initialize the disk controller logic and is given at the beginning of each command. The flip flops and registers cleared are given as follows:

INPQ	CLF
CBFQ	CBFQ
CRESQ	DBFQ
ENQ	CRESQ
	EOBQ
	ENQ
	AQ,BQ,CQ,DQ,EQ
	TAGQ
	BUSQ
	WRTBSYQ
	WRTDELQ
	EBSQ
	(BC12Q, BC00Q)
	CKSLDQ
	CODWQ
	DTQ
	RDFMTERRQ
	ROFQ
	WOFQ

CLF

BUSYQ
INDEXERRQ
CYLERRQ
OBFQ
CKSERRQ
CKS18Q - CKS01Q

4-131. Interrupt (S1R06J2*) - There are basically three different interpretations of the interrupt line.

The interrupt signal may be an indication of an error condition either in the disk controller or in the disk itself. These conditions are listed as

Read overflow	ROFQ
Write overflow	WOFQ
Cylinder error	CYLERRQ
Index error	INDEXERRQ
Disk unsafe	UNSAFEQ
Seek error	SEEKERRORQ
Read format error	RDFMTERRQ

A status check will show which of the above errors occurred. Note that if one of the above error conditions does occur, the command being executed is terminated. The resulting interrupt pulse is attributed to the Busy flip flop going inactive.

The interrupt signal is used during both read and write data transfers. In read mode, the interrupt signal signifies a word is currently in the output buffer. To answer the interrupt the computer forces the disk address line active and waits for the disk controller to set the IØDRDY bit. If the IØDRDY bit fails to set, check status for an error condition. This is shown in Dwg.SP5N8.

The write mode interrupt is a data request signal. The computer answers the interrupt by placing the requested data in I1, DADR10 and DCOMO active and setting the IØDRDY bit. If the DCU fails to clear IØDRDY, check status for an error condition. The above description is illustrated in Dwg.SP5N4.

Finally the interrupt signal may indicate the successful completion of a command. For the read or write command, this interrupt will occur after the end of block signal is given. For the seek or restore command, interrupt signifies the DCU's completion of the seek or restore. In this case the interrupt signal is attributed to Busy going inactive.

4-132. Error Detection - The type of error detection used in the disk controller is cyclic code checking---the same as that used in the IBM 2841. The checksum word is initialized by first setting the checksum register to all ones ---accomplished by CLF*N. The checksum word is tabulated by performing a 16

or 18 bit "exclusive or" of the contents of the checksum register with each data word. This is illustrated for 2 data words below. The code word is not included in the checksum.

Example 1

Initial Checksum		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Data Word #1	+	0	1	1	0	1	0	1	1	1	1	1	0	0	0	0	0
Checksum Word #1		1	0	0	1	0	1	0	0	0	0	0	1	1	1	1	1
Data Word #2	+	1	0	0	0	1	0	1	1	0	1	1	0	0	1	0	1
Checksum Word #2		0	0	0	1	1	1	1	1	0	1	1	1	1	0	1	0

In the write mode, the data from the input buffer is used to update the checksum word. The final checksum word (in the example - checksum #2) will be written on disk following the last data word. The write checksum word is written on disk by setting the CKSLD flip flop and enabling the shift input of the checksum register. The data source is thus changed from the dynamic shift register to the checksum register.

In read mode each data word is loaded into the output buffer. The data from the output buffer is used to update the checksum register. The last word which updates the checksum register is the write checksum word originally written on disk in write mode. This word is hopefully identical to the current read checksum word. If so, the final contents of the checksum register is zero. If not, a status bit called checksum error is posted.

A timing diagram is shown in Dwgs.SP5N5 & 9B for the write and read cases.

4-133. Status - The following are status flags indicating the status of both the disk controller and disk drive. All status flags relating to the disk controller are cleared in the initialization phase when a new command is issued. Those flags relating to the status of the disk interface are cleared in accordance with the Model 114 Disk Interface Specification.

Status can be checked at any time by the computer by holding the status line (DCOM1) and the disk address line (DADR10) active. They must be present for at least 200 ns. Note also that both the status and the read data are multiplexed over the same set of lines.

The status flags and their corresponding definitions are described below.

0. Checksum error (CKSERRQ)

   This flag, which may be set at the end of a real command indicates that the recorded write checksum word does not agree with the generated read checksum word. This flag may and may not be considered an error depending if the same number of words are read off disk as were originally written.

1. Read overflow (ROFQ)

   This flag can be set only in read mode. The disk controller has read a data word off disk so that the word is in the output buffer. The computer has failed to take the word within the allotted 5.5 μsec. The read overflow flip flop sets and the read is aborted.

2. Read Format error (RDFMTERRQ)

   This flag can be set in read mode only. This error flag indicates either of the following two error conditions: A one has been found in the read header extending from bit count 128 thru 256 or no code word has been found in the preamble. In either case, the read is aborted.

3. Write Overflow (WOFQ)

   This error condition means that the disk controller has requested a data word be supplied from the computer and the computer has failed to respond within the allotted 5.5 μs. Note this error flip flop is disabled when the EOB signal is sent by the computer.

4. Busy (BUSYQ)

   This signal is a level indicating that the disk is currently executing a command. Only when Busy in inactive may other commands be accepted. Restore is the exception and is not contingent on busy being active.

5. Index error (INDEXERRQ*)

   This error flag is set if data is being transferred in either read or write mode and the index marker appears.

6. Cylinder error (CYLERRQ)

   This error flag is set if the cylinder protect switch is on, and an attempt is made to write on cylinder 0. The write command is aborted immediately. The following three status bits represent status conditions within the Disk. A more detailed description follows in the "Specification Interface Model 114," Century Data 91801.

7. Seek Error (SEEKERRORQ)

   The disk drive has failed to generate seekready within 1 second of seek start.

8. Unsafe (UNSAFEQ)

   The disk drive is in an unsafe condition. Safety circuits within the disk, protect the recorded information when this line is active. (See specification interface Model 114 for causes). The unsafe error flag can be cleared by powering down the disk and then powering it back up again.

9. On-line (ONLINEQ)

this signal indicates the heads are extended and the disk drive is ready to be operated by the control unit. Note it requires approximately 45 seconds after the disk is powered up before this line is active.

4-134. INSTRUCTION DESIGN & IMPLEMENTATION

4-135. GENERAL

The MP-32A machine language instruction set is comprised of eight instruction subsets as shown in Table 1-1. Each instruction subset, except for the Mode 3 set CA operation, represents a family of hardware operations. Hardware operations within a subset are selected through field codes to form a single or multiple operation instruction which includes a 6-bit field specifying the address of the next instruction. The allowable hardware operations are selected within a subset by mode, within a mode by Op-Code, and within an Op-Code by D,C,B and A-field definitions. Individual instructions are formulated by assigning unique codes to the fields which constitute an instruction word.

4-136. INSTRUCTION TYPES

4-137. The types of instructions available in the MP-32A, along with the instruction format, was described in Section 1-60 through 1-69. The D,C,B and A-fields of the MP-32A machine language instruction differ significantly in scope from the fields associated with the classical multi-address computer instruction. In the latter, a field represents an operand address or some other singular portion of an overall computer operation. A field within an MP-32A instruction may designate a complete operation. For example, the D-field in the Normal Add Subset Mode-0, designates both 16-bit inputs to the Adder and the destination of the output. A machine operation is specified through a single field code or through a combination of two or more of the D,C,B and A-field codes. A single instruction may then specify from 1 to 4 operations.

4-138. MACHINE LANGUAGE INSTRUCTION SUBSETS

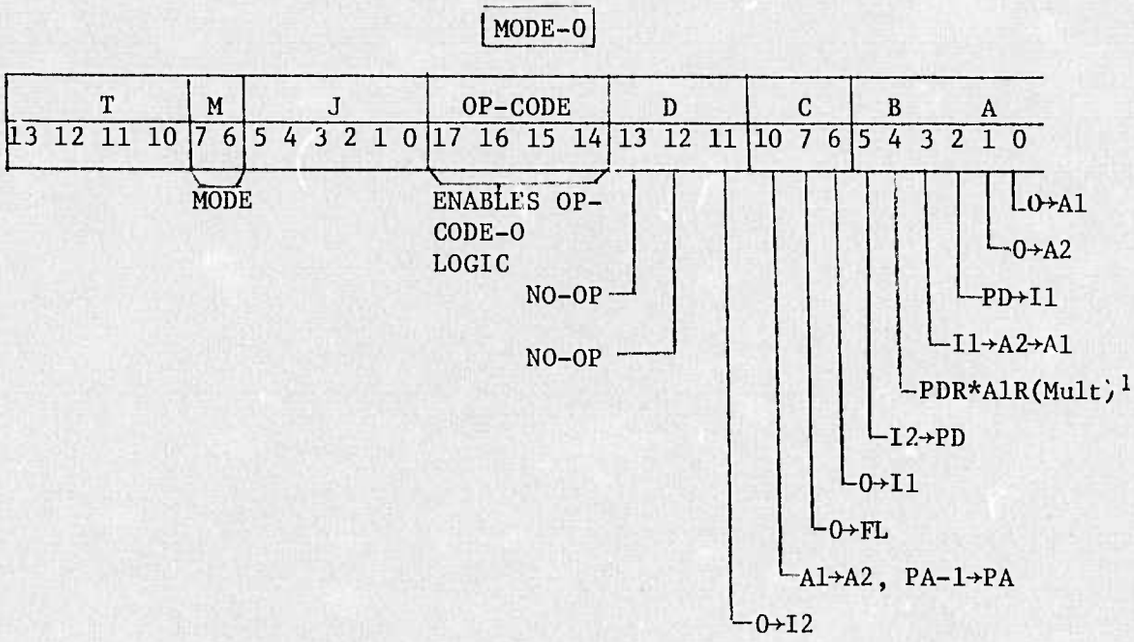
4-139. GENERAL. Concurrent operations per machine instruction require an instruction word which establishes a multiplicity of unique data paths and control terms and a component complement which allows overlapping of different operations. The multi-field instruction word provides the means to establish a set of parallel data paths in combination with control terms which selectively activate machine components. A distributed memory system, in combination with a plurality of arithmetic and logic modules has been incorporated in the equipment to allow overlap of different operations.

4-140. The distributed memory system provides three self-contained memories which can be exercised concurrently. Additionally, the contents of the memory address registers can be changed independent of the principal Adder. The arithmetic unit incorporates: Adder, Multiplier, Logical networks for register and single bit tests, and a complement of 7 high speed registers.

4-141. The major characteristics of the machine components and their relationship to field codes which allow multiple operations per machine instructions is best illustrated through the operation of the arithmetic registers. The arithmetic registers incorporate 4-way input gating. Three of the gates are 2-input structures, one data input and one control input. The fourth gate is a three-input structure, one data input and 2 control inputs. The additional control input is used to modify the normal 16-bit or 8-bit data transfers such as masking individual bits, half-word transfers, etc. The C, B, and A-fields

which control register operations are comprised of 3-bits each, providing 8 unique states per field. The 8 states are utilized as follows: one state defines no operation, four states define the normal data transfers, and the remaining three states effect alternate operations such as modification of the normal data transfers, clearing registers or portions of registers, setting controls and conditional tests, which can cause a program transfer. For register transfer operations the Op-Code, in general, determines those registers which are to receive data and the C,B, and A-fields select the data source.

4-142. SINGLE BIT DECODE OPERATIONS. The single bit decode operation subset incorporates a number of register operations and a special multiply. A unie operation is specified by the state of a single bit of the D, C, B & A-fields as shown by the example below. A single instruction can effect a plurality of parallel operations. The logical And between the signal derived from the decode of the Op-Code field and each of the relative bits of the instructionword is performed in a series of gates. The And term is mixed with other terms in Or gates. The output of the Or gates represent control terms which effect specific operations.



Execution Time

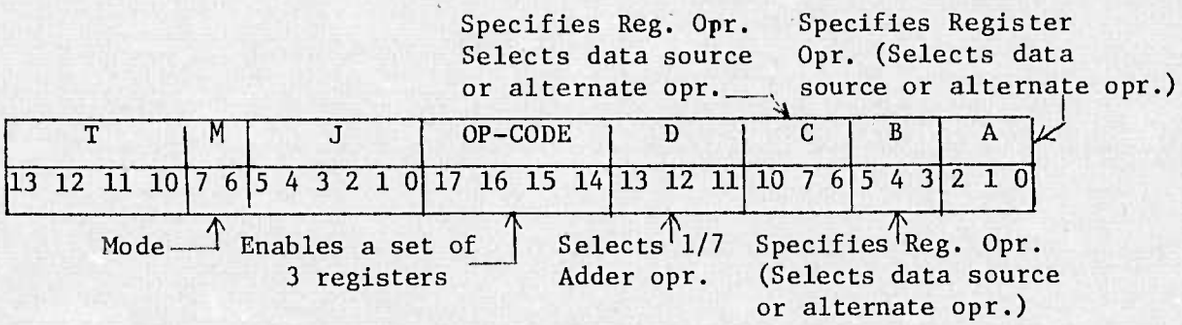
Register Operations:	1 clock
Multiply:	3 clocks

SPECIAL MULTIPLY

Note 1: PDR→M1, PD(17)→M1F1, A1R→M2, A1(17)→M2F1, MP/2→A2, MS→A2F1

4-143. SUCCINCT ADDER. The description of this subset does not apply to a single instruction but to the family of operations associated with a unique mode and Op-Code (D, C, B & A-field unspecified) as shown.

MODES: 0 & 1

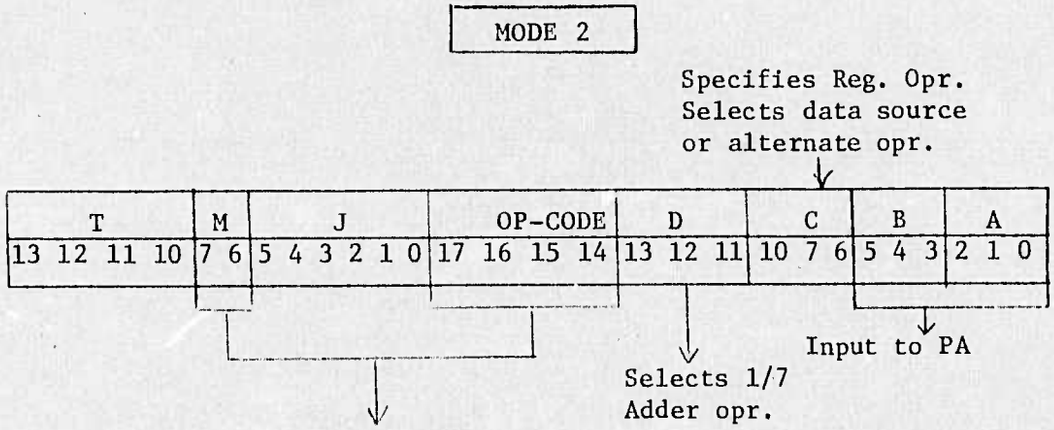


Op-Codes 1-5

Allows the execution of 1 of 7 adder functions in parallel with 3 register transfers or 3 alternate operations.

Remarks: The alternate operations include setting or clearing special controls, left and right half register swaps, data transfers to registers other than the designated registers. Device input bus to E1, device input bus to I1 via E1.

MODE 2



Op-Codes 1-4

Allows the execution of 1 of 7 adder operations in parallel with one register transfer or alternate operation and the load of PA from the low order 6-bits of IB.

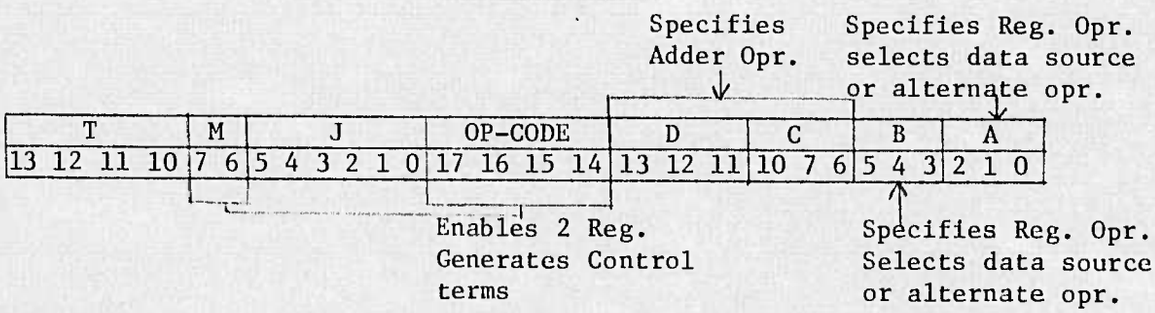
Op-Code 5

Transfers the contents of IBR bits (0-13) to the program source address (PSA) in the selected Array Processor (AP).

4-144. FULL ADDER. This subset differs principally from the previous subset in the allowable number of adder operations. The C and D-fields are combined to specify an adder operation. The combination of the two 3-bit fields provide 64 unique states. One state is allocated to no operation and the remainder to adder operations. One less register operation is allowed.

This subset is described, as shown below, in much the same way as the previous subset.

MODES: 0 & 1

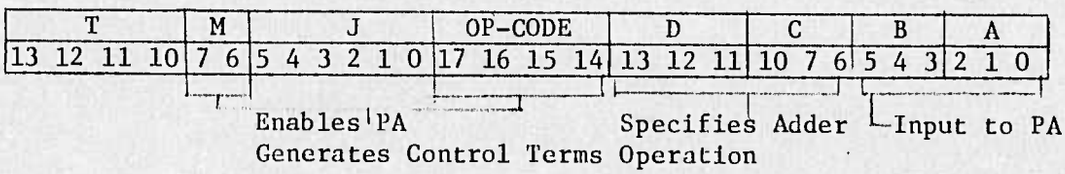


Op-Codes 6-13

Allows the execution of one of 63 adder operations in combination with 2 register transfers or 2 alternate operations.

Remarks: Adder destinations other than the ones specified by the C & D fields may be selected. The selected destination must be enabled by the particular Op-Code and the Adder output must go to the selected destination instead of the one specified by the C&D field.

MODE 2



Op-Code 6-13

Allows the execution of one of 63 adder operations in parallel with the load of PA from the low order 6-bits of IB.

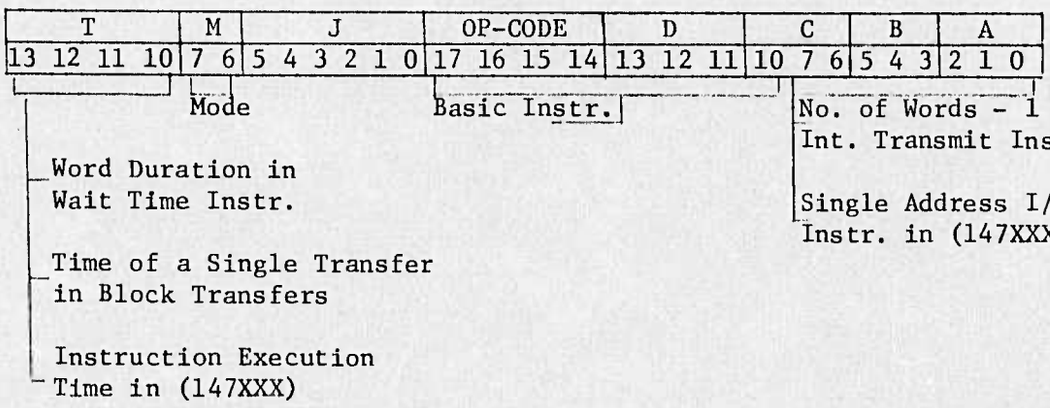
4-145 INPUT/OUTPUT & CONTROL

This instruction subset incorporates internal and external transmit instructions. The internal transmit instructions provide a means of transferring single words or variable length block of words between MP components. The external transmit instruction controls transfers between the MP and peripheral devices. All I/O operations are governed through the enable I/O instruction (147XXX). The bit assignments are shown below.

4-146 For internal transmit instructions, the T-field specifies the rate of individual transfers. The low order 8-bits of the instruction, 2 bits of C in combination with the A & B fields, specify the number of transfers. In transfers involving memory, the memory address register is incremented after each memory access. All internal transmit instructions incorporate the capability of terminating the transfer after a specified number of transfers have been made. The load macro instruction (1414) is representative of the internal transmit instructions.

4-147 The enable I/O instruction constitutes a family of I/O instructions for each peripheral device. The I/O instructions are described in detail in Sections 4-152 through 4-167 and briefly below.

MODE 0



DI→EI→PD, PA+1→PA. D=0, C=0. Data on the output lines of the devices input bus is transferred through EI to Pad Memory. PA is incremented after each transfer. This instruction provides synchronous transfer rates from the device Input Bus to Pad Memory: min. = 1.0 MHz, Max. = 2.67 MHz.

Load PD. D=1, C=0. A double word or a block of double words is transferred from MOS Memory to Pad Memory. The address registers of each memory is incremented by 2 for each transfer. The initial address locations of both memories must be established prior to execution of the Load PD instruction. Execution Times: Double word = .5 μs. Block = Variable (.5 N μs., N = No. of double words). Sequential Mode: Double word = 250 ns. Block = variable (.125 N μsec).

Store PD. D=2, C=0. Double words or a block of double words is transferred from Pad Memory to MOS Memory. The address registers of each memory is incremented by 2 for each transfer. The initial address location of PAD memory must be established prior to the execution of the Store PD instruction. Execution Times: Double word = .375 μs. Block = Variable (.25 N μs. N = No. of words).

Wait. D=5, C=0. Instruction causes a program delay which is variable in 125 ns. increments between the limits of 0.250 μs. and 0.256 ms. Remarks: Delay (μsecs) = .125 (T+1)(W+1)+.125, W = no. of words specified in the low order 8-bits of the instruction word.

IB(7-0) E2. D=6, C=0. The low order eight bits of the Instruction Buffer Register is transferred to the E2 Register. Execution Time: 1 clock Remarks: Register transfer type operation.

Enable I/O (Modes: 0 & 1). D=7, C=0. Remarks: This instruction governs all I/O operations. It is described in Sections 4-155 through 4-167.

MODE 1

E1 I1, E2 I2 (13,0) & 0 I2(17-14), I2 & I1 IR, Incr IA. D=0, C=4. The contents of the E1 and E2 Registers are transferred through I1 and I2 to the Instruction Memory, IR (27-0). IR(33-30) is set to zero. The Instruction Memory Address Register, IA, is incremented for each transfer. Execution Times: (.125 + .375N) μs, N = No. of words.

Load Macro. D=1, C=4. A single 28-bit word or a block of 28-bit words is transferred from MOS Memory to Instruction Memory. The MOS Memory Address Register is incremented by 2 per core access and the instruction Memory Address Register is incremented by one for each 28-bit word written into the Instruction Memory. Execution time: Single word: .5 μs. Block: Variable (.5N μs., N = No. of words written into Instr. Mem.). Remarks: The MOS memory word is 36-bits in length and the instruction word is 28-bits in length. Only the least significant 28 bits of the memory word is used. It is important to note that the last 28-bit word of the block transfer is written into the instruction memory, and additionally transferred to: TC, IMR, IA, and IB and executed if the instruction is Mode 0. If the instruction is Mode 1, CA is incremented by 2 and this instruction is transferred to TC, IMR, IA, IB and executed. The 1414 instruction example, see timing diagram, loads two 28-bit words into the Instruction Memory. Execution time: (.125 + .5N) μs, N = No. of words. Sequential Mode: (.125 + .125N) μs.

Load PD Macro. D=2, C=4. A single 28-bit word or a block of 28-bit words is transferred from Pad Memory to the Instruction Memory. Execution time: Double Word: .375 μs. Block: (.125 + .250N) μs, N = No. of words

written into Instr. Mem. Remarks: This instruction is basically the same as 1414 with the Pad Memory replacing the MOS Memory.

Execute I2, I1. D=4, C=4. Bits (13-0) of I2 is transferred to TC, IMR & IA. Bits (17-0) of I1 is transferred to IB. The composite of the two groups of data form a single instruction which is executed in the normal manner. Execution time .125 μsec plus time to execute instruction in I2, I1.

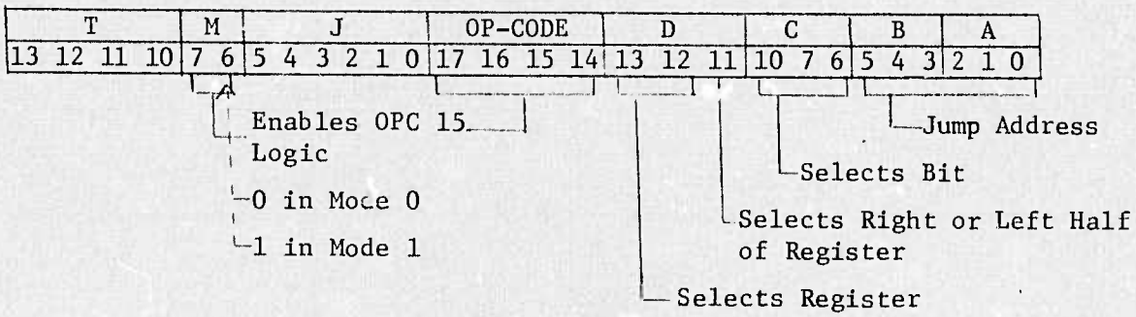
Execute IR, I1. D=5, C=4. Bits (17,0) of the I1 register is transferred to the Instruction Buffer Register and bits (33-20) of IR, at location J, is transferred to TC, IMR, & IA. The composite of the two groups of data form a single instruction which is executed in a normal manner. Execution Time: .125 μsec plus time to execute instruction in IR, I1.

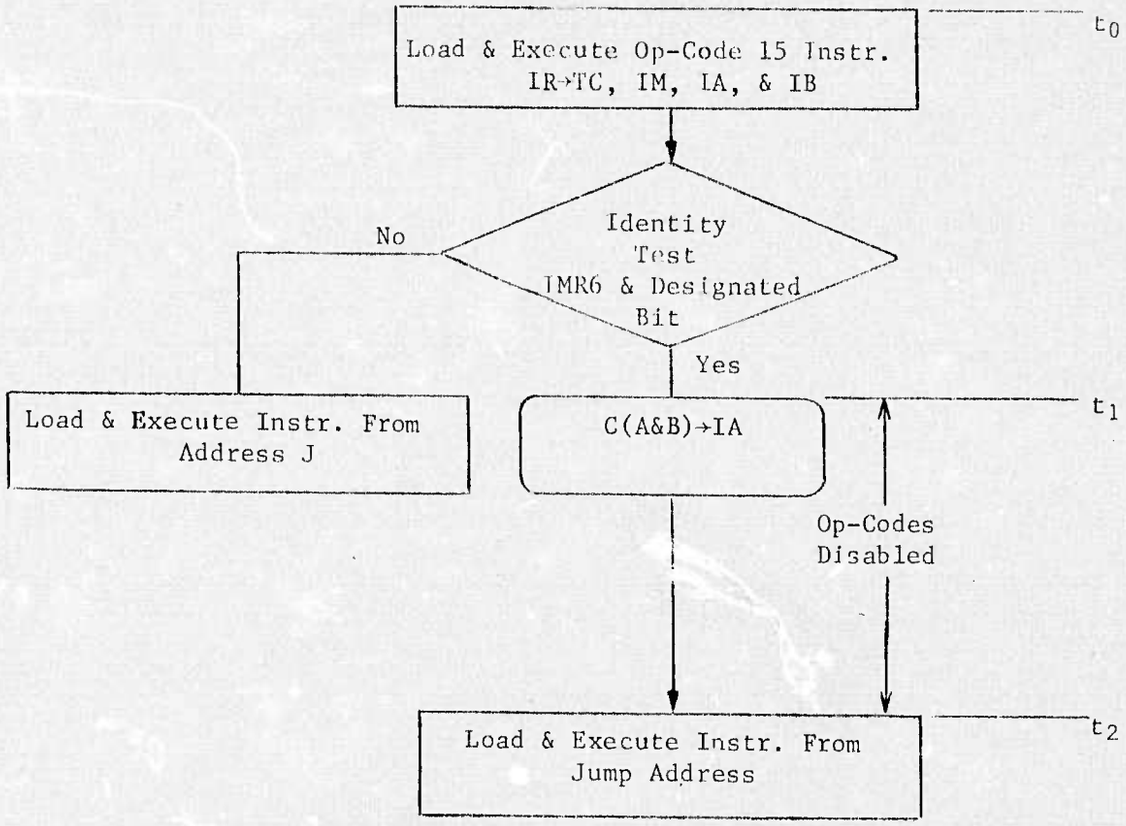
Execute I2, IR. D=6, C=4. Bits (13-0) of I2 is transferred to TC, IMR & IA. Bits (17-0) of IR is tranferred to IB. The composite of the two groups of data form a single instruction which is executed in the normal manner Execution time .125 μsec plus time to execute instruction in I2, IR.

Execute CD. D=7, C=4. Bits (27-16) of memory data (CD) is transferred to TC, IMR, & IA. Bits (15-0) of Memory Data (CD) is transferred to IB. CA is incremented by 2 and the instruction from memory is executed in a normal manner. Execution Time: .125 μs plus time to execute instruction from memory (if instruction was ready) Remarks: Sets memory execute mode (CEXQ=1). Allows instructions to be executed directly from MOS Memory.

4-148. BIT SELECT CONDITIONALS. The Bit Select Conditional instructions provide a program jump capability based on the state of individual bits in any one of four registers (Flag, A2, I1, & E1). In modes 0 & 1, the jump condition is satisfied if the designated bit in the selected register is in the same state as the low order mode bit, IMR06;IMR06 equals zero in Mode 0, and one in Mode 1. Mode 2 instructions provide a means of sequentially testing (scan) all bits in the selected register starting with the MSB (17). The jump condition is based on a bit in the selected register being in the one state. Flow and timing diagrams have been included as a part of the instruction subset description. The timing diagrams incorporate the names and corresponding functions of the principal control terms, (see Figure 4-7).

MODES: 0 & 1





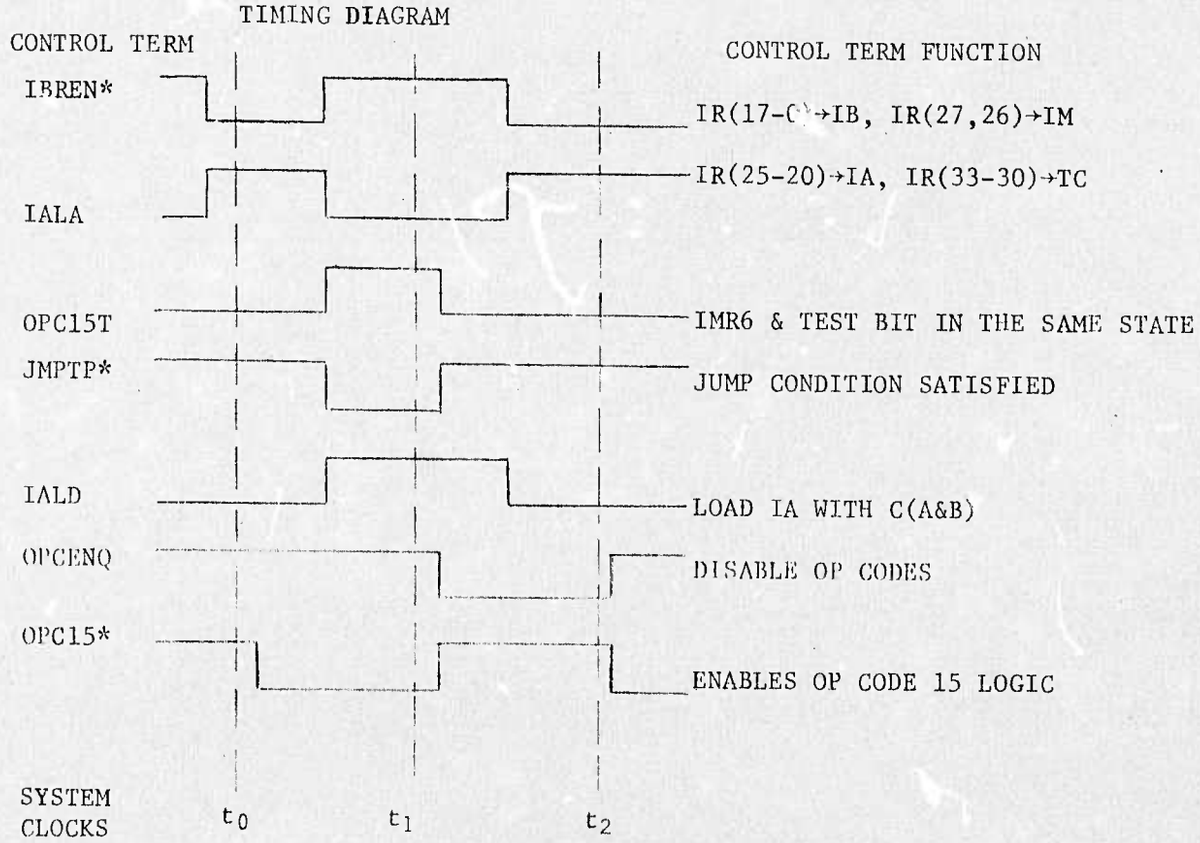


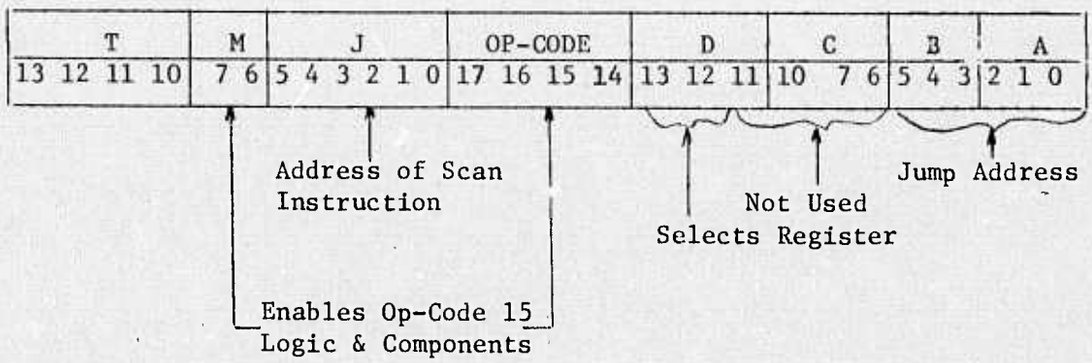
Figure 4-7. Flow & Timing Diagrams
Modes 0 & 1, Op-Code 15

The state of IMR06 is compared with the state of the designated bit in the selected register. If the state of the two bits is the same, jump condition met, the contents of the A & B fields are transferred to the Instruction Address Register. The next instruction is loaded from the jump address and executed. If the state of IMR06 and the designated bit differ, condition not met, the instruction specified by the J-field is loaded and executed.

Execution Time:

Condition met = 2 clock
Condition not met = 1 clock

MODE 2

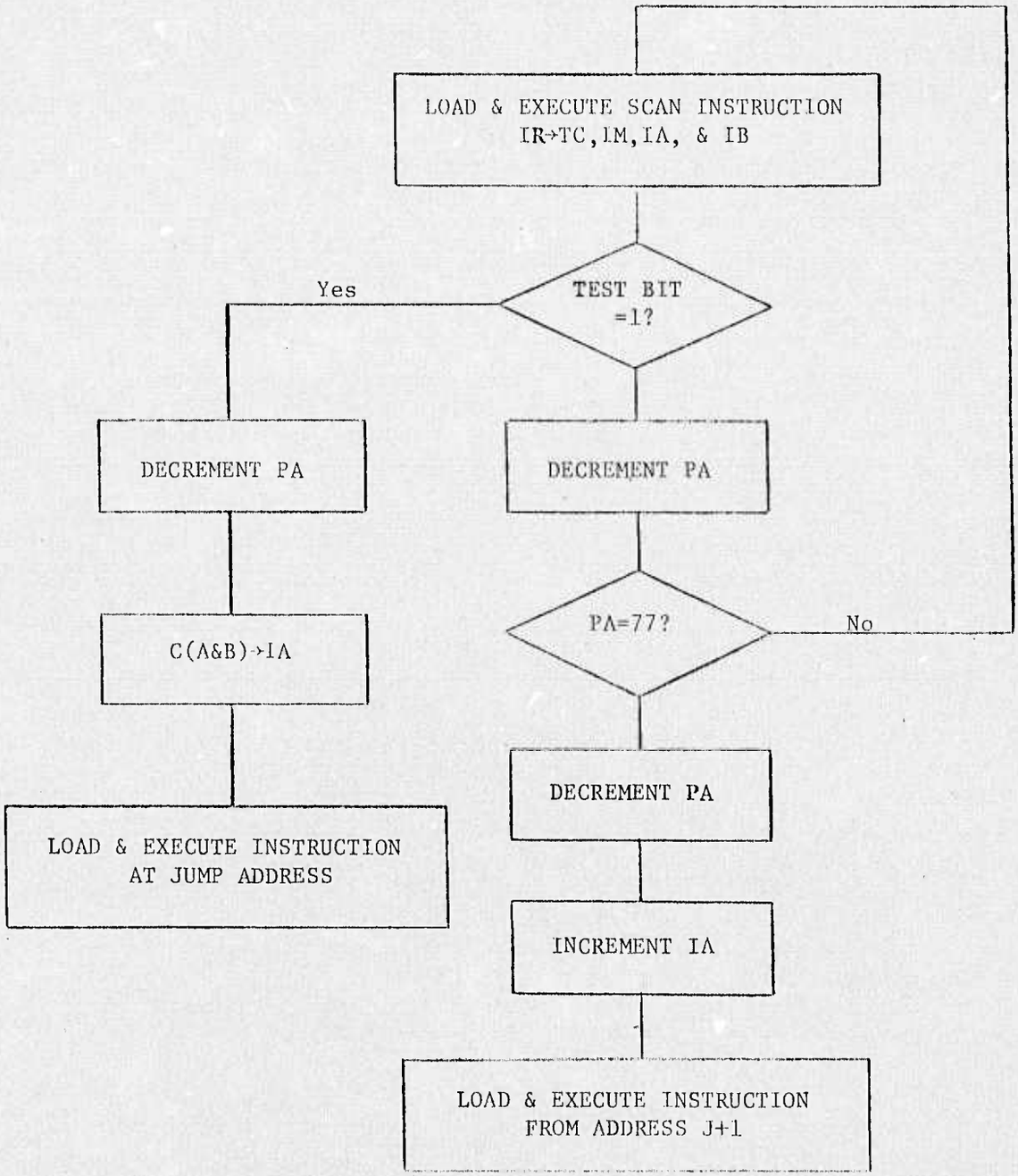


Note: The bit to be tested is designated by the contents of the Pad Address Register.

Each bit of the selected register is tested in sequence starting with the MSB(17). If the test is passed: PA is decremented, and the C(A & B) is loaded into IA. The next instruction is loaded from the jump address and executed. If the test is failed: PA is decremented and the instruction addressed by the J-field is loaded. For Scan, the J-field contains the address of the Scan Instruction, so it is executed continuously until the test is passed or PA reaches 77. If PA reaches 77, i.e. no bit position contained a One, IA is incremented, PA is decremented to 76. The instruction is loaded from the incremented address, J+1, and executed. Figures 4-8 through 4-11 describe the Scan instruction

Execution Time:

Min. = 2 clocks
Max. = 18 clocks



Note: 1. J-Field of Scan Instruction equals Address of Scan Instruction.

Figure 4-8. Flow Diagram Scan Instruction Mode 2, OPC 15

SYSTEM CLOCKS		NO. OF BIT TESTED AT CORRESPONDING CLOCK TIME	
INITIAL LOAD OF SCAN INSTRUCTION	PAD ADDR. REG. CONTENTS		
t_0			
t_1	0 0 1 1 1 1	17	
t_2	0 0 1 1 1 0	16	
t_3	0 0 1 1 0 1	15	
t_4	0 0 1 1 0 0	14	LEFT HALF REGISTER BITS
t_5	0 0 1 0 1 1	13	
t_6	0 0 1 0 1 0	12	
t_7	0 0 1 0 0 1	11	
t_8	0 0 1 0 0 0	10	
t_9	0 0 0 1 1 1	7	
t_{10}	0 0 0 1 1 0	6	
t_{11}	0 0 0 1 0 1	5	
t_{12}	0 0 0 1 0 0	4	RIGHT HALF REGISTER BITS
t_{13}	0 0 0 0 1 1	3	
t_{14}	0 0 0 0 1 0	2	
t_{15}	0 0 0 0 0 1	1	
t_{16}	0 0 0 0 0 0	0	
t_{17}	1 1 1 1 1 1	INC IA & EXIT FROM TEST	
t_{18}	1 1 1 1 1 0	LOAD NEW INSTRUCTION FROM ADDRESS J+1	

NOTES:

1. Pad Address Register set equal to 17_8 prior to loading scan instruction.

2. J-field of scan instruction equal to address of scan instruction.

Figure 4-9. Contents of PA V_s System Clocks in Scan Instruction

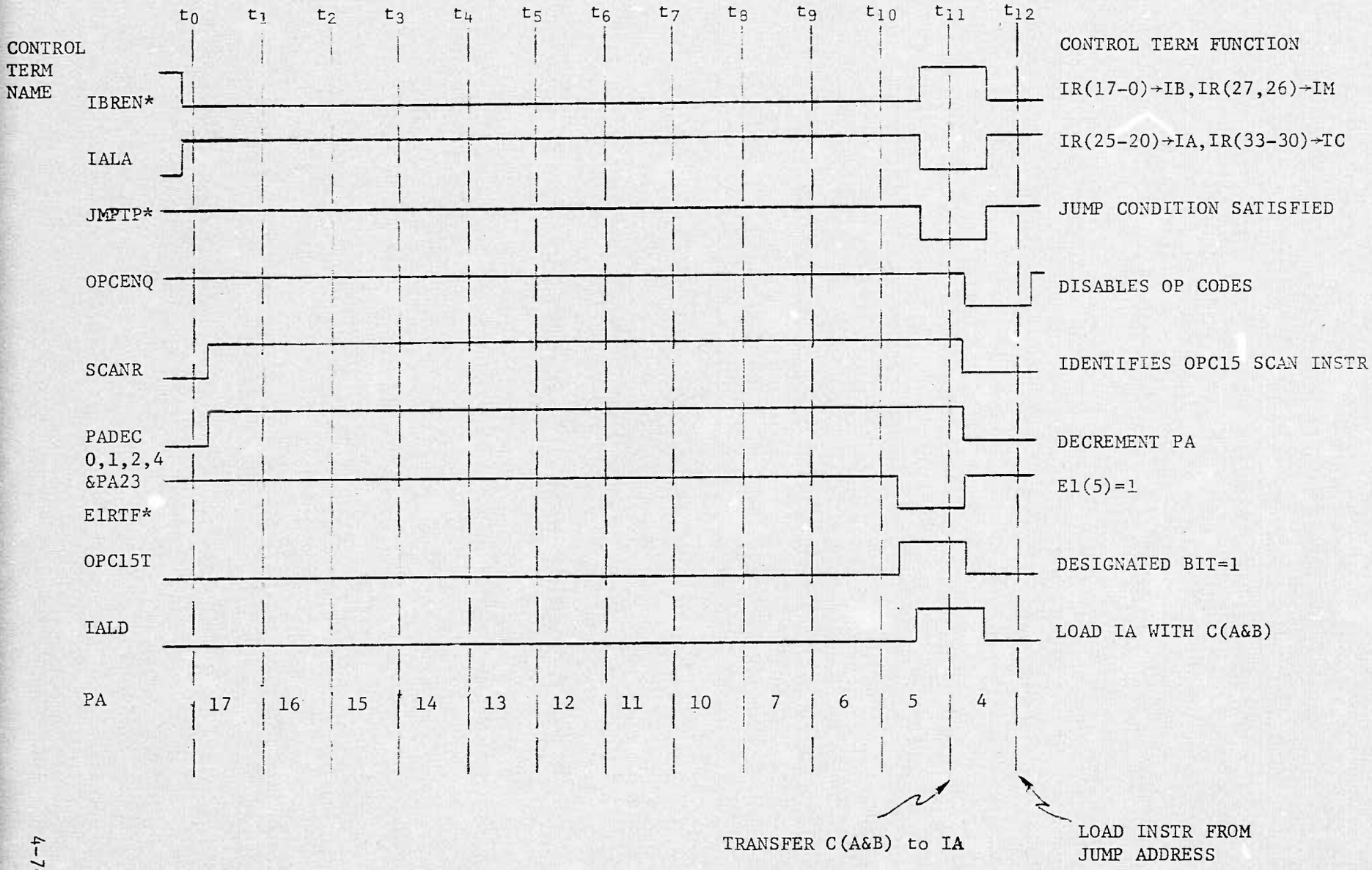


Figure 4-10. Scan Test - Condition Met - Bit(5)$_8$= 1

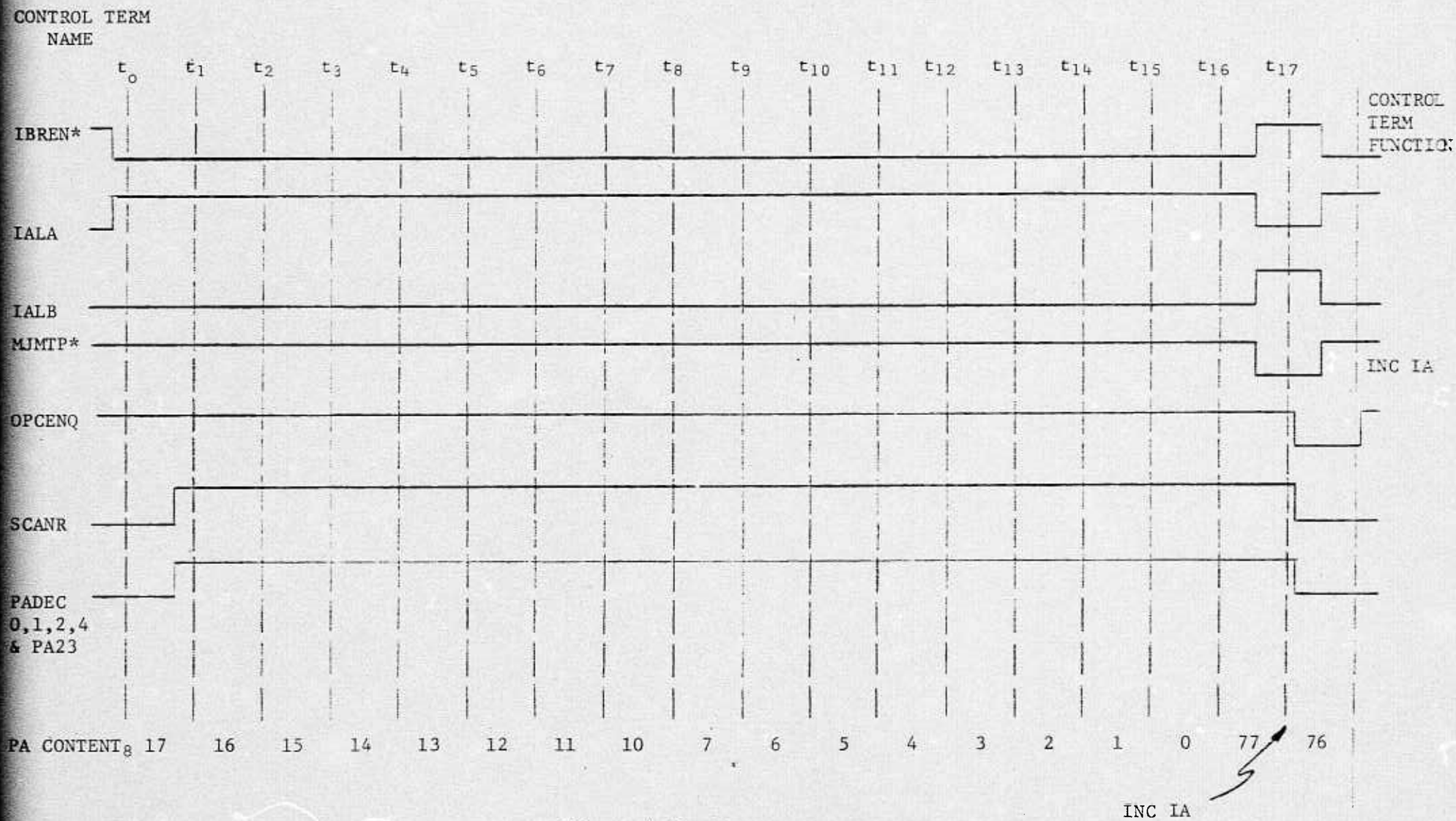


Figure 4-11. Scan Test - Condition not Met
All Bits in Selected Register=0
OPC=15

4- 149 LOGICAL OPERATIONS. This instruction subset incorporates a number of logical operations which can be grouped as follows:

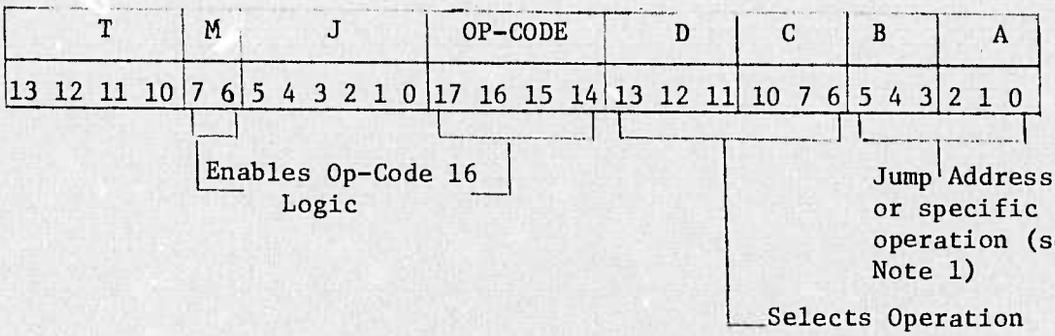
a. Conditional program jump and skip tests.

b. Register shifts, flag transfers, and special transfers to Data Pad Memory.

c. Operations which facilitate servicing I/O devices: test of Device Communication bit D(7), Data Ready bit, E1 Register, and S Register. The S-Register operations include single bit and scan tests.

The bit assignment is shown below:

MODES 0 & 1



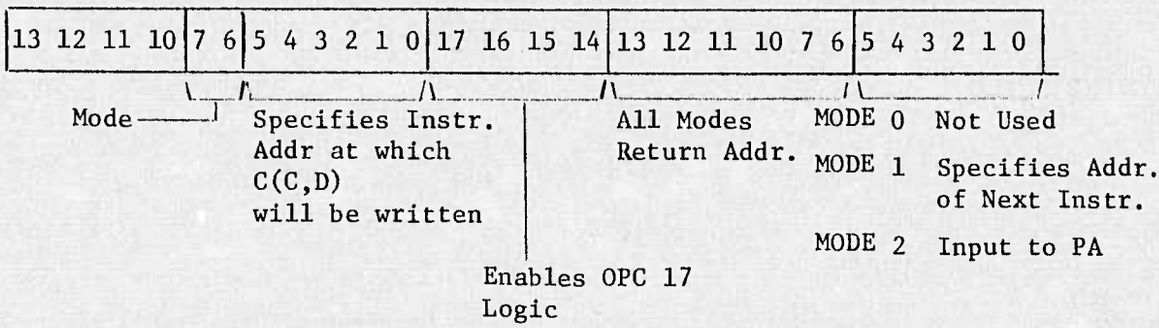
NOTE:

(1) Instruction with D & C-field codes of 40, 41 43, 44,47, 63 and 74 represent a family of operations in which individual operations are selected by a combination of the A & B fields or the A-field alone.

(2) Mode 2 incorporates a single instruction, scan for 1 in the S-Register. The Op-Code 16 scan instruction is identical to the Op-Code 15 scan instruction except for the register under test and the starting value in PA, which is 7 for scan 5.

4-150. LINKAGE OPERATIONS. The Link-Jump is a program control type instruction. These instructions provide the capability of inserting subroutines in the main program. The bit assignment is shown below:

Modes: 0, 1, & 2



Mode 0

The contents of the C & D fields is written into the address portion IR (25-20) of the Instruction Memory Cell addressed by the J-field. The contents of the IR(27,26) is first read and then re-written into the same cell of the Instruction Memory. IR(27 & 26). The Instruction Address Register is incremented and the next instruction is executed from address J+1.

Mode 1

Same as Mode 0, except instead of incrementing IA, the contents of the A & B fields is written into IA and the instruction at the jump address is executed next.

Mode 2

Same as Mode 0 instruction plus the load of Pad Address Register with contents of A & B fields.

EXECUTION TIME All Modes: 3 clocks, T field = 1

Remarks: The flow diagram, Figure 4-12 , shows the primary events for each mode.

4-151. MEMORY CONTROLS. The Set CA instruction is the lone mode 3 instruction. In the execution of Set CA, the low order 16-bits of the instruction word are transferred to the memory address register. The bit assignment is shown below.

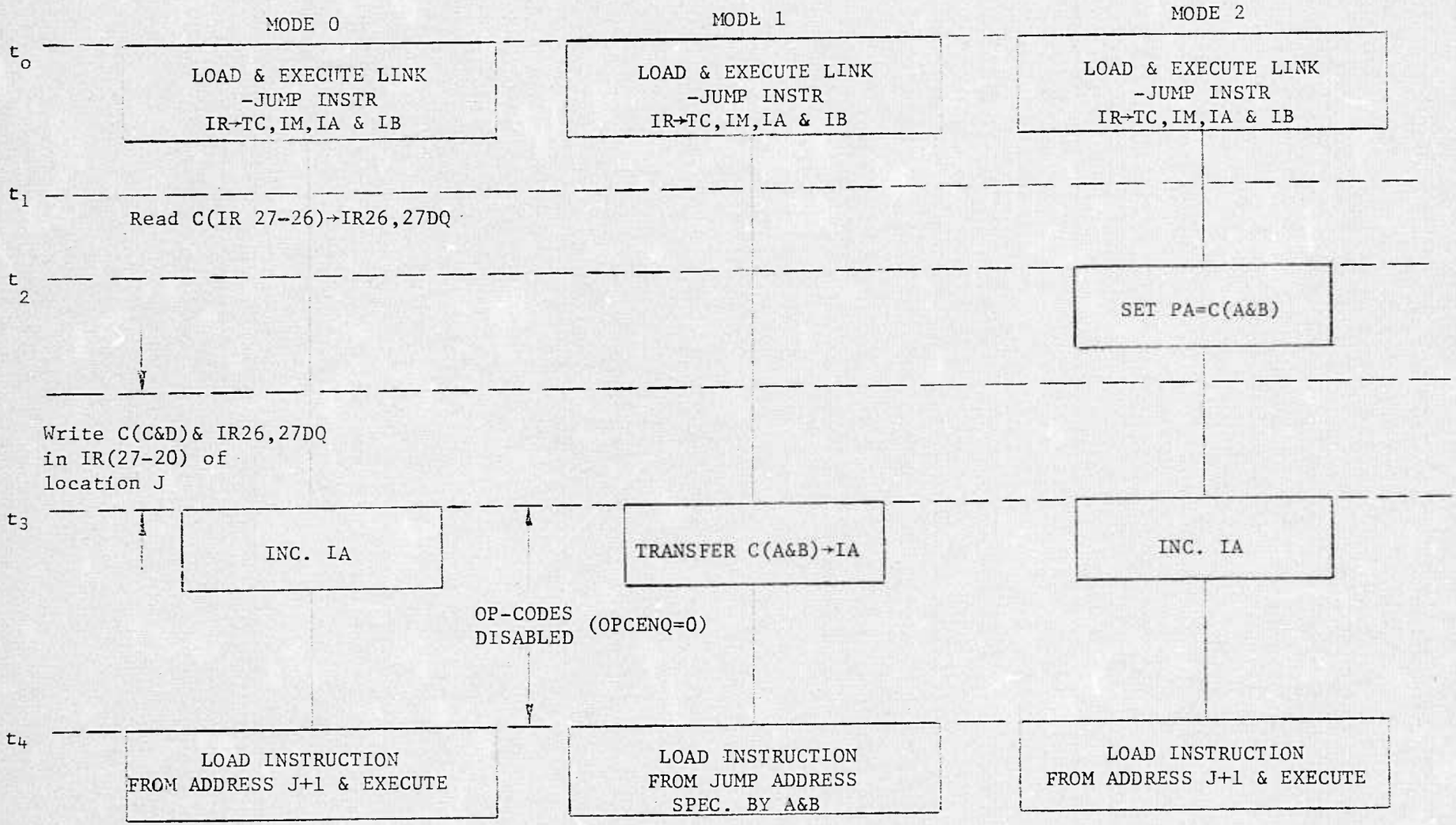
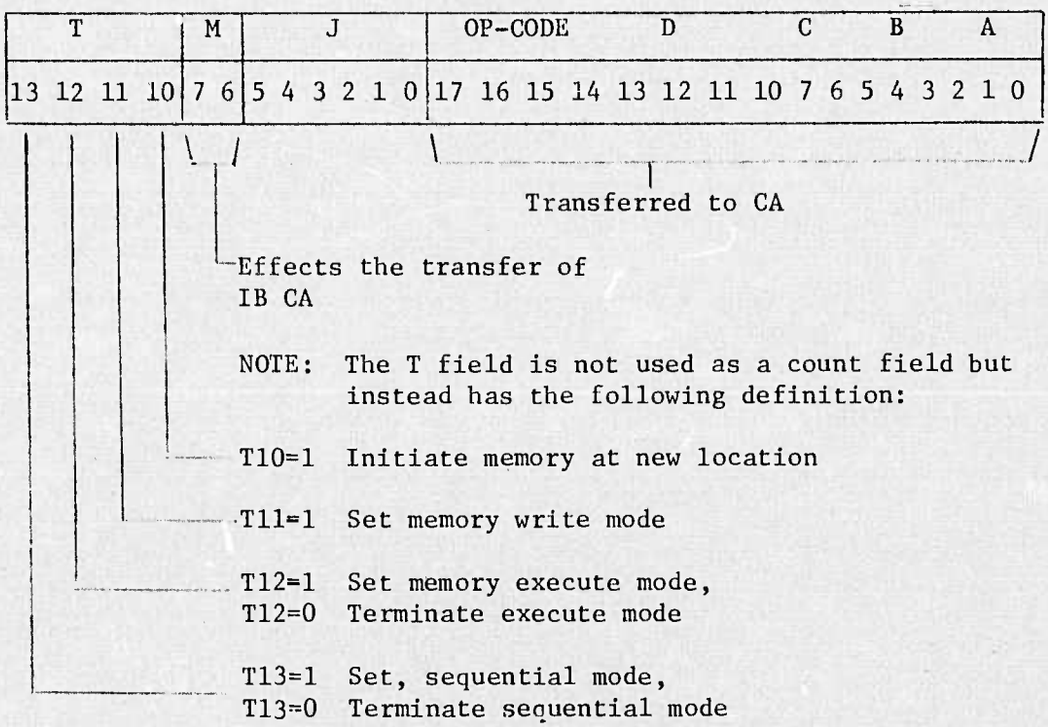


Figure 4-12. Link Jump Flow Diagram

MODE 3



NOTE:

For a standard system, 32K-words of 18-bit memory, 15-bits are sufficient to specify a unique address location. Bit 17 is not used in the standard system.

4-152. I/O INSTRUCTIONS

4-153. BASIC I/O INSTRUCTION TYPES. There are basically four types of I/O instructions: device status request, data transfer, instruction transfer and command execution. The four types are described below:

a. Device Status Request. The status request instruction causes the information on the device's controller status lines to be transferred to the Device Input Bus. The status of the device is then determined programmatically through a DI→E1 transfer and a test of the E1 register.

b. Data Transfer. There are two data transfer instructions: data-in and data-out. The data input instruction effects the transfer of the information on the peripheral device controller output data lines to the Device Input Bus. The data is then transferred to the MP programmatically through the DI→E1 transfer and a transfer of E1 to memory, data pad or a MP register. The data output command transfers the contents of the I1 register directly to the addressed device controller.

c. Instruction Transfer. The lower order 8-bits of the I/O instruction specifies that the content of I1 is to be interpreted as an instruction and transfers the contents of I1 to the addressed device. The device controller must provide buffer storage and decoding networks in order to utilize the contents of I1 as an instruction.

d. Command Execution. The command execution instruction provides the means for on-off type operations, mode set and clear, etc.

4-154. EXAMPLE OF I/O OPERATIONS. The disk instruction set includes two examples of each of the four basic instruction types as follows:

Command Code	Command	Instruction Type
0	Read disk data onto DI Bus	Data Transfer
0*	Write contents of I1 onto disk	Data Transfer
1	Read disk error status word onto DI Bus	Device Status Request
2	Execute Disk Command in I1	Instruction Transfer
3	End transfer	Command Execution
4	Reset head assembly to cylinder zero	Command Execution
5	Read disk position word onto DI bus	Device status Request
7	Select disk drive number from I1	Instruction Transfer

*[IBR(7)=1]

4-155. INSTRUCTION DESCRIPTION. I/O operations are effected through the I/O instruction: Op Code = 14, D-field = 7, C, B and A-fields variable. Bits (0-7) of the I/O instruction constitute a single address & variable command portion, which governs all transfers to and from peripheral devices. The address portion of the I/O instruction, bits (3-6), is unique to a particular I/O device. The command portion, bits (0-2), can have the same or different meanings among the various devices. The unique address ensures that only the selected device responds to signals from the MP.

4-156. Format: The I/O instruction is a "Load" type instruction with the following format:

33-30	27-26	25-20	17-14	13-11	10-6	5-3	2-0
T	M	J	OP	D	C	B	A
VAR	0 or 1	VAR	14	7	VAR	VAR	VAR

4-157. Commands. Bits (0-2) and bit (7) select the command. The interpretation of the command code for each I/O device is given in Section 4-160.

4-158. Device Addressing. Bits (3-6) address one of sixteen devices as follows:

Address	Device
0-3	User Stations 0 thru 3
4	Display (Status)
5-7	Open
10	Disk Storage Drive
11	Open
12	Analog I/O
13	Host (Full Duplex)
14	Timer
15	Host (Half Duplex)
16	DMA-1
17	Array Processor

4-159. Data Output Bit. Bit (7) is zero unless the instruction is a Data Output instruction where it is equal to one. (See for example Section 4-154). Bit 10 is always zero.

4-160. I/O Device Instructions

a. User Stations. There are 4 user station addresses 0-3 (B-field) used to address keyboards 0-3 respectively. The station controller executes the decoded command (A-field) as shown below:

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	0	Station number 0-3	0	Read keyboard entry onto DI Bus. If DI(10) =1, no data ready. (The keyboard has an eight bit code).
3	0	[]	14	7	0		1	Load A_2 into X DAC, I_2 into Y DAC and, after allowing a DAC settling time of 8 µs, write display for 24 µs. If in slew mode, no settling time is required & write display of 2 µsec allowed programmatically.
4	0	[]	14	7	0		2	Erase Display (Display Status=1 for 750 ms)
3	0	[]	14	7	0		3	Set Display in Non-Store Mode (no storage results from display output)
3	0	[]	14	7	0		4	Clear Non-Store Mode (storage results from display output)
3	0	[]	14	7	0		5	Turn on Keyboard STATUS REQUEST indicator (red light) to indicate user requests Analog I/O Channel

.....Continued

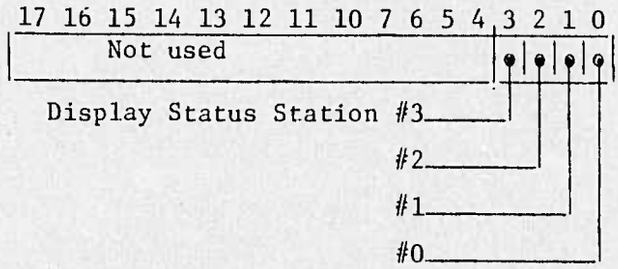
a. User Stations (continued)

T	M	J	OP	D	C	B	A	
3	0	[]	14	7	0	Station No. 0-3	6	Select Analog Channel. Clear red light and light STATUS LOCKOUT indicator (green light) to indicate user is connected to the Analog Channel.
3	0	[]	14	7	0		7	De-select Analog Channel, clear red and green lights.

b. Display (Status). The display commands are part of the station commands, but address 4 allows one to determine if a particular station is ready to accept an erase or write command. The status of all four displays is placed on the DI Bus. This can then be loaded into E1 and the appropriate bit tested.

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	0	4	0	Read display status word onto DI Bus
3	0	[]	14	7	0	4	1	A2→XDAC, I2→YDAC. In slew mode allow 2 μs per output.
3	0	[]	14	7	0	4	2	Enable display slew mode. Slew mode ends if no output for 32 μs.

The status word read onto DI by command code 0 has the format:



A "0" in one of the bits (0-3) indicates that the Display for that particular station is not busy. Conversely, a "1" indicates that the display is busy.

c. Disk Storage Drive. The disk address is 10. The disk controller executes the decoded command shown below.

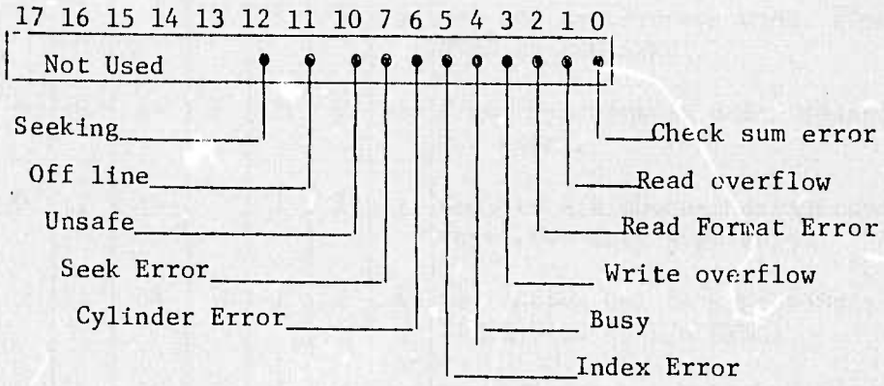
T	M	J	OP	D	C	B	A	
2	1	[]	14	7	1	0	0	Read disk data onto DI Bus. IØDRDY=0 (When DATA→DI, IØDRDY=1)
2	1	[]	14	7	3	0	0	Write contents of I1 onto disk. IØDRDY=1 (When I1→DATA, IØDRDY=0)
2	1	[]	14	7	1	0	1	Read disk status word onto DI; clear S(6) on DI→E1.

.....(continued)

c. Disk Storage Drive (continued)

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	1	0	2	Execute disk command in I1
4	0	[]	14	7	1	0	3	End transfer. If read, compare checksum. If write, store checksum.
4	0	[]	14	7	1	0	4	Reset head assembly to cylinder zero, reset status.
2	1	[]	14	7	1	0	5	Read position status word onto DI
2	1	[]	14	7	1	0	7	Select disk drive number from I1 (bits 10, 11,12)

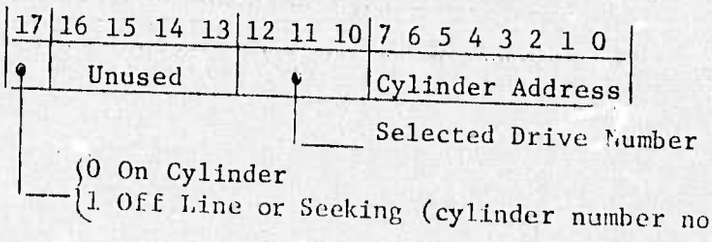
The disk status word read onto DI by command code 1 has the format:



The disk command word executed by command code 2 has the format:

	17	16	15	14	13	12	11	10	7	6	5	4	3	2	1	0
Position head to cylinder (seek)	0	0	0	0	0	0	0	0	Cylinder No.							
Select read head & read 18 bit words	0	0	0	0	0	0	0	0	0	0	0	Head No.				
Select write head & write 18 bit words	0	0	1	0	0	0	0	0	0	0	0	Head No.				
No-Op	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0
Select read head and read 16 bit words	0	1	0	1	0	0	0	0	0	0	0	Head No.				
Select write head and write 16 bit words	0	1	1	0	0	0	0	0	0	0	0	Head No.				
Read header	1	1	0	1	0	0	0	0	0	0	0	Head No.				
Write header	1	1	1	0	0	0	0	0	0	0	0	Head No.				

The read position status word read onto DI by command code 5 has the format:



d. Analog I/O. The Analog I/O address is 12. The Analog I/O Controller executes the decoded command as shown below:

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	1	2	1	Read A/D data onto DI Bus, Clear S(4) with DI→E1. Next channel determined by I1.
3	0	[]	14	7	1	2	3	Set A/D synchronous mode. (Connect TIMER SYNC to A/D SYNC).
3	0	[]	14	7	1	2	4	Clear synchronous mode. (Disconnect A/D & D/A sync).
3	0	[]	14	7	1	2	5	Convert A/D channel asynchronously; interrupt when data available.
3	0	[]	14	7	1	2	6	Set Analog Out to synchronous mode (Connect TIMER SYNC to D/A SYNC).
4	0	[]	14	7	1	2	7	Transfer contents of I2 to Analog DAC Buffer. Channel determined by I1.

Analog I/O allows the following channel possibilities:

I1 =
- 0 no channel
- 1 Channel #1
- 2 Channel #2
- 3 Channel #1 & 3

These are the "numbers" in I1 using bit positions 0 and 1.

e. Host Computer(Full Duplex)

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	1	3	0	Read data from host computer to DI
2	1	[]	14	7	3	3	0	Write data in I1 to host computer
2	1	[]	14	7	1	3	2	Execute I1 as a MP-32A/HOST instruction.

f. Timer

T	M	J	OP	D	C	B	A	
2	1	[]	14	7	1	4	0	Read timer Data→DI
3	0	[]	14	7	1	4	1	Set Synchronous Mode
3	0	[]	14	7	1	4	2	Clear Synchronous Mode
4	0	[]	14	7	1	4	3	Set timer from I1, interrupt when time=0

4-161. I/O DEVICE CONTROL. The operation of a peripheral device including timing is controlled through the address and command portions of the I/O instruction in combination with the device communication bit, the data ready bit, and the device service request bit. The basic I/O command causes the following action:

a. Transfer of the Instruction Buffer Register bits (0-6) to D(0-6).

b. Transfer of the Instruction Buffer Register bit (7) to IØDRDY.

c. Sets D(7) equal to one at the third clock time of the instruction.

The D register holds the device address and command codes until the next I/O instruction is initiated. I/O device address and function decoding is done from the contents of the D register.

4-162. Commands. The command portion of the I/O instruction, D(0-2), can have the same or different meanings among the various devices.

4-163. Addresses

a. Direct. The address portion of the I/O instruction, D(3-6), allows for direct addressing of 16 unique devices. Device addresses are given in Section 4-158. Additional peripheral devices can be accommodated by adding the appropriate controllers.

b. Indirect. The I/O capability can be easily expanded in terms of the allowable number of peripheral devices through the indirect address capability. Bits (0-7) of the I/O instruction provide the means for transferring the contents of the 16-bit I1 register to peripheral device controllers. The I/O instruction can further specify the contents of I1 as follows: (1) address, (2) single address instruction, and (3) data. Case (1) provides a 2^{16} device address capability. Case (2) provides both device address and function information. The number of allowable device addresses and functions depend on the bit allocation. Case (3) allows data to be transferred to the selected I/O device. Controller decoding is required for both case (1) and case (2). I/O devices serviced through the indirect address capability share a common service request bit and common address code.

4-164. Device Communication Bit. The primary function of the device communication bit D(7), is to indicate to a peripheral device that the contents of the D register represent a valid I/O instruction. The device communication bit is unconditionally set to one on the third clock time of the I/O instructions (mode 0 and mode 1). In the mode 0 instruction D(7) is reset to zero when TC=0. The mode 0 instruction allows the contents of the D register to remain valid for a variable time period ranging from 125 ns to .625 μs corresponding to a T-field between 3 and 7_8. In the mode 1 instruction, D(7) reamins on and is reset by the device controller to indicate command accept. The two instructions allow devices with widely different response times to be serviced efficiently and a single device to be serviced more efficiently when device response is a function of the operation to be performed.

4-165. Data Ready Bit. The data ready bit has the same meaning (i.e. data is ready to be transferred) in all data transfers between the MP and a peripheral device. In data output instructions, MP to a peripheral, bit (7) of the I/O instruction sets the data ready bit at TC=(T-1). The data must be in I1 prior to issuing the I/O command. The I/O device controller indicates data accept by clearing IØDRDY. When data is being transferred to the MP, the peripheral device controller sets the data ready bit to indicate data ready in controller buffer storage. The time in which the MPU must accept the data is determined by the characteristics of the peripheral device.IØDRDY is cleared with DI E1. Only the device that is currently addressed is allowed to control the Device Input lines and IØDRDY.

4-166. Service Request. Each peripheral device is assigned a dedicated service request bit. The service request bits are comprised of bits (0-7) of the S registers. The eight bits are arranged in a priority chain with bit 7 occupying the position of highest priority. (See Section 1-57 for bit assignment.) When a peripheral device requires service its controller sets its service request bit. The service request bits are tested by software and the appropriate macros called to process the requests. The instruction used to test for a particular request may be preceded by a test of S=0 . If S≠0, then either a scan can be made from left to right (7-0) or a particular bit position can be checked. I/O devices requiring service can set their assigned bit in the S register at any time whether they are addressed or not. The service request bit of a particular device is cleared when status is transferred from the DI bus to (or thru) the E1 Register by a program instruction, for I/O address >7.

V. MAINTENANCE

5-1. SCOPE

5-2. This chapter describes routine, special and preventive maintenance of the MP-32A Macroprocessor. The information contained in this chapter presupposes that the maintenance technician has been trained on, and is familiar with the equipment. The sections on routine maintenance describe the basic tools and procedures for maintaining the equipment. The sections on special maintenance supplement the instructions of the sections on routine maintenance by providing additional instructions pertaining to special maintenance situations. The sections on preventive maintenance describe specific maintenance tasks to be performed for the purpose of identifying and/or preventing a potential malfunction.

5-3. ROUTINE MAINTENANCE

5-4. TEST EQUIPMENT

5-5. Test equipment consists of a d.c.volt/ohm-meter (Simpson 260 or equivalent) and an oscilloscope with accessories. The oscilloscope recommended is a Tektronix 475 or equivalent with the following accessories:

Description	Quantity
Probe - 6' (P6075)	3
Folding View Hood	1
Scope-Mobile Cart	1
Probe Ground lead Adapter	3
Miniature Probe to #6-32 Adapter	3
Probe Pin Tip (0.025")	6
Probe Tip Flexible for 0.025 sq. pin	6

5-6. PRINTED CIRCUIT BOARD AND CONTACT CLEANING

5-7. Printed circuit boards should be cleaned at least once every 6 months.

CAUTION

Remove cards from assembly
before cleaning with pressurized air.

Dust removal from printed circuits is accomplished by spraying clear, oil-free, pressurized air (60 psi maximum) over boards.

5-8. LPS Instant Contact Cleaner has been approved for cleaning the contacts of printed circuit assemblies and baseplate connectors. It may be purchased locally in most cities as LPS, part number ICC-16 (16 oz. aerosol can or bulk containers). When printed circuit boards must be cleaned, hold the assembly with the contacts pointed down.

Thoroughly saturate contact area. While contacts are still wet, scrub with a soft natural bristle brush.

CAUTION

Under no circumstances should an eraser or other abrasives be used on gold plated contacts.

5-9. When baseplate connector contacts require cleaning, remove printed circuit board, clean, rewet contact area. Insert and withdraw several times into the connector. Wipe connector to determine if residue is present. Repeat this sequence until board contacts are free of residue.

CAUTION

Insertion of objects other than printed circuit boards may destroy contact surface or pressure characteristics.

5-10. TROUBLESHOOTING, REPAIR AND REPLACEMENT INSTRUCTIONS

5-11. General troubleshooting procedures should be followed whereby one first checks for loose plugs, connectors and IC's. If this does not eliminate the failure, the next step is to isolate the problem to a particular section of the unit. This can best be accomplished by executing the MP-32A Macroprocessor Diagnostic package and determining which routines fail. This will usually isolate the problem to one of the four major sections of the machine:

a. Control and Timing

b. Arithmetic

c. Memory

d. I/O.

The routine that failed may be placed in a loop mode and the error printout disabled. In this manner the actual failure can be pin-pointed to the individual gate or chip, which is at fault, with an oscilloscope. As an aid to the maintenance engineer, two sync points are provided on the main logic plane: IAEQL (B06-01-03) and CAREQL (B04-02-03). These test points are used to provide an external trigger to the scope at the time that the contents of the IA or CA registers is equal to the contents of the HALT register. In this manner the scope can be triggered on the actual instruction that is failing. If the problem is so basic in nature that the diagnostic routines won't run, a different procedure must be used as follows:

(1) Check that the address halt switch is in the center, or off, position.

(2) Check that the voltage indicator lights and the disk ready indicator, on the status panel are on.

(3) Press RESET -- Observe that all registers, with the exception of HALT, Instruction Register (control pad), and DSR, clear. The halted indicator should be on (STOPQ="1"). If no registers clear, check CLOKO*E (E15-02-02) and CLOKS*B (B09-01-02).

(4) See that both all ones and all zero's can be transferred to every register (including HALT, IR, PD & CD). If no registers (other than E) load, probable cause is I1, which transfers to almost every register. If a particular register won't load, check clock on the register chip. If CD won't load, check CDCAO* (A03-03-01). This signal should be active for about 70 ns with a repetition rate of 60 μs, indicating that MOS memory is being refreshed.

(5) See that CA, IA and PA increment from the control panel.

(6) Load the following instruction into address zero of control pad: 03 0 00 10 7 3 5 0. Load I1 with all ones -- press RUN (observe CA to be counting) and than HALT. Press RESET. Check the first 8 memory locations for all ones. Press RUN and then HALT. Check the same cells for all zeros. This checks all four memory modules -- address 0 and 1 are in module A, address 2 and 3 in module B etc. If a MOS storage card needs to be replaced, first power down the MP and then remove the card using the ejection tabs provided. Insert the new card making certain that the component side is to the left. Line up the connectors carefully and press card in firmly.

(7) Execute dead start routine -- press RESET, INITIATE, RUN. Observe that the operating system loads from the disk. If the system doesn't load, notice which instruction the program is spinning in and attempt to isolate the problem. If the instruction is a disk instruction, try powering down the disk and repeat. The dead start program may be checked by pressing RESET, INITIATE and then incrementing IA.

(8) If a chip needs to be replaced, power down the MP first and then remove the bad chip. Replace the I.C. with the same type (or a CHI approved equivalent) being careful to see that all of the legs are inserted into the socket without bending and that the identifying notch is pointing to the right (toward pin 1).

APPENDIX 2

AP-90 Reference Manual

PRELIMINARY

CHI SIGNAL SYSTEM

TMB8

TMB8

AP-90 REFERENCE MANUAL

CONTENTS:

1. INTRODUCTION
2. PHYSICAL DESCRIPTION
3. FUNCTIONAL DESCRIPTION
4. OPERATION
5. INPUT/OUTPUT CHARACTERISTICS
6. PROGRAMMING
7. MAINTENANCE

PRELIMINARY

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	10/75	pp.6-1 thru 6-8

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29204

© 1973
by Culler/Harrison, Inc.

TABLE OF CONTENTS

		Page
PREFACE		iv
1.	INTRODUCTION	1-1
	1.1 Configuration Selection	1-1
	1.2 System Overview	1-1
	1.3 Logical Units	1-3
	1.4 Programmable Components	1-5
	1.5 Execution Times	1-5
2.	PHYSICAL DESCRIPTION	2-2
	2.1 Maintenance Panel	2-2
	2.2 Transform Plane	2-2
	2.3 Multiply Extension Plane	2-3
	2.4 Data Memory (MOS)	2-3
	2.5 Power Supply	2-3
	2.6 Top Cover	2-3
3.	FUNCTIONAL DESCRIPTION	3-1
	3.1 Control Unit	3-1
	3.2 Data Pad Unit	3-2
	3.3 Adder Unit	3-3
	3.4 Accumulator Unit	3-3
	3.5 Data Memory Unit	3-4
	3.6 Multiplier Unit	3-6
	3.7 Table/Data ROM Memory Unit	3-6
	3.8 S-Pad Unit	3-7
	3.9 Shift Unit	3-7
4.	OPERATION	4-1
	4.1 Switches and Indicators	4-1
	4.2 Execution	4-3
5.	INPUT/OUTPUT CHARACTERISTICS	5-1
	5.1 General Features	5-1
	5.2 Functional Description	5-1
	5.3 Physical Description	5-5
6.	PROGRAMMING	
	6.1 Instruction Set	6-1
	6.2 AP-90 Cross-Assembler	6-1

Table of Contents (continued)

		Page
7.	MAINTENANCE	7-1
7.1	General	7-1
7.2	Accessability	7-1
7.3	Control/Maintenance Panel	7-1
7.4	Diagnostic Routines	7-1

TABLES:

1-1	Programmable Components	1-6
1-2	Fixed Point Program Execution Times	1-7
1-3	Floating Point Program Execution Times	1-7
1-4	Move & Convert Execution Times	1-8
6-1	General Description of Formats	6-4
6-2	Field Definitions	6-6
6-3	Field Selections	6-6
6-4	Typical AP-90 Short Form Instruction Mnemonics	6-8

FIGURES:

1-1	AP-90 Configurator	1-2
1-2	Programmable Components Block Diagram	1-4
2-1	AP-90 Assemblies	2-1
4-1	Control/Maintenance Panel	4-2
5-1	AP-90 I/O Ports Block Diagram	5-2

PREFACE

This manual is intended to provide a general description of the CHI AP-90 micro-programmable, fixed (16-bit/32-bit) and floating point, array and arithmetic processor. It provides information on the physical and functional characteristics, operation, input/output ports, programming and maintenance.

The reader is advised that the concepts and implementations involved in this device deviate significantly from that of current computer equipment INCLUDING the current versions of "micro" programmed processors.

The reader is URGED to allocate sufficient time for the reading of this manual to be able to assimilate the UNIQUE advantages of this device and the POWER of its organization and instruction set.

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No. 29204
© 1974
by Culler/Harrison, Inc.

The logical design principles embodied in the AP-90 are protected under U.S. Patent No. 3,771,141.

SECTION 1
INTRODUCTION

1.1 CONFIGURATION SELECTION

The AP-90 must be viewed in a considerably different light than other array or arithmetic processors.

The AP-90 should be considered in much the same way as we now consider general purpose computer systems. Basically, when a user selects a computer system he selects a "configuration" of hardware that best fits his own needs. Then, he selects and/or implements software which best solves the computational problems involved in his application.

It is in EXACTLY this same fashion that the AP-90 must be viewed. The AP-90 must be optimally "configured" hardware wise to the user's needs and software wise to the user's problems.

In a general purpose computer system the user commonly "configures" the system in terms of CPU, amount of core, number and types of I/O devices, etc. In "configuring" the AP-90 , we must consider the amount and type of program (control) storage, amount of data memory, amount and contents of table memory, I/O requirements, etc. Figure 1-1 exhibits the various possible selections of memories.

The AP-90 software must be selected and/or written to suit the problem. Basic operations are generally selected from a wide array of pre-programmed algorithms available from CHI. These are normally contained in read-only-memory (PS-ROM) as fixed programs (firmware).

Often the user finds his application contains basic algorithms which are used repeatedly and involve extensive computational efforts. Two approaches to an implementation of such programming exists:

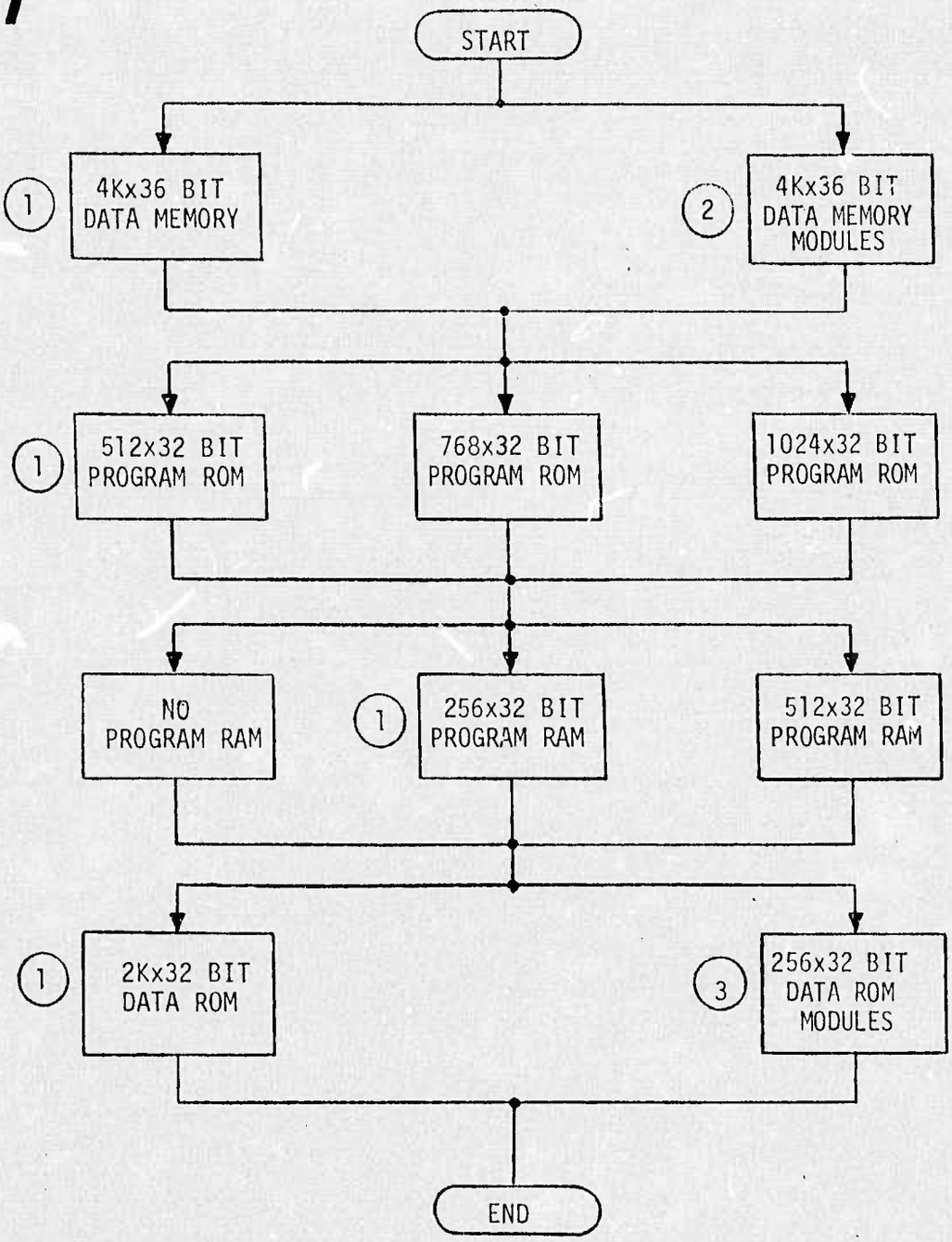
a. Have CHI program and store the algorithms in control memory (PS-ROM)

b. Program and store the algorithms in the host computer and transfer this code to the AP-90 read/write (PS-RAM) control store for execution when required.

1.2 SYSTEM OVERVIEW

The AP-90 is a high speed (167 ns cycle time), highly parallel (up to seven simultaneous operations), special purpose processor embodying many features of a general purpose computer.

The AP-90 is easily interfaced to any currently produced computer and can GREATLY ENHANCE the capabilities of the smallest mini to the largest maxi computer. The general purpose "host" computer to which the AP-90 is interfaced provides systems level control of data transfers between the AP and the host.



Notes:

(1) Basic configuration

(2) Up to 32Kx36 bits

(3) Up to 4Kx32 bits

Figure 1-1 AP-90 Configurator

The AP generally provides the detailed "bookkeeping" involved and performs the transfers on a "cycle-stealing" basis. The host generally is responsible for scheduling of operations and or initiation of complete processes by the AP on the transferred data.

The AP-90 's ability to perform as many as seven (7) operations in PARALLEL adds to the effectiveness of the machine, particularly in array operations where indexing can be "hidden" in arithmetic tasks.

TO FURTHER ENHANCE the use of the AP in signal processing environments such as seismic data analysis, speech research, and sonar analysis, the AP-90 has been designed to allow the host computer to request operations in two basic modes:

a. ARITHMETIC MODE---In this mode the AP is used as a peripheral arithmetic unit by the host computer. Execution is tightly controlled by the host.

b. ARRAY PROCESSING MODE---In this mode the host computer fills the data memory of the AP with information to be processed and turns control over to the AP-90. When such processing is completed, the AP interrupts the host and returns control.

EQUALLY IMPORTANT is the concept of PROGRAMMABILITY. Programmability is available at three levels:

a. Pre-programmed algorithms (such as FFT) furnished as standard ROM's (read-only-memory) with the system.

b. Special purpose algorithms which the user requests CHI to program into ROM's to be added to the system as supplemental operations.

c. Any type of algorithm which the USER may program and assemble through the Cross-Assembler. This assembler, written in FORTRAN, accepts coding in a readily understandable mnemonic language and translates it into machine instructions.

An operation, stored in ROM, is available which will transfer machine (AP-90) code from the host to the AP read/write control store and cause execution to commence as directed by the host.

1.3 LOGICAL UNITS

The AP-90 is composed of nine distinct sections termed "LOGICAL UNITS", as shown in Figure 1-2 and described in detail in Section 3. With the exception of the main data storage memory (MD) and the multiplier (MP), each of the other elements of the system can complete a programmed operation in a single clock time (167 ns). Many of these units may be operated in parallel thereby providing an APPARENT (when thought of in terms of conventional computers) instruction time as short as 25 nanoseconds.

The main data memory (MD) has a cycle time of 500 ns (3 clocks) and is equipped with a feature which "locks out" the user if he tries to access the memory at rates in excess of this speed. This feature, however, does not prohibit him from doing other operations once he has "initiated" a read or write.

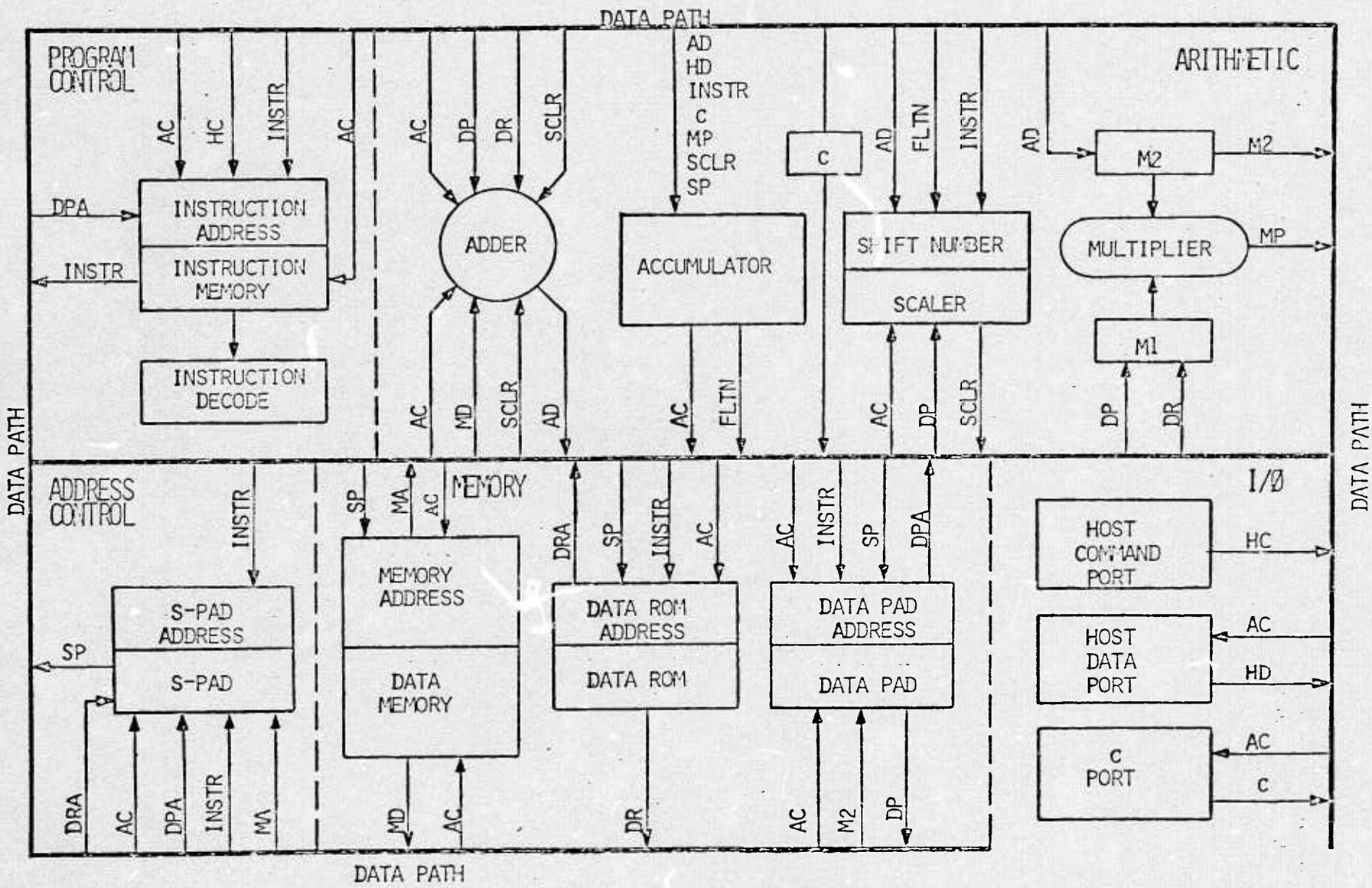


Figure 1-2 AP-90 BLOCK DIAGRAM (Programmable Components)

The multiplier (MP) can complete a 16 bit by 16 bit multiply and develop a 32 bit (or a 16 bit rounded) product in 2 clock times. After initiating a multiply, the extra clock time is available to the user for indexing or other operations. In the FLOATING POINT mode the multiplier develops a 24 bit rounded product for two 24-bit numbers in three clock times. Note that a full product is developed including rounding, not a truncated multiply.

PROGRAMS ARE EXECUTED from a bi-polar "program-source" storage which may be either ROM (PS-ROM) or read/write (PS-RAM) memory. The contents of the addressed program-source location are directed to a CONTROL BUFFER (CB) at the beginning of a machine cycle and the program source address (PSA) register is incremented by one. Command decoding is performed from the CONTROL BUFFER thus allowing overlap of accessing and decoding of instructions.

THE CULLER/HARRISON SUPPLIED ROM PROGRAMS include a suite of operations designed for signal processing of 16 bit fixed-point data. ALSO available are FLOATING POINT (8 bit scale, 24-bit mantissa) elementary functions and users host computer/AP-90 control, communications and data transfer operations. Although not programmed at this date, FLOATING POINT FFT operations have been examined and an estimate of less than 20 milliseconds for a 1024 real data point transform is assured.

ROM PROGRAM STORAGE is available in increments of 256 words. RAM program storage is available in increments of 256 words up to a maximum of 512 words. Combined ROM and RAM must not exceed 4096 words of memory. Both memories are organized in "pages" of 16 words. A minimum configuration consists of zero words of RAM and 512 words of ROM.

THE MAIN DATA MEMORY (MD) is available in increments of 4K words. The maximum configuration is 32K words. The memory is actually 36 bits wide but the additional 4 bits are transparent to the user. These additional bits are used with 16-bit FFT operations to insure the maintenance of MAXIMUM SIGNIFICANCE throughout the operation. Hardware detection of overflows during a FFT "pass" provides proper "scaling" of data for the next pass without the necessity of software testing for overflow and without requiring a scaling pass.

TABLE/DATA ROM (DR) contains often used constants such as the fixed-point roots of unity (stored as pairs), and initial values for elementary function approximations and is available in increments of 256 words from 2K to a maximum of 4K words.

1.4 PROGRAMMABLE COMPONENTS

Table 1-1 is a tabulation of the programmable components of the AP-90. The UNIT column indicates the mnemonic used in the AP-90 Assembly language to reference these elements. The type of IC (integrated circuit) device and the access/cycle times are indicated where applicable.

1.5 EXECUTION TIMES

Execution times for fixed point operations are given in Table 1-2. FFT uses the Cooley-Tucky (decimation in time) algorithm with a radix of 4. The point/pass time of 2μs leads to an execution time (no I/O) of 5.3 ms for the transform of 1024 real data point and 9.2 ms to 1024 complex data points. Execution times for floating point operations are given in Table 1-3 and for convert operations in Table 1-4.

Table 1-1 Programmable Components

Unit	Typical Usage	Type	Access/Cycle Time
AC	Accumulator for arithmetic operations	Bipolar-Schottky	--
AD	Fast adder	Bipolar-Schottky	--
C	Temporary storage and communication	Bipolar-Schottky	--
CB	Command buffer for instruction decoding and execution	Bipolar-Schottky	--
DP	Scratch pad	Bipolar	35ns
DR	Tables for transforms, filters, and fixed processes	Bipolar	60ns
EXITS	Settable exits for operations and subroutines	Bipolar	35ns
MD	Main memory, signal and transform buffer	MOS	500/500ns
MP	Multiplier	Bipolar	--
M1,M2	Input registers for Multiplier	Bipolar-Schottky	--
PS-RAM	Variable control programs and special operations	Bipolar	50ns
PS-ROM	Fixed micro programs and standard operations	Bipolar	60ns
SN,SCLR	N. position scaler	Bipolar-Schottky	--
SP	Parameters and index registers for subroutines	Bipolar	35ns
DPA	Address register for referencing data (scratch) pad registers		
SPA SPL	Registers associated with referencing of SP (subroutine parameters) registers.		
MA	Main memory address register		
DRA	Table/Data ROM address register		
PSA	Program counter (Program Source Address) register used to control sequence of control words directed to the command buffer.		

Table 1-2 Fixed Point Program Execution Times

Operation	Register to Register Execution Clock Times
Addition (real or complex)	1
Subtraction (real or complex)	1
Multiplication: real	2
complex	8
Convolution	2 x K† clock times/point
Fast Fourier Transform (Radix 4)	12 clock times/point/pass
Inverse FFT	12 clock times/point/pass

† K = the number of kernel points

Table 1-3 Floating Point Program Execution Times

Operation	AC to AC Execution Clock Times*
Floating Point Addition & Subtraction	3
Floating Point Multiplication	4
Floating Point Division	26
Floating Point Square	4
Floating Point Absolute Value	1
Floating Point Negate	1
Floating Poing Sgn	8
Floating Point Inverse	22
Floating Point Square Root,Even Scale	29
Floating Point Square Root,Odd Scale	35
Floating Point Log	47
Floating Point Exponential, $x \leq \ell n\ 2$	39
Floating Point Exponential, $x \geq \ell n\ 2$	42
Floating Point Sine	72
Floating Point Cosine	74
Floating Point Atan, $x<1$	75
Floating Point Atan, $x \geq 1$	107

*Standard linkage and interrupt is 3 clock times

Table 1-4 Move & Convert Execution Times

Operation	Execution Clock Times*
Move HD to MD	3
Move HD to DP	1
Move HD to DPL	1
Move HD to AC	1
Move MD to HD	3
Move DP to HD	1
Move HD to MD and Decimate	3
Convert Floating Point in AC to Integer	8
Convert Integer in AC to Floating Point	3

*or channel time, whichever is larger

NOTE:

Since all data is moved through the AP- 90's accumulator (AC), the move times, except for move HD to AC, are streaming times and not single word transfers.

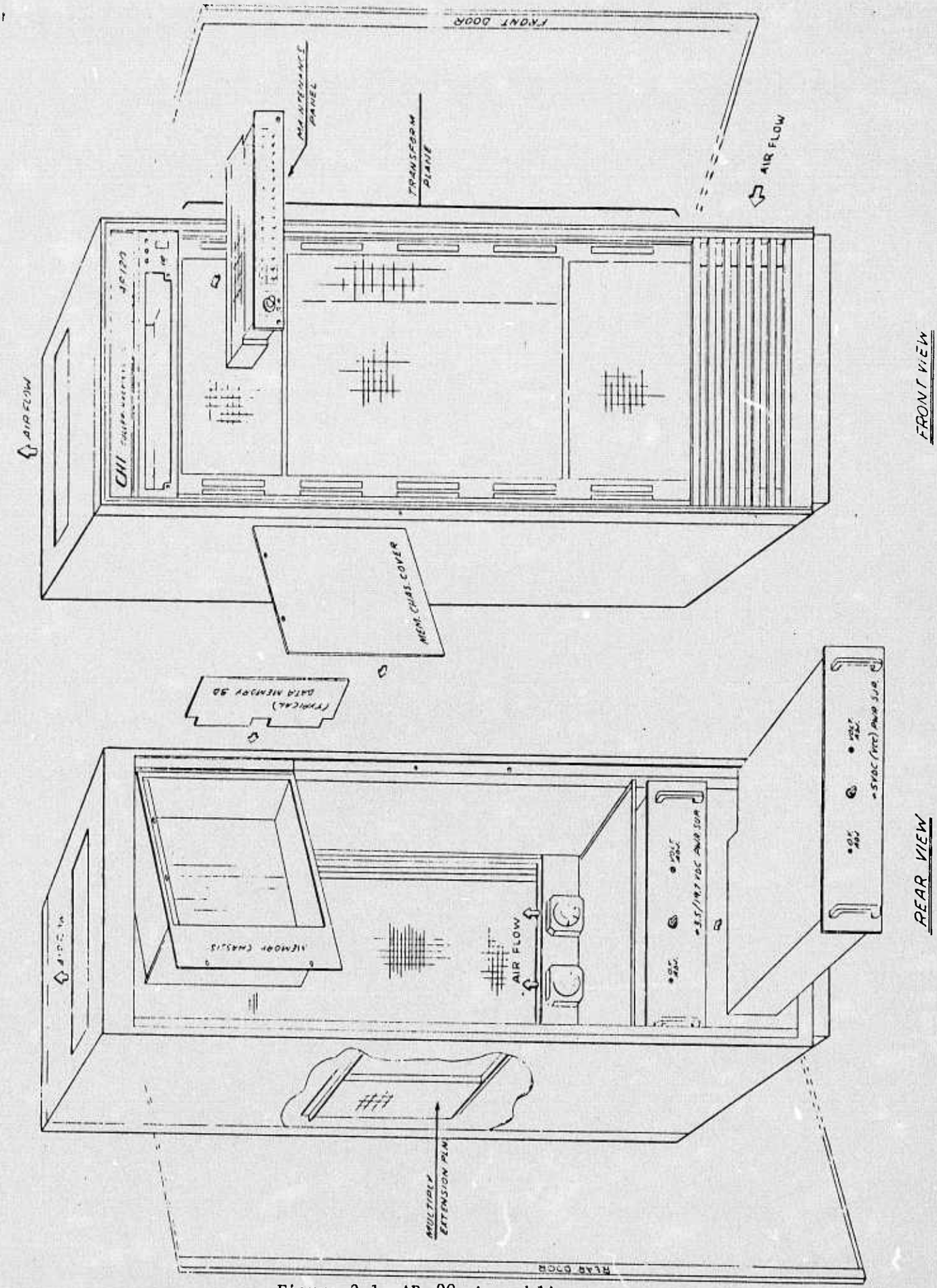


Figure 2-1 AP-90 Assemblies

SECTION 2
PHYSICAL DESCRIPTION

The AP-90 is comprised of six major assemblies, mounted in a custom-made electronic equipment cabinet measuring 61.5 inches high, 24.0 inches wide and 28.5 inches deep. The location of each assembly within the cabinet is shown in Figure 2-1.

The assemblies are interconnected either through standard-type connectors and cables or directly wired to form an integrated system.

2.1 MAINTENANCE PANEL

The maintenance panel incorporates a system power switch, +3.5VDC, +5VDC and +19.7VDC on-off indicating lights, and various controls and indicators which provide an operator the capability of data entry, program debugging, fault isolation and maintenance. These controls are more fully described in Section 4.

2.2 TRANSFORM PLANE

The transform plane is the major wire-wrap plane in the array processor and houses all of the integrated circuits used in the processor logic implementation except for those used in the multiply extension plane.

The plane is comprised of a dual plane power distribution system, an array of sockets, and wire wrap interconnections. The dual plane configuration consists of two low resistivity planes separated by a thin sheet of dielectric material with a relative permittivity of approximately 7. One plane is used to distribute the 5VDC supply voltage to the individual sockets and the second plane serves as a ground plane. The composite, dual conducting planes separated by a dielectric, provides distributed capacitance for VCC decoupling. The wire wrap interconnections in conjunction with the power and ground planes form a high frequency signal transmission system. Logic signal transmission between individual circuits on the plane is via short lengths of 30 gauge solid wire in close proximity with a ground plane. The power connection to sockets is made via clips that are soldered to the appropriate plane.

Individual circuit elements are plugged directly into sockets. This technique achieves elimination of possible circuit damage due to soldering when installing or removing parts, accessability for maintenance, and low cost complement of spares. The plane incorporates 1116 16pin sockets and 84 24-pin sockets. Fifteen 88-pin connectors may be mounted on the transform plane for routing signals to and from the plane.

The plane is partitioned into bays to facilitate locating individual circuit elements. An x-y coordinate system is used to specify the location of sockets within a bay. The equipment is labeled as to bay, column and row.

2.3 MULTIPLY EXTENSION PLANE

The multiply extension plane houses the circuits which extend the multiplier from 16 bits to 24 bits. The construction employed in this plane is identical to that of the transform plane. The plane incorporates 51 24-pin sockets arranged in 17 rows of three sockets each. Two 88-pin connectors are mounted on the plane for routing signals to and from the transform plane.

2.4 DATA MEMORY (MOS)

The data memory assembly houses up to 25 plug-in memory cards. In the standard configuration five slots are filled. The assembly incorporates fifty 80-pin connectors. Each of the memory assembly cards requires two connectors. Signals between the memory cards and the main logic plane are first routed, via back-plane wiring, between the memory assembly connectors and two 88-pin connectors attached to the side of the memory chassis, and then by cables between these connectors and the connectors on the main logic plane. The power supply outputs are routed to the connectors by direct wiring. Inter-connector wiring is accomplished with wire wrap techniques. One 117 cfm fan is mounted on the bottom of the chassis to cool the memory storage cards.

2.5 POWER SUPPLY

This assembly incorporates two power supplies:

a. A 5VDC supply to power the logic circuits of the memory cards, transform and multiply extension planes. Two dual pairs of 12 gauge wires bring the +5V output and its return to a pair of +5V and common buses Each +5V bus is connected to opposite sides of the +5V power plane via 4-12 gauge wires. The common buses are connected to the ground plane in an identical fashion. The +5VDC and its return are distributed to individual sockets via the power and ground planes. The return leads of all DC supplies (common) are tied to the cabinet (chassis ground).

b. A 20VDC supply, modified to output +19.7VDC and +23.2VDC, to power the MOS IC's of the Data Memory. The supply outputs and common lead are wired to terminals o the memory chassis.

The power supply assembly also includes two 117 cfm fans mounted as shown in Figure 2-1 and used to cool the transform plane.

2.6 TOP COVER

The top cover houses a 265 cfm fan which is used to cool the components on the memory chassis and exhaust air from the cabinet.

SECTION 3

FUNCTIONAL DESCRIPTION

The hardware of the AP-90 is composed of three distinct types of elements:

a. Memory Elements
b. Logical and Control Elements
c. Arithmetic Elements.

From these basic elements FUNCTIONAL UNITS have been designed. Each of these FUNCTIONAL UNITS is INDEPENDENT and can perform the programmed operations for which it was designed independently of and, in most cases, in PARALLEL with the other functional units.

The FUNCTIONAL UNITS (some with characteristics which may be defined by the user at the time of order) are described in the following sections:

3.1 CONTROL UNIT

The Control Unit consists of:

a. PROGRAM SOURCE (control) MEMORY (PS-ROM/RAM)
b. PROGRAM SOURCE ADDRESS REGISTER (PSA)
c. CONTROL BUFFER (CB) with decoding logic
d. PROGRAM SOURCE EXIT REGISTERS (EXITS).

The status and operations of the AP-90 are controlled by the bit configuration of a 32-bit micro-program instruction word which resides, initially, in PS-ROM or PS-RAM. The control word for the next instruction to be performed is selected by the address in the PSA register. At the initiation of the next clock cycle, this word is transferred to the CB and PSA is incremented by one unless the current instruction causes the PSA to "branch" to another location within the present page of PS-ROM/RAM. Control memory access and instruction decoding are overlapped so that machine operations may be executed at an 6 mega-hertz rate.

As mentioned above, instructions generally are accessed serially from control memory by the incrementing of PSA. However, if the current instruction involves a "TEST" condition and that condition is satisfied, the address for the next instruction is obtained by combining the most significant eight bits of the PSA register with the four-bit jump field of the test instruction. A special class of test instruction (especially for I/O) has been implemented in which NO jump occurs until the test condition is satisfied.

In order to provide ABSOLUTE control of the AP-90 by the host computer, interface designs normally provide a direct path from the host computer to the PSA (program source address) register. A LOAD PSA command from the host computer overrides the current address in PSA and causes the AP to begin processing at the loaded address.

The "linking" of operations programmed in ROM requires the use of some type of dynamic (alterable) storage external to the "hard-wired" ROM.

This capability is provided through sixteen 12-bit word registers termed EXITS. Each ROM subroutine is specified to have a specific return EXIT. To link ROM subroutines together, a control program running in RAM, may load the applicable EXIT with an address in RAM or the address of the next ROM subroutine to occur in the particular processing sequence. With this capability the user has full access to all preprogrammed AP operations and can program significant algorithms in a minimum number of words of RAM.

The CONTROL UNIT thus provides the program storage area, instruction sequencing logic and the instruction (32-bit control word) decoding logic. Specific operations which are set up by the decoding logic are discussed in Section 6.

3.2 DATA PAD UNIT

This unit consists of thirty-two (32) 32-bit "addressable", high speed (35 ns) scratch pad registers. Access to data in this area is controlled by the Data Pad Address (DPA) register. Logic associated with the DPA register allow the user the following manipulations of the data pad addresses:

a. Set DPA to a value specified by the current instruction or by the current contents of ACR
b. Increment DPA by one.
c. Decrement DPA by one.
d. Decrement DPA by eight.
e. Load DPA with data contained in a specified SP (subroutine parameters) register.
f. Add a value from a specified SP register to the value of DPA and return this sum to DPA.
g. Swap the value in DPA with the value in a specified SP register.
h. Add a value from a specified SP register to the value of DPA, increment the resultant sum by one and return this result to DPA.

(Note: Arithmetic operations on DPA are modulo 32 with no carry being indicated. All of the indicated operations result in a VALID DPA during the same cycle within which the address modification is specified.)

The decoding section contains logic to provide certain tests of the VALUE contained in the DPA register.

For convenience in 16-bit data manipulation, logic is provided to access or store either the low order (RIGHT) or high order (LEFT) sixteen bits of the addressed data pad register without disturbing the other sixteen bits.

Note, however, that a full 32-bit path is also provided to most of the other logical units of the system.

Data paths, controlled by the instruction decode logic, between data pad (DP) registers and other logical units of the AP-90 are:

a. Accumulator (AC) to data pad (DP_{DPA})
b. Data pad (DP_{DPA}) to adder (AD)
c. Data pad (DP_{DPA}) to multiplier input register #1 (M1)
d. Multiplier #2 (M2) into Data Pad Right
e. Instruction (CB) to Data Pad Right

Although the number of paths and direction of transfer may seem limited, an examination of detailed arithmetic operations will indicate these paths to be a majority of those which could possibly be needed.

3.3 ADDER UNIT

This high speed (Schottky TTL) adder is used primarily to combine data from DP registers and the accumulator (AC). Data from DP registers or AC may also be combined with data from the data memory output register (MD). The complement (one's) of data from DP registers may be added to data from AC or to data in MD.

Additionally, the following quantities or data may be combined with the accumulator.

a. Fixed point zero.
b. Floating point zero.
c. Floating point one.
d. Floating point negative one.
e. Output of shift network.
f. Table/Data ROM data.
g. The accumulator with itself.

The speed of the adder is such that a full 32-bit add can be accomplished and the result be true in a single clock (167 ns) time.

In SOME operations (specified in the instruction set), the adder (AD) becomes a PAIR of adders. In these operations each add causes two sums to be developed about the LEFT and RIGHT boundaries indicated above. In floating point operations of this type the adders are respectively 8 bits for the LEFT and 24 bits for the RIGHT. In fixed point operations of this type two 16-bit sums are developed. Consequently there are operations which really require four input terms to develop the two sums. In some of these instructions, data paths have been implemented which facilitate the summation of "crossed" terms. These latter features are particularly useful in 16-bit complex operations associated with the FFT.

Because the output of an add operation can be programmatically selected as left or right, even if "nonsense" sums develop as the result of an add they can be ignored.

3.4 ACCUMULATOR UNIT

The Accumulator Unit is comprised of two registers. The first, the central register of the AP-90 is the accumulator (AC). In addition to holding the results from an adder operation, the accumulator provides a maximum number of communication paths with other LOGICAL UNITS of the system. For I/O data the AC is the communication point for 32-bit transters to the outside world through the HD (host data) port.

To reiterate, the accumulator has two configurations when referencing the RIGHT(ACR) side and/or the LEFT side (ACL) of the AC. Accumulator Left (ACL) will be either the high order 8 or 16 bits depending on whether the floating point condition is set. ACR will be either the low order 24 or 16 bits.

Transfers which can be made to and from the accumulator are:

a. Multiplier product (MP) 16 or 24 bits to ACR.
b. Adder to accumulator (check the instruction format section for all combinations).
c. Host computer data lines to and from AC.
d. C Register to and from AC.
e. Shift network (SCLR) to ACR.

(Note: The results of an adder operation may be stored within the accumulator within the same clock time as the adder operation. However, since the adder output is stable until another adder operation is specified, the adder output may be transferred to AC on any subsequent instruction prior to specifying another adder operation.)

A significant amount of additional logic is associated with the accumulator. Data contained in ACR may be converted to its 2's complement from sign-magnitude format or vice versa. Additional operations include the rounding of floating point numbers and the lfoating of fixed point quantities. This latter operation serves to "normalize" floating point values also.

As an aid in floating point additions and subtractions, an instruction exists to difference the scales of two numbers and transfer the result to the scale number (SN) register. SN will then contain a value indicating the amount and direction by which shifting must occur to justify the two floating point numbers. Associated logic sets up the gating for the proper floating point quantity to be shifted.

The second register in the Accumulator Unit is the C-Register. The C-Register is 32 bits wide, but many operations with it are 16 bit operations. The C-Register provides a 16 bit Iϕ path to and from the outside world. Transfers to and from the C-Register are:

a. Multiplier product (32 least significant bits) to C
b. Accumulator (32 bits) to and from C
c. Host lines (16 bits) to and from C Left
d. C Left to C Right, to build up a 32 bit number from the Host
e. Program Source (PS) to C for diagnostic purposes

3.5 DATA MEMORY UNIT

The main memory of the AP is a unit which operates entirely independent of other logical units of the system once a memory INITIATE operation has occurred.

The memory unit contains a memory input and a memory output register. Both registers are referred to with the mnemonic MD. The decoding logic automatically selects the proper register depending on whether an operation logically places data into or expects data from memory.

The memory address (MA) register points to the location in memory from which data is to be read or into which data is to be written. The MA register, through logic associated with the S-PAD unit, allows significant modifications

of the memory address under program control. These operations are:

a. Transfer the value of a selected SP register to MA as the address for the memory operation programmed.
b. As in (a) but also increment the value by one.
c. As in (a) but also decrement the value by one.
d. Increment the MA register by one.
e. As in (a) but also increment the value by eight.
f. Add the value from a selected SP register to the value in MA and place this sum in MA.

In referencing memory for a read or write operation, the selected operation is "INITIATED" by any change of the memory address register--UNLESS--the no memory initiate option is selected at the time MA is changed. When a write is initiated, the data to be written into memory is transferred (in that one instruction time) to the proper MD register. Three clocks (500 ns) later this data in MD has been written into main memory. This organization allows the programmer to use the additional two clock times to perform any operation available in the AP except a memory initiate operation. CONVERSELY, once a memory read has been initiated, the desired data referenced by MA IS NOT correct until the third clock time after initiation of the read. Again there are two intervening clock times available to the programmer for other operations. To optimize the operations of the AP-90 it is necessary for the programmer to "look ahead" and initiate reads prior to the actual time data in MD is to be used in an operation.

The system provides a "memory lock-out" which will not allow a memory initiate instruction to be executed until the completion of a memory cycle initiated by a prior instruction. This function only serves to insure that erroneous writes and reads (OF THE MEMORY ITSELF) do not occur. However, it does prevent execution of write instructions too rapidly for the system to handle. This feature DOES NOT insure that the programmer cannot erroneously combine data from MD with other data in the system. IT IS MANDATORY that the programmer continuously be aware of the timing of validity of data READ from memory.

Once thethree clock times have elapsed as required by the memory, the data in MD is static and remains correct until changed by an AP-90 operation which initiates another memory cycle.

With the exception of systems which are equipped with direct memory access channels (DMA), all transfers into and from main memory must be via the accumulator.

3.6 MULTIPLIER UNIT

The multiplier is a set of logic which is continuously forming the product of the contents of the two multiplier input registers (M1 and M2). The 32-bit product of two 16-bit numbers is true in two clock times and may be transferred two clock times after M1 and M2 are loaded. The 48-bit product of two 24-bit numbers requires an additional clock time. Only the rounded 24-bit product is available from a 24 x 24-bit operation.

The 16-bit rounded product OR the full 32-bit product is available from a 16 x 16-bit operation.
The rounded 16-bit product or the rounded 24-bit product output (MP) of the multiplier is available to the accumulator.
The unrounded 32-bit product is available to the C register.
Input to M1 may be from the scratch pad (DP) or from the table/data ROM (DR).
Input to M2 comes only from the adder.

3.7 TABLE'DATA ROM MEMORY UNIT

In signal processing environments, large tables of values are constantly being referenced. The data ROM (DR) facilitates this by its organization and data content. Data may be read from any selected location by setting a specific value in the data ROM address (DRA) register. Once DRA has been set to an

address, the addressed data is available one clock later and will remain available until DRA is again changed.

For tables of length 128 or 256 entires, special reference may be made indirectly through the contents of the accumulator. The high order 7 or 8 bits of AC specify the displacement from a base address in the instructions which provide for this type of reference.

The following constants and tables are contained in the first 2K words of the "standard" system data ROM.

a. Fixed point complex roots of unity
b. Floating point approximations of the following functions:

   Inverse
   Square Root
   Log
   Sine
   Atan
   Exp

3.8 S-PAD UNIT

This unit contains sixteen 16-bit, directly addressable registers. The output of these registers pass through a special adder associated with this unit. The output of the adder is directed to a S-PAD LATCH (SPL) register.

The output of SPL can be an input to a selected S-PAD register. This output can also be transferred to the memory address (MA), data ROM address (DRA) registers, or data pad address (DPA).

The adder, in addition to the contents of the selected S-PAD register, may have as the other input the following:

a. Table/data ROM address (DRA) register.
b. Main memory address (MA) register.
c. Plus one.
d. Negative one.
e. Plus eight.

The control section contains logic to test the value of the most previously selected register to be =0 or ≠0.

These tests and operations make the S-PAD registers obvious targets for use as index and count registers.

3.9 SHIFT UNIT

This element of the system consists of an 8-bit shift number (SN) register and a shifting network (SCLR). This logic can shift the ACR or an addressed DPR (16 or 24 bits as appropriate). Shifts can occur left or right and may be of any length up to the full length of the register. Shifts of any length can be performed in time to present the true value of SCLR to the adder or the accumulator in sufficient time to perform an operation on the data at the next clock time.

SECTION 4
OPERATION

4.1 SWITCHES AND INDICATORS

The AP-90 Control/Maintenance Panel (Figure 4-1) contains all of the switches and indicators needed for operating the system.

4.1.1 Power Switch

The Power Switch is a two-position switch that controls the 115VAC line voltage to the processor power supplies.

In the OFF position, the ac line voltage is removed from the input of the power supply.

In the ON position, the ac line voltage is applied to the power supplies. Both the processor and control/maintenance panel are fully operational and the three power supply status indicators are lit.

4.1.2 STEP, RUN/STOP, RESET Switches and HALT Indicator

Pressing the STEP switch executes one instruction at a time from Program-Source (ROM or RAM). The HALT indicator will appear to remain lit.

The RUN/STOP switch is an alternate-action switch that switches the processor alternately to the RUN and STOP modes. In the STOP mode, the HALT indicator is lit.

Pressing the RESET switch places the processor in the halt state, clears AC and all address registers, and resets internal flip-flops.

4.1.3 REGISTER SELECT Switch, L/R Switch and LEFT Indicator

The REGISTER SELECT switch is a 6-position switch that allows selection of each of the following registers:

AC	Accumulator
MD	Data Memory (MOS)
SP	S-Pad
DP	Data Pad
PS	Program Source
DR	Data ROM

The L/R switch is an alternate-action switch that switches the data which can be viewed between the right 16 or 18 bits (R) and the left 16 or 18 bits (L). The LEFT indicator lights when the left 16 or 18-bit position is selected.

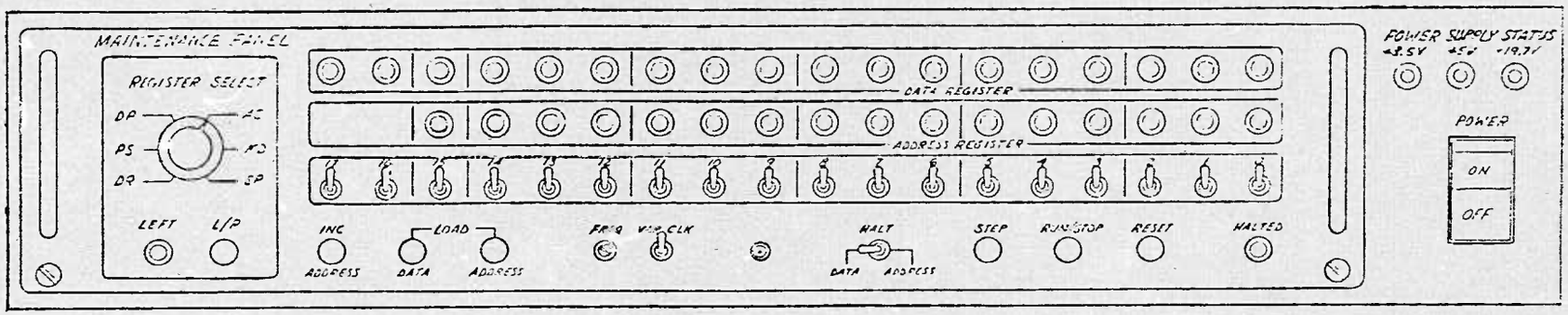


Figure 4-1 AP-90 Control/Maintenance Panel

4.1.4 DATA REGISTER and ADDRESS REGISTER Indicators

The 16 ADDRESS REGISTER indicators indicate the contents of the address register indicated by the REGISTER SELECT position as follows:

Position	Address Register
AC	PSA
MD	MA
SP	SPA
DP	DPA
PS	PSA
DR	DRA

The 18 DATA REGISTER indicators indicate the contents of the register selected by the Register Select Switch. If the LEFT indicator is lit the contents of the left 18-bits are displayed; otherwise the right 18-bits are displayed. In the case of SP selection, SP will appear regardless of the LEFT indicator since S-Pad is 16 bits wide.

4.1.5 Panel Switch Register, LOAD DATA Switch, LOAD ADDRESS Switch and HALT DATA/ADDRESS Switch

The Panel Switch Register switches allow specification of 18 bits of information by putting the switches into the up position for one and the down position for zero.

To load a data word or address register the user selects the appropriate register by means of the REGISTER SELECT switch and the L/R switch. The information is then transferred from the Panel Switch Register by pressing the LOAD DATA switch or the LOAD ADDRESS switch as appropriate.

The 3-position HALT DATA/ADDRESS switch allows a transfer from the run to halt state when the information in the selected data or address register matches the information indicated by the data/address entry switches. When the switch is in the center position, no halt will occur. In this position an address sync test point is available for de-bugging.

4.1.6 INC ADDRESS Switch

The INC ADDRESS switch allows the user to step the address of a particular register by one. (SPA cannot be incremented)

4.1.7 VAR CLK Switch, FREQ Adjust and Sync Test Point

These controls are for use in maintenance of the processor only and are described in Section 7.

4.2 EXECUTION

4.2.1 Stand-Alone Mode

To run in stand-alone mode, the user presses POWER ON and, if the POWER SUPPLY STATUS indicators are lit, then presses RESET. He then enters the starting PSA of the program he wishes to run and pushes the RUN/STOP Switch (or STEP) in order to execute.

4.2.2 HOST/AP-90 MODE

When the AP-90 is communicating with a host computer, the user presses POWER ON, RESET, enters the starting PSA (for this mode) and presses RUN/STOP.

SECTION 5

INPUT/OUTPUT CHARACTERISTICS

5.1 GENERAL FEATURES

The AP-90 offers several interfacing options to the user as shown in Figure 5-1. There are two programmatic ports into and out of the array processor. One of these inputs, Host Data (HD), is 32-bits wide and is usually connected to the data output lines of the host computer. The accumulator (AC) output is the source of data offered to the host computer by the Array Processor. The other programmatic input port C-Register Input/Output (CIØ), is 16-bits wide and is often connected to an external source of data such as A/D converters. Data from the AP is supplied from the C-Register (C).

In addition to the two programmatic ports, there are facilities to handle one or two Direct Memory Access (DMA) ports into and out of Memory Data (MD), the MOS memory of the Array Processor. With the additional DMA logic the user can read from or write into Memory Data (MD) on a cycle-stealing basis. The DMA ports have memory cycle priority assignments higher than the Array Processor.

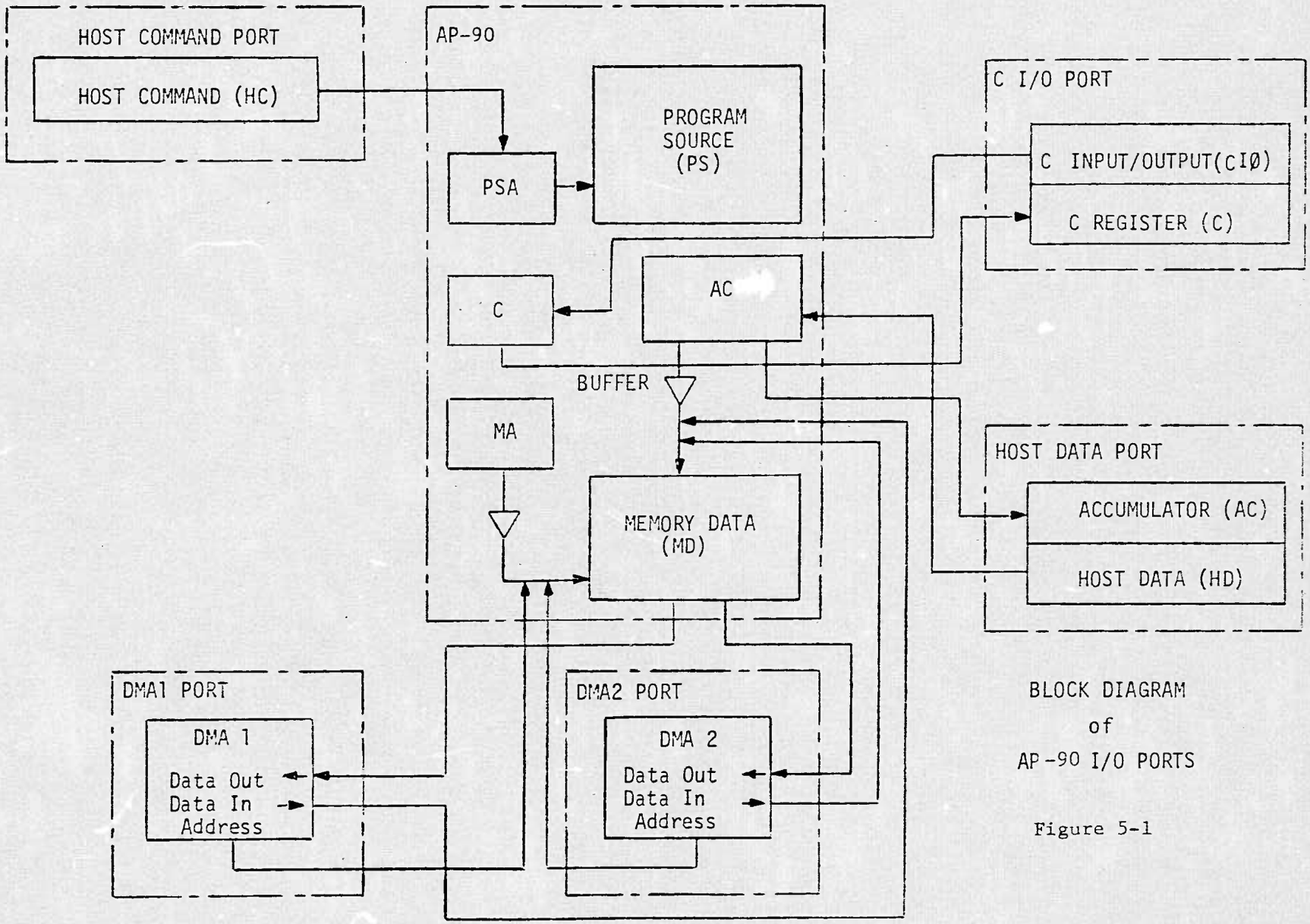
The Array Processor is sent command information over the Host Command (HC) lines. These lines transfer into Program Source Address (PSA) which is the address register that fetches the next instruction out of Program Source (PS). The Host Command path into PSA is assigned the highest priority. So whenever the Host Command path becomes active, the AP immediately stops doing whatever it was doing and starts executing the operation at the address offered by the Host Command lines. These Host Command (HC) lines can be some of the Host Data (HD) lines that supply data to the AP, or they can be unique lines from the host. Included in the instruction set of the Array Processor is the capability to interrupt the host computer. Therefore, any operation or process in the AP can interrupt the host computer upon its completion, simply by including the interrupt host instruction at the end of the operation or process.

5.2 FUNCTIONAL DESCRIPTION

5.2.1 Host Data Port

Input from Host

Name of Lines:	Host Data--HD(00-31)*
Number of Lines:	32 (or 16)
Levels: (TTL)	Logical 1 = 0-0.4 volts sinking 16 ma Logical 0 = 2.4-5 volts
Transfer Rates: (not into MD)	Up to 6 MHz for 32 bit data (167nsec/32 bit word)
Transfer Rates: (into MD)	Up to 2 MHz for 32 bit data (500 nsec/32 bit word)



BLOCK DIAGRAM
of
AP-90 I/O PORTS

Figure 5-1

Transfer Control: Host Data Ready Flip Flop (HDRDY)
j-ed by host when HD lines are stable
k-ed by AP as HD lines are sampled into AC

(Negative transition of Q informs host that HD lines have been read by the AP.)

Output to Host

Name of Lines: Accumulator--AC(00-31)*2

Number of Lines: 32 (or 16)

Levels: (TTL) Logical 1 = 0-0.4 volts sinking 16 ma
Logical 0 = 2.4-5 volts

Transfer Rates:
(not from MD) Up to 6 MHz for 32 bit data (167 nsec/32 bit word)

(from MD) Up to 2 MHz for 32 bit data (500 nsec/32 bit word)

Transfer Control: Host Data Ready Flip Flop (HDRDY)
j-ed by AP when AC contains data for host
k-ed by host when AC lines are sampled by host
(Negative transition of Q informs AP that AC lines have been read by host)

5.2.2 IØ Port

Input from I/Ø

Name of Lines: C Input Output-CIØ(00-15)*

Number of Lines: 16

Levels: (TTL) Logical 1 = 0-0.4 volts sinking 16 ma
Logical 0 = 2.4-5 volts

Transfer Rates:
(not into MD) Up to 6 MHz for 32 bit data (167 nsec/16 bit word)

Transfer Rates:
(into MD) Up to 4 MHz for 16 bit data (500 nsec/3? bit word)

Transfer Control: IØ Data Ready Flip Flop (IØDRDY)
j-ed by IØ when IØ lines are stable.
k-ed by AP as IØ lines are sampled into AC
(Negative transition of Q informs the IØ that the data lines have been read by the AP)

Output to IØ

Name of Lines: Accumulator-- C(16-31)*

Number of Lines: 16

Levels: (TTL) Logical 1 = 0-0.4 volts sinking 16 ma
Logical 0 = 2.4-5 volts

Transfer Rates:

(not from MD)	Up to 6 MHz for 16 bit data (167 nsec/16 bit word)
(from MD)	Up to 4 MHz for 16 bit data (500 nsec/32 bit word)
Transfer Control:	IØ Data Ready Flip Flop (IØDRDY) j-ed by AP when AC contains data for the IØ k-ed by the IØ when AC lines are sampled by the IØ (Negative transition of Q informs the AP that the data lines have been ready by the IØ.)

5.2.3 Direct Memory Access Ports

Input into DMA Logic from External Device

Name of Lines:	Data Input--DI(00-31)*
Number of Lines:	32 (or 16)
Levels: (TTL-Tristate)	Logical 1 = 0-0.4 volts sinking 16 ma Logical 0 = 2.4-5 volts Hi Z = High Impedance source
Transfer Rates:	Up to 2 MHz for 32 bit data (500 nsec/32 bit word)
Transfer Controls:	DMA Data Ready Flip Flop (DMADRDY) j-ed by external device when DI lines are valid k-ed by DMA when DI lines are sampled (Negative transition of Q informs external device that DI lines have been read by DMA.)

Output to External Device from DMA

Name of Lines:	Data Output--DØ(00-31)*
Number of Lines:	32 (or 16)
Levels: (TTL-Tristate)	Logical 1 = 0-0.4 volts sinking 16 ma Logical 0 = 2.4-5 volts Hi Z = High Impedance source
Transfer Rates:	Up to 2 MHz for 32 bit data (500 nsec/32 bit word)
Transfer Controls:	DMA Data Ready Flip Flop (DMADRDY) j-ed by DMA when DØ lines are valid k-ed by external device with DØ lines have been sampled. (Negative transition of Q informs DMA that DØ lines have been read by the external device.)

5.2.4 Host Command Port

Input into Program Source Address Register

Name of Lines:	Host Command--HC(00-11)*
Number of Lines:	12

Levels:	
(TTL) for HC(00-11)*	Logical 1 = 0-0.4v sinking 16ma Logical 0 = 2.4-5v
Transfer Rates:	Up to 4 MHz
Transfer Control:	Host Command Ready Flip Flop (HCRDY) j-ed by host when HC lines are valid k-ed by AP after HC lines are sampled

5.3 PHYSICAL DESCRIPTION

External connections to the AP-120 are made through SAE Dip Stick 88 pin connectors.

The CI/Ø port connection pin assignments are flexible to meet particular requirements of the IØ device.

The DMA logic is separate from the AP logic and has its own connectors. The memory side of the DMA connects directly onto the memory data input bus, address bus and output bus. Two control signals interconnect the DMA and the AP logic.

SECTION 6

PROGRAMMING

6.1 INSTRUCTION SET

The instruction set for the AP-90 comprises fourteen basic instruction formats. Each of these formats offers one "primary" operation, plus up to six "secondary" operations which are all performed in parallel during one machine cycle (167 ns) except for the multiply, DR read, and MD read operations. For each of the primary and secondary operations, the user may select from a variety of different manipulations (there are 15 different possibilities for multiplication, for example).

A general description of the formats is given in Table 6-1; usually, the first operation listed for a format in this general description is regarded as the "primary" operation, although in some formats (such as 5 and 6) the distinction between primary and secondary operations is somewhat hazy. The format names are the assembly language format mnemonics.*

*For details, see the AP-90 Programming Manual

Table 6-1 General Description of Formats

Format Name	Format #	Operations
SETDPR	A	1. Load right half of Data Pad with value given in instruction. 2. Various data transfer options.
SETREG1	B	1. Load a selected address register with value given in instruction. 2. Various data transfer options. 3. Perform operation on Data Memory Address Register.
SETREG2	C	1. Load Program Storage or selected S-Pad register with contents of accumulator. 2. Various data transfer options. 3. Perform operation on Data Memory Address Register. 4. Load Accumulator. 5. Load Data Pad.
SETTBL1	D	1. Access Data ROM for an entry in a table of type 1 (128 entires). 2. Various data transfer options. 3. Perform operation on Data Memory Address Register.
SETTBL2	E	1. Access Data ROM for an entry in a table of type 2 (256 entries). 2. Various data transfer options. 3. Perform operation on Data Memory Address Register.
SETSP	1	1. Load selected S-Pad register with value given in instruction. 2. Various data transfer options. 3. Perform operation on Data Memory Address Register.
ACR→REG	2	1. Load selected register with contents of Accumulator. 2. Various shifts of selected S-pad register, other data transfer options. 3. Perform operation on Data Memory Address Register. 4. Load accumulator. 5. Perform fixed-point add. 6. Perform operation on Data Pad Address register.
ACR→SP	3	1. Transfer contents of Accumulator to selected S-Pad register. 2. Perform operation on Data Memory Address Register. 3. Load Accumulator. 4. Perform fixed-point add. 5. Perform operation on Data Pad Address register.

Format Name	Format #	Operations
SP→SP	4	1. Transfer data between selected S-Pad registers. 2. Various shifts of S-Pad, other data transfers (7 options). 3. Perform operation on Data Pad Address register. 4. Load Accumulator. 5. Load Data Pad. 6. Perform fixed-point add. 7. Perform operation on DPA.
TEST3	5	1. Perform conditional jumps within current 16 word page of Program Storage. 2. Perform operation on Data Memory Address register. 3. Load Accumulator. 4. Load Data Pad. 5. Initiate fixed-point multiply. 6. Perform operation on Data Pad Address register.
SHIFT or TEST4 or FLAG	6	1. Shift Accumulator or current Data Pad word. 2. Special-purpose set/clear for FFT,IFT,host interrupt, etc. 3. Conditional jump within current page of Program Storage.
TEST0 or FADD	7	1. Floating/fixed point add. 2. Conditional jump within current page of Program Storage. 3. Various operations on Accumulator. 4. Load Data Pad. 5. Perform operation on Data Pad Address register.
TEST1 or MULT	10	1. Floating/fixed point multiply. 2. Floating/fixed point add. 3. Conditional jump within current page of Program Storage. 4. Various data transfer operations. 5. Perform operation on Data Memory Address register. 6. Load Accumulator. 7. Perform operation on Data Pad Address register.
TEST2	20	1. Conditional jump within current page of Program Storage. 2. Various data transfer operations. 3. Perform operation on Data Memory Address register. 4. Load Accumulator. 5. Load Data Pad. 6. Perform fixed-point add. 7. Perform operation on Data Pad Address register.

Table 6-2. Field Definitions

FORMAT NAME	FORMAT NUMBER	31 30	29 28 27	26 25 24	23 22 21	20 19 18	17 16 15	14 13 12	11 10 9	8 7 6	5 4 3	2 1 0
SETDPR	A		0	DPR	SPA	X2	VALUE					
SETREG1	B		0	REG1	SPA	X2	MA	Ø	VALUE			
SETREG2	C		0	REG2	SPA	X2	MA	AC1		ADDER		DPA
SETTBL1	D		0	TBL1	SPA	X2	MA	Ø	VALUE		Ø	
SETTBL2	E		0	TBL2	SPA	X2	MA	Ø	VALUE		Ø	
SETSP	1		1	Ø	SPA	X2	MA	Ø	VALUE			
ACR→REG	2		2	REG	SPA	XO	MA	DP	AC2	ADDER		DPA
ACR→SP	3		3	MULT	SPA	XO	MA	AC1		ADDER		DPA
SP→SP	4		4	SPAW	SPAR	XO	MA	DP	AC2	ADDER		DPA
TEST3	5		5	JUMP	SPA	TEST3	MA	AC1		DP	C Ø	DPA
TEST4 or SHIFT or FLAG	6		6	JUMP	Ø TEST4	F SC	STATUS	FLAG		Ø	C SHIFT	
TESTO or FADD	7		7	JUMP	TESTO	F A	ADDI	ACO		DP	C ADDIFN	DPA
TEST1 or MULT	10	1	TEST1	JUMP	MULT	F X1	MA	AC1		ADDER		DPA
TEST2	20	2	TEST2	JUMP	SPA	X2	MA	DP	AC2	ADDER		DPA

Table 6-3. Field Selections

CODE	ADDER A,B INPUT	ADDERFN	DPA	MA	TEST1	TEST2	TEST3	TEST4	XO	X2
0	DP,AC	A-1	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP
1	DP',AC	A	INC	SP	AC→C	SP=0	JMP	JMP	AC→C,NØMI	NØMI,AC→C
2	AC,ACX	A+B	DEC	SP+1	DPALSC=7	SP≠0	AC22=1	DP22=1	SP→SP	WRT
3	0,ACX	A+B+1	DPA-8	SP-1	JMP	DPA≠7	AC23=1	DP23=1	MAFN→SP	MAFN→SP
4	DP,ACX	A-+B	SP	MA+1	DPA≠37	DPA≠37	DPA≠37	ADC23=1	MA→SP	MA→SP
5	ACL/DPR,ACR/0	A-B	DPA+SP	SP+8	HDRDY=0	HDRDY=0	HDRDY=0	MDAQ=1	SP*2→SP	DPA→SP
6	0,MD	A+-B	SWAPSP (DPA→SP)	SP+MA	IØDRDY=0	JMP	Ø	MDBQ=1	SP/4→SP	DPA+SP+1→DPA
7	DP'/DPR,ACL+1/ACR	A+1	WRT	DRA+SP DP	IØDRDY=1	HDRDY=1	HDRDY=1	Ø	SP 8→SP	AC→PS

Field Selections (continued)

CODE	ACO	AC1	ADDI A,B INPUTS	ADDIFN FUNCTIONS	FLAG	MULT M1,M2	REG	TESTO
0	NØ-ØP	NØ-ØP	DP,AC	A-1	NØ-ØP	NØ-ØP	PSA	NØ-ØP
1	AD→AC	Ø	DP',AC	A	FTQ=1	M1,ADR	DPR	ØVFL=0
2	HD→AC	Ø	AC,AC	A+B	CLRFLGS	M1,ADL	DRA	ØVFR=0
3	C→AC	C	0,AC	A+B+1	IFTQ=1	DRM,ADR	MA	ADTEST=1
4	FP→AC	MP→ACL	SCLR,AC	Ø	FTP2Q=1	DPR,M2	DPA	JMP
5	RNDAC	AD→ACL	FPO,AC	A-B	FTP2Q=0	DPL,M2	SPA	AL=BL
6	FLØTAC	HD→ACL	FP1/2,AC	AL+BL+1,AR	SP→DRA	DRR,M2	NA	AL>BL
7	ADR→ACR	CL→ACL	FP-1,AC	A+1	SPAX=11	DRL,M2	NA	AL<BL
10	ADL→ACL	MP→ACR	DP,MD		IØDRDY=0	DPR,ADR	CALLSUB	NA
11	SCLDPR→ACR	AD→ACR	DP',MD		IØDRDY=1	DPL,ADR	MP→C	NA
12	ADL→SN	HD→ACR	AC,MD		HDRDY=0	DRR,ADR	PS→C	NA
13	DSCL→SN	CR→ACR	0,MD		HDRDY=1	DRL,ADR	NO-OP	NA
14	MP→ACR	MP→AC	Ø		SENDHI	DPR,ADL	Ø	NA
15	SP→SCL	AD→AC	DR,AC		MDOVF→MDS	DPL,ADL	LØADPS	AR=BR
16	SMAG→2C	HD→AC	Ø		DECIMATE	DRR,ADL	SCL→SP	AR>BR
17	2C→SMAG	C→AC	Ø		Ø	DRL,ADL	SETEXIT	AR<BR

CODE	AC2	DP	SC	X1
0	NØ-ØP	NØ-ØP	ACR	NØ-ØP
1	AD→AC	ACL→DPL	ACR,SN	MAFN→SP
2	HD→AC	ACR→DPR	DPR	MA→SP
3	C→AC	AC→DP	DPR,SN	M2→DPR

CODE	REG1
0	PSA
2	DRA
3	MA
4	DPA
10	CALLSUB
13	NØ-ØP
17	SETEXIT

CODE	REG2
11	MP→C
12	PS→C
15	LØADPS
14	NØ-ØP
16	SCL→SP

CODE	SHIFT
n	R:n
40+n	L:n

CODE	DPR
1	DPR

CODE	TBL1
6	TBL1

CODE	TBL2
7	TBL2

CODE	C
0	NO-OP
1	AC→C

CODE	A
0	NORMAL
1	DBLE

SECTION 7

MAINTENANCE

7.1 GENERAL

The facilities described in this section, together with the design documentation and an oscilloscope*, allow complete maintenance of the unit without the host computer or any other additional equipment.

7.2 ACCESSIBILITY

All internal control and adjustments are readily accessible. All of the unit's circuitry and memories are easily accessible. All wiring connections, integrated circuits, and test points are easily reached with a normal oscilloscope probe while the unit is operating.

7.3 CONTROL/MAINTENANCE PANEL

The maintenance panel displays the contents of the important registers and memories, loads the contents of a panel switch register into the important registers and memories, and executes instructions held in the maintenance panel or in other external sources. Other features of the maintenance panel are a RUN/STOP button, a STEP button, a variable oscillator to slow down or speed up the clock rate, and a HALT switch. When the HALT switch is activated, the unit halts when the contents of the selected register match the contents of the panel switch register. If the HALT switch is not activated, the Sync Test Jack may be used to test equality of Panel Switch Register and Selected Address Register. Typically, this is used as an external trigger for the oscilloscope. Details of all switches and indicators are given in Section 4.

7.4 DIAGNOSTIC ROUTINES

Diagnostic routines are supplied with the AP-90 and are stored in Program-Source ROM. Each major element listed below is tested by executing all possible functions relating to that element, and with a sufficient variety of data so as to locate any individual failure within the element under test:

Adder	DPA
Scaler	MA
Register Transfer	S-Pad
Data Pad	DRA
Memory	Multiply

The separate tests are chained together by placing the appropriate addresses in EXITS. When a failure is encountered, three options are available:

a. Spin at the test where the failure occurred.
b. Increment a cell in S pad dedicated to that particular test.
c. Branch to a routine that will pass the current state of the machine to the host computer.

*Tektronix Model 475 or equivalent.

A test that is failing can be placed in loop mode by changing the appropriate cell in EXITS and making the test that is failing a no-op. If option b is selected, at the completion of the entire diagnostic package, a test for zero of the appropriate cells in S pad is made to determine if any failures have occurred. The run time of the complete diagnostic package is generally less than one-half hour.

APPENDIX 3

The Very Distant Host (VDH) Connection for the MP-32

CHI
SIGNAL SYSTEMS

TMB7 THE VERY DISTANT HOST (VDH) CONNECTION FOR THE MP-32 REFERENCE MANUAL

CONTENTS:

I. INTRODUCTION

II. REFERENCES

III. GENERAL DESCRIPTION

IV. INPUT CONTROL FLOW

V. OUTPUT CONTROL FLOW

VI. INPUT TIMING

VII. OUTPUT TIMING

VIII. GLOSSARY

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	10/75	pp. Title page, I-1, VI-1, VI-3, VII-2, VIII-1, VIII-3,VIII-4,VIII-5

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29104

© 1973
by Culler/Harrison, Inc.

I. INTRODUCTION

The Very Distant Host (VDH) Connection for the MP-32 is responsible for moving information between the processer and a communication system. The communication system operates in a full-duplex, serial manner with a Binary-Synchronous transmission format. Further, that data presented to the communication line will be in the form of 8-bit characters, or codes.

In the Bi-Sync form of communication, certain characters have been designated as control codes that are used by the communication controllers to maintain synchronism or to detect the beginning and end of a message. For example, the SYN code is used to represent a "quiescent" line, upon which no information is being sent; the DLE code is used to inform controllers that a message is about to begin or end.

The Bi-Sync method also allows for the transmission of binary information. It must be seen that in this case, information on the communication system will resemble any one of the special, dedicated codes. This problem is circumvented by use of the DLE code. The following paragraph will illustrate the technique:

> It is a rule of Bi-Sync operation that once a message has begun no special codes are recognized by a receiving terminal except the DLE. To begin a message, a code-pair is sent (DLE/STX), to end a message a differant code-pair is sent (DLE/ETX). If the message itself contains a code that resembles a DLE then the combination DLE/DLE is sent. One of these DLE's is actually a data-code and the other is a special code that is inserted by the transmission controller to cause the receiving controller to ignore the code. It is understood that the receiving controller will discard the second DLE.

As a full message is sent, its contents are "Summed" on a bit-by-bit basis in a special counter. This Sum is referred to as the Cyclic-Redundancy Check Sum or CRC. The CRC is 24-bits (3 characters) in length. The CRC is appended to the outgoing message after the DLE/ETX code-pair.

At the receiving site the same message passes through a CRC "Summing" process as well. The CRC created by the transmitting site and the CRC at the receiving site must sum to "0". The receiving controller takes care of this final summation. If a non-zero sum occurs, the message is assumed to contain an error.

With the VDH, it has been agreed upon that once a message has been transmitted, the sending site will wait a suitable length of time for a receiving site to take in the message, check the CRC, and send an acknowledgement. If no response is returned, the transmitting site is obligated to send the entire message again. Receiving sites are not allowed to respond to an erroneous message.

II. REFERENCES

A. Report No. 1822--Appendix F
"Specification For The Connection of a Host to an IMP"
Available from Bolt, Beranek and Newman, Inc.

B. Data-Set Model 303--Technical Manual
Available from Bell Laboratories

III. GENERAL DESCRIPTION

The controller consists of two sections, one of which is responsible for moving data from the CPU out into the communication system (Transmit or Output) and the other that is responsible for moving data from the communication system into the CPU (Receive or Input). These two sections have been blended into one package that is connected to the CPU data bus and interrupt logic on one side and to the modem for the communication system on the other.

Although these two controllers are within the same package there is very little logic that can be considered common to both. This common logic consists of the term AOK which indicates that the modem is up and operational, the gating which produces an interrupt to the CPU (SRQP) and the input data bus multiplexer circuitry.

A review of the glossary will show that the active elements and critical gating for the input side contain an "I" within their logical names and those concerned with the output controller contain an "0". It might also be well to note that the four letter entries are flip-flops (F/F) while the three letter entries are gates.

Non-critical gating has been numbered with the series 700 and 800 for the Input Controller while 500 and 600 series are used for the Output Controller. The common gating that drives the interrupt F/F, uses 900 series.

The transfer of information between the modem and either controller takes place in a serial, synchronous manner. The signal lines are as follows:

A. Receive Data (RD) - Incoming data signal from the modem. A data-bit is valid and may be "sampled" when the SCR signal (below) makes a positive (↑) transition and may change upon a negative (↓) transition.

B. Serial Clock Receive (SCR) - The square-wave signal that signifies the time that a data-bit may be sampled. The modem, or data-set, provides this signal.

C. Send Data (SD) - The outgoing data line that is responsive to the contents of the Output Shift Register. This line may be changed upon the negative (↓) transition of the SCT (below) and must remain steady during the positive (↑) transition.

D. Serial Clock Transmit (SCT) - This square-wave signal is provided by the modem to indicate when a new bit may be placed on the Send Data (SD) line.

E. Data-Set Ready (DSR)
Clear to Send (CTS)
AGC Lock (AGC) - These signals indicate that the modem is operational. They are grouped into gate AOK to be tested by the CPU in status.

F. Request to Send - This signal is required at the positive (+) level before the modem will transmit.

The signal lines between the controllers and the CPU are summarized below.

A. Data-Bus to CPU (E1X00D* thru E1X17D*) - These 16 signal lines are driven by open-collector gates and are used to transfer an 8-bit parallel character from the Input Buffer Register plus a set of Status lines including INRQ which indicates the Input Controller needs service and OURQ indicating that the Output Controller needs service.

B. Interrupt Signal (SR02J2*) - This signal is sent by SRQP being set. It lasts for one clock interval and causes the CPU to be aware of a need for service by either controller.

C. Address Signal (DADR5*) - This signal is presented to the controllers to make them respond to one of the Command lines from the CPU. This is a unique address signal, only held by the VDH.

D. CPU Command Lines (DC0M0 thru DC0M7) - These 8 command lines are only sampled by the VDH when DADR5* (above) is raised by the CPU. See the Glossary (Section VIII) for interpretation and duration.

E. Timing Pulse (EEXTO) - This particular CPU timing pulse is used by the VDH to know the exact time that the Input Data and Status lines have been sampled.

F. Data-Bus from CPU (I1R00Q 3 thru I1R07Q 3) - These 8 lines are used by the CPU to present a character for transmission to the Output Controller.

The controllers make use of a pair of unique devices. These are: the Motorola MC2257L Terminal Transmitter chip which contains an 8-bit Output Buffer Register, a Serial Shift Register, and Timing Logic to conform to various output formats; and the Motorola MC2259L Terminal Receiver chip which contains an Input Shift Register, a separate parallel, 8-bit Buffer Register, and Timing and Control circuitry to handle various formats of incoming data. Much of the control logic in both the input and output controllers is used to operate these chips.

IV. INPUT CONTROL FLOW

Figure IV illustrates the sequence of Input Controller operation while it is moving information from the data-set, or modem, into the CPU. Briefly it begins in a "Search" mode, moves to a "Character" mode and then to a "Message" mode when a message arrives. It returns to Character mode by CPU command each time a message is completed. It returns to Search mode upon abnormal conditions or by command.

The VDH input section deals with a Binary-Synchronous form of communication. In this technique a quiescent condition (non-data transfer) on the incoming signal line is indicated by a series of 8-bit codes referred to as sync codes (SYN). The Input Controller begins its operation by "Searching" to detect the presence of these SYN codes. It does this by decoding the 8-bit Buffer Register after each new bit is shifted in, to see if its contents are in fact a SYN code. If not, the controller remains in the Search mode. Once a SYN is detected the Controller advances to "Character" mode and only decodes the Input Buffer each 8 bit-times (or character-time).

In Character mode the Input Controller begins looking for a non-SYN code. However, it is understood that at least two SYN codes are sent prior to any message. The Input Controller is obligated to receive these codes before a non-SYN code is allowed to reach the CPU.

Upon receipt of the first non-SYN code after two SYNs, the Input Controller passes this non-SYN into the CPU and moves into the "Message" mode. It is assumed that this first non-SYN will be the DLE of a DLE/STX code-pair. The program within the CPU must assess this occurrence, however, and will reset (IRST) the Input Controller to the Search mode if an error is detected.

Once in the Message mode, the Input Controller passes all characters to the CPU as received. When the CPU finally perceives the DLE/ETX code-pair, plus the CRC, it will move the Controller back out of Message mode but leaves it in Character mode (See Gate IDN).

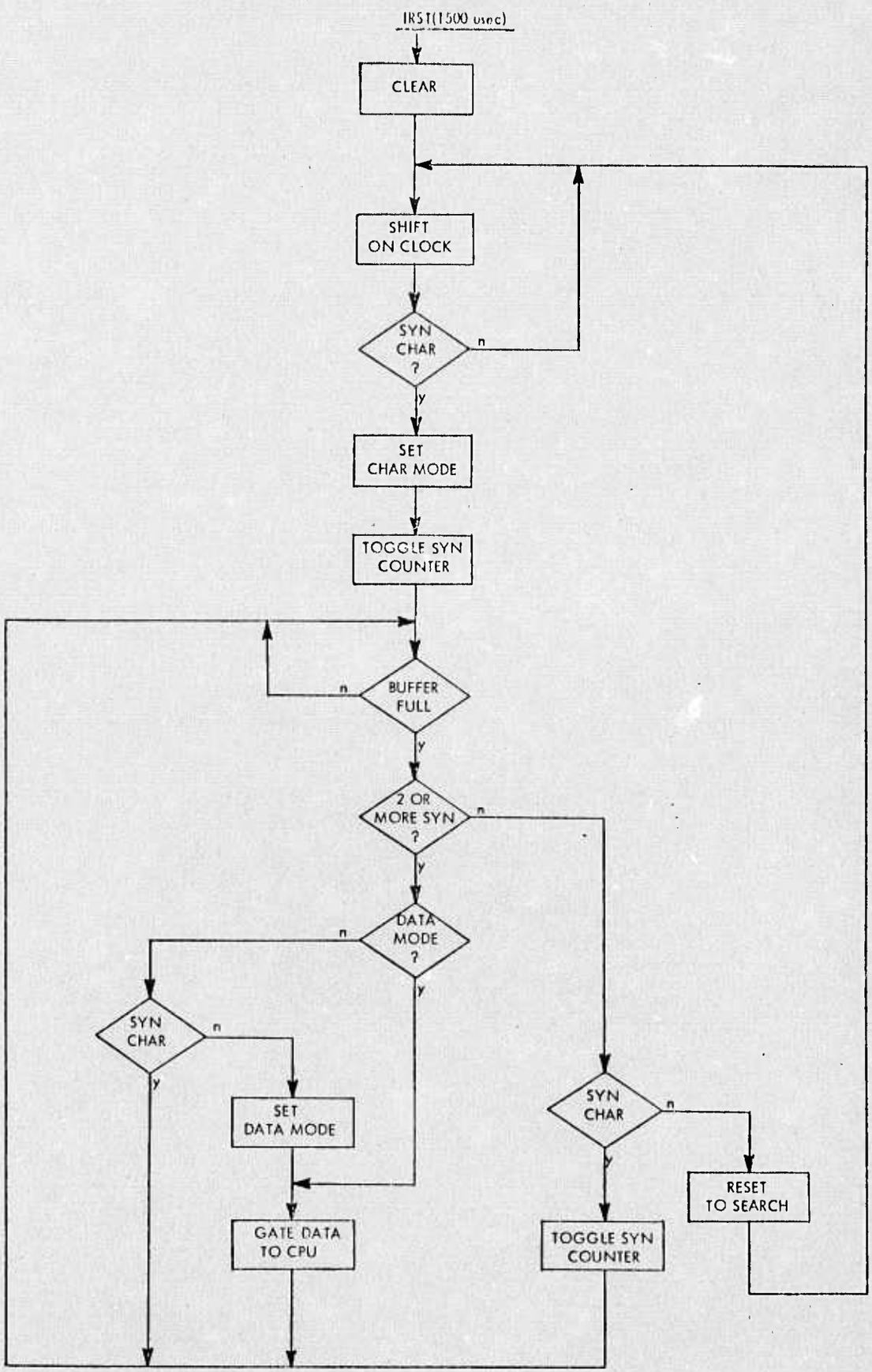


FIGURE IV - INPUT CONTROL FLOW

V. OUTPUT CONTROL FLOW

Similar to the input section, the output section begins in a "Cleared" state. This is done when the CPU raises Command Line DCOM7 with Address Line DADR5*. This combination raises the signal ORS which in turn sets flip-flop ORST to clear the logic of the Output Controller.

To conform to the Binary-Synchronous mode of transmission the controller must place SYN codes on the output signal line. These codes are gated to the Output Buffer Register each time the Terminal Transmitter chip indicates that its buffer is empty (OBE). This will continue to occur until the CPU places the Output Controller into data mode. Subsequent to that time data from the CPU will be placed into the Output Buffer.

Referring to Figure V it is seen that the logic flow proceeds from the Cleared state to the "Buffer Empty Test". If the buffer is empty a second test is made to see if two SYN codes have been sent. This not being the case a SYN code is set into the Output Buffer and the SYN counter is toggled. The Output Controller then resumes its test to see if the buffer is empty. This cycle is repeated once more and the SYN counter is toggled to indicate that two SYN codes have been sent.

With two SYN codes safely on their way, the test to see if the controller is in data mode occurs each time the buffer is empty. If the CPU has not placed a controller into data mode a subsequent SYN code is placed in the Output Buffer.

Transmission proceeds in this manner until the CPU has completed the data transfer followed by the CRC. The CPU informs the Output Controller of completion by raising an "End of Send" (EOS) indication. This is done by using command line DCOM5. The EOS toggles the SYN counter once more (resets to 00) so that two more SYN characters must be sent before data mode is allowed.

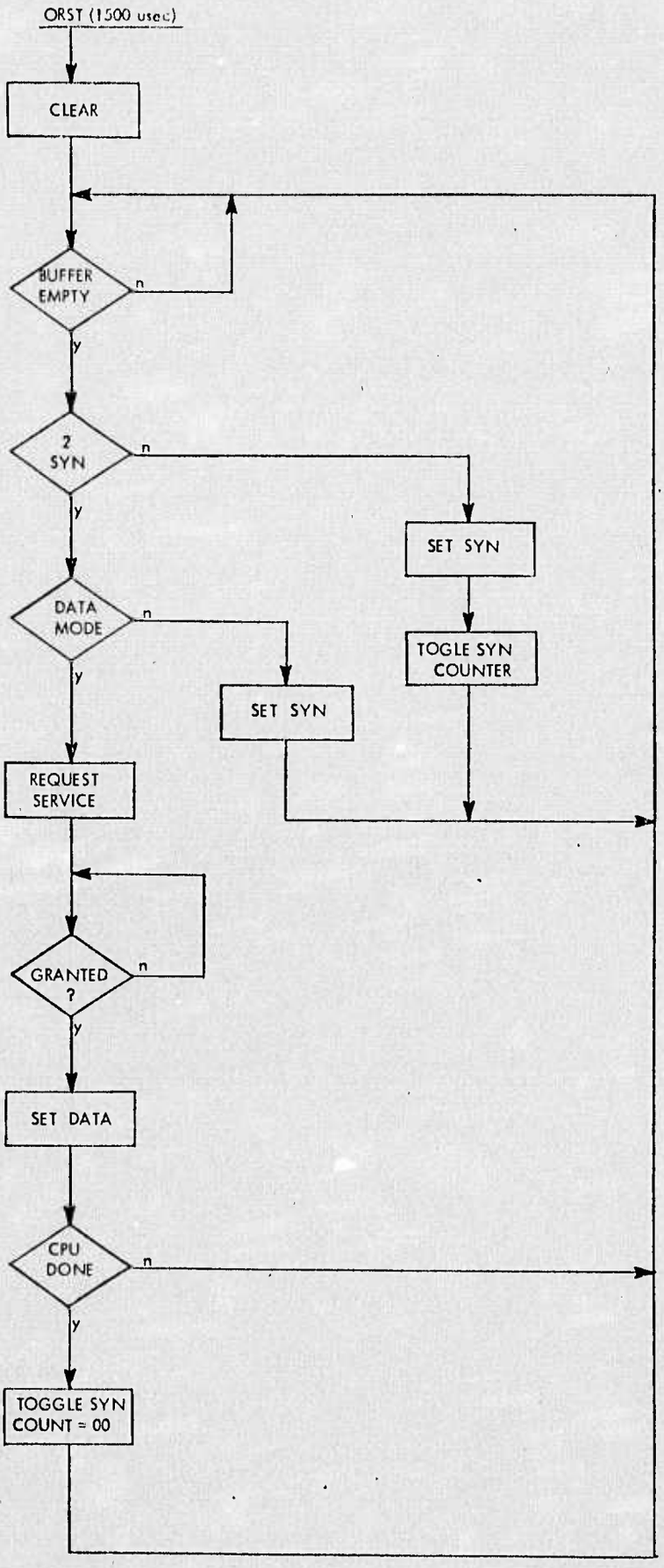


FIGURE V - OUTPUT CONTROL FLOW

VI. INPUT TIMING

Refer to the Input Timing Diagram. The external signals that are important to the Input Controller will be the Serial Clock Receive (SCR), and the response to an interrupt by the CPU indicating that data has been taken out of the Input Buffer Register.

The SCR is a square-wave that indicates when the data line is in transition or when the data can be considered valid. The controller captures the asynchronous signal with a F/F (ISCR)*. ISCR is used to synchronize all subsequent operations in the Input Controller.

The two flip-flops ISY1 and ISY2 make up the SYN counter. When ISY2 "true" it indicates that two SYN characters have been received. Until that time and until data mode is established, the term ISYT (SYN Test) is raised each time the data is valid in the Input Buffer. This F/F is used to "center-sample" the data in the buffer at the time it is considered quiescent.

Beginning with the Search mode, the pulse shown on ISYT in the Timing Diagram occurs every bit-time. During one of these bit-times it is seen that a SYN code resides in the Input Buffer and the latch ICHA is turned on to indicate character mode. From this point on the only valid time to assess the data within the Input Buffer is when the Input Buffer Full (IBF) signal is raised. It is seen that at the same time that ICHA is "set" the SYN counter is toggled by means of gate 799.

*Note: All flip-flops are clocked by the 6 MHz clock.

At the next occurrence of IBF, when ISYT tests the buffer's contents, it is expected that a subsequent SYN character will reside in that buffer. Upon this test the SYN counter will be toggled again causing ISY1 to be reset. If the character is indeed a SYN character then ISY2 will be turned on by means of gate 802. If it is not a SYN character then both ISY1 and ISY2 will be "0" and the Search mode will begin again.

From this point onward, and each time the buffer is full, ISYT will perform the test. At the moment the buffer has a non-SYN character the test will result in the setting of ISY1 and therefore data mode will be entered (See IDM).

Since this first character is assumed to be, and should be, a data character, the flip-flop INRQ is set so that the CPU can take the character. The one restriction on the setting of this flip-flop is that it cannot occur during the time the CPU is reading from the controller (IDG must not be "true"). However, IDG will only be true for a short period, and INRQ will be set subsequently.

The setting of INRQ also causes an interrupt to be sent to the CPU if no interrupt was presently outstanding. The interrupt is caused by the setting of SRQP. The indication that no interrupt was outstanding is that neither INRQ nor OURQ have been raised (gate 900).

Throughout the operations above the flip-flop IDTL is required to properly define the period during which a sample may be taken

from the Input Buffer Register. Each time the sample is utilized, IDTL is reset. IDTL is set each time an input shift takes place and is reset by a SYN test or the taking of the data by the CPU.

The data is gated to the Input Data-Bus of the CPU by the term IDG. IDG is produced by command line DC0M0 which is raised for a period of about 667 nsec. This long duration gives the input bus lines time to settle and the strobe EEXT0, which occurs approximately 333 nsec after the rise of DC0M0, indicates the actual time the data is taken. It is this occurrence that resets both IDTL and INRQ (gates 807 and 905).

VII. OUTPUT TIMING

As in the case of the input section the modem produces a square-wave clock to indicate when a data bit can be changed and when the modem is "center sampling". This clock, SCT, is brought into synchronization with internal logic by the flip-flop OSCT. All internal transfers are in concert with OSCT.

As previously described the Output Shift Register must constantly be shifting some intelligible character. Therefore, each time the buffer of the Terminal Transmitter chip becomes empty (OBE) a character must be inserted into that buffer. The time to allow the insertion of such a character is governed by the amount of time it takes to shift the 8 bits of the previous character out of the register. A data rate of 200 K bits/second allow approximately 35 usec to accomplish this.

Once cleared, both OSY1 and OSY2 will be in the "0" state, indicating no SYN characters have been sent. The flip-flop OSYS will be set each time the buffer is empty if OSY2 has not been turned on (indicating two SYN codes have been sent) or if OSND has not been turned on (indicating that the CPU desires to send data - See Gate 601).

When the first SYN code is loaded into the Output Buffer, OSY1 is set. When the second SYN code is loaded into the buffer, OSY1 is reset and OSY2 is turned on. From this point the CPU may send data by raising the signal SOS which in turn sets the flip-flop OSND. The next time that the buffer is empty (OBE) the CPU is interrupted and a data character is requested (OURQ).

There is only one restriction on the setting of OURQ. It cannot be allowed to occur while the Input Data and Status lines are being read by the CPU. This is prohibited by the term IDG* in gate 606. Similarly, the interrupt SRQP will be raised simultaneously with the setting of OURQ unless a prior interrupt is already in process.

When the CPU has a character available it places it on the Output Bus lines from the CPU and raises Address Line (DADR5*) with Command Line (DCOM6). The command must remain in effect for a period of 1167 nsec. The data must remain quiescent for at least 333 nsec following the drop of the Command Line.

The data is strobed into the Output Buffer by ODS. The strobe term OBS is either set by the raising of OSYS or the raising of ODS. When OBS goes "true" the data lines OB0 through OB7 are gated into the Output Buffer. The code which is present on these data lines is the output of a multiplexer which takes its source either from the CPU Data-Bus or from a fixed, hard-wired, SYN code. The decision of which code source to place into the buffer is controlled by the term OGD.

VIII. GLOSSARY

ADR	Receiver and amplifier for DADR5*.
AGC*	Receiver for AGC Lock signal from modem.
AOK	Indicates modem is Ready and Clear To Send.
CTS*	Receiver from modem for Clear To Send signal.
DADR5*	Address line from CPU. Used with DC0M0 thru DC0M7.
DC0M0	Command line from CPU for "Read Data and Status".
DC0M1	Command line from CPU for "Set Search Mode" (12 clocks).
DC0M2	Command line from CPU for "Drop Data Mode" (2 clocks).
DC0M3	Command line from CPU (Unused).
DC0M4	Command line from CPU for "Start Of Send" (2 clocks).
DC0M5	Command line from CPU for "End of Send" (2 clocks).
DC0M6	Command line from CPU for I_1 equals data (7 clocks).
DC0M7	Command line from CPU for "Transmit Reset" (12 clocks).
DSAGC	Signal line from modem for AGC Lock.
DSCTS	Signal line from modem for Clear To Send.
DSDSR	Signal line from modem for Data Set Ready.
DSRD	Signal line from modem for Read Data (incoming data).
DSRQS	Signal line to modem for Request To Send.
DSSCR	Signal line from modem for Serial Clock Receive.
DSSCT	Signal line from modem for Serial Clock Transmit.
DSSD	Signal line to modem for Send Data (output data).
EEXTO	Signal line from CPU. Used with DC0M0. Indicates data taken.
EOS	Gates ADR with DC0M5 to indicate End Of Send.

E1X00D* thru E1X17D*	Input bus drivers for Data and Status to CPU.
IBB	Input Break signal from receiver chip.
IBF	Input Buffer Full signal from receiver chip.
IBO	Input Buffer Overflow from receiver chip.
IBP	Input Buffer Parity Error from receiver chip.
IB0 thru IB7	Input buffer data from receiver chip, IB0 is Least Sig. Bit.
IB0* thru IB7*	Inverters for IB0 thru IB7.
ICHA	Input character mode F/F.
ICHA*	(Same as above)
IDA	Input data from modem.
IDA*	Receiver for input data signal from modem.
IDG	Gated signal for ADR and DC0M0, causes Data and Status to be gated to Input-Bus.
IDG*	(Inverter for above)
IDM	Input data mode, all data is for CPU.
IDM*	(Inverter for above)
IDN	Gate for ADR and DC0M2. Indicating CPU is done receiving data so drop data mode.
IDTL	Data Time Latch F/F. Defines sample time for Input Buffer.
IDTL*	(Same as above)
IRS	Gate for ADR and DC0M1. Causes reset to take place.

1RS*	(Inverter for above)
IRST	Reset F/F. Causes input side to be cleared back to "Search" mode.
IRST*	(Same as above)
1SCR	Receive Clock F/F. Brings modem clock for receive into synch with internal logic.
1SCR*	(Same as above)
ISY	1nverter for ISY*
1SY*	Decodes Input Buffer to detect SYN character.
ISYT	Input Sync Test F/F. Causes test to be made for SYN character at proper time.
1SYT*	(Same as above)
ISY1	SYN Character Counter (LSB) toggles when SYN code received (first two only) "set" for data mode.
ISY1*	(Same as above)
ISY2	SYN Counter F/F (MSB) "set" by second SYN character received.
I1R00Q*3 thru I1R07Q*3	Output Data Bus Lines from CPU.
JODA	Special gate to drive output data transistor from Output Data signal.
JOD2	Jumper from emitter of output driver to series diode for Send Data signal to modem..
JOD3	Jumper for output driver transistor collecter to its 47 ohm load.
OBE	Output Buffer Empty signal from transmitter chip.
OBS	Output Buffer Load Strobe. Gates either SYN code or output data to the buffer.

OB0 thru OB7	Miltiplexed signals for presentation to Output Buffer.
ODA	Output Data signal from transmitter chip.
ODS	Gate for ADR and DCOM6. Drives OBS when data is present on lines from CPU.
OGD	Indicates data transfer mode.
ORS	Gates ADR and DCOM7, causes reset of the Output Controller under CPU control.
ORS*	(Inverter for above)
ORST	Output Reset F/F. Set by ORS signal.
ORST*	(Same as above)
OSCT	Transmit Clock F/F. Brings Serial Clock Transmit from modem into synch with internal clock.
OSCT*	(Same as above)
OSND	Send Mode F/F. Set by SOS from CPU.
OSND*	(Same as above)
OSYS	Gate SYN code to Output Buffer.
OSYS*	(Same as above)
OSY1	SYN Character Count F/F. Toggles when first or second SYN is sent or "set" to enter data mode.
OSY2	SYN Code Counter. Indicates two or more SYN codes have been sent (or data mode with OSY1).
OSY2*	(Same as above)
OURQ	Output Request F/F. Indicates to CPU that data is needed for output.
OURQ*	(Same as above)
P8M	8MHz clock signal as used within Input and Output Controllers.
SCR	Receive Clock from modem.

SCR*	Receiver for Serial Clock Receive signal from modem.
SCT	Transmit clock from modem.
SCT*	Receiver for Serial Clock Transmit from modem.
SOS	Gate for ADR and DC0M4. Indicates Start Of Send (SOS) operation from CTU.
SRQP	Interrupt F/F. Indicates that either input request or output request are active. Drives SR02J2* (125 nsec).
SRQP*	(Same as above)
SR02J2*	Interrupt driver to CPU.

APPENDIX 4

Specification of Analog Input/Output Capability for Serial 2 MP

CHI

CULLER-HARRISON INC.

~~5770 Thornwood Drive~~, Goleta, California 93017
150-A Aero Camino

Telephone (805) ~~967-0424~~
968-1813

CHI-TN-74-38
December 19, 1974
Ray Bjorkman

SPECIFICATION OF ANALOG INPUT/OUTPUT CAPABILITY FOR SERIAL 2 MP

1. Analog Input

There will be up to eight analog input channels. Each channel will have its own instrumentation type amplifier, Sample and Hold unit, A/D converter, opto-isolators, and Tri-state output buffer register. Each channel is housed in its own separate module which may be mounted in a common rack or case. The back plane wiring will be a Tri-state uni-bus structure with addressing being done from data pad address register. An input and an output connector will be provided on the back of the case for daisy chaining or terminating the digital lines.

2. Analog Output

There will be three output channels. Each channel will have its own D/A converter, opto-isolators, and a pair of input buffer registers, buffer A and buffer B. At load time buffer B is loaded from buffer A, which had been loaded previously from the data bus with channel addressing being provided by PA and transfer control from an analog output command.

3. Analog Timing

The A/D conversion rate and D/A output rate are to be governed by the sync counter, a 12-bit synchronous down counter and buffer register. This will provide sampling rates of from one clock to 4096 clocks, with a resolution of one clock, (250 ns - 1024 μsec). The sync time will be set from PD, with an analog output command, and PA=13, and shall be $\geq$20 μsec. When the sync counter reaches zero, the Load line is dropped putting the sample and hold units in hold mode. At this same time a count of 32 is loaded into the convert counter which counts down to zero. When this counter equals 24 the convert line is dropped, starting the A/D conversion. At a count of zero the load line is raised which transfers all A/D outputs to their respective buffers, transfers all D/A buffer A's to their respective buffer B's, and places the sample and hold units in sample mode. The computer receives an interrupt (S5) and has until the next load time (a minimum of 20 μsec) to service all analog input and output channels.

4. Commands

The device address of the analog input/output module is 12. A standard I/O command of 2 1 J 14 712_ or 7 0 J 14 712_, with the channel address coming from PA and data (in the case of output) coming from PD.

COMMAND DEFINITION

A Field	Description
0	Connect selected A/D buffer register to the DI bus
1	Same as 0, but in addition clear interrupt bit, S5
2	Transfer current PD contents to selected analog output channel's buffer A (input buffer)
3	Same as 2, but in addition clear interrupt bit, S5.

With commands 0 or 1 a DI→E or DI→E→PD command with the channel address in PA must be used. Input address = PA 0-7.

With commands 2 or 3, PA address of 10-17 should be used with channel assignments as follows:

PA=10 DAC 1
PA=11 DAC 2
PA=12 DAC 3
PA=13 Sync Module
PA=14 Stepper Motor (mike positioning)

5. Typical Command Sequence

1) Load PD_{13} with desired sync time (time= (PD_{13}) clocks or (PD_{13}) x 250 ns).

2) PD to Analog output, clear S5.

3) Load D from corresponding A's with first output point.

4) Wait for interrupt, S5. Interrupt occurs when convert counter reaches zero and the following has taken place:

   (a) A/D value, from each channel→each A/D output buffer.

   (b) Sample signal→all A/D's.

   (c) Transfer all D/A buffer A's to buffer B.

5) Load D/A's from corresponding PD locations.

6) Read desired A/D channels.

7) Test for interrupt (interrupt at this point indicates overrun).

8) Jump to step 4.

6. Signal and Control Lines from Analog Controller to Channels

16 PR bi-directional data lines
6 PR address lines (buffered PA)
1 PR load
1 PR convert
1 PR enable

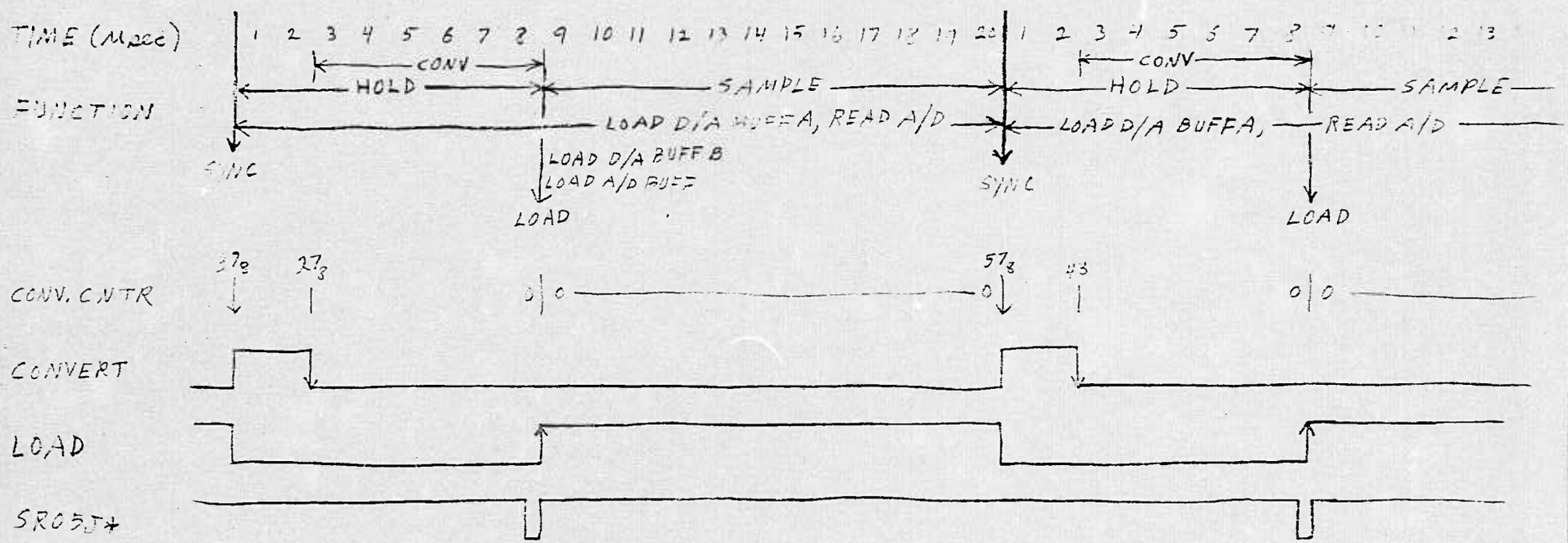
7. New Signal Paths Required Between MP and I/O Plane

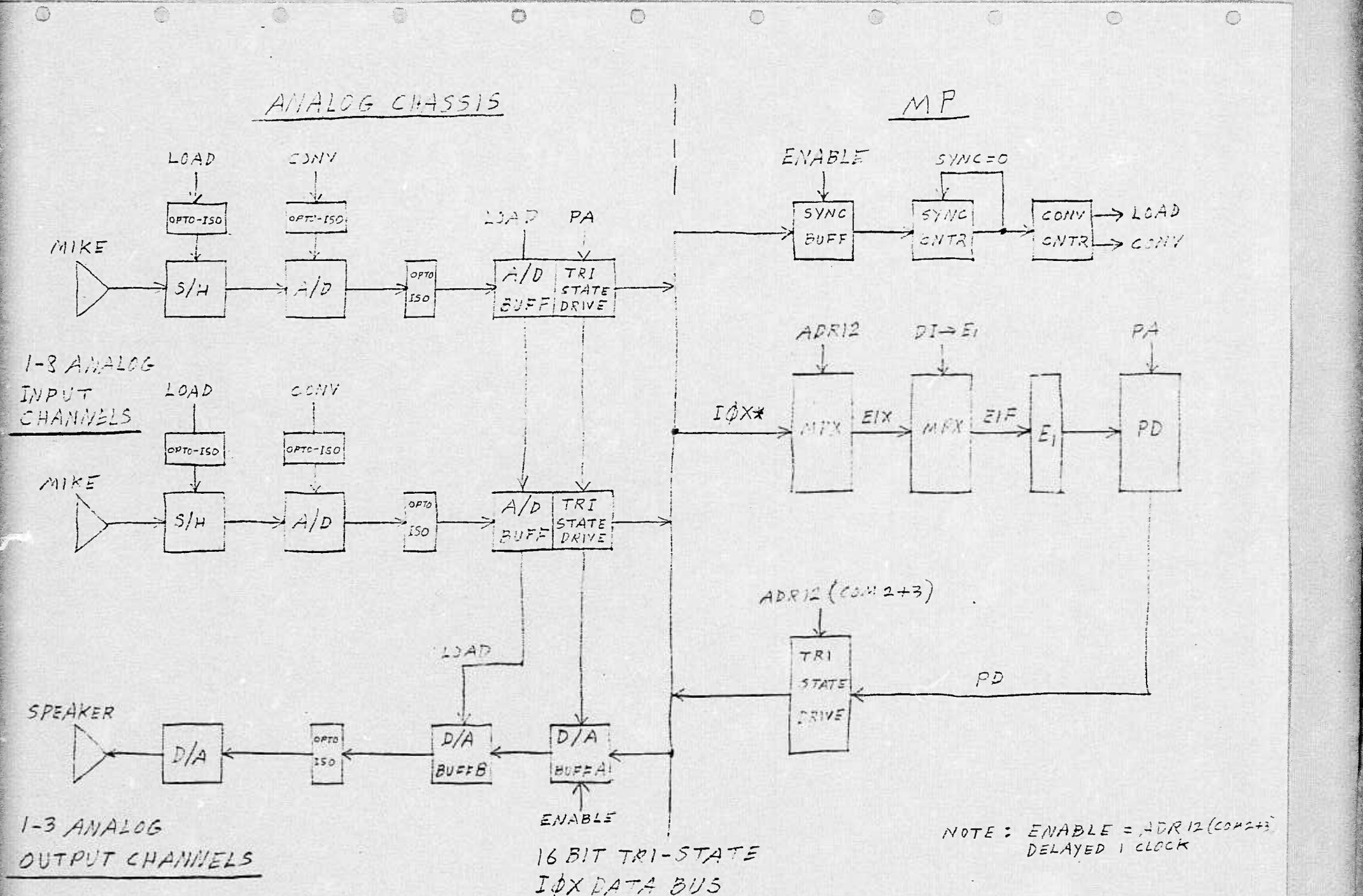
16 PR Data Pad
6 PR PA
1 PR SR05J*

8. Optically Coupled Isolators

To prevent ground loops and isolate the inherent digital noise from the analog system opto-isolators will be provided between: the A/D outputs and their respective buffer registers, the CONVERT line and the A/D's, the LOAD line and the Sample and Hold units, and the D/A's and their respective buffer B's. Two separate 5V power supplies -- one for digital and one for analog will be used.

ANALOG TIMING




ANALOG CHASSIS
MP
LOAD
CONV
ENABLE
SYNC=0
OPTO-ISO
MIKE
S/H
A/D
OPTO ISO
LOAD
PA
A/D BUFF
TRI STATE DRIVE
SYNC BUFF
SYNC CNTR
CONV CNTR
LOAD
CONV
1-8 ANALOG INPUT CHANNELS
ADR12
DI→EI
PA
IØX*
MPX
EIX
EIF
EI
PD
ADR12 (CONV 2+3)
TRI STATE DRIVE
SPEAKER
D/A
D/A BUFFB
D/A BUFFA
ENABLE
1-3 ANALOG OUTPUT CHANNELS
16 BIT TRI-STATE IØX DATA BUS
NOTE: ENABLE = ADR12 (CONV 2+3) DELAYED 1 CLOCK

APPENDIX 5

User Manual

CHI
SIGNAL SYSTEM

software
A2- USER MANUAL

CONTENTS:

1 INTRODUCTION

2 LIBRARY & DOCUMENT OPERATIONS

3 TEXT CONSTRUCTION & EDITING

4 MATHEMATICAL PROGRAMMING

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	12/19/73	Appendix C released.
B	3/28/74	pp. B-6,B-7,C-1,C-2
C	7/7/75	pp. Cover page,ii,iii,1-3,1-5,3-6,B-6,B-7,Chapter 4
D	10/22/75	pp. 1-9,1-10,1-11,2-4,3-2 thru 4-1,4-3,4-11,4-14,4-15, 4-16,A-1, 3-1,B-2 ,1-8,2-7 .

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29105

© 1973
by Culler/Harrison, Inc.

PREFACE

This manual is intended to provide the user of a CHI Signal Processing System with a description of the tools made available to him through the SIGNAL software system.

Included in this manual are descriptions of: the hardware and software features of the CHI Signal System, the Library and Edit sub-systems and the Mathematical Programming sub-system. Instructions for signing on the System are also included.

Other technical manuals which may be utilized for the development of maximally efficient execution sequences (when a process is well defined and its frequent use warrents it) are:

TMA3 Macro-programming Manual
TMA4 Micro-programming Manual
TMA8 AP Programming Manual

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No.
© 1973
by Culler/Harrison, Inc.

CONTENTS

Page

PREFACE ii

Chapter 1. INTRODUCTION 1-1

1.1 Overview 1-1
1.2 User Station 1-3
1.3 User/System Interaction 1-3
1.4 User Manual Conventions 1-6
1.5 Registration and Sign-on 1-7
1.6 System Operations 1-8

Chapter 2. LIBRARY AND DOCUMENT OPERATIONS 2-1

2.1 Introduction 2-1
2.2 Library Operations 2-2
2.3 Document Operations 2-5

Chapter 3. TEST CONSTRUCTION AND EDITING 3-1

3.1 Introduction 3-1
3.2 Text Buffer Operations (T Submode) 3-2
3.3 Edit Buffer Operations (E Submode) 3-4
3.4 Text Buffer/Edit Buffer Transfer Operations (T or E Submode) 3-8

Chapter 4. SIGNAL INTERACTIVE MATH SYSTEM 4-1

4.1 Introduction 4-1
4.2 Entering the Interactive Math System 4-1
4.3 Data Definition Statements 4-1
4.4 Data Transfer Statements 4-7
4.5 Mathematical Operation Statements 4-9
4.6 Integer Scalar Operations 4-13
4.7 Display Operations 4-14
4.8 Special Purpose Operators 4-15
4.9 Programming Keyboard Operations 4-16

FIGURES: 1. Structural Diagram of the Signal System 1-2
2. Schematic Layout of Typical Signal System 1-4
3. Schematic Layout of Signal System Keyboard 1-5

APPENDIX A. Operations Index A-1

APPENDIX B. Error Messages B-1

APPENDIX C. System Operators C-1

Chapter 1. INTRODUCTION

1.1 OVERVIEW

The SIGNAL SYSTEM is a combination of software and hardware specifically designed to provide an interactive signal processing facility with the following key features:

A. Software Features

- Independent controllers, one for each I/O unit

  1. Keyboard
  2. Display
  3. Analog
  4. Timer
  5. Disk

- Modular management programs organized as coordinated subsystems

  1. Operating
  2. Library
  3. Editing
  4. Mathematical
  5. Utility
  6. Diagnostic

- Multi level programming languages which permit the user to focus his efforts to enhance both his and the systems performance

  1. Basic system compiler (MP micro and macro language)
  2. AP micro assembler
  3. Program compiler
  4. Interactive interpreters

- Specialized user language tailored to the needs of interactive signal processing

  1. Signal processing operators
  2. Array process organization
  3. User operator extension
  4. User system definition

A structural diagram of the SIGNAL SYSTEM is shown in Figure 1.

B. Hardware Features

- Higher level micro-instruction set with multiple operations simultaneously executed.

- Multiple memories, simultaneously controlled by micro-instructions

- Multiple internal registers with simultaneous transfer

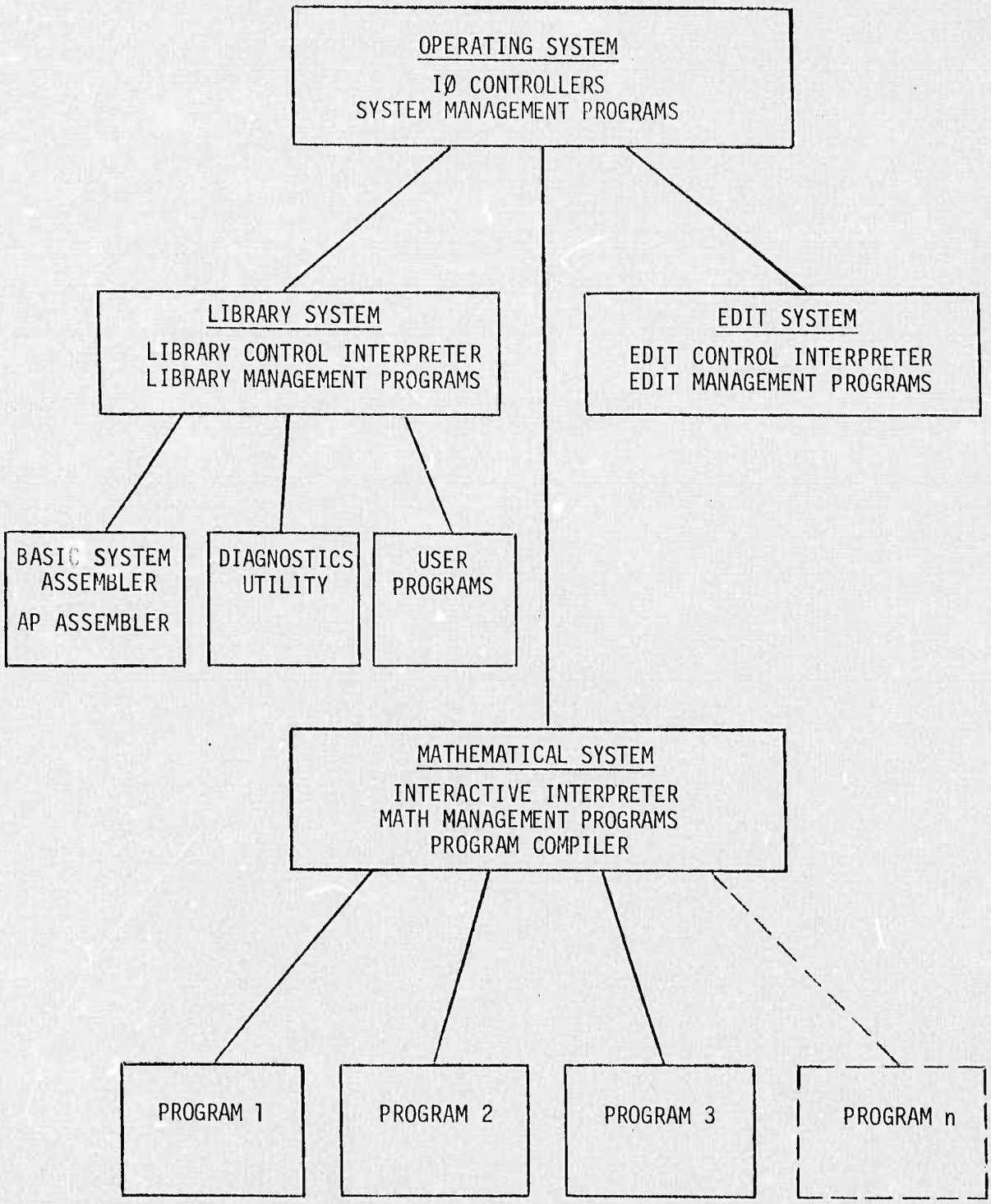


Figure 1. STRUCTURAL DIAGRAM OF THE SIGNAL SYSTEM

- Multiple program statement sources to minimize memory interactions for programs and data

- Simultaneous execution of MP and AP programs

- Firmware programs for arithmetical operations, mathematical functions and basic signal processes

A typical hardware configuration is shown in Figure 2.

This USERS MANUAL will explain how a signal process researcher may use the SIGNAL SYSTEM to develop his own processes, to experimentally test their performance and to interactively increase his repertoire of problem solving facilities.

When processes are well defined and when their utilization warrents the development of maximally efficient execution sequences, the reader is referred to manuals 3, 4, 8 of this software documentation sequence. There he will find all the information required to produce the desired programs.

If the macro programming language is sufficient for his purpose, then only manual 3 will be required. When this is not the case, he may elect to add his own macro instructions by producing new primitives in either the MP or AP or both; in this case mauals 4 and/or 8 may be used.

1.2 USER STATION

The Model 100 User Station consists of a console keyboard and display screen. The display screen offers the capability for character string display and graphical display of functions. Conceptually, it is divided into two areas; one for displaying the commands being issued by the user, and the other for displaying the material the user is working on. The size and location of these areas are controlled by the user.

The keyboard, shown schematically in Fig. 3, offers the standard "typewriter" keyboard, with upper and lower case alphanumeric and special characters. In addition, the keyboard contains an expanded set of control keys used to request system action. These keys are color-coded according to function as nearly as possible. In general, the black keys of the keyboard afford access to text editing and library operations. The blue keys are used to access a variety of mathematical and special-purpose functions. The gray keys denote the standard keyboard.

1.3 USER/SYSTEM INTERACTION

The SIGNAL System Software offers three main modes of operation to the user: Library Mode, Edit Mode, and Math Mode.

In the Library Mode, the user has access to units of information called libraries and documents. These he may create, access, display and delete using library operations. Libraries may contain text (character strings representing source programs or documentation), programs (executable and object programs), or data (sample data, fixed and floating point data, including real and complex values).

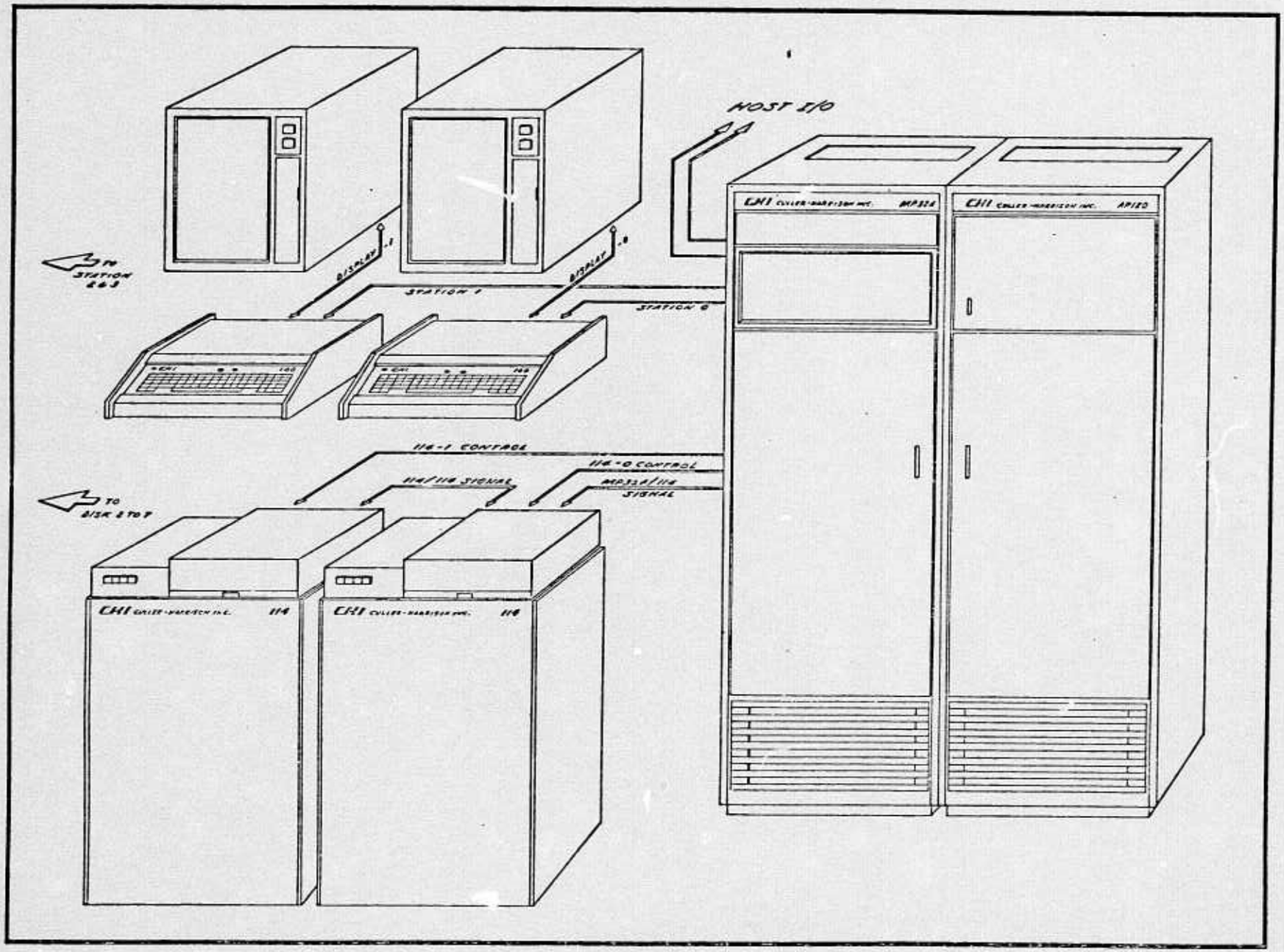


Figure 2. Schematic Layout of Typical SIGNAL SYSTEM

25%

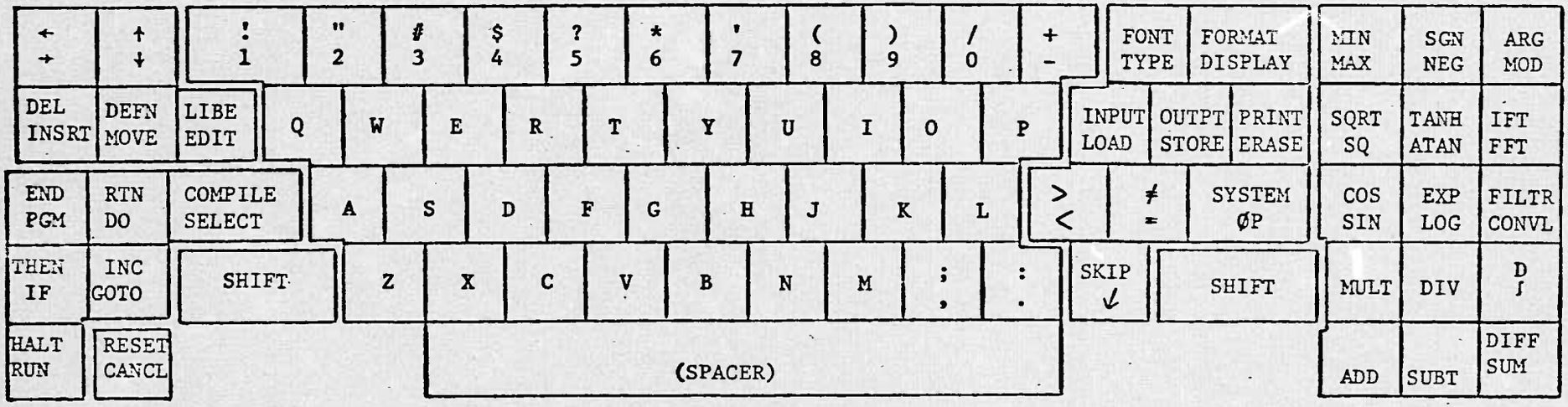


Figure 3. Schematic Layout of SIGNAL SYSTEM Keyboard

In the Edit Mode, the user has access to a formatted text buffer where he has the capability of creating, modifying, and deleting textual information using editing operations.

The Math Mode allows the user to perform the basic operations required for modern signal analysis. These operations may be controlled directly from the keyboard or by user programs using mathematical and display operations.

Chapter 2 of this user's manual deals with the operations available in the Library Mode. Chapter 3 discusses Edit Mode operations, and Chapter 4- describes the capabilities available to the user in the Math Mode.[1]

1.4 USER MANUAL CONVENTIONS

Several format conventions used in this manual are described below:

1. Control keys (the black, blue and red keys) will be shown enclosed in an oval or circle to distinguish them from alphanumeric characters. For example:

   (LIBE) (shift key + LIBE/EDIT key)

   (↵) (carriage return)

2. Underlines are used to denote gray-character keys which must be typed as shown to effect the desired result. For example:

   :P

   T

3. Lower-case letters with no underlines are used to indicate names supplied by the user. For example:

   libe-name

   entity

4. A "space", ◯, is represented by the underline character (_) and is mandatory where it appears.

5. Braces and vertical "stacking" are used to denote a choice of specifications, one of which must be selected. For example:

   { T / P / D }

6. Brackets are used to indicate optional specifications. For example:

   [:P]

1. Appendix A contains a summary of all operations.

Stacking may also be used here to indicate a choice of optional specifications. For example:



1.5 REGISTRATION AND SIGN-ON

1.5.1 Registration

The System provides facilities for up to 375 registered users. To register a new user, enter

(LIBE) (INSERT) (LIBE) user-name :U (↵)

Where user-name is from one to nine alphanumeric characters, the system will respond, "PASSWORD?"

Enter the desired password, which must be from one to nine alphanumeric characters, and (↵). This causes the System to enter the user's ID into the Directory of Users, and to create a Reference Library Directory for the User. The System then checks the directory each time an individual signs on to ensure he is a valid System user, and to locate his Reference Library Directory.

1.5.2 Sign-On

Once registered, the user may gain use of the System from any console by entering (↵). The system will respond "PLEASE IDENTIFY YOURSELF." The User's User Name should then be entered, followed by (↵). If the User Name is registered, the system will respond "WHAT IS YOUR PASSWORD?" The user should enter his password, followed by (↵). At this point, normal sign-on is complete and the System is left in Library Mode. However, if the last session at this console was terminated without Sign-Off by the same user, the System will offer the user a chance to recover his temporary libraries and text buffer contents from the last session. This will be indicated by the message, "ENTER R TO RECOVER OLD CONTEXT OR P TO PURGE IT." The user should respond, R (↵) to recover or P (↵) to start fresh. If either the User Name or Password entered are invalid, the system will respond "INVALID SIGNON."

1.5.3. Sign-Off

To indicate to the System that he wishes to terminate further action, the user enters:

(LIBE) (OP) SIGNOFF (↵).

The system will respond "ANY TEMP LIBES TO BE SAVED? ENTER N IF NOT." The user then should use N (↵). Any other response will abort signoff action and wait for next instruction.

1.6 SYSTEM OPERATIONS

A user may communicate directly with the operating system by using system operations. These operations are common to all interactive systems and permit the user to:

A. Manage his keyboard input ((CANCEL) , (RESET) , (↵))

B. Manage his display output ((FORMAT) , (ERASE))

C. Select an interactive system ((LIBE) , (EDIT) , (SYSTEM))

D. Initiate job execution ((OP) , (PGM))

1.6.1 Keyboard Input

When a key is depressed on the SIGNAL SYSTEM keyboard, an interrupt is posted in the MP-32 mainframe and the keycode is buffered in the user station interface. When this interrupt is serviced, the keycode with appended user station number is stored in a common input buffer and a KB-input service request is posted. The operating system flushes this buffer within 90 ms and distributes all keycodes into appropriate character string buffers, one associated with each user station. The name of the key depressed is then displayed in the echo section of the user's display field. The incomplete statement being constructed by the user includes all keycodes previously entered on the current echo display line. The last keycode in the incomplete statement may be cancelled by pressing (CANCEL) and this may be repeated, or the whole incomplete statement may be cancelled by (RESET). A statement is completed whenever (↵) is depressed. In that case, execution of the statement is initiated and its keycodes may no longer be cancelled.

1.6.2 Display Output

Each user's console has associated with it two typing display fields. One of these is an echo field used for displaying the names of keys as they are depressed; the other is used for general system typing and provides communication from the SIGNAL SYSTEM to the user. Both of these areas receive a standard definition when a user signs on. The keyboard echo area is currently five lines of up to 56 characters at the bottom of the screen. The system typing area is currently 56 lines of up to 80 characters occupying the upper part of the screen.

When the user wishes to clear the display tube he may press (ERASE) (↵).

When the (SKIP) character appears between words in any general system typing, the character cursor is advanced to the next tab stop setting as determined by the skip table for the console. The skip table entries are set by the (FORMAT) operation:

(FORMAT) $t_1, t_2, \ldots, t_n$.

Up to 15 tab stops can be specified in increasing sequence. The stop value in the number of character positions to the left of the tab stop. If tab stop values greater than the number of character positions on a line are specified, the line is "wrapped-around" onto the next line. However, a tab stop value may not exceed 255.

The user is able to change various other typing parameters from the keyboard. Below is a list of those parameters which can be changed and how to change them.

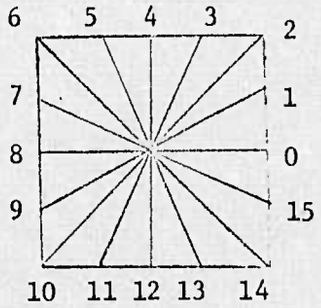
a. FØRMAT (a,b) — Locate upper left hand corner of page at (a,b) $0 \leq a \leq 4000$ $0 \leq b \leq 3800$

b. FØRMAT S=5 — Use polygonal segments with five overlapping points

c. FØRMAT W=3 — Use segments of width with three overlapping points

d. FØRMAT Q=2 — Use quill direction 2.

QUILL DIRECTIONS



e. FØRMAT C=64 — Use 64 characters per line

f. FØRMAT L=15 — Use 15 lines per page

g. FØRMAT DX=5 — Use 5 direction 0 segments for character frame width

h. FØRMAT DY=7 — Use 7 direction 4 segments for character heights

Any combination of the above FØRMAT arguments is allowed if they are separated by semi-colons.

If the entire list of arguments is preceded by K: or E: then the ensuing parameter changes will be made to the keyboard echo area or the edit buffer display area, respectively.

None of the display parameter changes are saved when a user signs off.

1.6.3 Interactive System Selection

The interactive operations of the SIGNAL SYSTEM have been grouped together according to their application. These groupings are called interactive systems. A user gains access to the operations in an interactive system by the following operations:

a. (LIBE) (/ select interactive operations from the library system

b. (EDIT) (/ select interactive operations from the editing system

c. (SYSTEM (/ select interactive operations from the current mathematical system

1.6.4 Job Execution Control

The SIGNAL SYSTEM provides facilities which permit a user to submit jobs for background execution and to be interactively connected to such jobs at selected checkpoints. In general, when a user is running in the interactive mathematical system, he has available for his use the operations (OP operator-name operand) compiled within that system and the programs (PGM program-name parameters) that have been constructed to run under that system. In addition, he has a repertoire of mathematical control and display operations;except for these, execution generally takes place as high priority background jobs. These jobs are automatically submitted and executed as a normal function of the operating system and, as such, do not require any job execution control by the user. The operations below pertain only to jobs specifically designed to run as background processes; usually, they will involve little or no interaction with the user. TMA3, the MP-32 MACRO PROGRAMMING MANUAL, describes how to design such jobs. This section covers their initiation.

Two methods of job initiation are provided, the OP and PGM facilities. The use of these facilities is described here only as they exist as system facilities, initiated from the Library System.

a. OPs

Jobs initiated with OP are generally system facilities, including diagnostic routines, utilities, and compilers. These jobs are resident on the system pack. To initiate execution of an OP , enter

(OP opname[_parameters] (/

The parameters for a particular OP are given with the description of that operation. A summary of the standard operators is given in Appendix C.

b. PGMs

Jobs initiated with PGM are fetched directly from the user's current program library (see section 3.3). They must be system modules prepared to execute as background jobs. To initiate execution of a PGM , enter

(PGM progname[, pad 0 value [, priority]] [_parameters]

The pad 0 value, if specified, is a non-negative decimal integer which will be placed in data pad cell 0 when the program is initiated. If not specified, data pad cell 0 will be 0. This value is available for use by the program. The priority is a number between 0 and 255 which will be used as the program's priority to determine the order in which it will be executed relative to other background jobs. If omitted, the priority will be specified as zero, the lowest priority. The job with the highest priority will be initiated first when more than one job is waiting to execute. The parameters are available to the background job for its own interpretation.

Chapter 2. LIBRARY AND DOCUMENT OPERATIONS

2.1 INTRODUCTION

These operations allow the user to access information categorized as text (character strings), data (numeric values), or programs (executable object code). Generally, the user deals with broad units of information called libraries. Within libraries, the user deals with more specific units of information called documents. Before describing the operations available to the user for the creation and modification of libraries and documents, we must define some general terms.

2.1.1 Shared and Reference Libraries

Two "main libraries" are at the user's disposal in creating and accessing information: the Shared Library and his own Reference Library.

The Shared Library provides permanent storage generally accessible to all users (although libraries within the Shared Library may be designated as accessible to only those users with the correct password). The user's own Reference Library provides permanent storage for his individual use.

These "main libraries" are made up of what may be thought of as "branch libraries." These branch libraries are organized according to contents (text, data, or programs) and contain entities called documents.

2.1.2 System Disk Facilities

The System Pack resides on one disk drive at all times; it contains all System Programs (the operating system, assembler, etc.), space for system-maintained directories (see Sec. 2.1.4), buffer areas (see Sec. 2.1.3), and library work space directly accessible by the user into which libraries may be copied for transfer or modification. The remaining disk drive(s) are used for mounting Reference Library Packs or Working Storage Packs. Reference Library Packs contain Reference Libraries and/or Shared Libraries. All such libraries may be physically located on the same disk pack, or on separate disk packs, if required by space limitations or the user's convenience. A Working Storage Pack is used for temporary mass data storage. The user is responsible for physically changing these packs as necessary to achieve the desired transfers. The SIGNAL SYSTEM will request mounting of these Packs as they are needed.

2.1.3 Text and Edit Buffers

Two buffers are maintained on the System Pack as user working areas for text construction and modification. In these buffers, the user can construct textual information which may form a document within a text library. He may also load text documents or pages from text documents into his text buffer for modification or combination with other documents.

The Edit Buffer is a working space which provides temporary storage for the user lines of text; here he may create and edit text in a line, word, and character structure. The Text Buffer provides an intermediate storage area between the Edit Buffer and his current text library; here he may create and modify text in a page and line structure.

For example, suppose the user wishes to modify a document in one of his text libraries. He first transfers the library from his Reference Library to the System Pack (Library MOVE Operation, Section 2.3.1). He may then load the document into the Text Buffer (Load Document Operation, Section 2.3.2). He may then select a portion of the document to be loaded into the edit buffer for modification (Text Editing Operations, Section 3).

The document operations in this chapter allow the user to transfer a specified document between his current library on the System Pack to and from his Text Buffer. The library operations allow the user to tranfer a specified library between his Reference Library and the System Pack.

Operations in the Edit Mode described in the next chapter allow the user to access portions of documents in the Text Buffer and Edit Buffer.

2.1.4 System Directories

To regulate user access to libraries and documents, the SIGNAL SYSTEM maintains four types of directories.

The Directory of Users contains a list of all recognized users of the interactive system. A user is entered in this directory when he is registered with the system (see Section 1.4). An entry in this directory contains the user's ID and a pointer to his library directory (see below).

A Reference Library Directory is created for each registered user of the system. It lists the names of the libraries accessible to the user and points to their location within the reference library. This directory resides on the System Pack during a users execution time.

A Shared Library Directory is maintained by the system, listing the names of all libraries in the Shared Library and their location on disk. This directory resides permanently on the System Pack.

One Document Directory exists for each library in the user's Reference Library and for each library in the Shared Library. It lists the names of all documents which comprise that library and points to their location on disk. The document directory for a particular library resides on the same physical disk pack as the library itself.

2.1.5 Library Mode Indication

(LIBE) (↓)

This command indicates that the user wishes to operate in the Library Mode; i.e., this operation allows the user to issue the library and document operations described in Sections 2.2 and 2.3.

2.2 LIBRARY OPERATIONS

These operations provide access to entire libraries of information; they provide for the transfer of information between the work area on the System Pack and a Reference Pack.

The "libe-name" parameter in these operations may consist of any combination of up to nine of the following characters:

A - Z upper case alphabetic characters

a - z lower case alphabetic characters

0 - 9 numeric characters

2.2.1 Move Library

This operation provides the means for transferring libraries as units.



The S option refers to the shared library, the R option refers to the users reference library. If no option is selected then the user's library on the System Pack is indicated. If a password was assigned for a library indicated in a move statement, then the System will request validation of that password before the move is permitted. It will request

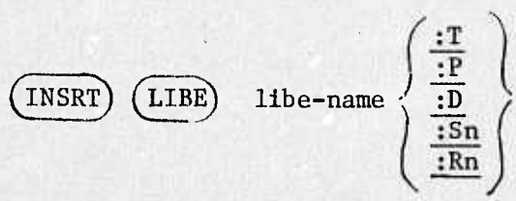
PASSWORD for libe-name?

and the user must type the necessary password, terminated by a carriage return. When all requested passwords have been supplied, the system will transfer the information as specified.

If the library named is not listed in the appropriate library directory, or if the user makes an invalid response to the password request, an error message is displayed (see Appendix B) and no transfer takes place.

2.2.2 Insert Library

This command indicates that the user wishes to define a new library. No actual transfer takes place with this operation; the system merely updates the appropriate library directory according to the user's specifications.



The S option causes the system to update the Shared Library Directory; the R option causes the system to update the user's Reference Library Directory on pack n. If option T, P or D is selected, then the System assumes the library is to be created on the System pack and generates an initial library structure of the type indicated (text, program or data). If option S or R is selected, the system next displays the message:

PASSWORD?

If no password is desired, the user simply presses the carriage return key. Otherwise, the user types the desired password, which may consist of up

to nine characters from the set:

A - Z upper case alphabetic characters

a - z lower case alphabetic characters

0 - 9 numeric characters

and terminates it with a carriage return.

2.2.3 Delete Library

With this operation, the user requests deletion of an entire library.

(DEL) (LIBE) libe-name [:R / :S] (↙)

The R option indicates that the named library is to be deleted from the user's Reference Library; the S option indicates the Shared Library. If neither is specified, the library is to be deleted from the System Pack.

If the named library is password protected, the system next displays the message:

PASSWORD?

The user must then respond with the correct password and a carriage return.

Before performing the actual deletion, the system gives the user a "second chance" by displaying the message:

ENTER Y TO DELETE { TEXT / PGM / DATA } LIBRARY libename

If he still wishes to delete the library, the user types Y, followed by a carriage return; anything other than a Y indicates that the user does not wish the deletion to occur.

After these queries and responses, the system deletes the named library from the appropriate library directory, and makes the space allocated to the library available for use.

If an invalid request is given, or if the user gives an invalid response to the "PASSWORD?" query, the system displays an error message (see Appendix B) and awaits another operation.

2.2.4 Display Library

This command is used to list the directory of documents in a given library, to list the names of the user's reference libraries, to list the names of all shared reference libraries or to list the user's System Pack libraries.

(DISPLAY) (LIBE) [libename] [:S / :R] (↙)

The S option refers to the shared library directory, the R option refers to the user's reference library directory. If no option is selected then the user's System Pack library directory is used. If a libename is specified, a listing of the names of all documents in that library, together with the track and record numbers of their header items, is displayed. If no libename is specified, the directory itself is listed.

If the library named is not listed in the appropriate library directory, an appropriate error message is displayed (see Appendix B).

2.3 DOCUMENT OPERATIONS

The operations described in the previous section provided access to information in units called libraries; transfers were allowed between two Reference Packs. The operations in this section provide access to documents within libraries. These operations deal with the transfer of information within the System Pack, and with the transfer of text documents between the library and the user's Text Buffer. The Text Buffer allows the user more direct access to his information; it provides an interface with the Edit Buffer via operations described in Chapter 3.

The "document-name" parameter in these operations may consist of any combination of up to nine of the following characters:

A - Z upper case alphabetic characters

a - z lower case alphabetic characters

0 - 9 numeric characters

2.3.1 Selection of Current Library

(LIBE) libe-name (↵)

This operation selects the "current" library of the specified type on the System Pack; this is the library on which ensuing document operations are to be performed. The Load and Store operations apply only to the current Text Library, whereas Insert, Delete, Move and Display apply to documents regardless of type.

2.3.2 Load Document

This operation requests the system to transfer a document or a portion of a document from the current text library to the user's Text Buffer.

(LOAD) document-name [:n_1, n_2, n_3] (↵)

The parameters n_1, n_2, and n_3 are positive integers defined as follows:

n_1 - specifies the starting page* for transfer from the library

*for definition of "page" see Section 3.1.

n_2 - specifies the starting destination page in the Text Buffer, if omitted transfer starts at page 1

n_3 - specifies the total number of pages to be transferred, if omitted all remaining pages are transferred

For example, the operation

(LOAD) MARY:3,5,2 (↵)

would cause a total of two pages to be transferred from the document MARY in the current library to the Text Buffer; pages 3 and 4 from MARY would be stored as pages 5 and 6 in the Text Buffer.

If this option is not used, the entire document is loaded into the Text Buffer.

If the document named does not exist in the current text library directory, or if the specified source or destination page does not exist, the system displays an error message (see Appendix B) and awaits further instruction.

2.3.3 Insert Document

(INSRT) document-name (↵)

This operation causes the system to add the document-name specified to the current document directory; no transfer of information between the library and Text Buffer takes place. This operation must be specified before any new document can be stored in the library.

If the document-name already exists in the current library directory, the system displays an error message (see Appendix B).

2.3.4 Store Document

(STORE) document-name (↵)

This operation stores the contents of the Text Buffer as an entire document in the user's current text library on the System Pack. The document-name must already exist within the document directory of the current text library (see Section 2.3.2) or an error message will result (see Appendix B).

With the STORE document operation, the system protects the user from inadvertently destroying useful information as follows: if information has previously been stored under this document name, it now becomes the "secondary document" under this name. The material currently being stored becomes the "primary document." All further document operations referring to this document name will reference the primary document, until the primary document has been deleted (see Section 2.3.5).

If a STORE operation is executed with a document name for which there already exists a primary and secondary document, (1) the contents of the text buffer are stored as the new primary document, (2) the old primary document becomes the new secondary document, and (3) the old secondary document is purged.

If the document named is not in the directory of the current text library, the system displays an error message (see Appendix B).

2.3.5 Delete Document

(DELETE) document-name [:E] (↵)

This operation deletes a document from the current library; two options are available.

Option 1: No E Specified. If the E option is not specified, the material stored under this document name is purged, but the entry for this document is left in the directory for the current library. If both a primary and secondary document exist under this name, DELETE purges the primary document and recovers the secondary document as the new primary document. In this case the message "BACK UP RESTORED" is displayed.

Option 2: E Specified. If the E option is specified the entry in the document directory for this document is removed. This means that whatever material is stored for this document is made permanently inaccessible and the name is no longer defined.

If the document named is not in the directory of the current library, the system displays an error message (see Appendix B).

2.3.6 Move Document

(MOVE) document-name$_1$, libe-name$_1$→document-name$_2$, libe-name$_2$ (↵)

This operation copies a document from one library to another on the System Pack; the document still remains within the source library.

The parameters document-name$_1$ and libe-name$_1$ define the source document and library. The parameters document-name$_2$ and libe-name$_2$ define the destination document and library.

The MOVE document operation does not alter the contents of the text buffer; the current library remains the same as it was defined before the MOVE operation.

If an invalid command is given, e.g., if either the source or destination documents are not consistently defined in the appropriate directories, the system issues an error message (see Appendix B).

2.3.7 DISPLAY

The library display operations are interpreted to have meaning appropriate to the type of current library. Displays of directories and symbol tables are ordered alphanumerically and are full page, 61 line outputs. If an option selection specifies the beginning, then it and one page of its successors are displayed. If the beginning is unspecified, the first page is displayed. For all three types of libraries (text, program and data) the following operation displays the current library's directory.

(DISPLAY) [:alphanumeric] (↵)

where the alphanumeric option specifies the leading character of the first name to be displayed.

a. Text libraries

(DISPLAY) document-name [:n] (↵)

This operation displays as many pages as will fit on the screen starting at page n.

b. Program libraries

1) SYSTEM documents

(DISPLAY) document-name [:alphanumeric] (↵)

This operation displays the symbol table of the document specified.

2) PGM documents

(DISPLAY) document-name (↵)

This operation displays compilation status information for the program specified.

c. Data libraries

(DISPLAY) data doc-name (↵) (↵)

This operation displays location on disk of the named variable.

Chapter 3. TEXT CONSTRUCTION AND EDITING

3.1 INTRODUCTION

In the SIGNAL SYSTEM, characters are encoded. The ASCII representation is used for alphanumerics, punctuation and logical symbols with operator symbols occurring in the octal range 200-277. The text data structure used distinguishes pages, lines and words.

A. A word is defined to be a string of alphanumeric characters followed by a single non-alphanumeric character and preceded by a character count. For example,

Stop it!

is two words listed as follows:

5Stop_	3it!

The alpha-numeric character string may be empty, in which case the character count is 1. The only exception to this is the special case of a string of blank (or space) characters, these are permitted to form a blank word with the number of blanks in the first byte followed by the succession of blanks, e.g.

3 _ _ _

B. A line is an ordered set of words whose last word has (↓) as its terminator. The pointer structure for a line has the form

$$n\ w_1\ w_2 \ldots w_n$$

where n is the number of words on a line and the w's are local displacements to the respective words.

C. A page is an ordered set of lines and as the pointer structure:

$$L\ m\ \ell_1\ \ell_2 \ldots \ell_m$$

where L is the total length (measured in 16-bit words) of the page, m is the number of lines of the page and the ℓ's are local displacements to the respective lines pointer structures.

The operations defined in this chapter allow the user to create and edit text, that is, character strings, within a text document. As mentioned in Section 3.1.3, two buffer areas for creating and editing text material are provided by the system: the Text Buffer and the Edit Buffer.

The Edit Buffer provides a working area for the creation and editing of material; the Text Buffer serves as an intermediate storage area between the Edit Buffer and the current text library.

3.1.1 Edit Mode Indication

(EDIT) (↵)

This operation indicates that the user wishes to perform operations in the Edit Mode.

3.1.2 Text and Edit Buffer Indication

In the EDIT system, the user may define whether ensuing operations are to access the Text or Edit Buffers by pressing T (for text) or E (for edit), followed by a carriage return.

3.2 TEXT BUFFER OPERATIONS (T Submode)

3.2.1 Display Text Buffer

(DISPLAY) [n] (↵)

This command displays one page of information from the Text Buffer. If the n option is specified, the n^{th} page is displayed (where n is a positive integer) and becomes the current page. Otherwise, the current page of the Text Buffer is displayed. If the specified page does not exist, an error message is displayed.

3.2.2 Set Pointer Operations

These commands allow the user to refer to a subset of information within the Text Buffer by setting pointers which define the first and last lines of the subset the user desires. The "top pointer" refers to the first line of the subset; it is displayed on the screen as ‾. The "bottom pointer" refers to the last line of the subset; it is displayed on the screen as _. The options available are listed below; the control keys ↑ and ↓ are used to indicate movement of the pointers "up" and "down", respectively.

(↑) / (↓) [n] (↵)

Moves both the top and bottom pointers n lines above or below the current position of the top pointer. Access is limited to the page currently being displayed. If n exceeds the number of lines available on the current page, the pointers are moved to the appropriate page extremity. If there are more lines on the page than in the display field, the display is organized to put the pointer in a central location within the display field.

(↑) / (↓) (/) [n] (↵)

Moves either the top pointer [indicated by "("] or the bottom pointer [indicated by ")"] n lines above or below its current position. Only the current page of the text buffer is accessed. If n exceeds the number of lines available, the first or last line is pointed to.

In either of the above cases, "n" may be replaced with the letter "s" to indicate "start" or "e" to indicate "end." In these cases, the appropriate pointer will be moved up to the start or down to the end of the text buffer. Otherwise, n is a positive integer. If not specified, n is assumed to be one.

In the last option, where top and bottom pointers are being moved separately, the top line pointer cannot be moved below the bottom line pointer. Likewise, the bottom line pointer cannot be moved above the top line pointer. Should such an operation be specified, both pointers are moved to the indicated position.

Below, n is a positive integer only.

(↑) (↓) P[n] (/)

Moves both the top and bottom pointers to point to the first line of the page, n pages preceding or following the current page. If n exceeds the number of pages available, the first or last page is accessed. The page indicated is displayed and becomes the current page.

Examples

Suppose the document being accessed contains three pages of information as follows:

```
Page 1, Line 1    THIS IS A WHIMSICAL
        Line 2    AND HOPEFULLY ENLIGHTENING
        Line 3    EXAMPLE OF THE WAYS
        Line 4    IN WHICH POINTERS
        Line 5    CAN BE SET
        Line 6    IN THE TEXT BUFFER.

Page 2, Line 1    ONCE THESE POINTERS
        Line 2    GET SET,
        Line 3    WE CAN BEGIN TO
        Line 4    MOVE THINGS AROUND.

Page 3, Line 1    I CAN TELL
        Line 2    YOU'RE REALLY
        Line 3    LOOKING FORWARD TO THAT!
```

A series of commands and their consequences are shown below.

(DISPLAY) 1 (/)

```
THIS IS A WHIMSICAL
AND HOPEFULLY ENLIGHTENING
EXAMPLE OF THE WAYS
IN WHICH POINTERS
CAN BE SET
IN THE TEXT BUFFER.
```

(↓) p1 (/)

```
ONCE THESE POINTERS
GET SET,
WE CAN BEGIN TO
MOVE THINGS AROUND.
```

(↓))2 (/)

```
ONCE THESE POINTERS
GET SET,
WE CAN BEGIN TO
MOVE THINGS AROUND.
```

(↓) (e (/)

```
ONCE THESE POINTERS
GET SET,
WE CAN BEGIN TO
MOVE THINGS AROUND.
```

3.2.3 Deletion in the Text Buffer

(DEL) (/)

This command deletes the portion of the Text Buffer between the top and bottom pointers.

For example, suppose the Text Buffer pointers are set as follows on the current page:

```
INFORMATION CAN BE STRICKEN,
SUCH AS:
QUOTH THE RAVEN,
NEVERMORE!
FASCINATING, WHAT?
```

The operating (DEL) (/) would have the following effect:

```
INFORMATION CAN BE STRICKEN,
SUCH AS:
FASCINATING, WHAT?
```

Note that the top and bottom pointers now bracket a null line.

3.3 EDIT BUFFER OPERATIONS (E Submode)

These operations deal with entering and deleting information in the Edit Buffer. To perform these operations, the user must, of course, be in EDIT mode, and must have indicated by typing E followed by a carriage return that he wishes to perform operations in the Edit Buffer.

3.3.1 Display Edit Buffer

(DISPLAY) (/)

This operation displays the current contents of the Edit Buffer.

3.3.2 Set Caret Operations

These commands allow the user to refer to a subset of information within the Edit Buffer by setting a pointer which defines the first character of the subset. The pointer is called a caret; it is displayed on the screen as ^. The options available are listed below. The control keys (←) and (→) indicate movement to the left or right; (↑) and (↓) indicate movement up and down.

(→) / (←) [n] (/) — Moves the caret n characters to the right or left, but not beyond the line the caret is currently in. If n exceeds the number of characters available in the line, the caret is positioned at the rightmost or leftmost character.

← s ↵
→ e

Moves the caret to the first or last character of the line in which the caret is currently positioned.

←
w[n] ↵
→

Moves the caret n words to the right or left, but not beyond the line the caret is currently in. The caret is positioned at the first character of the word.

← w ↵

Moves the caret to the first character of the word in which it is currently positioned.

↑
[n] ↵
↓

Moves the caret up or down n lines, to point to the first character of the specified line. If n exceeds the number of lines available, the first or last line in the Edit Buffer is accessed.

↓ e ↵

Moves the caret to the first character of the last line in the Edit Buffer.

↑ s ↵

Moves the caret to the first character in the Edit Buffer.

Example 3.1

Suppose the Edit Buffer contains the following:

```
Line 1: ^UNDER A SPREADING
     2:  CHESTNUT TREE
     3:  THE CITY SMITHY
     4:  STANDS
     5:  THE SMITH A MIGHTY
```

A series of operations and their cumulative effect are described below:

→ 5 ↵

```
UNDER^ A SPREADING
CHESTNUT TREE
THE CITY SMITHY
STANDS
THE SMITH A MIGHTY
```

↓ 3 ↵

```
UNDER A SPREADING
CHESTNUT TREE
THE CITY SMITHY
^STANDS
^THE SMITH A MIGHTY
```

↑ s ↵

```
^UNDER A SPREADING
^CHESTNUT TREE
THE CITY SMITHY
STANDS
THE SMITH A MIGHTY
```

→ w2 ↵

```
UNDER A ^SPREADING
CHESTNUT TREE
THE CITY SMITHY
STANDS
THE SMITH A MIGHTY
```

(↓) e →e (↵)

UNDER A SPREADING
CHESTNUT TREE
THE CITY SMITHY
STANDS
THE SMITH A MIGHTY^

3.3.3 Insert to Edit Buffer

(INSERT) text-material (INSERT) (↵)

This command causes the typed text-material to be inserted in the Edit Buffer at the place where the caret is currently positioned. The new contents of the Edit Buffer are then automatically displayed.

Example 3.2

Suppose the Edit Buffer contains the material noted in Example 3.1, and that the caret is positioned as after the last command in that example. Suppose further that the user issues the following commands:

(INSERT) (↵) MAN IS HE (↵) WITH LARGE (↵) AND SINEWY HANDS (INSERT) (↵)

(↑ 5 (↵)

(→ 3 (↵)

INSERT_VILLAGE INSERT (↵)

The new contents of the edit buffer would be displayed as follows:

UNDER A SPREADING
CHESTNUT TREE
THE VILLAGE^ CITY SMITHY
STANDS
THE SMITH A MIGHTY
MAN IS HE
WITH LARGE
AND SINEWY HANDS

3.3.4 Delete from Edit Buffer

(DELETE) entity (↵)

This operation allows the user to delete a specified portion of the Edit Buffer.

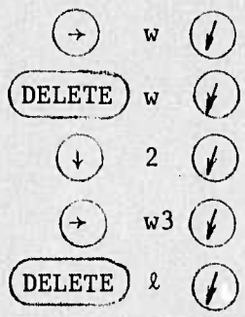
The entity specified defines the amount of the Edit Buffer to be deleted; the options available are as follows:

n	Specifies the number of characters to be deleted, beginning with the character at which the caret is currently pointing.

wn	Specifies the number of words to be deleted, beginning with the word at which the caret is currently positioned. The carriage return is not deleted with this option. If a whole line is deleted, its position in the text will remain, creating a "skipped" line consisting of the line terminator character (carriage return).
e	Specifies that the remainder of the line in which the caret is currently positioned is to be deleted, and its position in the text to remain.
ℓn	Specifies the number of lines to be deleted, beginning with the line in which the caret is currently positioned. (ℓ denotes lowercase letter ℓ).
ℓ	Specifies that the remainder of the line in which the caret is currently positioned is to be deleted and replaced by the next line.
b	Specifies that the entire contents of the Edit Buffer following the caret is to be deleted.

Example 3.3

Suppose that the Edit Buffer contains the material noted in Example 3.2, with the caret positioned as shown at the end of that example. Suppose further that the user issues the following commands:



The new contents of the Edit Buffer would be displayed as follows:

```
UNDER A SPREADING
CHESTNUT TREE
THE VILLAGE SMITHY
STANDS
THE SMITH A^MAN IS HE
WITH LARGE
AND SINEWY HANDS
```

3.4 TEXT BUFFER/EDIT BUFFER TRANSFER OPERATIONS (T or E Submode)

These operations allow the user to transfer information between the Text Buffer and the Edit Buffer. They may be called from either the T or E submode within the Edit Mode.

3.4.1 Load Edit Buffer



This operation loads the contents of the Text Buffer between the top and bottom pointers into the Edit Buffer and displays the Edit Buffer. The old contents of the Edit Buffer are destroyed. This operation leaves the user in the E submode (i.e., further Edit Mode operations will be assumed to deal with the edit buffer).

3.4.2 Store Text Buffer



This operation stores the entire contents of the Edit Buffer into the Text buffer at the position of the top pointer, and displays the contents of the Text Buffer.

This operation leaves the user in the T submode (i.e., further Edit Mode operations will be assumed to deal with the Text Buffer).

Example 3.4

Suppose the Text Buffer contains one page of information, with pointers positioned as shown:

```
THERE MUST BE SOME EXAMPLE
WHERE A LINE
WOULD MAKE SENSE WHEREVER
IT WAS PUT
```

Suppose further that the Edit Buffer contains one line of information:

```
(BUT I CAN'T THINK OF ONE)
```

The operation



would cause the contents of the Text Buffer to be modified as follows:

```
Page 1      THERE MUST BE SOME EXAMPLE
            (BUT I CAN'T THINK OF ONE)
            WHERE A LINE
            WOULD MAKE SENSE WHEREVER
            IT WAS PUT
```

Chapter 4. MATHEMATICAL PROGRAMMING

4.1 INTRODUCTION

A preliminary version of the Signal Interactive Mathematical System is available for use. This system is limited to single operations specified as transformations of declared data arrays and the plotting or printing of these arrays. No arithmetic expressions are supported. The operations available are the elementary functions and operations on floating point real data and the FFT and IFT on complex fixed point data.

4.2 ENTERING THE INTERACTIVE MATH SYSTEM

To enter the Interactive Math System from the Library System, declare a current data library, then push (SYST) (↵). The response to this will be the message: READY, displayed on the screen. To return to other system activities from the Interactive Math System, push (LIBE) or (EDIT) and (↵).

4.3 DATA DEFINITION STATEMENTS

There are two data definition operators, (DEFN) and (SELECT). (DEFN) is used to define arrays of one or more dimensions, indexes, or disk variables. (SELECT) defines scalar variables and assigns an initial value to them.

A. Declare a simple variable (SELECT)

The format of this operator is:

(SELECT) var-name: type code = value

The allowable data types and their codes are:

Type	Format	Type-Code
Integer	16-bit integer	I
Real	32-bit floating point	R
Complex Floating	2 32-bit floating point	C

If no type code is entered the data type will default to be Real.

Data variable names are from one to nine alphanumeric characters long; the first character must be alphabetic.

Value is a decimal specification of the desired initial value for the variable. The allowed forms of value for each type code are:

Type Code	Value
I	[+] n where n is an integer $0 \leq n \leq 32767$
R	[+] d ([E][+]e) where d is a decimal number, which may include a decimal point and e is an integer.
C	a pair of values as in R, separated by a,

A simple variable may also be defined in terms of an already existing variable:

SELECT var-name = { var-name / var-name[↓subscript list↑] }

The type of new variable will be the same as the type of the old variable. The value of the new variable will be the value of the old variable, or element, at the time the (SELECT) statement is made. A subscript may be used whenever the old variable is an array; it is specified as a set of integers, separated by commas. The first element of an array (in each dimension) has a 0 subscript (in that dimension). If the old variable is an array and no subscript list is given, the first element of the array is used as the source.

Several simple variables may be defined in one (SELECT) operation, each new variable name is separated from the previous definition by a single space.

Examples:

Valid (SELECT) statements

(SELECT) JOE:I=-37_hot:R=3.7E-5_cold=-3.7E-5 (↵)

(SELECT) j:I=0_h23=hot_LAST=BIG↓15↑ (↵)

Invalid (SELECT) statements

(SELECT) Structured:I=5 (↵) "Name too long

(SELECT) Harry:c=j (↵) "No new type-code permitted

B. Declare an Array - (DEFN)

The format of this operator is:

(DEFN) var-name:type-code=dim_1[,dim_2,...]

The data types currently supported for array variables are:

Type	Format	Type-Code
Scaled Integer	16-bit integer/entry with power of 2 scale	I
Scaled Complex	2 16-bit integers/entry with power of 2 scale pair	C
Real	32-bit floating point /entry	R
Complex Floating	2 32-bit floating point values/entry	C

If no type code is entered the array will default to be Real.

The dim_i are positive decimal integer values specifying the number of elements in each dimension. Up to seven dimensions may be specified. This operation only sets aside space for an array, it does not initialize it in any way.

For scaled lists, the first component (index 0) is assumed to be the scale, thus, scaled arrays are normally specified with n+1 components, where n is the number of data values.

An array may also be defined in terms of an existing array:

(DEFN) var-name[:type code]=var-name[↓subscrip list↑];dim_1[,dim_2,...]

If no :type-code is given, the type of the new array will be the same as the type of the old array. The initial point (index (0,0,...) of the new array will be the point selected by the subscript list. If the subscript list is omitted, the initial points of both arrays will be the same. The dimensions of the new array need not have any relationship to those of the old array, except that the entire new array must be a subset of the old array. Unlike the relative definition of simple variables, an array variable defined in this way does not create a copy of the old array variable's data, it is a re-description of some of the same data space. Any change in the new array, or the common subset of the old array, affects both variables.

Several array variables may be declared in a single (DEFN) operation. Each new variable name is separated from the prior declaration by a single space.

Once declared, an array variable which has its own storage allocation cannot be altered. Arrays declared relative to other arrays can be re-defined at any time, but must not change in either type or number of dimensions.

Each absolute array declaration results in an allocation of MOS memory to hold that array. This allocation is taken from a pool of about 28,000 16-bit words. Since all allocated memory must be saved on disk whenever swapping is necessary, service time may be noticeably worse when several large arrays are declared. An attempt to describe an array for which there is insufficient space results in an error message.

Examples:

Valid (DEFN) statements:

(DEFN) Alpha:R=3000_Beta=A;1000_Gamma=1000 (↵)

Array Beta is another name for the first 1000 elements of Alpha; both are real arrays. Gamma is another, separate, real array containing 1000 elements.

(DEFN) S:c=4097_D:I=500_Delta=S↓100↑;1_IS:I=S;4097,2

Array Delta is another name for the 101th element of S. Unlike a simple variable, Delta's value will always be the same as S_{100}, no matter how S or Delta are transformed, unless Delta is redescribed. Array IS has 2 columns, one containing the real parts of array S and the other containing the imaginary parts.

Invalid (DEFN) statements:

(DEFN) C=Alpha↓4090↑ (↵)

Array Alpha has only 3000 elements; also, no length was specified for C.

(DEFN) A:c=512 A:R=1000, B=A↓500↑;12 (↵)

The second declaration of A is a duplicate definition, thus not permitted. The comma (,) before B is not a valid delimiter. The array B extends beyond the extent of A by one element.

C. Define a Disk Variable (DEFN)

The format for this operation is:

(DEFN) var-name:D=#tracks

The number of tracks specified are allocated, but are not initialized in any way. Each track can hold up to 3600 16-bit words. No format is assumed for data written on disk. Disk variables defined in this way and the disk tracks associated with them form the user's current data library. When the math system is entered, all variables from the current data library directory are automatically defined.

Disk variables may also be defined relative to existing disk variables:

(DEFN) var-name=disk-var-name[↓subscript list↑];dim$_1$[,dim$_2$,...]

These relative disk variables may have multiple dimensions. Each entry is a disk track. As is the case for other types of relative definitions, the new variable must be a subset of the existing disk variable. Relative disk variable definitions are not preserved in the data library.

Example:

(DEFN) source:D=10_Fetch=source;5,2 (↵)

Source will be added to the current data library directory with an allocation of 10 tracks. Fetch is an alternate description of the same 10 tracks as five rows of two tracks each. Fetch is not placed in the data library directory.

D. Define an Index Variable (DEFN)

An index variable is defined by:

(DEFN) var-name:n=[starting value,] length[;increment]

An index variable functions as an integer in subscript expressions. The value of this integer is its current value. The current value is initially equal to the starting value (0 if no starting value is given). The current value is altered by the (INC) operation, which adds the increment (whose default value is 1) to the current value if the result will be less than starting value+ length, in which case the current value is unchanged.

Examples:

(DEFN) i:n=32 "i will assume values 0,1,...,31

(DEFN) j:n=5,33 "j will assume values 5,6,...,37

(DEFN) skip:n=32;5 "skip will assume values 0,5,10...,30

E. Variable References

A one-dimensional subset of any array variable may be selected for use in most of the arithmetic operations. These subsets are chosen by specifying for each index whether it is to be a given fixed value or is to range over all defined values. The latter indexes will be called free indexes. The only restriction on the choice of free indexes is that they must be contiguous. That is, it would not be permissible to specify that the first and third indexes were free while the second index was fixed.

Within the selected subset, processing will occur in lexicographic order. That is, the last free index will increase most rapidly, whenever a free index reaches its limits, it is reset to 0 and the next free index is incremented.

The array subset is selected by giving the array name followed by a subscript expression which begins with '↓' and is terminated by '↑'. Fixed indexes are specified using simple integer arithmetic (integer or index variables, literal integer values, with + or - operations). Free indexes are specified by entering an '*'. Each index specification except the last given is followed by a ','. If all indexes are not specified before the terminating '↑', the unspecified indexes are assumed to be free.

Examples:

Assume an array A has dimensions 2,3,4. Then the following specifications select elements from A in the order shown:

A $A_{0,0,0}$, $A_{0,0,1}$, $A_{0,0,2}$, $A_{0,0,3}$, $A_{0,1,0}$, $A_{0,1,1}$, $A_{0,1,2}$, $A_{0,1,3}$,
$A_{0,2,0}$, $A_{0,2,1}$, $A_{0,2,2}$, $A_{0,2,3}$, $A_{1,0,0}$, $A_{1,0,1}$, $A_{1,0,2}$, $A_{1,0,3}$,
$A_{1,1,0}$, $A_{1,1,1}$, $A_{1,1,2}$, $A_{1,1,3}$, $A_{1,2,0}$, $A_{1,2,1}$, $A_{1,2,2}$, $A_{1,2,3}$,

A↓0↑=A↓0,*↑=A↓0,*,*↑ $A_{0,0,0}$, $A_{0,0,1}$, $A_{0,0,2}$, $A_{0,0,3}$, $A_{0,1,0}$, $A_{0,1,1}$,
$A_{0,1,2}$, $A_{0,1,3}$, $A_{0,2,0}$, $A_{0,2,1}$, $A_{0,2,2}$, $A_{0,2,3}$,

A↓1,2↑=A↓1,2,*↑ $A_{1,2,0}$, $A_{1,2,1}$, $A_{1,2,2}$, $A_{1,2,3}$

A↓*,0,3↑ $A_{0,0,3}$, $A_{1,0,3}$

A↓0,*,1↑ $A_{0,0,1}$, $A_{0,1,1}$, $A_{0,2,1}$

All variables will be preserved when leaving the math system and returning. Their definitions and values will remain unchanged, even if the user signs off (saving his context) and signs back on again (recovering).

F. Delete a Variable (DEL)

The format of this operator is:

(DEL) var-name

All variables may be deleted from the data library except for disk variables. An attempt to delete a disk variable will result in a diagnostic.

Several variables may be deleted with one (DEL) statement by separating the variable names with a space.

It is possible to delete all variables except non-relative disk variables at once by pushing the button sequence (SYST) ↓ .

Examples:

Valid (DEL) statements:

(DEFN) A:R=4000_B:I=200_C=A

(DEL) A

(DEL) B_C

Invalid DEL statements

(DEFN) disk:D=10

(DEL) disk

Since disk is a disk descriptor, it may not be deleted.

4.4 DATA TRANSFER STATEMENTS

Transfer of data between one variable and another, including transfers between disk variables and MOS memory variables, is accomplished using the MOVE operator.

A. Transfers between disk and memory

These transfers do no format conversion, but simply transfer the data as unformatted 16-bit words. Data is transferred to or from contiguous memory locations, regardless of how the memory variable is specified. Only the first element address and displacement from the first element selected to the last element selected are relevant. The disk variable may be fully qualified to select a one-dimensional subset.

Data will be written on disk in 3600 16-bit word tracks, except for the last track, which may not be full. It is not possible to begin transfers in the middle of a track. The transfer ends when either variable is exhausted.

Examples:

(DEFN) A:R=2000_disk:D=10_d2=disk;5,2 (↓)

(MOVE) A(→)disk

Transfers 4000 16-bit words from A to disk. (Each entry of a Real list is two 16-bit words.) The first track of disk would receive 3600 words. The second track of disk would receive 400 words. Nothing would be written on the remaining 8 tracks.

(MOVE) disk↓0↑→A "3600 16-bit words would be transferred.

(MOVE) d2↓0↑→A "4000 16-bit words would be transferred.

B. Memory to Memory Transfers

(MOVE) var_1(→)var_2[;scale]

These transfer operations permit conversion of scaled integer variables to real variables and vise versa, as well as initializing variables to given values.

If var_1 is a scaled integer variable, the first element of the selected subset of var_1 is assumed to be its binary scale. If var_2 is real, the conversion takes place by converting each remaining element of var_1 to floating point, incorporating the scale, and transferring it to var_2. Note that in this case, var_2 should have one less element than var_1.

If var_1 is real and var_2 is scaled integer, the conversion takes place in the opposite direction. In this case, a scale factor must be specified which gives the binary scale to be associated with each element of the integer variable. This scale is also stored as the first element of the list. If no scale is given, a scale of 0 is assumed, and the list elements will be the integer parts of the floating point list elements. In this case, var_1 should have one more element than var_2.

If var_1 and var_2 are the same type, no conversion or scaling is performed. If var_1 is entered as a literal value, it is treated as a floating point variable with length greater than the length of the destination variable.

If either var_1 or var_2 is a scaled complex list, the move may be restricted to either the real components or imaginary components by qualifying the variable name with :r or :i.

Multiple move operations may be performed on the same line of input by separating the different move operations by a space.

Examples:

(DEFN) A:R=2000 L:I=2001

(MOVE) 10 → L "Each element of L except the first is set to integer 10, the scale is 0.

(MOVE) L → A_A → L;-5 "$A_i = L_{i+1} * 2^{L_o}$, then "L_o=-5

"L_i=Integer part($A_{i-1}/2^{-5}$)i=1,2000

(DEFN) sig:c=2001 B:R=2000,2

(MOVE) sig → B "transfer real parts of sig into "B↓*,0↑,imaginary parts into B↓*,1↑.

(MOVE) A → sig:i;-15 "transfers into imaginary parts of sig only

(MOVE) B↓*,1↑ Sig:r;-15 "transfers second column of B into "real parts of sig.

4.5 MATHEMATICAL OPERATION STATEMENTS

There are three types of mathematical operations: unary floating point operations, binary floating point operations and fixed point operations. The format of each type is given below; the allowable operators of each type are given in tables 1, 2 and 3. Multiple mathematical operations can be specified in one statement by separating each by a single space.

A. Unary Floating Point Operations

The general format of these operations is:

(op) var_1[→ var_2]

(op) may be any of the operators given in Table 1. Var_1 and var_2 must be real floating point variables, either arrays or scalar variables. Each variable is considered to be a one-dimensional array, with scalar variables treated as having one element. The operation is carried out to produce as many results as the dimensions of var_2, with the last (or only) element of var_1 being used repeatedly when needed to provide enough components. In addition, a literal value may be specified for var_1; it is treated like a scalar variable.

The rules for unary op arguments may be thought of as follows: there is an input argument and an output argument, either or both of which may be omitted. If the input argument is omitted, the input will be taken to be the previous output argument. If no output argument is specified, the input argument will be used as the output argument, unless the input argument was a literal value, in which case an error message will occur.

In summary, the following cases are allowed with the indicated results:

(op) $var_1 \rightarrow var_2$	The op applied to var_1 will be stored in var_2
(op) var_1	The op applied to var_1 will be returned to var_1
(op) $\rightarrow var_2$	The op applied to the previous output will be stored in var_2
(op)	The op applied to the previous output will be returned to the previous argument.

Table 1. Unary Floating Point Operators

In this table, x_i is the ith (or last) component of var, Y_i is the ith component of var_2.

Operator Name	Operation
(SQ)	$x_i * x_i \; Y_i$
(SQRT)	$\sqrt{x_i} \rightarrow Y_i$ if $x_i > 0$; $0 \rightarrow Y_i$ otherwise
(LOG)	$\ln(\lvert x_i \rvert) \rightarrow Y_i$
(EXP)	$\exp(x_i) \rightarrow Y_i$
(SIN)	$\text{sine}(x_i) \rightarrow Y_i$
(COS)	$\text{cosine}(x_i) \rightarrow Y_i$
(ATAN)	$\text{arctangent}(x_i) \rightarrow Y_i \; (-\pi/4 < Y_i \leq \pi/4)$
(NEG)	$-x_i \rightarrow Y_i$
(SGN)	$Y_i = \begin{cases} -1 \text{ if } x_i < 0 \\ 0 \text{ if } x_i = 0 \\ 1 \text{ if } x_i > 0 \end{cases}$

Table 1 (continued)

Operator Name	Operation
(MOD)	$\lvert x_i \rvert \rightarrow Y_i$
(DIFF)	$\lvert x_i - x_{i-1} \rvert \rightarrow Y_i, i>0$
(SUM)	$\sum_{j=0}^{i} x_j \rightarrow Y_i$
(MAX)	$\text{MAX}\ (x_j) \rightarrow Y_i$ $j=0,\ldots,i-1$
(MIN)	$\text{MIN}\ (x_j) \rightarrow Y_i$ $j=0,\ldots,i-1$

Examples: In these examples, assume all variables beginning with lower case letters are simple real variables and all variables beginning with upper case alphabetics are real array variables.

(SIN) ALPHA (→) ALPHA (↓) "ALPHA=SIN(ALPHA)

(SUM) 5-3 (→) LIST (↓) "$\text{LIST}_i = \sum_{j=0}^{i} 2$

(SIN) LIST (→) B_ SQ COS LIST (→) C_ SQ (↓) "$B=(\text{SIN}(\text{LIST}))^2$ $C=(\text{COS}(\text{LIST}))^2$

(EXP) one (→) e_ SUM e (→) Big_ LOG Big (→) Count (↓) "$\text{Count}_i=1+\ell n(i)$

B. Binary Floating Point Operations

The general format of these operations is:

(op) var_1,var_2[(→) var_3]

The variables named must all be real floating point; however, both var_1 and var_2 may be specified as literal values. Again, the number of elements in var determines the number of computed results. The last component of either var_1 or var_2 will be used as the value for that operand when necessary. (op) may be any of the operators given in Table 2.

The rules for binary op arguments are exactly the same as for unary ops, where the "input argument" is the first of the two arguments and the "output argument"is the destination argument. There must be at least one argument specified for all binary ops.

The following cases are allowed with the indicated results, using subt as a typical binary operator:

(subt) Arg1,Arg2→Arg3	"Arg3 = Arg1-Arg2
(subt) Arg1,Arg2	"Arg1 = Arg1-Arg2
(subt) Arg2→Arg3	"Arg3 = Previous output argument-Arg2
(subt) Arg2	"Previous output argument = Previous output argument-Arg2

Table 2. Binary Floating Point Operators

In this table, x_i and y_i are the ith (or last) components of var_1 and var_2 respectively. Z_i is the ith component of var_3.

Operator Name	Operation
(add)	$x_i+Y_i \rightarrow Z_i$
(subt)	$x_i-Y_i \rightarrow Z_i$
(mult)	$x_i*Y_i \rightarrow Z_i$
(div)	$x_i*Y_i^{-1} \rightarrow Z_i$

Examples:

(add) Y,1 (→) Z (↵)

(mult) Big,a (→) Biga (↵)

(div) Z,Biga_ (add) 3 (↵) $Z = \left(Z/Biga\right) + 3$

C. Fixed Point Operations

Two fixed point transform operations are presently defined. The FFT of either real or complex data may be calculated. The Inverse FFT of a complex fixed array may be calculated.

1. FFT

(FFT) n,m,var_1[:r] (→) var_2 (↵)

n is the $\log_2$ of the number of elements in each block to be transformed.

m is the $\log_4$ of the total number of points (normally the length of var_1•$0<n\leq 2m, 0<m\leq 6$.

var_1 is a scaled complex array
var_2 is a scaled complex array

The FFT of one or more blocks of data contained in var_1 is calculated. The results are returned in similar blocks of var_2. The real and imaginary parts of the first element of var_2 are set to the real scale of var_1 plus the change in scale from the FFT.

If :r is specified, the real and imaginary parts of each block of var_1 are treated as separate real arrays. The transform of these arrays is returned as a complex list of length one-half the specified blocksize in each block of var_2. The output for the real components precedes that for the complex components in each block.

Examples:

(FFT) 12,6,list→Spectrum (↵) "4K complex

(FFT) 10,6,list→Spectrum (↵) "4x1K complex

(FFT) 10,5,list:r→Spectrum (↵) "4x1K real

(FFT) 11,6,list→Spectrum (↵) "2x2K complex

2. IFT

(IFT) n,m,var_1(→)var_2 (↵)

n, m, var_1, var_2 are all as described for the FFT. The input blocks are always complex.

Table 3. Fixed point Operations

FFT n,m,x[:r](→)Y

IFT n,m,x(→)Y

4.6 INTEGER SCALAR OPERATIONS

Two operations are defined for integer scalars or index variables only. These operations are (ADD) and (SUBT) (upper case). The format of these operations is:

(op) var_1,var_2[(→)var_3]

If an index variable is referenced, its current value is used. If no var_3 is specified, the (→) must also be omitted, and the result of the operation will be placed in var_1.

Integer scalars and index variables may be used in subscript expressions to select subsets of an array variable. Integer scalars may be tested in programs using (IF).

Table 4. Integer Scalar Operations

Operator Name	Operation
(ADD) x,y→Z	Z = x+y
(SUBT) x,y→Z	Z = x-y

4.7 DISPLAY OPERATIONS

Two types of display of variables are possible: numerical values or plots. In addition, the descriptor of any variable, giving its type and dimensions or value, may be displayed.

1. Numerical Display

(PRINT) var

The numerical values of a scalar variable or any array are displayed. For array variables, only the first n components are displayed, where n is 500 for integer, 300 for real floating, 250 for scaled complex, and 100 for complex floating arrays. If no argument is specified, the last output argument will be printed.

2. Graphical Display

(DISPLAY) var [:i / :r] [; scale] (↙)

Only array variables may be plotted, for scalar variables DISPLAY is identical to PRINT. For real or integer array variables, the plot uses equally spaced abscissa values. For complex arrays, the real parts are plotted as the abscissas and the imaginary parts are plotted as the ordinate values. The scale is a value greater than the maximum expected data value, and defines the upper screen value. For complex variables only, a :i or :r will cause the plot to be of the imaginary or real values only, with equally spaced abscissa values. For complex plots, the scale factor must be at least 30% greater than the maximum abscissa value, since the screen is rectangular. All fixed point values are considered integers for the purposes of scaling. If the variable name is omitted, the last output argument will be displayed. If the scale is omitted, a value of 32768 is used.for integer arrays and the least power of two greater than or equal to the maximum absolute value in the list is used for floating point arrays.

3. Display of a Variable's Descriptor

(DISPLAY) (DEFN) Var (↙)

The variable name, its type code, and either its value or its base and limit addresses in MOS and its dimensions are printed. More than one variable's definition may be displayed by separating the variables' names by single spaces.

4. Display a Waterfall Distribution

a) Data is on disk

(OP) WFALL var [↓*,n↑],var;scale factor, number of lines [,starting line] (↵)

The first variable name is a disk descriptor.
The second variable name is an array descriptor.
The scale factor is a power of two.
Number of lines is the number of lines to be displayed.
Starting line (optional) is the number of the first line to be displayed counting from line 0.

b) Data is a floating point array in memory

(OP) WFALL var;scale factor,number of lines[,starting line] (↵)

The variable name must reference a two-dimensional array of floating point numbers. Other parameters are defined in Section 4a above.

4.8 SPECIAL-PURPOSE OPERATORS

There are several systems operators which either do not have a key so are called in a special way or are unique non-mathematical functions. Their use is outlined below.

1. Analog Input

(INPUT) Arg1 (↵)

This operator performs Analog-to-digital conversions of audio input through the selected microphone and writes the data on the disk data set given as its argument.

Variable "Arg1" must be a disk variable, as many tracks long as desired. The tracks will be length 3584 (7 sets of 512 points each) and the data will be fixed point, with absolute value less than 2048.

2. Analog Output

(OUTPT) Arg1 (↵)

Performs digital-to-analog conversion of the disk data set described by "Arg1" through the selected audio output device. The D/A converters are 10-bit converters, so the output data should have absolute value less than 2048.

3. Simultaneous Analog Output and Input

(OUTPT) Arg1→Arg2 (↵)

Performs D/A conversion of the integer data set described by Arg1, sends it out through the selected audio output device, and simultaneously inputs through the selected microphone and A/D converters into the integer data set described by Arg2. The arguments must be separate memory-resident data sets. Again, the output data set must contain values all of whose absolute values are less than 2048.

4. Setting the Analog Conversion Rate

(ØP) rate=n

It is possible to set the analog sampling rate to within one microsecond by using this command. The argument is an integer number of microseconds which will be used to determine the A/D sampling rate and the D/A output rate. The rate will remain at the value specified until it is changed again, and it starts out at 150 μsec at system initialization.

5. Generating a Ramp Function

(ØP) ID→Arg1

A list of equally-spaced values x, $-1 \leq x \leq 1$, is generated and placed in the real list described by Arg1. Arg argument other than a real list will result in an error message.

6. Generating Random Numbers

(ØP) random→Arg1

The next pseudo-random number or set of pseudo-random numbers x, $0 \leq x \leq 1$, will be generated and placed in the variable described by the argument. The output may be an integer variable or array or a real variable or array.

4.9 PROGRAMMING KEYBOARD OPERATIONS

A. User Programs (PGM)

Programs can be written using any valid sequence of mathematical operations previously described except (LIBE) and (EDIT). In addition, there are three program sequencing and test operations which can be used.

Programs are prepared using the system text editor and stored as documents in a text library. The same sequence of keys which are pushed to perform an operation manually are used for programs. In addition, statements may be labeled with a single alpabetic character followed by

(SKIP). This label may be referenced in (GOTO) operations. Also multiple operations may be written on one statement, with each operator key separated from preceding operations by a single space or (SKIP).

In the math system, a program is executed by pushing (PGM) pgm-name (↓), where pgm-name is the name of the document in the user's current text library containing the program. Programs are limited to one page.

B. Program Control Operations

1. Increment Index Variable or Skip (INC)

(INC) var_.....

Var must be an index variable. Its current value is incremented by its increment. If the result is less than the limit value, the operation is complete. If the result is greater than or equal to the limit value, the current value of the variable is reset to the starting value and the rest of the statement is skipped.

2. Test a Variable (IF)

(IF) $var_1 \left\{ \begin{matrix} \leq \\ = \\ \neq \\ > \end{matrix} \right\} var_2$_.....

All types of single-element variables may be compared using the IF statement. The only restriction is that both variables be of the same type (index variables are considered integers for this purpose). If the comparison specified is true, the rest of the statement is processed, otherwise, the rest of the statement is skipped.

Examples:

```
IF A↓12↑>3.14159 MOVE 3.14159→A↓12↑   "Assures that element
                                       A↓12↑ is less than or
                                       equal to 3.14159 (assuming
                                       A is a real array)

IF tly = a↓o↑ SUBT tly,1              "Subtracts one from the
                                       value of index variable tly
                                       in the case that tly was
                                       equal to the scale of fixed
                                       point array a.
```

3. Transfer to a New Statement (GOTO)

(GOTO) label

If a program is being executed, control is transferred to the beginning of the named statement. "Label" is a single alphabetic character. If this operation appears in a keyboard entry statement, the transfer is back to the beginning of the current statement. In the latter case, the label must be present, but is ignored.

4. Terminating a Program (END)

(END)

If a program is being executed it is terminated by either reaching the (END) key or the last line of the program.

APPENDIX A

OPERATIONS INDEX

OPERATION	SYSTEM OPERATIONS	REF
(LIBE)	Access library operations	1.6.3a
(EDIT)	Access editing operations	1.6.3b
(SYSTEM) [system-name]	Access mathematical operations	1.6.3d
(CANCEL)	Cancel input character	1.6.1
(RESET)	Cancel input line	1.6.1
(RESET)	Cancel the System	
(ØP) opname	Execute named system operation	1.6.4b
(PGM) prog-name [,pd0,pri]	Execute named job from pgm libe	1.6.4b
(ERASE)	Erase display	1.6.2
(FORMAT) $t_1,t_2,\ldots,t_n$	Set SKIP Table	1.6.4a
(LIBE)(ØP) SIGNOFF	Terminate further action	1.5.3

OPERATION	LIBRARY OPERATIONS	REF
(INSRT) (LIBE) User-name:U	Introduce new user	1.5.1
(INSRT) (LIBE) libe-name { :T :P :D :Sn :Rn }	Initiate new library	2.2.2
(MOVE) (LIBE) libe-name [:S :R] →libe-name [:S :R]	Library transfer	2.2.1
(DEL) (LIBE) libe-name [:R :S]	Purge Library	2.2.3
(DISPLAY) (LIBE) [libename] [:S :R]	Display library directory	2.2.4
(MOVE) doc-name$_1$,libe-name$_1$ →doc-name$_2$,libe-name$_2$	Document transfer	2.2.6
(LIBE) libe-name	Establish current library	2.3.1
(LOAD) doc-name [:n_1,n_2,n_3]	Load Text-buffer from library	2.3.2
(STORE) doc-name	Store Text-buffer as a document	2.3.4
*(INSRT) doc-name	Initiate new document	2.3.3
*(DEL) doc-name [:E]	Purge document	2.3.5
*(DISPLAY) [:alphanumeric]	Display the current library's directory	2.3.7
*(DISPLAY) doc-name [:n]	Display text document	2.3.7a
*(DISPLAY) doc-name [:alphanumeric]	Display system Symbol table (of system program document)	2.3.7b-1
*(DISPLAY) data-doc-name [:options]	Display variable (of data library document)	2.3.7c
*(DISPLAY) doc-name	Display program compile status (of AP Program document)	2.3.7b-2

*all display, insert and delete operations apply only to the current library

OPERATION	EDIT OPERATIONS	REF
A) T submode		
(T)	Access Text Buffer	3.1.2
(DISPLAY) [n]	Display Text Buffer	3.2.1
[↑ ↓] [n]	Move top and bottom pointer simultaneously	3.2.2
[↑ ↓] [( )] [n]	Move top and bottom pointer separately	3.2.2
[↑ ↓] p [n]	Move both top and bottom pointer to a new page	3.2.2
(DEL)	Delete a specified portion of text buffer	3.2.3
(FORMAT) $t_1,t_2,\ldots t_n$	Set SKIP Table	1.6.4a
B) E submode		
(E)	Access Edit Buffer	3.1.2
(DISPLAY)	Display contents of Edit Buffer	3.3.1
[→ ←] [n]	Move caret n characters right or left	3.3.2
[← s / → e]	Move caret to the first or last character of the current line	3.3.2
[← →] w[n]	Move caret n words right or left	3.3.2
← w	Move caret to the first character in the current word	3.3.2
[↑ ↓] [n]	Move caret up or down **n** lines	3.3.2
↓ e	Move caret to first character of last line	3.3.2

EDIT OPERATIONS (continued)

OPERATION	EDIT OPERATIONS	REF
(↑) s	Move caret to first character	3.3.2
(DEL) { [n] / w[n] / e / ℓ[n] / b }	Delete a specified portion of Edit buffer	3.3.4
(INSRT) text-material (INSRT)	Insert typed text material into Edit buffer	3.3.3
C) T or E Submode		
(LØAD)	Load specified contents of Text Buffer into Edit Buffer	3.4.1
(STØRE)	Store contents of Edit buffer into text buffer	3.4.3

APPENDIX B

B.1 Library System Error Messages

Document Operations — Operation

1. NØ LIBRARY DECLARED! — INSRT, DEL, DISPLAY

   There is no current library declared. Use the LIBE operation to declare the desired library.

2. NØ TEXT LIBRARY DECLARED! — LOAD, STORE

   There is no current text library declared. Use the LIBE operation to declare the desired text library.

3. NØ NAME GIVEN — ALL DOCUMENT

   No document name was specified. Repeat the operation specifying the name of the document.

4. "docname" IS NOT DEFINED! — LOAD, STORE, DEL

   Document "docname" is not defined in the current library. If the correct library has been declared as the current library, the document name must be defined using the INSRT operation.

5. TEXT BUFFER IS EMPTY! — STORE

   A STORE document operation was attempted with an empty text buffer. The operation was not performed.

6. NOT ENOUGH ROOM ON PACK! — STORE

   There are not enough unallocated contiguous tracks available on the system pack to store the current text buffer. To make more room on the pack, delete any temporary libraries which are not immediately needed using the DEL LIBE operation.

7. LIBRARY STORE FAILED!! — STORE

   Inconsistent information was detected during allocation of disk space for the document. This is system error. The system must be restarted from the machine panel. If recovery is attempted, the current text buffer contents will be valid; however, the current text library cannot be stored into.

8. SOURCE PAGE DOES NOT EXIST! — LOAD

   The specified starting source page in the LOAD document operation does not exist.

9. ONLY 10 PAGES ALLOWED IN THE TEXT BUFFER! — LOAD

   The specified destination page number was greater than ten.

		Operation
10.	"docname" ALREADY EXISTS	INSRT

The "docname" to be defined already exists in the current library.

11.	"docname" WAS NOT STORED	DEL

Document "docname" is defined, but unused, hence there is no copy to delete. If the document entry is to be removed, specify :E after the document names.

12.	-----SYNTAX ERROR	DEL, MOVE

The operation parameters were incorrectly specified. See Section 2.3 for a brief summary of the proper syntax for each operation.

13.	DOCUMENT "docname" IS NOT DEFINED	DISPLAY

Document "docname" is not a member of the current library. The operation (DISPLAY)(/) will print a list of all documents which are defined in the current library.

14.	DOCUMENT "docname" PAGE "n" DOES NOT EXIST	DISPLAY

The page "n" which was requested does not exist in document "docname."

15.	DOCUMENT "docname" - UNKNOWN TYPE!	DISPLAY

A program library document is not a recognized type (SYSTEM, AP-PGM). This error should not occur.

16.	"libename" does not exist!	LIBE

No library with the requested "libename" has been defined as a temporary library. (DISPLAY)(LIBE)(/) will print a list of all temporary libraries which are defined.

17.	"libename" is not a valid type library!	LIBE

The library "libename" is defined, but its type is not TEXT, PGM or DATA. This error should not occur.

20.	LIBE DOES NOT EXIST!	MOVE

Either the source or destination library named is not defined as a temporary library.

21.	TYPES DO NOT AGREE	MOVE

The source and destination libraries are not of the same type.

22. Header type error! MOVE

The source document was not a recognized type.

23. IMPOSSIBLE ERROR! MOVE

Inconsistent information was discovered during the MOVE operation. The operation was aborted. The destination library should not be stored or moved into.

Library Operations

24. SYNTAX ERROR libename:t where t is T,P,D,S pack# or R pack# INSRT LIBE

The library type, if temporary, or class (Shared or REFERENCE) and the desired pack number, if permanent, must be specified as shown. See Section for a more complete description of these parameters.

25. "libename" ALREADY EXISTS ON THIS PACK! INSRT LIBE

A library with the given "libename" already is defined on the specified pack. A new library name or a different pack should be selected.

26. "libename" CANNOT BE DEFINED ON THIS PACK! INSRT LIBE

There are no disk tracks available on the specified pack or the maximum number of libraries allowed for on the pack already have been defined. Delete one or more existing libraries from the pack or select a different pack.

27. LIBE IS UNDEFINED DEL LIBE

The library to be deleted does not exist in the appropriate directory of libraries.

28. INCORRECT PASSWORD DEL LIBE

The password response does not match the password associated with the library.

29. IMPOSSIBLE! DEL LIBE

The library name was formed in the appropriate directory of libraries, but was not found in the table of contents for the disk pack on which it was supposed to be defined. This error should not occur.

30. "code" CODE NOT FOUND! DEL LIBE

The allocation code listed in the library's directory was not listed as assigned to any library.

This error should not occur.

31. TEMP / REFERENCE / SHARED — LIBRARY DOES NOT EXIST — DISPLAY LIBE

The library name given is not defined in the appropriate directory of libraries.

32. "libename" LIBE DOES NOT EXIST! — MOVE LIBE

Library "libename" is not defined in the appropriate directory of libraries.

33. Pack full, only part of LIBRARY moved — MOVE LIBE

Insufficient space was available on the destination pack to move the entire library. The operation was completed with some documents in the library not moved. Use DISPLAY LIBE to print a list of the documents moved. To obtain space for the entire library, delete one or more other libraries which are located on the pack.

34. Header type error! — MOVE LIBE

A document was encountered whose type is not recognized. The MOVE was aborted. This error should not occur.

35. ACCESS DENIED — MOVE LIBE

The password response did not match the library's password.

36. IMPOSSIBLE ERROR! — MOVE LIBE

Inconsistent information was discovered during the operation. The operation was aborted. The destination library should not be used. Instead, repeat the move operation.

System Operations

37. IS NOT A VALID OP — OP

The operator name specified is not recognized as a system operator. (OP) (PRINT) (/) will print a list of all recognized operators.

38. NO PROGRAM LIBRARY SPECIFIED — PGM

No current program library has been declared. Use the LIBE operation to declare the desired library.

39. NO PGM NAME GIVEN — PGM

No program name was specified.

40. NO SUCH PGM IN CURRENT LIBRARY PGM

The program name specified is not defined in the current program library.

41. IS NOT A VALID SYSTEM PGM

The program specified is not a SYSTEM type program.

42. SYST SA does not agree with BBASE PGM

The program specified was assembled with a starting address which was not equal to BBASE. In order to use it as a program, it must be re-assembled with the base address specified in its definition document equal to BBASE.

43. Syst not run PGM

The program specified was not executed.

B.2 Edit Error Messages

1. "EDIT...Word Øverflow"

   There are too many words on the current line. To resolve the problem, start a new line.

2. EDIT...Line Øverflow

   There are too many lines in the Edit buffer. To resolve, store the Edit buffer and clear it out before continuing.

3. EDIT BUFFER ØVERFLØW

   There is too much data in the Edit buffer. To resolve, store the Edit buffer and clear it out before continuing.

4. Text Buffer Øverflow

   The text page has too much data. To resolve, begin a new page.

5. Page Undefined

   The user caused a page to be displayed which did not exist. The purpose of the message is merely to notify the user that the page does not exist.

6. Text Buffer Pointer Error

   The text pointers were moved too far apart. To resolve the problem, access a smaller portion of the Text buffer.

B.3 Disk Controller Messages

1. MOUNT PACK n

   Disk pack n was required for a requested disk action, but the pack was not locateable on any available drive. The job or other process requiring the pack was aborted.

   Response: Mount the requested pack on a disk drive and make the drive ready. To avoid this message, mount disk packs before they are needed. If the pack to be mounted is the system residence pack, the system must be restarted from the computer panel.

2. DISK CHECKSUM ERROR

   A disk track being read had a computed checksum word which differed from that computed when the track was last written. Error recovery is performed automatically by re-reading the track up to seven times. If not successful after these retries, the disk controller will abort the current job with a "DISK ERROR S 0" message.

   Response: Occasional CHECKSUM ERROR messages are normal, corresponding to the manufacturer's rating of the disk drives. If a retry of the read is successful, the data will be correct. If the retries are not successful, and the read is aborted, the information on the track is normally not recoverable.

3. DISK ERROR ...FORMAT PACK=p TRACK=t

   An attempt was made to read a disk track in a format (16 or 18 bit) different from that in which it was written. The job or other process requesting the read is aborted.

   Response: This error most frequently occurs when pointer information to the disk track has been destroyed.

4. DISK ERROR ...LENGTH PACK=p TRACK=t

   An attempt was made to write more information on a disk track than there was room for. The limits are 3600_{10} 16-bit words or 3200_{10} 18-bit words per track. The job or other process requesting the write is aborted.

   Response: Decrease the number of words being written. Again, the Error may result from destruction of reference data.

5. DISK ERROR..TRACK PACK=p TRACK=t

   An attempt was made to access track t , which was out of range. The maximum track number which can be read is 4059_{10}. The job or other process requesting the access is aborted.

   Response: If the request made was too large, decrease it. Otherwise, the error may result from destruction of reference data.

6. DISK ERROR... POSITION PACK=p TRACK=t

The track number recorded in the disk track header of the track selected did not agree with the track requested. The job or other process requesting the access is aborted.

Response: This error indicates a hardware failure in the disk drive and should be reported for corrective action.

7. DISK CHECK FAILED

The data recorded on a disk track did not match the information in MOS when read immediately after writing. The system must be restarted from the computer control panel.

Response: This error may occur either through failure of the MOS memory or a recording error which was not detected by the checksum code. The contents of PD 41-47 should be recorded before restarting the computer.

8. DISK ERROR...STATUS PACK=p TRACK=t

A non-zero error status word w was received from the disk. The error status codes are given in the appendix to TMA4: MP-32 Microprogramming Manual. The job or process requesting the disk action is aborted.

Response: The error status word w should be recorded, together with any status information printed, and reported for corrective action.

APPENDIX C

SYSTEM OPERATORS

1. Print a disk track in octal

(OP) PDISK pack, track[,starting word] (↵)

pack and track are decimal
starting word is octal.

600_8 words will be displayed, with flags

2. Copy one or more disk tracks.

(OP) MOVETRK spack, strack→ dpack,dtrack[:#tracks] (↵)

a. All values are decimal.

b. If no:#tracks is not given, 1 track will be moved.

c. The track format and length of each source track determine the format and length of the destination pack.

d. If the destination track number (dtrack) is less than 20, the disk write protect switch must be in the disabled position.

e. If the destination track number (dtrack) is greater than 3999, the Cyl 200-202 Wrt Enable switch must be in the enable position.

3. Modify one or more words on a disk pack (octal patch).

(OP) SETTRK pack, track, starting word: $data_1$[,data;] (↵)

a. Pack and track are decimal, starting word is octal, and data is specified as [#f_] value, where f is the desired flag, which may be omitted, and value is the octal data value desired.

b. Leading Os need not be specified unless the value is identically 0.

c. As many words may be entered as will fit on one statement.

d. The track length will be extended, if necessary to include the specified values. If it will not be shortened, nor will the format be changed.

e. If track is less than 20, the disk write protect switch must be in the disabled position.

f. If the destination track number (dtrack) is greater than 3999, the Cyl 200-202 Wrt Enable switch must be in the enable position.

4. Display a track in symbol table (directory) format:

(OP) PRINT [Rn,] track (↵)

a. n and track are decimal.

b. If R n is specified, n is the desired pack number, otherwise, the system pack is assumed.

c. If the track requested is not in symbol table format, the information displayed and system state are unpredictable.

5. Display a snapshot dump. A snapshot dump is made by entering (RESET) (↵) or by manually setting CA=5 and executing at IA 0.
(OP) PSNAP starting address (↵)

a. Starting address is octal.

b. 600_8 words will be displayed, beginning at the specified starting address.

6. Establish the number of pages in the text buffer

(OP) p=n

a. Sets the number of pages in the text buffer to n, where $0 \leq n \leq 10$. This op may be used to initially clear the text buffer when beginning a new document, or to decrease the number of pages in the current document.

7. Assemble a macro-processor system

(OP) ASM (SYST) sysdoc[;a-priori sys]:pgmdoc (↵)

a. See manual TMA3, Macroprogramming Manual, for an explanation of this op.

8. Assemble an AP-120 Microprogram

(OP) APASM doc1[,doc2,doc3,...]:pgmdoc (↵)

a. See manual TMA6, AP-120 Programming Manual, for an explanation of this op.

9. Terminate a console session

(OP) SIGNOFF (↵)

a. The system will respond:

ANY TEMP LIBES TO DELETE? ENTER N IF NOT.

The response should be N (↵). Any other response will abort the signoff procedure. If N is entered, the user's temporary libraries are deleted and the station is available for SIGNON by a new user.

10. Define a new operation

(OP) ADDOP opname,track #[-parameter]

a. The opname given is adder to the operator table. Track # is the decimal track on the system pack where the program which defines the operator resides. Parameter is a one-word decimal value which will be placed in pad cell 0 when the op is called. If the track # is less than 20, it will automatically be relocated to the current system cylinder.

11. Delete an existing operation

(OP) DELOP opname

a. The given opname is removed from the operator table.

12. Set Program Source Address

(OP) SETPSA PSA

a. Sets the AP PSA to the octal value entered through the keyboard.

13. Write Program Source

(OP) WRTAPPS Name, PSA

a. Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Program RAM starting at the RAM address specified by the PSA parameter.

If the program does not exist in the current program library a message will be displayed which reads:

"Item Name" is not in current program library.

14. Write AP Memory

(OP) WRTAPMD Name, MA

Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Memory, starting at the AP Memory Address specified by the parameter MA.

15. Write AP Data Pad

(OP) WRTAPDP Name, DPA

a. Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Data Pad, starting at the AP Data Pad address specified by the parameter DPA. It writes every word of the program into Data Pad (which has only octal 40 32-bit words) so it is possible that DPA may "wrap around."

16. Write AP S-Pad

(OP) WRTAPSP SPA, Arg_1

a. Moves the argument, which is treated as a general argument, into the AP S-Pad cell specified by the SPA parameter.

17. Snapshot AP Data ROM

(OP) SNAPDR DRA

a. Moves the AP Data ROM image, starting with the 32-bit word whose address in Data ROM is specified by the DRA parameter, onto track number 3999 decimal of the system pack. The transfer is terminated by the event which occurs first of the following: either DRA begins to wrap around or the track becomes full.

18. Snapshot AP Memory

(OP) SNAPMD MA

a. Moves one disk track's worth of the AP Memory, starting at the address specified by the MA parameter, onto track 3999 decimal of the system pack.

19. Snapshot AP Data Pad

(OP) SNAPDP

a. Moves the AP Data Pad contents starting at DPA 0, onto track 3999 of the system pack.

20. Snapshot AP S-Pad

(OP) SNAPSP

a. Moves the AP S-Pad contents, starting at SPA 0, onto track 3999 of the system pack. The only path out of S-Pad takes the right most 8 bits from S-Pad into the leftmost 8 bits of ACL, so the data is written on disk that way. For example, the left half of the first 16-bit word on disk will be what was in the right byte of S-Pad cell 0, and the left byte of the third 16-bit word on disk will be what was in the right byte of the second S-Pad cell, etc. The other 3 bytes of each double word on disk will be random.

21. Restore Disk Track Header

(OP) WRTHDR pack,track (↵)

a. The disk track header for the specified track is rewritten. This operation should be used only when it is suspected that a particular track has had its header destroyed.

22. System Diagnostics

APTEST, CDTEST, DISKTEST and SYSTEST are described in TN 73-008.

23. Print Diagnostic Log

(OP) PLOG (↵)

a. The log prepared by the system diagnostic routines is printed. A pause occurs at the end of each page to allow examination or copying of the information. A (↵) will cause the next page to be displayed. At the end of the log, the message ENTER P TO PURGE LOG will appear. Respond P (↵) if the log is to be purged, or (↵) if the log is to be saved.

APPENDIX 6

System Operation Manual

CHI
SIGNAL SYSTEMS

TMA9

TMA9

SYSTEM OPERATION MANUAL FOR SN2

CULLER-HARRISON INC.

PREFACE

This manual is intended to contain those technical notes and appendices useful in the operation of the CHI SIGNAL SYSTEM. In addition to notes common to all systems, such as sign-on, sign-off procedures and error messages, those notes that apply specifically to this system are also included. Interface support software and unique operators would be documented in this manual.

Reference is made to the following manuals for details characteristic of the CHI Signal Processing System:

TMA2 User's Manual
TMA3 Macroprogramming Manual
TMB2 MP-32A Macroprocessor Reference Manual

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No. 29110
© 1975
by Culler/Harrison, Inc.

System Operation Manual for SN/2

TABLE OF CONTENTS

Section	Document	Title
1	CHI-TN-74-004	Sign-on, Sign-off Procedure
2	CHI-TN-73-003	Job Management Data Structures
3	CHI-TN-74-006	Library Data Structure
4	Appendix C of Users Manual	Utilities and System Operators
5	CHI-TN-74-007	Disk Controller
6	CHI-TN-73-008	Signal System Diagnostics
7	Appendix B of Users Manual	Error Messages
8	CHI-TN-73-005	Signal System Acceptance Tests
9	CHI-TN-74-010	Disk Pack Initialize Procedure (PGM DASDI)
10	CHI-TN-74-003R	System Relocatability on the System Pack
11	CHI-TN-74-008	System Disk Backup Recommendations
12	CHI-TN-74-011	System Problem Report Procedure and Forms
13	CHI-TN-74-013	Basic Disk Commands
14	CHI-TN-74-014	System Message Facility
15	CHI-TN-75-014	Operating System Philosophy - SN/2
A		Software Change Requests
B		Software Change Notices

CHI
CULLER-HARRISON INC.
~~5770 Thornwood Drive~~, Goleta, California 93017
150-A Aero Camino
Telephone (805) ~~967-0434~~
968-1064

CHI-TN-74-004R
10/29/75

SIGN-ON, SIGN-OFF PROCEDURE

M. McCammon/J. Vanderford

The system library and signon facilities are implemented as described in the User's Manual.

1. To sign-on to the system, the following sequence is used:

   a) Push (/) to awaken the system. The screen will be erased and the message

      "PLEASE IDENTIFY YOURSELF"

      will appear.

   b) At this point, enter a valid user name. Follow the user name by (/). If the user name given is valid, and a password is required, the message

      "WHAT IS YOUR PASSWORD?"

      will appear.

   c) Respond to the password request, if any, by entering the correct password and (/). If the password is accepted the signon procedure is complete, and the system will be in LIBE mode. However, if the last session at any console was terminated without (OP) SIGNOFF or with (OP) SIGNOFF followed by the S (/) response, the system will give you a chance to resume activities at the point where the last session was terminated. This allows recovery of all TEMP libraries as well as the contents of both the text and edit buffers. The message

      "ENTER R TO REOVER OLD CONTEXT OR P TO PURGE IT"

      will appear if recovery is possible.

   d) Respond R (/) to recover or P (/) to purge. The system will be left in LIBE mode with no activity underway for the user.

e) If either the user name given is not valid or the password incorrect the system will respond:

"INVALID SIGNON!"

and return to the rest state.

2. Library usage conventions

a) Each user name has associated with it a private directory of reference libraries. These libraries are accessable only by that user.

b) To permit exchange of information between users operating under different user names, a directory of shared libraries is available. Any library in this directory is accessable by all users who know its password, if any. At present, all system libraries are defined as shared libraries, without passwords.

c) The following procedure is proposed when two or more users are updating the same shared library.

i) Define a temporary library of the proper type (e.g. (INSRT) (LIBE) EDIT1:T(/)).

ii) Copy the desired shared library into this library (e.g. (MOVE) (LIBE EDIT:S(->)EDIT1).

iii) Declare this temporary copy as your current library ((LIBE) EDIT1). Perform whatever modifications, tests, etc. on the documents of this library.

iv) Do not move this library back over the shared library. Instead, define a second temporary copy, move the original shared library into it as above, and then move documents that you have updated into the second copy. Then immediately move the now updated second copy back over the shared library.

INSRT LIBE EDIT2:T ↵

MOVE LIBE EDIT:S→EDIT2 ↵

MOVE TDISP2,EDIT1→TDISP2,EDIT2 ↵

...other documents moves...

MOVE LIBE EDIT2→EDIT:S ↵

DEL LIBE EDIT2 ↵ Y ↵

v) The purpose of updating a second copy and immediately transferring it back is to ensure that any intervening changes in other documents in the shared library are not lost when you make your update.

vi) The procedures described in step iv) are only necessary for libraries which more than one user may be simultaneously updating. Otherwise, the original temporary library could be simply moved back over the shared library.

3. To sign-off from the system:

a) In LIBE mode, enter OP SIGNOFF ↵ the system will respond with the message:

"TO SAVE CONTEXT ENTER S--TO PURGE ENTER N"

This is to remind you to move any temp libraries whose contents you want to save onto a private or shared reference library.

b) If you respond S ↵ , the system will sign you off as a current user but will save all of your temporary libraries, as well as your text buffer and edit buffer. The screen will be erased to signify completion of the signoff procedure.

c) If you respond N ↵, all temporary libraries you have defined on the system pack will be deleted as well as your text and edit buffers. The screen will be erased to signify, completion of the signoff procedure.

d) If you respond with anything other than S ↵ or R ↵, the system will abort the signoff procedure, leaving you in LIBE mode. The screen will not be erased in this case.

JOB MANAGEMENT DATA STRUCTURES

Background jobs:

1. A background job is a process which runs on the MP-32 with the system in the background state. It is written as a macro subroutine or collection of subroutines. Background jobs may be resident, that is, already present in main memory, or disk-based. If disk-based, they will be loaded at BBASE, and should be compiled to run at that location.

2. Memory Space: A background job may use all memory between BBASE and the contents of location BKLIMIT. This may be as much as 24K for a 32K memory or 56K for a 64K 18-bit word memory (at present, these limits are 20K and 52K respectively). The actual amount of space available for a given application will be determined by the amount of space required for resident jobs or communication between jobs.

3. Swapping: A background job may be swapped out, its entire memory load is onto disk with data pad preserved. This swapping may take place any time that the background job is waiting for input from the user's console, 500 millisecond after erasing the console display screen, or at the job's request through use of the BREAK macro instruction. When any of these events occur, the job's memory load will be swapped out only if another background job or an interactive user is ready to run in its place. Otherwise, no swapping will occur and the job will remain in memory.

   When a disk-based job is swapped, all memory between BBASE and the BKBND value set by the last call to SETBBND by the job will be written out. Resident jobs are never swapped.

4. Input and Output to the User's Console

   Each background job has associated with it a user console; normally, the console from which the job was initiated. A job may at any time output information to the display screen of the console by using the system output macro instructions and subroutines. However, if the job is to either erase the display screen or request console input, it is necessary to avoid conflicts with other jobs making similar requests for the same console. Erasing of the display screen is accomplished by using the ERASE macro instruction.

Input from the user console is obtained by using the GETNAME, GETINTGR, GETOCTAL or GETKEY macro instructions to obtain input from the current input statement. When the end of the current statement (the carriage return) has been received, further input can be obtained only by first requesting the next statement, using the GETSTMNT subroutine, which will delay the job until a complete statement has been input. After the GETSTMNT call, a new current statement is available, and the above mentioned calls to obtain input from that statement may be used.

5. Job Priorities:

Each background job has associated with it a priority between 0 and 255. Each time a different background job is to be executed the job with the highest priority among all those ready to be run will be selected for service. This choice is made independently of whether a new job is selected or existing job is resumed. The selection process takes place when the current job completes or is temporarily delayed (see Swapping). Interactive service is always given priority over background jobs at selection times.

6. Submitting a job: A background job is submitted from an executing program by using the JOB subroutine. The JOB subroutine takes four parameters: a job-descriptor, the address of a resident exit subroutine, a one-word parameter value which will be passed to the job at execution time in pad cell 0, and a priority number between 0 and 255. The job-descriptor is two words long. The first word is the starting track where the job is located for a disk-based job; or 0 for a resident job. The second word is the length in 18-bit words of a disk-based job or the base address for a resident job. The exit subroutine specified will be called by the job scheduler when the job has completed. The exit subroutine must return to the job scheduler.

7. Job Format:

Each job must begin with a 64 word data-pad save area. The last word in this area must be the address of the beginning of a push-down stack. The first word in this area is the entry point address for the job. The job will be executed as a subroutine of the job scheduler. It must end by a RETURN to return control to the job scheduler. Data pad cells 1 to 67_8 will contain the content of words 1 - 67_8 in the data-pad save area, pad cell 0 will contain the parameter word provided when the job request was submitted. The job should begin with a mode setting macro instruction.

8. Job Element

One for each job in the system.
Contains the information necessary to schedule and locate the job at any stage when it is not being executed.
May be on JOBSTK or WAITSTK, depending on whether the job is ready for processor service or waiting for some event.
Unused job elements are kept on the JOBELT queue.
Each job element is 8 words long.

Word 0: LINK - address of next job element on queue, 0, if last element on a queue.
Word 1: Station Number (JSTN)
2: Track of program (JTRK)
3: Length of program (JLENO)
4: Completion exit (JCOMP)
5: PARAMETER (JPAR)
6: SLIST INDEX/Priority (JPRI)
7: Length of Swap area (JLEN1)

9. Job Queue (JOBSTK)

The job queue (JOBSTK) is a linked list of the job elements of all jobs which need processor service. This queue is ordered by priority, with the job with the highest priority first. The primitive subroutine ENQUE will place a job element whose address is in A2 on JOBSTK according to its priority.

10. Wait Stack (WAITSTK)

The wait stack is a linked list of the job elements of all jobs which are waiting for some external event before they are readyfor processor service.

11. Job Element Stack (JOBELT)

The job element stack is a linked list of available job element blocks. The macro subroutine JOB will remove an element from JOBELT, fill it in to describe a job, and insert it in priority order on JOBSTK.

12. Swap Area List (SLIST)

A list of the starting track numbers of the disk swap areas defined for use by RUNJOB in swapping background jobs. It is ordered by size, with the smallest swap areas first. A secondary index, called SBNDS is a list of the discrete swap area sizes (in words, biased by 100000), followed by the address of the first SLIST entry for swap areas of that size.

An SLIST entry is flagged by turning on bit 17 of the entry whenever the entry is in use. The current allocation of swap areas is:

1 - 3 track (22600_8 words) area
2 - 4 track (31000_8 words) areas

2 - 6 track (45400_8 words) areas
2 - 17 track (152200_8 words) areas

13. Global Variables Related to Job Management

a) Interactive mode flag (INTF)

0 when background mode active
≠0 when Interactive mode active

b) Data pad save area pointer (SAVPD)

Contains the address of a 64 word save area where the current contents of data pad may be saved or previous contents loaded from. If flag 1 is 0, the area is empty, if 1, the area is full. This variable is used by interrupt routines to locate a place to save the interrupted program's data pad.

c) Current interactive station (ISTN)

This word gives the current interactive station number. If negative, no interactive station information is loaded at IBASE.

d) Current background area occupant (BSTN)

This word gives the job element address of the background job currently loaded at BBASE. If 0, no background job is loaded at BBASE.

e) Current background job (CJOB)

This word gives the job element address of the background job currently receiving processor service. If 0, no background job is receiving processor service.

f) Background area swap limit (BKBND)

This word defines the upper limit of the background area which needs to be swapped out if a job is to be pre-empted. The instruction SETBBND should be used to set this bound as well as check that it doesn't exceed BKLIMIT.

g) Background area upper limit (BKLIMIT)

This word contains the upper limit of the area which can be used for background jobs. All space above BKLIMIT is reserved for inter-job communication and/or resident programs.

h) Background job request address (BRQST)

The BRQST is an array containing one entry for each interactive terminal. When a job requests interactive input, its job elements address is recorded in the BRQST entry for its associated interactive terminal. Similarly, background requests to erase the display screen cause BRQST to be set with their address. When the events necessary for the job to

proceed have occurred, this provides the job element address to enable transfer of the job element from the wait stack to the job queue.

i) Station Flag Words (STNFWB)

STNFWB is an array, with one entry for each interactive terminal, in which some conditions primarily affecting interactive scheduling are recorded. The bit assignments and their use are:

Bit 17_8 - Display Busy — Set during erase interval to defer echoing of input keys.

Bit 15_8 - Background Request — Set when a background job has requested a statement from an interactive console. Valid only if interactive inhibit bit set. Cleared when request satisfied.

Bit 12_8 - Background Wait — Set when a background job has erased the screen and is waiting for the erase interval to expire.

Bit 11_8 - Interactive Inhibit — Set whenever interactive service for this terminal is inhibited. May be either during an erase interval initiated interactively or while a background job is communicating with a terminal.

j) Interactive station track (STNBS)

STNBS is an array with one entry for each interactive terminal giving the track number on the system pack where the current interactive system data for the user is located.

Michael McCammon
CHI-TN-74-006
3/1/74

Library System Data Structures

A. DISK ALLOCATION

1. Allocation Tracks - Tracks 20 and 21 on each pack.

Record which tracks are allocated, and for what use.
Written in 16-bit format.

Word 0 - Use code - highest number is current track. Whenever an allocation track is updated, its use code is incremented by one and it is written on the alternate track.

Words $1-3721_8$: Allocation record for tracks $0-3999_{10}$ on the pack. One half-word per track. If entry $\geq 200_8$, track is allocated. If entry $< 200_8$, track is available. All permanently reserved tracks have entry 377_8.

2. Pack Table of Contents Track - Track 22 on each pack.

A directory of all reference libraries on the pack. Also contains a list of allocation codes which may be assigned to libraries.
Written in 18-bit format.

Word 0: Word number of last used word or track: track length -1

Word 1: Upper limit for track space, normally 6170_8.

Words 2-77 : List headers for directory entries beginning with 0-9, A-Z, a-z respectively.

Words 100-101: Pack and track numbers of this directory.

Word 102: Type code associated with this directory = 376_8

Words 114-275: Available allocation code list. Contains codes $14-175_8$. If an allocation code is being used, 200_8 is added to its entry in this list.

Word 276: Marker for end of allocation codes = 177777_8.

Word 277: Entries for all permanent libraries defined on the pack. The data portion of each entry is three words long. The first word is the track number of the directory for the library. The second word is the track number of the directory of libraries which references the library. The third word is the allocation code for the library.

B. LIBRARY DIRECTORY

This format is used for all library directories, including the system and user directory of libraries.

Word 0: Next available word on track -1.

Word 1: Limit on available space. Normally 6170.

Words 2-77$_8$: Addresses of first directory entry beginning with a given character, in the order 0-9, A-Z, a-z.

Words 100-101$_8$: Pack and track numbers of this directory.

Word 102$_8$: Type code for the directory. 0=text library, 1=PGM library, 2=DATA library, 4=Directory of library, & 376=Table of Contents

Words 103-107$_8$: Protection code word. Word 103 is 0 if there is none.

Words 110-112: Residual track information. The track #, last record # and length of the last record structure track entry for this library.

Words 113: Allocation code for all tracks allocated for this library (except the directory track itself, which has a code of 376$_8$).

Words 114: Document Entries: Each entry has the following format:

Word 0: Link to next entry with same first character

Word 1-j=4: 2nd through last characters in label, up to 9 character labels max.

Word j+1: FLAG 1 on, data portion is track of header for this document.

Word j+2: Right half is record number of header on the track, left half is record number of backup copy header.

Word j+3: Track of backup copy header.

C. LIBRARY DIRECTORY STRUCTURE

1. System Directory of Users: Track 23 on system pack.

One entry for each user, gives track number of his Directory of Libraries. Accessed during signon to establish access right to system.

2. User's Directory of Libraries

One for each valid user name. Located on system pack. Password for user is protection code in this directory, checked at SIGNON. Entry in this directory for each private reference library defined by the user. This entry gives the pack number of the library. The library is located on the pack through the pack table of contents.

3. Shared Directory of Libraries: Track 38 on system pack.

   One entry for each shared reference library. This entry gives the pack number of the library. The library is located on the pack through the pack table of contents.

4. Directory of Temporary Libraries

   This directory is part of each user's Library Interactive track, beginning at word 240_8.

   It contains one entry for each temporary library which gives the track of the library's directory, the type of the library, and the allocation code assigned to the library.

5. Library Directory

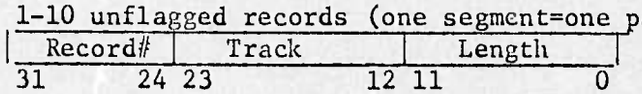
   Contains one entry for each document in the library. This entry gives the track and record number of the header for the document. A backup header pointer is also part of temporary library directories.

6. Document Header

   A document header describes the type of document, gives the locations of each segment of the document, and may contain additional information about the document. All document headers begin with a doubleword which gives the document type and the number of segments. This is followed by one doubleword for each segment which gives the segment location and length. Additional doublewords may follow.

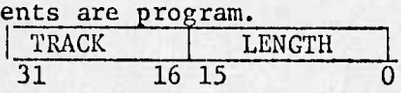
   a) Text documents

      Type code: 0
      Number of segments: 1-10 unflagged records (one segment=one page)
      Segment descriptor:

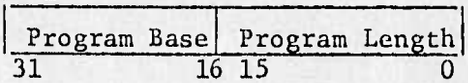


   b) MP Programs:

      Type code: 5
      Number of segments: 2-18 flagged tracks
           First segment is global symbol table
           Remaining segments are program.
      Segment descriptor:



      Additional Doublewords: 1



c) AP Programs:

Type code: 4
Number of segments: 1 unflagged track
Segment Descriptor:

TRACK	LENGTH
31 16	15 0

Additional Doubleword:

1 PS-Base	Length
31 16	15 0

7. Record Format for Track

RECORD FORMAT FOR TRACK

Word No.	Contents
1	n = number of records on track
2	location of next available word on track
3	displacement to record 1 (words to be skipped + 1)
4	length of record 1, in words
5	displacement to record 2 (words to be skipped + 1)
6	length of record 2, in words
⋮	⋮
2n+1	displacement to record n (words to be skipped + 1)
2n+2	length of record n, in words
2n+3	Record 1
⋮	Record 2
⋮	⋮
⋮	Record n
	(Unused portion of track, if any)

8. Segment Structure for Text Documents

For text documents, a record contains a segment. A segment is a page of text.

```
NA          -Total length of text page
#lines(n)   -Number or lines in the text page
LPTR1       -Offset from here to start of line one (Line 1, below)
LPTR2       -Offset from here to start of line two
  .
  .
  .
LPTRn       -Offset from here to start of line n (last line defined)

Line 1 #words(m) -Number of words on line 1
       WPTR1     -Offset from here to word 1 line 1
       WPTR2     -Offset from here to word 2 line 1
         .
         .
         .
       WPTRm     -Offset from here to word m line 1
Line 2 #words(p) -Number of words in line 2
```

NOTES: a) A"word", as used here, is structured as below:

n C1	where these are halfwords, n is the total number
C2 C3	of characters in the word and C1,...C_n are the
:	characters in the word. C_n is the terminator for
:	the word. If n is even, the final memory cell in
C_{n-1} C_n	the word will have both halves equal to C_n; if n is odd only the right half will be C_n.

b) There are always exactly as many line (word) pointers as there are lines (words) defined.

c) The pointers in the text page are called "start displacement" pointers and are always the distance in memory cells from where the pointer in question is to where the object being pointed at is.

d) Each mnemonic (i.e., NA, LPTR1,...WPTRm) above represents one 16-bit computer word.

1. Print a disk track in octal

(OP) PDISK pack, track[,starting word] (↵)

pack and track are decimal
starting word is octal.

600_8 words will be displayed, with flags

2. Copy one or more disk tracks.

(OP) MOVETRK spack, strack→ dpack,dtrack[:#tracks] (↵)

a. All values are decimal.

b. If no:#tracks is not given, 1 track will be moved.

c. The track format and length of each source track determine the format and length of the destination pack.

d. If the destination track number (dtrack) is less than 20, the disk write protect switch must be in the disabled position.

e. If the destination track number (dtrack) is greater than 3999, the Cyl 200-202 Wrt Enable switch must be in the enable position.

3. Modify one or more words on a disk pack (octal patch).

(OP) SETTRK pack, track, starting word: $data_1$[,data;] (↵)

a. Pack and track are decimal, starting word is octal, and data is specified as [#f_] value, where f is the desired flag, which may be omitted, and value is the octal data value desired.

b. Leading Os need not be specified unless the value is identically 0.

c. As many words may be entered as will fit on one statement.

d. The track length will be extended, if necessary to include the specified values. If it will not be shortened, nor will the format be changed.

e. If track is less than 20, the disk write protect switch must be in the disabled position.

f. If the destination track number (dtrack) is greater than 3999, the Cyl 200-202 Wrt Enable switch must be in the enable position.

4. Display a track in symbol table (directory) format:

(OP) PRINT [Rn,] track (↵)

a. n and track are decimal.

b. If R n is specified, n is the desired pack number, otherwise, the system pack is assumed.

c. If the track requested is not in symbol table format, the information displayed and system state are unpredictable.

5. Display a snapshot dump. A snapshot dump is made by entering (RESET) (↵) or by manually setting CA=5 and executing at IA 0.
(OP) PSNAP starting address (↵)

a. Starting address is octal.

b. 600_8 words will be displayed, beginning at the specified starting address.

6. Establish the number of pages in the text buffer

(OP) p=n

a. Sets the number of pages in the text buffer to n, where $0 \leq n \leq 10$. This op may be used to initially clear the text buffer when beginning a new document, or to decrease the number of pages in the current document.

7. Assemble a macro-processor system

(OP) ASM (SYST) sysdoc[;a-priori sys]:pgmdoc (↵)

a. See manual TMA3, Macroprogramming Manual, for an explanation of this op.

8. Assemble an AP-120 Microprogram

(OP) APASM docl[,doc2,doc3,...]:pgmdoc (↵)

a. See manual TMA6, AP-120 Programming Manual, for an explanation of this op.

9. Terminate a console session

(OP) SIGNOFF (↵)

a. The system will respond:

ANY TEMP LIBES TO DELETE? ENTER N IF NOT.

The response should be N (↵). Any other response will abort the signoff procedure. If N is entered, the user's temporary libraries are deleted and the station is available for SIGNON by a new user.

10. Define a new operation

(OP) ADDOP opname,track #[-parameter]

a. The opname given is adder to the operator table. Track # is the decimal track on the system pack where the program which defines the operator resides. Parameter is a one-word decimal value which will be placed in pad cell 0 when the op is called. If the track # is less than 20, it will automatically be relocated to the current system cylinder.

11. Delete an existing operation

(OP) DELOP opname

a. The given opname is removed from the operator table.

12. Set Program Source Address

(OP) SETPSA PSA

a. Sets the AP PSA to the octal value entered through the keyboard.

13. Write Program Source

(OP) WRTAPPS Name, PSA

a. Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Program RAM starting at the RAM address specified by the PSA parameter.

If the program does not exist in the current program library a message will be displayed which reads:

"Item Name" is not in current program library.

14. Write AP Memory

(OP) WRTAPMD Name, MA

Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Memory, starting at the AP Memory Address specified by the parameter MA.

15. Write AP Data Pad

(OP) WRTAPDP Name, DPA

a. Retrieves the program identified by the Name parameter from the current program library and moves it into the AP Data Pad, starting at the AP Data Pad address specified by the parameter DPA. It writes every word of the program into Data Pad (which has only octal 40 32-bit words) so it is possible that DPA may "wrap around."

16. Write AP S-Pad

(OP) WRTAPSP SPA, Arg1

a. Moves the argument, which is treated as a general argument, into the AP S-Pad cell specified by the SPA parameter.

17. Snapshot AP Data ROM

(OP) SNAPDR DRA

a. Moves the AP Data ROM image, starting with the 32-bit word whose address in Data ROM is specified by the DRA parameter, onto track number 3999 decimal of the system pack. The transfer is terminated by the event which occurs first of the following: either DRA begins to wrap around or the track becomes full.

18. Snapshot AP Memory

(OP) SNAPMD MA

a. Moves one disk track's worth of the AP Memory, starting at the address specified by the MA parameter, onto track 3999 decimal of the system pack.

19. Snapshot AP Data Pad

(OP) SNAPDP

a. Moves the AP Data Pad contents starting at DPA 0, onto track 3999 of the system pack.

20. Snapshot AP S-Pad

(OP) SNAPSP

a. Moves the AP S-Pad contents, starting at SPA 0, onto track 3999 of the system pack. The only path out of S-Pad takes the right most 8 bits from S-Pad into the leftmost 8 bits of ACL, so the data is written on disk that way. For example, the left half of the first 16-bit word on disk will be what was in the right byte of S-Pad cell 0, and the left byte of the third 16-bit word on disk will be what was in the right byte of the second S-Pad cell, etc. The other 3 bytes of each double word on disk will be random.

21. Restore Disk Track Header

(OP) WRTHDR pack,track (↵)

a. The disk track header for the specified track is rewritten. This operation should be used only when it is suspected that a particular track has had its header destroyed.

22. System Diagnostics

APTEST, CDTEST, DISKTEST and SYSTEST are described in TN 73-008.

23. Print Diagnostic Log

(OP) PLOG (↵)

a. The log prepared by the system diagnostic routines is printed. A pause occurs at the end of each page to allow examination or copying of the information. A (↵) will cause the next page to be displayed. At the end of the log, the message ENTER P TO PURGE LOG will appear. Respond P (↵) if the log is to be purged, or (↵) if the log is to be saved.

Disk Controller

NAME: DISK

DESCRIPTION: Provides input and output facilities for transfer of either 16-bit or 18-bit data words from or to named disk packs. Also provides the ability to initiate the positioning of a disk access mechanism. Parameters for the DISK controller must be set up in data pad as follows:

Data Pad	Contents
41	OPTION (see list of options below)
42	PACK - the logical pack number
43	TRACK - the sequential track number on the pack
44	BUFFER ADDRESS - the memory address of the data area
45	LENGTH - the length of the data area
46	EXIT - the return address where control is to be transferred upon successful completion of the disk operation. Must be an executable primitive.

In addition, the read record options require a record number, this is passed as:

47	RECORD NUMBER.

The permissible values of the OPTION parameter, and the disk action performed, are:

OPTION	ACTION
0	COMPARE FLAGGED - Read in 18-bit format the data from the specified disk track and compare it word by word, including flags, with the contents of the memory data area. If they match, data pad 47 will contain a zero upon exit.
1	READ FLAGGED - Read the specified disk in 18-bit format into the memory data area.
2	WRITE FLAGGED - Write the contents of the memory data area, including flags, onto the specified track in 18-bit format.
3	READ FLAGGED RECORD - Read in 18-bit format only the record specified of the disk track named into the memory data area.
4	COMPARE DATA - Read in 16-bit format the specified data track and compare it, word by word excluding flags, with the contents of the memory data area. If they match, data pad 47 will contain a zero upon exit.

OPTION	ACTION
5	READ DATA - Read in 16-bit format the specified disk track into the memory data area, setting the flags to 0.
6	WRITE DATA - Write the contents of the memory data, excluding flags,onto the specified track in 16-bit format.
7	READ DATA RECORD - Read in 16-bit format only the record specified of the disk track named into the memory data area. The flags are set to 0.
10	READ - Read the specified disk track in the format which it was written into the memory buffer specified. Sets data pad cell 47 to 0 if track was flagged, or 5 if track was not flagged. This option will take 25 milliseconds longer than option 5 if the track is not flagged.
100	SEEK - Position the disk access mechanism of the disk drive on which the specified pack is mounted to the specified track. This operation completes as soon as the seek is initiated.

Notes:

1. The following conditions are recognized as user errors:
   a) An attempt to read a track in the wrong data format (also applies to compare)
   b) A request for a non-existent track number
   c) An attempt to write more words than will fit on one track
   d) A request for a disk pack which is not mounted.
2. The following conditions are recognized as system errors:
   a) A header track number which does not match the track requested
   b) A checksum error which is not correctable in eight retrys
   c) Any disk error status indication except checksum error or, in some cases, drive busy.
3. For all read and compare operations, data pad cell 50 will contain the actual data length. However, in no case will more words be transferred to memory than the buffer length specified.

Disk Information Formats:

1. TRACK HEADER

   Three words long:

   PACK NUMBER
   TRACK NUMBER
   CHECKSUM WORD.

2. Data Portion of a Disk Track

   N+2 words long
   Data length (N) with bit 17=1 if 16-bit format
   bit 17=0 if 18-bit format
   N data words
   Checksum word

   A word is 18 bits long in 18-bit format 16-bits long in 16-bit format.
   Only the N data words are ever transferred into memory.

3. Record Structure of data words

   Record header
   Record 1
   Record 2
   :
   Record K
   CHECKSUM WORD

4. Record Header

   2K + 2 words long, where K is number of records.
   K
   N
   *DISPLACEMENT TO RECORD 1
   LENGTH OF RECORD 1

   *DISPLACEMENT TO RECORD 2
   LENGTH OF RECORD 2
   :
   *DISPLACEMENT TO RECORD K
   LENGTH OF RECORD K

Other Data Structures:

1. PAKTAB - One entry for each physical disk drive, gives the pack number of the pack believed to be mounted on the drive.

2. DSTAB - One entry for each physical disk drive. The entry is 0 if the drive is available or contains the station number +1 of the user station for when the drive is currently being used.

The Disk controller is organized into functional modules as follows:

1. DISK - Main entry point, searches PAKTAB to locate drive on which pack requested is mounted. Reserves that drive for the operation if it is available, selects the drive and calls SEEKC. If the pack is not found in PAKTAB, calls MOUNT to locate it.

2. MOUNT - Reads a track header from each drive to update PAKTAB until the requested pack is located. Reserves the drive for the operation and selects the drive, then calls SEEKC. If the pack is not mounted, prints a mount request message. If the pack requested is not the system pack, the user is reset to allow repetition of the operation when the pack has been mounted.

3. SEEKC - Calculates the cylinder and head number for the specified track. Checks that the track number is valid. Sends the seek command. If the operation is SEEK, calls DSKX. Exits to SWAIT if the access arm was already in place, otherwise calls DISKRDHD.

4. SWAIT - Waits for seek to complete, then calls DISKRDHD.

5. DISKRDHD - Reads the track header and verifies that the correct pack is mounted. If not, it calls MOUNT. The track number is checked to verify positioning. The appropriate OPTION routine is called to perform the requested disk operation. This module contains command issuing and end of block subroutines used by the subsequent routines.

6. DSKX - Clears the DSTAB entry for the currently selected drive and exits to the macro whose address is in data pad 46.

7. DOREAD - Common full track input routine. Reads the specified track in the format specified by the low order 3 bits of the option (0,1 = 18-bit; 4,5 = 16-bit). Assumes previously loaded routines will process input data and verify format correctness. When number of words indicated by length word on the track have been input, sends end of block, verifies that no checksum errors were detected, and exits to DSKX.

8. DOWRITE - Common full track output routine. Writes the length word and the specified number of data words in the format specified by the low order 3 bits of the option (2 = 18-bit, 6 = 16-bit). Assumes previously loaded routines will fetch data and provide address of maximum track length and format flag.

9. DORDR - Common single record input routine. Reads the specified track in the format specified by the low order 3 bits of the option (3 - 18-bit, 7 = 16-bit). Verifies that the record exists on the track, skips over words to the header entry for that record, skips to the beginning of the record. Then transfers the record itself using an assumed routine. Finally, any remaining words on the track are skipped over and end of block is sent.

CHI-TN-73-008
Dale Taylor

CHI SIGNAL SYSTEM DIAGNOSTICS

11/19/73

The SIGNAL System diagnostics are chained together so that they can either run successively or be called separately. The diagnostic routines available are:

1) PADTEST, CDTEST
2) DISKTEST, SEEKTEST
3) APTEST
4) SYSTEST

1. PADTEST, CDTEST

a) PADTEST - Tests every bit of every cell of the MP-32A Data Pad for both one and zero and worst case pad access for every cell, as well as 1024 (decimal) complete padloads of random numbers.

b) CDTEST - There are two phases of CDTEST. The first phase writes a count, count bar*, and random numbers throughout memory and reads and compares each memory load for accuracy. Before it reads to compare, it waits eight seconds in order to test the MOS refresh facility. After waiting and reading, it waits eight more seconds and reads again for each memory load. The second phase writes count, count bar, random and random bar into memory, then reads and compares without waiting.

To call these tests, the user must sign on the system and push the buttons:

ØP C D T E S T (/)

This sequence will cause both the PADTEST and the CDTEST to run. The second phase of CDTEST will run continuously until the user pushes STØP on the control panel. The first phase runs just once.

Failures

SPIN IN IA23: The value read from a pad cell was not the value written.

Procedure: Push STOP on the control panel, and record PA, PD, and A2; then push IA INC and RUN.

SPIN IN IA54: The value read from a memory cell was not the value written.

Procedure: Push STOP on the control panel, and record CA, I1 and A2; then push IA INC and RUN.

2. DISKTEST, SEEKTEST

a) DISKTEST - Writes and reads 16-bit count and count bar, random and random bar, then 18-bit count and count bar, random and random bar,

*bar indicates a complement

according to the code word specified in the call. The above functions are written on every track of cylinders one through 199, unless the user specifies otherwise in the call (see below). The version of DISKTEST which runs during SYSTEST works as follows: Upon reading to compare and funding a failure the error is recorded in the log and the track is re-read and compared up to eight times. If it still fails the comparison after the eighth re-read, the diagnostic goes on to the next track, after entering a message (in the error log) that the previous track has failed the test.

b) SEEKTEST - Seeks cylinder zero, then a random cylinder, then one, then a random cylinder, etc. up to cylinder 199.

The diagnostic disk pack must be mounted before the disk tests are called. This may be done before, after, or during the sign on procedure. To call the disk tests, the user must push the following buttons:

ØP DISKTEST _ CODE [,starting cylinder no., final cylinder no. [;head no.]]

where:

Code is an octal number whose bits have the following interpretations:

1	Bit 0 (LSB)	= 1 = >Disable WRITE (if doing read only, the comparison is disabled)
2	Bit 1	= 1 = >Disable READ
4	Bit 2	= 1 = >Disable SEEKTEST
10	Bit 3	= 1 = >Disable COUNT
20	Bit 4	= 1 = >Disable COUNT BAR
40	Bit 5	= 1 = >Disable RANDOM
100	Bit 6	= 1 = >Disable RANDOM BAR
200	Bit 7	= 1 = >Loop forever in write/read portion (SEEKTEST disabled)
400	Bit 10	= 1 = >Disable 16-bit portion
1000	Bit 11	= 1 = >Disable 18-bit portion

Cylinder numbers (optional) are specified in decimal. [It is possible to specify cylinder zero in the parameters for DISKTEST. In this case, the user must turn off the disk-protect switch on the maintenance panel before running the diagnostic.]

The head number is only allowed if cylinder numbers are specified. There are 20 heads, number 0-19.

It is possible, at any time during execution of this test, to disable the printing of error messages by setting E3(10)=1 on the control panel. It is not necessary to stop the machine to do this; merely touching the button will set the bit and touching it when the bit is on will clear it, and allow error messages to display again.

Examples

1) ØP DISKTEST _ 0 (↵)

would do the complete DISKTEST and SEEKTEST program.

2) ØP DISKTEST _ 144,31,40 (↵)

would write and read, in turn, 16-bit count and count bar, then 18-bit count and count bar, on every track on cylinders 31 through 40, and would not do the SEEKTEST when through.

3) ØP DISKTEST _ 601,135,135;2

would read, in 18-bit format, one track; cylinder 135 decimal, head 2 (or track number 2702 decimal). No data comparison would be made; the program would keep reading the track over and over. It should be noted that it is possible to get a checksum error if the program is reading in 18-bit format a track which was written in 16-bit format (or vice-versa). The message STATUS ERRØR...STATUS=1 appearing continuously is an indication of this. To remedy, stop the test, sign on, and specify read-only in the other format (for instance, 16-bit format in this example).

Failures

In case the data read does not compare with that written, a message will be displayed showing the error and pertinent information about it. After trying 8 times, the test will be continued. Other errors cause:

SPIN IN IA33: Status was not clear at a time when it should have been. E1 will contain the status.

SPIN IN IA66: Received an interrupt on a WRITE and IØDRDY≠0

SPIN IN IA65: Received an interrupt on a READ and IØDRDY≠1

SPIN IN IA32: Status not clear after a SEEK. E1 will contain the status.

SPIN IN IA40: No interrupt after READ HEADER mode was entered and the READ command was given.

Procedure: Push STØP on the panel, and record IA, E1, and I2. Then set PA=1 and record PD, then set PA=13 and record PD. Finally, push RESET INITIATE RUN on the panel and you may call the test again.

The message "DØNE" will be printed on the screen upon completion of the test.

3. APTEST

APTEST - Writes and reads every cell of the AP's S-Pad, Data Pad & MD and checks that every bit can be both a one and a zero. Then it reads Data RØM and compares it against an image on disk.

The APTEST which runs during SYSTEST is slightly different. It writes and reads first whatever is in memory at the time it executes, then count and count bar, random and random bar. The first time it runs during each pass of SYSTEST, it will write and read the random bar numbers left from CDTEST, and each pass thereafter it will write and read the Data ROM image.

There are two options for calling the APTEST. It can be executed once by pushing

ØP A P T E S T (↵)

or it can be caused to repeat indefinitely by pushing

ØP A P T E S T _ R (↵)

Failures

SPIN IN IA64: A word did not compare after being transferred over and back.

Procedure: Push STOP on the control panel and record I1, A2, CA, and PD. Then push IA INC, RUN on the panel.

SPIN IN IA70: A cell of S-PAD did not compare with what was transferred over.

Procedure: Push STOP on the control panel, and record E1, A2, and CA. Then push IA INC, RUN on the panel.

4. SYSTEST

SYSTEST is an acceptance test which uses the above diagnostics to determine whether the system is usable or not. It runs each of the above diagnostics except SEEKTEST repeatedly, and records any errors found in an error log on the system disk pack, as well as displaying on the screen the pertinent information about the errors. The format of these displays is, in general, first the element which failed (for instance CD), then the address, the expected value, and the value read. If the screen no longer has room for more error messages, the test continues running and recording errors, but they are no longer displayed. When the error log on disk becomes full, the test is aborted and the system enters a spin state at a CA near 131260.

The SYSTEST runs the CDTEST 512 times per pass, the DISKTEST (without SEEK) once per pass and the APTEST 892 times per pass. This amounts to 2048 complete CD write/reads per pass, 8 complete write/reads of the disk per pass and 4460 complete MD write/reads per pass. A complete pass takes about 35 minutes.

The SYSTEST program will run until the error log becomes full, and will print on the screen (if there is room) when it starts each pass. The diagnostic pack must be mounted to run SYSTEST.

To call this hardware acceptance test, the user must sign on the system and push the buttons:

ØP S Y S T E S T (↵)

To display the error log recorded by SYSTEST, the user must sign on the system and push the buttons:

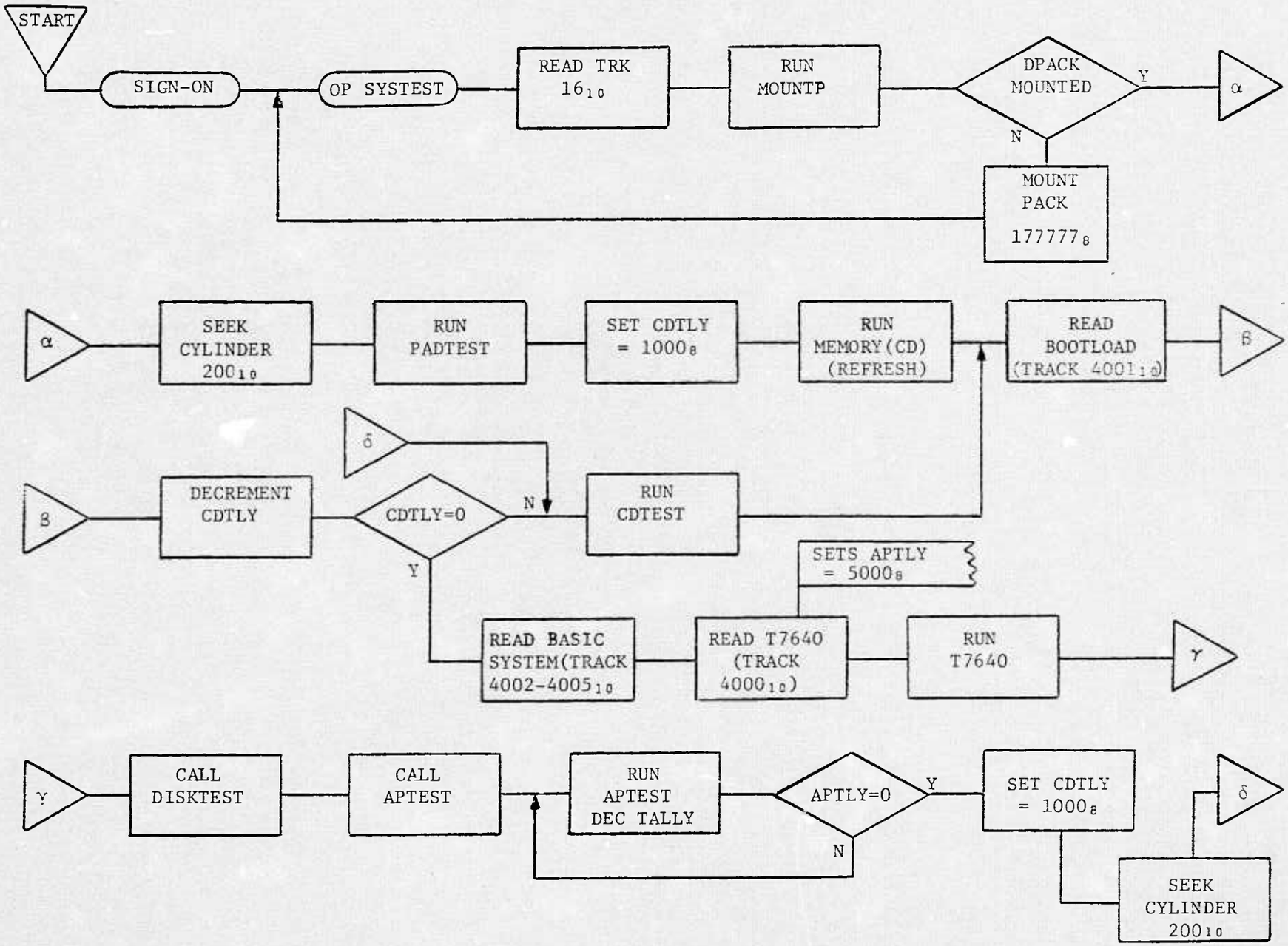
ØP P L Ø G (↵)

This will cause first the total number of passes of SYSTEST which ran to be displayed, then the errors grouped according to pass number, until either the end of the error log is found or the display screen becomes full.

In the latter case, the next page may be displayed by pushing the (↵) key, and in the former the message

DØNE...ENTER P TO PURGE LØG

will appear. The error log will be purged if a "P" is entered at this point; if any other character is entered the log will not be purged and the next running of SYSTEST will add its error messages, if any, onto the log already in existence.



SYSTEST

CHI 10/10/73

APPENDIX B

B.1 Library System Error Messages

Document Operations	Operation
1. NØ LIBRARY DECLARED!	INSRT, DEL, DISPLAY

There is no current library declared. Use the LIBE operation to declare the desired library.

2. NØ TEXT LIBRARY DECLARED!	LOAD, STORE

There is no current text library declared. Use the LIBE operation to declare the desired text library.

3. NØ NAME GIVEN	ALL DOCUMENT

No document name was specified. Repeat the operation specifying the name of the document.

4. "docname" IS NOT DEFINED!	LOAD, STORE, DEL

Document "docname" is not defined in the current library. If the correct library has been declared as the current library, the document name must be defined using the INSRT operation.

5. TEXT BUFFER IS EMPTY!	STORE

A STORE document operation was attempted with an empty text buffer. The operation was not performed.

6. NOT ENOUGH ROOM ON PACK!	STORE

There are not enough unallocated contiguous tracks available on the system pack to store the current text buffer. To make more room on the pack, delete any temporary libraries which are not immediately needed using the DEL LIBE operation.

7. LIBRARY STORE FAILED!!	STORE

Inconsistent information was detected during allocation of disk space for the document. This is system error. The system must be restarted from the machine panel. If recovery is attempted, the current text buffer contents will be valid; however, the current text library cannot be stored into.

8. SOURCE PAGE DOES NOT EXIST!	LOAD

The specified starting source page in the LOAD document operation does not exist.

9. ONLY 10 PAGES ALLOWED IN THE TEXT BUFFER!	LOAD

The specified destination page number was greater than ten.

Document Operations	Operation

10. "docname" ALREADY EXISTS — INSRT

    The "docname" to be defined already exists in the current library.

11. "docname" WAS NOT STORED — DEL

    Document "docname" is defined, but unused, hence there is no copy to delete. If the document entry is to be removed, specify :E after the document names.

12. -----SYNTAX ERROR — DEL, MOVE

    The operation parameters were incorrectly specified. See Section 2.3 for a brief summary of the proper syntax for each operation.

13. DOCUMENT "docname" IS NOT DEFINED — DISPLAY

    Document "docname" is not a member of the current library. The operation (DISPLAY) (/) will print a list of all documents which are defined in the current library.

14. DOCUMENT "docname" PAGE "n" DOES NOT EXIST — DISPLAY

    The page "n" which was requested does not exist in document "docname."

15. DOCUMENT "docname" - UNKNOWN TYPE! — DISPLAY

    A program library document is not a recognized type (SYSTEM, AP-PGM). This error should not occur.

16. "libename" does not exist! — LIBE

    No library with the requested "libename" has been defined as a temporary library. (DISPLAY) (LIBE) (/) will print a list of all temporary libraries which are defined.

17. "libename" is not a valid type library! — LIBE

    The library "libename" is defined, but its type is not TEXT, PGM or DATA. This error should not occur.

20. LIBE DOES NOT EXIST! — MOVE

    Either the source or destination library named is not defined as a temporary library.

21. TYPES DO NOT AGREE — MOVE

    The source and destination libraries are not of the same type.

22. Header type error! MOVE

The source document was not a recognized type.

23. IMPOSSIBLE ERROR! MOVE

Inconsistent information was discovered during the MOVE operation. The operation was aborted. The destination library should not be stored or moved into.

Library Operations

24. SYNTAX ERROR libename:t where t is T,P,D,S pack# or R pack# INSRT LIBE

The library type, if temporary, or class (Shared or REFERENCE) and the desired pack number, if permanent, must be specified as shown. See Section for a more complete description of these parameters.

25. "libename" ALREADY EXISTS ON THIS PACK! INSRT LIBE

A library with the given "libename" already is defined on the specified pack. A new library name or a different pack should be selected.

26. "libename" CANNOT BE DEFINED ON THIS PACK! INSRT LIBE

There are no disk tracks available on the specified pack or the maximum number of libraries allowed for on the pack already have been defined. Delete one or more existing libraries from the pack or select a different pack.

27. LIBE IS UNDEFINED DEL LIBE

The library to be deleted does not exist in the appropriate directory of libraries.

28. INCORRECT PASSWORD DEL LIBE

The password response does not match the password associated with the library.

29. IMPOSSIBLE! DEL LIBE

The library name was formed in the appropriate directory of libraries, but was not found in the table of contents for the disk pack on which it was supposed to be defined. This error should not occur.

30. "code" CODE NOT FOUND! DEL LIBE

The allocation code listed in the library's directory was not listed as assigned to any library.

This error should not occur.

31. {TEMP / REFERENCE / SHARED} LIBRARY DOES NOT EXIST — DISPLAY LIBE

The library name given is not defined in the appropriate directory of libraries.

32. "libename" LIBE DOES NOT EXIST! — MOVE LIBE

Library "libename" is not defined in the appropriate directory of libraries.

33. Pack full, only part of LIBRARY moved — MOVE LIBE

Insufficient space was available on the destination pack to move the entire library. The operation was completed with some documents in the library not moved. Use DISPLAY LIBE to print a list of the documents moved. To obtain space for the entire library, delete one or more other libraries which are located on the pack.

34. Header type error! — MOVE LIBE

A document was encountered whose type is not recognized. The MOVE was aborted. This error should not occur.

35. ACCESS DENIED — MOVE LIBE

The password response did not match the library's password.

36. IMPOSSIBLE ERROR! — MOVE LIBE

Inconsistent information was discovered during the operation. The operation was aborted. The destination library should not be used. Instead, repeat the move operation.

System Operations

37. IS NOT A VALID OP — OP

The operator name specified is not recognized as a system operator. (OP) (PRINT) (/) will print a list of all recognized operators.

38. NO PROGRAM LIBRARY SPECIFIED — PGM

No current program library has been declared. Use the LIBE operation to declare the desired library.

39. NO PGM NAME GIVEN — PGM

No program name was specified.

40. NO SUCH PGM IN CURRENT LIBRARY PGM

The program name specified is not defined in the current program library.

41. IS NOT A VALID SYSTEM PGM

The program specified is not a SYSTEM type program.

42. SYST SA does not agree with BBASE PGM

The program specified was assembled with a starting address which was not equal to BBASE. In order to use it as a program, it must be re-assembled with the base address specified in its definition document equal to BBASE.

43. Syst not run PGM

The program specified was not executed.

B.2 Edit Error Messages

1. "EDIT...Word Øverflow"

   There are too many words on the current line. To resolve the problem, start a new line.

2. EDIT...Line Øverflow

   There are too many lines in the Edit buffer. To resolve, store the Edit buffer and clear it out before continuing.

3. EDIT BUFFER ØVERFLØW

   There is too much data in the Edit buffer. To resolve, store the Edit buffer and clear it out before continuing.

4. Text Buffer Øverflow

   The text page has too much data. To resolve, begin a new page.

5. Page Undefined

   The user caused a page to be displayed which did not exist. The purpose of the message is merely to notify the user that the page does not exist.

6. Text Buffer Pointer Error

   The text pointers were moved too far apart. To resolve the problem, access a smaller portion of the Text buffer.

B.3 Disk Controller Messages

1. MOUNT PACK n

   Disk pack n was required for a requested disk action, but the pack was not locateable on any available drive. The job or other process requiring the pack was aborted.

   Response: Mount the requested pack on a disk drive and make the drive ready. To avoid this message, mount disk packs before they are needed. If the pack to be mounted is the system residence pack, the system must be restarted from the computer panel.

2. DISK CHECKSUM ERROR

   A disk track being read had a computed checksum word which differed from that computed when the track was last written. Error recovery is performed automatically by re-reading the track up to seven times. If not successful after these retries, the disk controller will abort the current job with a "DISK ERROR S 0" message.

   Response: Occasional CHECKSUM ERROR messages are normal, corresponding to the manufacturer's rating of the disk drives. If a retry of the read is successful, the data will be correct. If the retries are not successful, and the read is aborted, the information on the track is normally not recoverable.

3. DISK ERROR ...FORMAT PACK=p TRACK=t

   An attempt was made to read a disk track in a format (16 or 18 bit) different from that in which it was written. The job or other process requesting the read is aborted.

   Response: This error most frequently occurs when pointer information to the disk track has been destroyed.

4. DISK ERROR ...LENGTH PACK=p TRACK=t

   An attempt was made to write more information on a disk track than there was room for. The limits are 3600_{10} 16-bit words or 3200_{10} 18-bit words per track. The job or other process requesting the write is aborted.

   Response: Decrease the number of words being written. Again, the Error may result from destruction of reference data.

5. DISK ERROR ...TRACK PACK=p TRACK=t

   An attempt was made to access track t , which was out of range. The maximum track number which can be read is 4059_{10}. The job or other process requesting the access is aborted.

   Response: If the request made was too large, decrease it. Otherwise, the error may result from destruction of reference data.

6. DISK ERROR... POSITION PACK=p TRACK=t

The track number recorded in the disk track header of the track selected did not agree with the track requested. The job or other process requesting the access is aborted.

Response: This error indicates a hardware failure in the disk drive and should be reported for corrective action.

7. DISK CHECK FAILED

The data recorded on a disk track did not match the information in MOS when read immediately after writing. The system must be restarted from the computer control panel.

Response: This error may occur either through failure of the MOS memory or a recording error which was not detected by the checksum code. The contents of PD 41-47 should be recorded before restarting the computer.

8. DISK ERROR...STATUS PACK=p TRACK=t

A non-zero error status word w was received from the disk. The error status codes are given in the appendix to TMA4: MP-32 Microprogramming Manual. The job or process requesting the disk action is aborted.

Response: The error status word w should be recorded, together with any status information printed, and reported for corrective action.

CHI-TN-73-005
11/28/73

SIGNAL SYSTEM ACCEPTANCE TESTS

Execution of the acceptance test for the signal System's mathematics may be initiated from the console by entering:

(pgm) MCHK_# cycles (/) or (pgm) FPTEST # cycles (/)

One cycle of the acceptance test MCHK takes 25 minutes. Each cycle of FPTEST takes 2.5 minutes. MCHK includes subtests C,D,E,F and G; FPTEST includes subtests A and B. A summary of the various tests is given by Table 1.

This acceptance test is a consistency check of the Signal System's mathematics. If the consistency criterion of any subtest fails to meet previously established consistency criterion (which have been permanently stored in the program), an error message will be printed and logged.
For the floating point tests, the consistency criteria are the sums of the errors; for the FFT, IFT and convolution tests, the consistency criteria are check-sums (the bit-by-bit exclusive-or of all the errors). A more detailed description of each subtest and the error computation it employs is found below.

A. FPMATHCHK

FPMATHCHK utilizes pre-selected input to test each of the hardware options in floating point arithmetic (add, multiply and subtract). For this test the consistency criterion is the sum of the differences between the calculated values and the expected values. The inputs and the expected outputs are given in Table 2.

B. FPØPCHK

FPØPCHK utilizes random number input for eight floating point operator identity tests. For each identity, 32,767 relative errors are calculated. In each test, the consistency criterion is the sum of the errors. The identities are:

1. $\ell n(X_1X_2)=\ell n(X_1/X_2)+2\ell nX^2$
2. $\ell nX^{1/2}=1/2(\ell nX)$
3. $e^{x_1}e^{x_2}=e^{(x_1+x_2)}$
4. $(e^x)^3=e^{3x}$
5. $X^5=e^{5\ell n\ x}$
6. $\sin^2x + \cos^2x=1$
7. $\arctan X_1 + \arctan X_2 = \arctan (X_1+X_2/1 \cdot X_1X_2)$
8. $\arctan (\sin X/\cos X)=X$

C. FFTFFTCHK

FFTFFTCHK checks the consistency of the FFT with itself. Each input function generated is a Kronecker delta function ($\delta_j(n)=0$ for $n\neq j$, $\delta_j(n)=1$ for $n=j$) and is 4096 complex points (1 complex point = 32 bits) long. A total of 2048 different input functions ($j=0,\ldots,2047$) are transformed. The complex FFT (the FFTC) of each input function is conjugated and the FFTC of the result calculated. Except for a multiplicative constant (the delta scale factor), the final transform is identical to the original input function. For each of the 2048 input functions, the absolute value of the difference between the input function value $\delta_j(j)$ and the transform value $A_j(j)$ is determined (denoted $\varepsilon 1(j)$), and the bit-by-bit exclusive-or of all $A_m(n)$, $n\neq j$ is determined (denoted $\varepsilon 2(j)$). Finally, for all 2048 input functions, the bit-by-bit exclusive-ors of $\varepsilon 1$ and $\varepsilon 2$ are calculated. The consistency criteria for this test are these last two check-sums.

D. FFTERRCHK

FFTERRCHK utilizes the following input and output functions: $f(k)=\left(\frac{1}{2\pi}\right) e^{\frac{i\pi kj}{n}}$ and $F(s)=\delta(s-j)$, where $k=0,\ldots,n-1$ and n=blocksize. For each input function, this test compares the complex FFT with the analytic result by computing the difference between the transform value $A_j(j)$ and the predicted value $\delta_j(j)$ (denoted $\varepsilon 1(j)$) and computing the bit-by-bit exclusive-or of all the $A_j(n)$, $n\neq j$ (denoted $\varepsilon 2(j)$). It also computes the IFT (FFT) and compares it with the original input function by computing the bit-by-bit exclusive-or of the difference (denoted $\varepsilon 3(j)$). Five input function blocksizes are exercised. They are, respectively, 16, 64, 256, 1024 and 4096 points long. For a given blocksize N, $(N/2)LOG_4(N)$ passes are made through the calculation, as j goes from 0 to $(N/2)-1$ in steps of $LOG_4(N)$. After the last pass, the bit-by-bit exclusive-ors of $\varepsilon 1$, $\varepsilon 2$ and $\varepsilon 3$ are calculated. The consistency criteria for this test are these last three check-sums.

E. IFTERRCHK

IFTERRCHK utilizes the input step function

$F(k)=\begin{cases}1, k<L\\ 0, k\geq L\end{cases}$ whose inverse transform is

$$f(t)=\left(\frac{L}{N}\right) e^{\frac{i\pi t(L-1)}{n}} \left(\frac{\text{Sin}\ \frac{\pi tL}{n}}{\text{Sin}\ \frac{\pi t}{n}}\right), \text{ where } t=0,\ldots,n-1 \text{ and } n=\text{blocksize}.$$

This test compares the real part of the IFT with the analytic result (from the equation above) calculated in floating point. Five input function blocksizes are exercised. They are, respectively, 16, 64, 256, 1024 and 4096 points long. For a given blocksize N, $(N/2)LOG_4(N)$ passes are made through the calculation, as L goes from 1 to N/2 in steps of $LOG_4(N)$. On each pass, the bit-by-bit exclusive-or of the difference between the inverse transform value and the predicted value is determined (denoted $\varepsilon 1(L)$). After the final pass, the exclusive-or of $\varepsilon 1$ is calculated. The consistency criterion for this test is this final check-sum.

F. FFTRX2

Using four 2K (K=1024 pts) real delta functions, FFTRX2 checks the consistency of the real FFT (the FFTR) using an odd power of two blocksize.

S_1 = K-delta at T
S_2 = K-delta at 2T-1
S_3 = K-delta at T+1
S_4 = K-delta at 2T-2

The delta functions are interleaved in pairs to form two 2K pseudo complex input functions i.e. S_1S_2 and S_3S_4. FFTRX2 compares the S_i (for i=1,2,3,4) portion of FFTR (S_1S_2, S_3S_4) with FFTC(S_i) (the FFTC of each input function taken separately). 1024 passes are made through the calculation, as T goes from 1 to 1024. On each pass, for each i, the bit-by-bit exclusive-or of FFTR(S_1S_2,S_3S_4) (S_i)-FFTC(S_i) is determined (denoted εi). After the 1024th pass, the bit-by-bit exclusive-or of each εi is calculated. The consistency criteria for this test are these last four check-sums.

G. CNVERR

CNVERR checks the consistency of CØNVØLVE by means of the following identity: f*h = IFT(FFT(h)·FFT($\bar{f}$)). The input functions are:

$$f = e^{-5\left(\frac{k}{15}\right)^2} \Big/ \sum_{k=-15}^{15} \left| e^{-5\left(\frac{k}{15}\right)^2} \right|$$

$\bar{f}$=f expanded to 4096 points with negative values of k mapped into 4096+k points.

$h_j = \cos(2\pi Nk/64)$ where j=64N+k. $\bar{f}$ and h are 4096 real points long; whereas f is 31 points long. Only one pass is made through the calculations. On this pass, the bit-by-bit exclusive-or of the difference between CØNVØLVE (f,h) and IFTC (FFTR(h)·FFTR($\bar{f}$) is determined. This check-sum is the consistency criterion for this test. This test is repeated 40 times during any cycle.

Table 1

TEST	BLOCKSIZE	INPUT	OPERATION	REPETITIONS
A. FPMATHCHK		preselected	Floating point ADD, SUBT, MULT	1
B. FPØPCHK		random numbers	Floating point operators DIV, SQ, INV, SQRT, LN, EXP, SIN, COS and ATAN	32,767 for each of 8 identities
C. FFTFFTCHK	4096 points	δ_j, j=0,...,2047	complex FFT(the FFTC)	1
D. FFTERRCHK	16,64,256,1024 and 4096 points	$\left(\frac{1}{2\pi}\right) e^{\frac{i\pi kj}{n}}$, k=0,..., n-1 and n=blocksize	FFTC,IFT and floating point MULT,SIN, DIV,ADD,COS	5 cases
E. IFTERRCHK	16,64,256,1024 and 4096 points	$F(k)=\begin{cases}1, k<L\\0, k\geq L,\end{cases}$	IFT and Floating Point SIN, COS, MULT, DIV, ADD	5 cases
F. FFTPX2	4x2048 points	$S_1=\delta_j, S_3=\delta_{j+1}$ $S_2=\delta_{2j-1}, S_4=\delta_{2j+1}$ j=1,...,1023	Real FFT and Complex FFT	1
G. CNVERR	4096 points x 31 points	$f=e^{-5\left(\frac{k}{15}\right)^2}\Big/\sum_{k=-15}^{15}\left|e^{-5\left(\frac{k}{15}\right)^2}\right|^2$ $h_j=\cos 2\pi Nk/64$ where j=64N+k	CONVOLVE, Real FFT,IFT, and Floating Point EXP,MULT, ADD	40

Table 2

ADDITION		SUBTRACTION		MULTIPLICATION	
Input	Output	Input	Output	Input	Output
.5,.5	1.0	1.0,-1.0	2.0	-1.0,-1.0	1.0
.5,.5+ε	1.0+ε	.5,.5-ε	1.0+ε	1.0-ε*1.0-ε	1.0-2ε
-1.0,-1.0	-2.0	-1.0,1.0-ε	-2.0	$385*2^{-9}*385*2^{-9}$	$148225*2^{-18}$
-2.0,-1.0+3ε,	-3.0+ε	-2.0,1.0-3ε	-3.0+ε	-1.0*.75	-.75
.5,1.0	1.5	2.0-ε,.5	1.5-ε	$385*2^{-9}*.74-3*2^{-14}$	$-284.9*2^{-23}$
1.0,.5+ε	1.5+ε	2.0-ε,.5-2ε	1.5	.5*.5	.25
-2.0,.5	-1.5	-1.0,1.0	-2.0	$257*2^{-9}*513*2^{-10}$	$131841*2^{-19}$
-4.0,.5+3ε	-3.5+ε	-1.0+3ε,2.0	-3.0+ε	-1.0*.5	-.5
2.0,-1.0	1.0	1.5,1.0	.5	$257*2^{-9}*-.5+2^{-14}$	$-128.5*2^{-23}$
2.0,-1.0+ε	1.0+ε	2.0,1.0-ε	1.0+ε		
-1.0,.5	-.5	.5,1.0+ε	-.5-2ε		
-4.5,.5+3ε	-4.0+ε	1.0-ε,4.0	-3.0		
1.5,-1.0	.5	1.0-ε,4.0	-3.0		
1.5,-1.0+ε	.5+ε	$2^{21}+ε,2^{21}$	.5		
-4.0,3.0	-1.0	$2^{22},2^{22}-ε$	.5		
-2.5,1.5+ε	-1.0+2ε	$2^{22},2^{22}+ε$	-1.0		
-1.0,1.0	0	$2^{23}-ε,4.0$	-1.0		
-1.0+ε,1.0	2^{-10}	$2^{127}-ε,2^{127}-ε$	0		
		$2^{12},2^{12}-ε$	2^{-11}		

For the above input and output $ε=2^{-23}$

CHI-TN-74-010
Jan Vanderford
3/6/74

DISK PACK INITIALIZE PROCEDURE

In order to use a disk pack in a CHI system, it is necessary to format the pack; that is, to write the software header on every track. A sequential count will be written on the data portion of every track. The software header is comprised of two words, the pack number and the track number. The pack number is identical on all tracks of a pack, while the track number is a sequential count from zero to 2059. Initialization of a pack is not something to be done casually and should have to be done only once for each pack. If for some reason a particular software header should degrade or become incorrect, it should be corrected individually by using OP _ WRTHDR (see Appendix C of the User Manual, TMA2).

The following steps should be done to initialize an unformatted disk pack:

1. Sign-on and insert a temporary program library and define it as your current program library.

2. Move the most up-to-date version of the shared library PGM:S into the temporary program library.

3. Call (PGM) DASDI ↓ (DASDI stands for Dynamic Access Storage Device Initialize). The macroprocessor should spin in IA=76.

4. Reset the MP-32A by pushing the red RESET button on the front panel.

5. Mount the pack to be initialized. Be sure it is the desired pack, as it will be totally written. Notice the drive number of the drive that the pack is mounted.

6. Power-down the other disk drive, using the white button on the drive front.

7. At the front panel of the MP-32A, load the following registers with the desired data.

   Load IA with 10_8

   Load E1 with the pack number for the pack to be initialized

   Load E2 with the drive number from 5 above

   Load E3 with 10_8 if a check is desired following the initialization
   with 0 if no check is desired.

8. Move the disk protect switch(es) (located on the MP-32A maintenance panel) to the center position to allow disk writes on all cylinders.

9. Push RUN on the front panel. The program should be running and the disk heads should be stepping thru cylinders. The program should finish in 2-3 minutes (double if check is enabled).

10. When the program has run to completion, move the disk protect switch(es) back to the protected position. Power-on the other disk drive. The intialization of the unformatted pack is complete.

CHI
CULLER-HARRISON INC.
~~5770 Thornwood Drive~~, Goleta, California 93017
150-A Aero Camino
Telephone (805) ~~967-0424~~
968-1064

CHI-TN-74-003R
October 29, 1975

SYSTEM RELOCATEABILITY ON THE SYSTEM PACK

Michael McCammon/Dale Taylor

The following system components are collected on one cylinder:

1. Resident System Nucleus (SYSONE)
2. Editor programs
3. Source for interactive EDIT and LIBE tracks
4. Libe programs
5. All system OPS and OP symbol table

These normally reside on cylinder 0 of the system pack, and must reside there in order to use the ROM bootstrap directly to load the system.

However, all system references to these components on disk are relocated by adding the contents of a variable called SYSBASE, which is currently word 100011_8 of the resident system nucleus. Thus, it is possible to have more than one version of the system on one disk pack, whether for checkout or backup.

To establish a new system cylinder, it is necessary to allocate the desired tracks through the disk allocation system using an allocation code of 377_8. Then each component can be compiled and moved to the desired cylinder using the MOVETRK OP. The relative track numbers for the standard system components are:

0	Bootstrap Loader (BOOTSTRAP) - Not needed on alternate cylinders
1-3	System Nucleus (SYSØNE)
4-5	Editor (EDITØR)
6	Libe Interactive Track (LIBETRK)
7	Math Interactive Track (MATHTRK)
8-9	Unused
10	Libe operations (LIBEØPTRK)
11	Utility ops (UTILITY)
12	System compiler (CMP)
13	AP Assembler (APASM)
14	AP Assembler tables (APTABLES)
15	AP Utilities (APMOVE)
16	Diagnostic Routines (DIAGNSTCS)
17	AP Link Loader (APLINK)
18	Edit Interactive Track
19	Operator Symbol Table (Move from good copy)

To initialize a system from other than cylinder 0, the following steps are necessary:

a. Select Cylinder
   Press RESET.
   Enter cylinder number in E2.

b. Enter normal bootstrap
   Press INITIATE to enter ROM mode
   Press RUN.

System Disk Back Up Recommendations

Besides the read-only operating system components, which reside on cylinder 0, the following other information is vital to successful operation:

1) Tracks 20-21: Two copies of track allocation record. Track with larger first entry is current. Allocation record for track N is in word N/2 + 1. For system pack, these can be re-created in initial state (tracks 0-119 have code 377_8) as long as no permanent libraries are resident. It would also be necessary to allocate any tracks used for users' directory of libraries.

2) Track 22: Pack Table of Contents. A directory giving the location, type and allocation code of each permanent library directory residing on the pack. In addition, words 114_8 to 275_8 contain a list of allocation codes 14_8 to 175_8 , followed by 177777_8. Any codes which are assigned to permanent libraries on the pack should have bit 7 (200 bit) on.

3) Track 23: System Directory of Users. A directory giving the location of the user's directory of libraries for each user known to the system. This track should be backed up each time a new user is added.

4) Track 38: Directory of shared Libraries. A directory giving the pack number for each shared permanent library known to the system. This should also be backed up regularly.

5) Tracks 101 & 102: System Message Tracks. These may be restored by moving page 1 of MESSAGE0 and MESSAGE1 onto them. MESSAGE0 and MESSAGE1 are text documents in library BASIC.

6) User's Directories of Libraries: A directory giving the pack number of each permanent library defined by a user, as well as his password. These should be backed up periodically. Reconstruction of these tracks is theoretically possible, since the pack table of contents entry for each library contains the track number of the user or shared directory of libraries which points to it.

Reference Disk Pack Backup

Items 1 and 2 above exist on each reference pack. However, only the pack table of contents is problably worth backing up.

Summary

BackUp	When
1. System Directory of Users (23)	Each time new user defined.
2. Directory of Shared Libraries (38) Users' Directories of Libraries	Periodically-frequency depends on rate at which new libraries defined, old ones deleted.
3. Pack Table of Contents (22 on all packs)	Periodically as above.

CHI System Problem Report (SPR) Procedure

Who initiates the form?

The customer.

When is the form initiated?

Whenever the customer experiences a system problem during operation.

How is the form filled out?

The customer completes the following:

1. Name of person initiating the report and affiliation.
2. Date the problem occurred and site.
3. Brief description of the problem.
4. The state of the system before problem occurred.
5. The state of the system after problem occurred.
6. Whether the system is restartable.
7. Whether the problem is repeatable.
8. Any other comments which might assist CHI personnel in locating and correcting the problem.
9. Signature of initiator.

Where is the form sent?

The customer should make one copy of the form for his records and send the original to CHI, 150-A Aero Camino, Goleta, Ca. 93017.

What further action takes place?

CHI personnel: 1) Note "date received" in "CHI Use" box, 2) When action is started on problem, note date and initiator in box along with the document kind(s) and number(s) which describes the action -- Status Report (SR), and Software Change Order (SCO) or Engineering Change Order (ECO) as appropriate.

Customer: When customer receives a copy of the referenced Status Report, he should enter the action, SR# and date in the "Customer Use" box.

BASIC DISK COMMANDS

The following commands may be used to read or write data on disk, or to position the disk access arm. All assume the following parameters are set up in data pad. In addition, data pad cells 41 and 46-50 are used.

Pad Cell	Parameter
42	Pack Number
43	Track Number
44	A(Buffer)
45	Buffer Length

DRD16

Read in 16-bit mode. The track must have been written in 16-bit mode or a format error will occur. Data pad cell 50 will contain the actual track length. The flag portion of each data word in memory is set to 0.

DRD18

Read in 18-bit mode. The track must have been written in 18-bit mode or a format error will occur. Data pad cell 50 will contain the actual track length.

DRDQ

Read track in format in which it was written. Data pad cell 47 will contain 5 if the track is written in 16-bit format or 0 if it was written in 18-bit format. Data pad cell 50 will contain the actual track length.

DWRT16

Write track in 16-bit mode. The buffer length specified must be $\leq 7020_8$ or a length error will occur. In this format, only the data portion of each 18-bit word is written.

DWRT18

Write track in 18-bit mode. The buffer length specified must be $\leq 6200_8$ or a length error will occur.

DCHK16

Write track in 16-bit mode, followed by a read-back from disk and word by word comparison of the data written. An error message will be printed and the computer must be manually restarted if the data does not compare exactly.

DCHK18

Write track in 18-bit mode, followed by a read-back from disk and word by word comparison of the data written. An error message will be printed and the computer will have to be manually restarted if the data does not compare exactly.

DSEEK

A seek action is initiated to position the access arm of the disk drive on which the requested pack is mounted to the cylinder containing the specified track. Only pad cells 42 and 43 need be specified. Once the seek is initiated, the operation terminates and subsequent macro operations may be overlapped with the positioning. No data transfer takes place.

DRDR16

Data pad cell 47 must contain the record number of the record to be read. The track must be structured as a record structure track. The specified record will be transferred to memory at the address given. Data pad cell 50 will have the actual length of the record. If the track was not written in 16-bit format, an error message will be printed.

DRDR18

Data pad cell 47 must contain the record number of the record to be read. The track must be structured as a record structure track. The specified record will be transferred to memory at the address given. Data pad cell 50 will have the actual length of the record. If the track was not written in 18-bit format, an error message will be printed.

Basic Disk Allocation Commands

The following commands may be used to allocate or release tracks on a disk pack. All assume the following parameters are set up in data pad, and all use pad cells 37-50.

Pad Cell	Parameter
33	Pack Number
34	Number of Tracks
35	Allocation Code
36	Buffer Address

GETTRK

Allocates a track or set of tracks using the allocation code passed in pad 35, and returns the starting track number in pad cell 32. There is one argument, which is the exit for the case that the requested allocation could not be made.

Example: <GETTRK, >NØRØØM

This would attempt to allocate the number of tracks indicated in pad cell 34 and exit to label >NØRØØM if it were not possible.

RELTRK

Releases a track or set of tracks which are described by the above parameters with the addition of one more: pad cell 32 contains the starting track number. It will set pad cell 35=0 if the tracks are successfuly released.

RELCDTRK

Releases the entire set of tracks on a pack with a specified allocation code. Uses all of the parameters listed above except the number of tracks.

CHI-TN-74-14
April 10, 1974
McCammon

SYSTEM MESSAGE FACILITY

A system message facility is supported which allows any program to call for the printing of any one of a set of system messages on the current display line. All system messages end with a carriage return.

System Message Request

CALL, MESSAGE, message #

The message specified is printed on the display screen, beginning at the current character position.

System Message Listing

The system messages are defined in documents MESSAGE0 and MESSAGE1 in the BASIC library. Each line of the first page of these documents represents one message. The message number of a message is the line number where that message appears in the document, beginning with line 1. Message numbers for messages in document MESSAGE1 are 400_8 plus the line number.

Adding New System Messages

To add a new system message, edit either document MESSAGE0 or MESSAGE1 by adding the new message as a line at the bottom of the first page. The document should then be stored back in the library BASIC. It is also necessary to update the appropriate system message track by moving a copy of the updated page to the track.

Message 0 resides on track 101_{10}

Message 1 resides on track 102_{10}

of the system pack. The suggested procedure is to use the MOVETRK OP to move the first track of the station's text buffer to the indicated message track.

CHI

CULLER-HARRISON INC.
~~5770 Thornwood Drive~~, Goleta, California 93017
150-A Aero Camino

Telephone (805) ~~967-0424~~
968-1064

CHI-TN-75-014
October 20, 1975

OPERATING SYSTEM PHILOSOPHY
SN/2
Dale E. Taylor

There are two modes of operation which the operating system uses: Interactive mode and Background mode. The system switches back and forth between the two modes according to the system environment at any particular time.

From the user's point of view, interactive mode is used for short, simple tasks such as editing, displaying a library directory or member, and loading and storing documents in a library. Background mode is used for larger, more significant and time-consuming processes such as the math system, moving a library, or running a user-defined job.

Memory is laid out in such a way as to allow both a background job and an interactive station to be resident at the same time (see Figure 1). Thus, even if a background job is interrupted to service an interactive user, the background area need not be swapped onto the disk.

When a user signs on, he is immediately assigned 13 tracks of space on the disk. These include his three interactive tracks (Edit, Library and Math) and 10 tracks for his text buffer. On the Edit Interactive track are the user's pointer information for both his text and edit buffers, and his entire edit buffer. On the Library Interactive track are the directory of temporary libraries as well as the programs for decoding library operation requests, library display, library load and store, insert and delete, and several other quick and simple library operations. On the Math Interactive track is simply the logic to begin the math job for the calling station.

There are two things which are common to all interactive tracks: the current statement (the statement being executed at the time) and the display parameters for typing on the screen (including character position, character size, etc.). When the interactive track changes because the user changes systems (by pushing the EDIT key from the library system for instance), the current statement and the station display parameters move from the first Interactive track to the second (Library track to Edit track for instance).

Processing of Keyboard Input

When a keyboard interrupt occurs, the keyboard interrupt handler reads every keyboard and writes every new key which has been input into a single keyboard input buffer, along with the number of the keyboard associated with the key. The keyboard interrupt handler's job is complete at this time.

The system timer, which gets control because of a timer interrupt approximately once every 87 milliseconds, examines the keyboard input buffer for new keys. Upon finding a key, the key is displayed on the screen associated with it, written in a keyboard execution list associated with the station from which it came, and examined for certain values; namely RESET, CANCEL, and Carriage Return. If the key is not one of these values, its processing is complete as far as the system timer is concerned, so the keyboard input buffer is further examined, and all keys are removed from it and passed out into their respective keyboard execution lists. If the key is one of the above values, certain processing occurs before the keyboard input buffer is examined further. In the cases of RESET and CANCEL, the keyboard execution list for the associated station is adjusted accordingly. In the case of Carriage Return, a flag is set whose meaning is, "invoke the interactive scheduler after the keyboard input list is exhausted."

Interactive Scheduling

The interactive scheduler has two principle functions. The first is to grant service to all interactive jobs which require it. Whenever the interactive scheduler is invoked, it places the system in "interactive mode", and does not leave interactive mode until all interactive requests have been fulfilled. The second function of the interactive scheduler is to remove from wait state all jobs whose requests for interactive service have been satisfied. So, for instance, if a background job had requested keyboard input and entered wait state until the input had been done, the interactive scheduler would remove the job from wait state so that the background scheduler could cause the job to resume execution.

Part of memory is reserved for the exclusive use of interactive jobs. This is the area from 62000_8 (called IBASE) to 70200_8. Another part of memory must always be available to an interactive job, and this is the area from 70200_8 (called IBUFF) to the top of memory. The entire area from 14000_8 (BBASE) up to 62000_8 is available for the use of the background job. (See Manual TMA3, Section 4.2.)

Initiating a Background Job

There are several ways to initiate a background job. For instance, many of the library system operators are run as background jobs by the system because they are rather time-consuming operations, hence should not be run as interactive jobs. Including in this category are "Move library", "Insert library", "Delete library", and a few others. Also, all "OP's" and user programs are run as background jobs, as well as the math system.

So when the user calls for any of the above operations, he has caused a background job to be initiated. What it means to initiate a background job is the following: the job-submission subroutine first compiles a block of relevant information (the job element) about the job including starting address, priority, disk tracks, size, etc., (see TN-73-003, TMA7) and places this job element on the system run stack, ordered according to the job's priority; at the same time the station from which the job was submitted is placed in an interactive wait state which simply means that the station will receive no more interactive service until the job has completed execution. The background scheduler will then execute the job when its turn comes. The job's turn will come when all the jobs which are on the run stack with a higher priority than the job in question have been removed from the run stack. Jobs are removed from the run stack when they have either terminated or entered wait state (wait state will be explained later). Once the job has terminated and been removed from the run stack, the background schedule will remove the submitting station from interactive wait state, allowing it interactive service once again.

Background Scheduling

The operating system's fundamental mode is to be running the background scheduler. This scheduler will execute the job with the highest priority which is ready to run (a job is ready to run if it has been placed on the run stack), if there are any jobs ready to run. It may be necessary to swap out an old job in order to make room for the new job to run; this is done by the background cycler.

Background jobs may both receive input from the keyboard and display on the screen of the station from which they were initiated. There are system macro calls to perform the various input and output commands which may be required (consult manual TMA3, "MP-32 Macro-Programming Manual").

It is possible that a background job may be interrupted while in execution. There are two types of interruption which can occur. The job may be temporarily interrupted in order to allow interactive service to be granted; if this happens, the job does not, in general, get swapped out onto the disk. The interactive and background areas of high-speed memory are not overlapped so that an interactive user may be resident at the same time as a background job. When a background job is interrupted to allow interactive service, all interactive service requests are processed before the background job is resumed. Interactive service may be granted any time that the system examines interrupts, which is between every two macro-instructions.

In addition to interrupting a background job to provide interactive service, the job may be interrupted to run another background job. The nature of this type of interruption is slightly different from the former in that while the former type of interruption may occur at any time in the job, the latter may only occur at specific times in the job, namely those times during which the job is waiting for some type of I/O to complete or else when the job has executed the BREAK macro. The BREAK macro is explicitly for the purpose of allowing other jobs with higher priority which

are on the run stack to execute. So, if the user prepares a job which does a large amount of computing with very little keyboard input, erasing the display screen, or DMA activity, and he wants to allow other jobs a chance to execute every so often, he should insert a call to BREAK every so often in his code (see manual TMA3, section 13.3). A request for an input statement from the keyboard when a statement is not yet completed (CALL GETSTMNT), erasing the display screen (ERASE), and waiting for the completion of a DMA transfer (WAITDMA) all cause the calling job to be put in wait state. Wait state is different from the state the job is in when BREAK is invoked in that any other job which is ready to execute may run, as opposed to only higher priority jobs. The job which entered wait state for any of the above reasons is automatically re-scheduled for execution by the job scheduler upon completion of the event for which it was waiting. When any of the above four macro calls (BREAK, CALL GETSTMNT, ERASE, and WAITDMA) are used, the job must be in a state in which it is swappable, i.e., the system must know the amount of memory space the job is using. (See Manual TMA3, section 4.2.) There is no other time when it is possible for the job to be swapped onto disk.

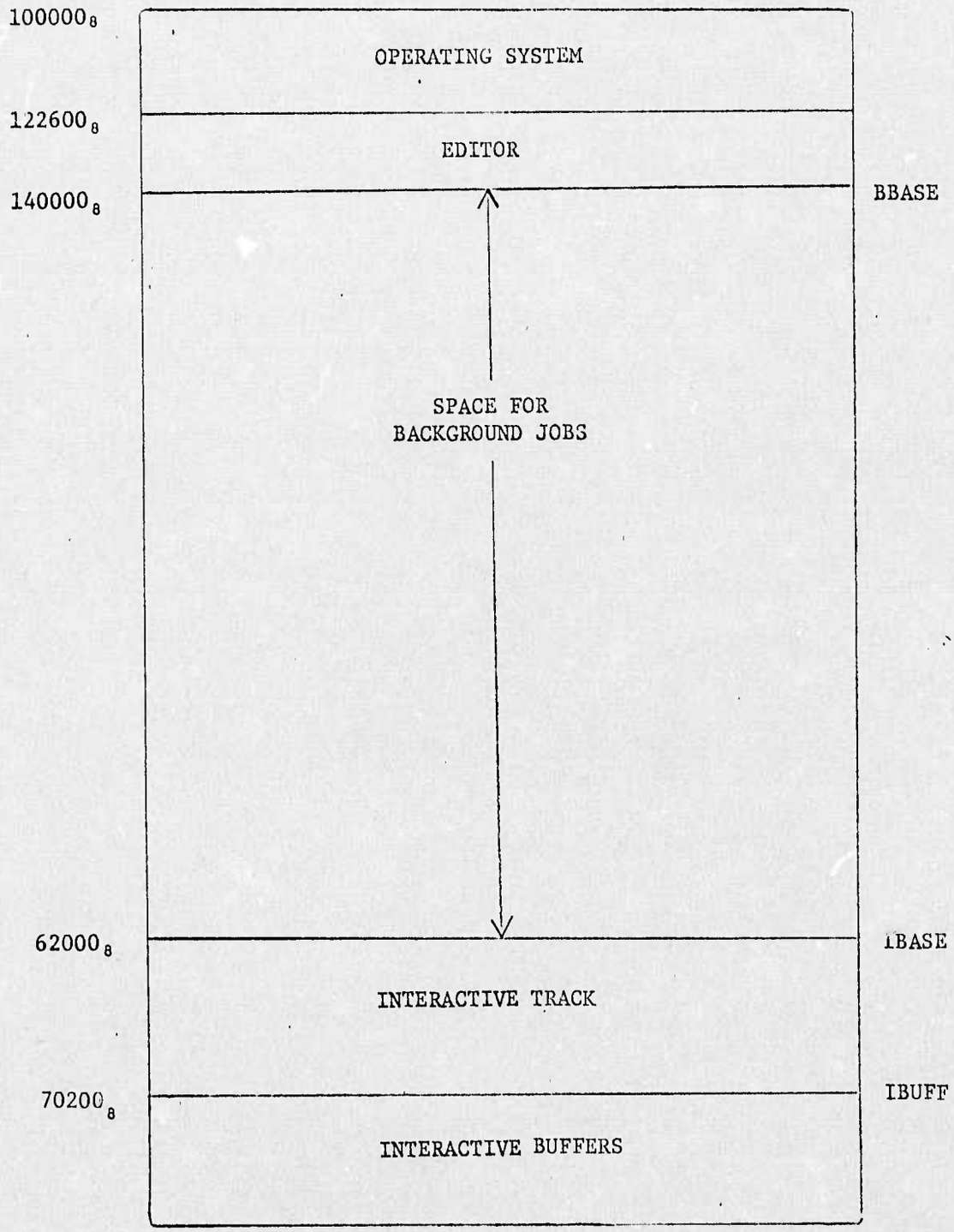


Figure 1. Memory Layout for Background Jobs

APPENDIX 7

Macroprogramming Manual

TMA3

CHI
SIGNAL SYSTEMS

SOFTWARE

TMA3 MP-32 MACRO-PROGRAMMING MANUAL

CONTENTS:

1. INTRODUCTION
2. THE MACRO LANGUAGE
3. THE SYSTEM ASSEMBLER
4. MACRO LANGUAGE PROGRAMMER'S GUIDE

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	4/15/74	p. 4-3 and 24 pages of the Appendix
B	6/4/74	p. 3-2 , 3-5a and 18 pages of the Appendix
C	10/23/75	pp. 2-1,3-6, and 9 pages of the Appendix

Address comments about this manual to:

C ller/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29106

© 1973
by Culler/Harrison, Inc.

Rev C

PREFACE

This document contains the information necessary for programming the MP-32 Macroprocessor in the macro language. Chapter 1 gives a brief overview of the MP-32 as a macroprogrammable computer. The programmable components of the MP-32 and the macroinstruction set are described briefly in Chapter 2, with the macroinstruction set fully described in the Appendix. Chapter 3 describes the System Compiler and its facilities for assembling both macroinstructions and microinstructions. Chapter 4 gives the information necessary for preparing macro language programs for execution as programs, operations, or systems. The complete set of macroinstructions is given in Appendix A.

This manual assumes familiarity with the contents of TMA2, the User Manual. Information on the preparation of new macroinstructions can be found in TMA4, the Microprogramming Manual.

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No. 29106
© 1973
by Culler/Harrison, Inc.

CONTENTS

		Page
PREFACE		ii
Chapter 1.	INTRODUCTION	1-1
Chapter 2.	THE MACRO LANGUAGE	2-1
	2.1 Introduction	2-1
	2.2 Data Pad Usage	2-1
	2.3 Main Memory (MOS)	2-1
	2.4 Operand References	2-1
	2.5 Subroutine Structure	2-1
	2.6 Detailed Instruction Description	2-2
Chapter 3.	THE SYSTEM ASSEMBLER	3-1
	3.1 Introduction	3-1
	3.2 Source Statement Organization and Character Set	3-1
	3.3 Symbols	3-1
	3.4 Self-Defining Term	3-2
	3.5 Location Counter Reference	3-2
	3.6 Expressions	3-2
	3.7 Data Types	3-2
	3.8 Statement Format	3-3
	3.9 System Assembler Pseudo Operations	3-5
	3.10 Using the System Assembler	3-6
	3.11 Assembler Error Messages	3-7
Chapter 4.	MACRO LANGUAGE PROGRAMMER'S GUIDE	4-1
	4-1 Background Jobs	4-1
	4-2 Program Structure for Background Jobs	4-2
	4-3 Processing Mode Control	4-3
APPENDIX		

Chapter 1. INTRODUCTION

The CHI MP-32 Macroprocessor is a high-performance general-purpose processing unit. The MP-32 is designed to be programmable at several different levels, with the level of programming used dependent upon the complexity and timing requirements of the task. In particular, the macro language described in this manual provides a convenient tool for the programming of most system processes. It includes the facilities of an assembly language for a multiple register computer and powerful table search, formatting and input-output instructions. The macro language can be easily extended by the definition of new macroinstructions by microprogramming the primitive instruction set of the macroprocessor. At this instruction level the full facilities of the macroprocessor may be used to create tailored macroinstructions for optimizing execution speed or programming convenience. The macro language level, on the other hand, permits rapid programming of complex tasks by eliminating much of the need for detailed concern with the internal structure of the processor, as well as generally decreasing the memory requirements for a program.

This manual describes those components of the MP-32 which are directly accessible with the standard macroinstruction set. A detailed description of the processor at the microinstruction level is given in TMA4.

Chapter 2. THE MACRO LANGUAGE

2.1

The macro language provides facilities for directly accessing Data Pad and MOS, the main memory unit of the MP-32. Access methods are provided which permit data transfers to or from the disk and other peripheral units. Operations using the AP-90 Array Processor may also be called for.

A macroinstruction consists of an operation code followed by any operand specifications required. The standard macroinstruction set provides for arithmetic operations on the registers and MOS, subroutine calls, conditional branches, table searches, format conversion, input/output with an interactive console, disk input/output, and AP operations. In addition, standard subroutines support disk library facilities, allocation of disk tracks and scheduling of separate processes.

2.2 DATA PAD USAGE

Data Pad is a set of 64 programmable 16-bit registers. Eight of these registers (70_8-77_8) are reserved for system usage in macroinstruction execution. The remaining 56(0-67_8) may be used as programmable registers. The reserved registers may be examined using macroinstructions, but should not be changed. Their usage is as follows:

Registers 70 - 73*	Work space
Register 74	Interactive console number
Register 75	Current mode
Register 76	Execution address macroinstruction address register
Register 77	Subroutine push-down stack pointer

2.3 MAIN MEMORY (MOS)

Main memory consists of 32K to 64K 18-bit words (16 data bits and two flag bits/word). This memory may be referenced for data operands, and addresses in the macro language. All macro instructions are executed from MOS.

2.4 OPERAND REFERENCES

Most macroinstructions permit operands to be specified in one of four ways:

(1) Directly, as part of the instruction.
(2) In a register specified in the instruction.
(3) In MOS, at an address specified in the instruction.
(4) In MOS, at an address given in a register.

Two flags, associated with each memory word, determine how the operand will be located.

---

* all references to data pad cells are in octal.

2.5 SUBROUTINE STRUCTURE

Structuring of macro language programs into sub-programs, including recursive use of sub-programs, is supported through a system maintained stack permitting the re-establishment of the execution environment of a program when a called sub-program returns. Special macroinstructions are provided for passing parameters to sub-programs using the same conventions as are supported for macroinstruction operands.

2.6 DETAILED INSTRUCTION DESCRIPTION

The Appendix to this manual contains a detailed description of the basic macroinstruction set. Most of these macroinstructions are resident; that is, the primitive microinstruction sequences which define them are part of the resident system. Additional macroinstruction definitions, including those defined by the user, are normally made a part of the system in which they are used.

Chapter 3. THE SYSTEM ASSEMBLER

3.1

The System Assembler is a symbolic assembler capable of processing both microinstruction statements and macroinstruction statements to produce executable code. A set of input text documents are assembled into a single, non-relocatable MOS-image program module which is stored as a program document. Symbolic specification of all fields in both microinstruction and macroinstruction statements is permitted, and statements may be labeled for reference in other statements.

3.2 SOURCE STATEMENT ORGANIZATION AND CHARACTER SET

The input to the System Compiler must be in the form of text documents in a single text library. The character set used is:

(1) Upper and lower case alphabetic: A - Z, a - z
(2) Numeric: 0-9
(3) Punctuation: blank, comma, ;, :, -, +, (/) ,(SKIP)
(4) Special Characters: ', ", *, $, <, >

In addition, the characters !, #, ?, (,), =, ≠, ↑, ↓, ←, →, and / may be used in comments or character string data statements.

3.3 SYMBOLS

Symbols consist of either < or >, followed by from one to nine alphanumeric characters. Two types of symbols are used in the System Assembler:

(1) Local symbols, which may be referenced only within the document in which they are defined, begin with >,

(2) Global symbols, which may be referenced in any document which is a part of the system in which it is defined, or in any document in a system compiled with respect to that system. These symbols begin with <.

Symbols are defined when they are either used as a statement label (Sec. 3.8) or appear as the first argument of an Equate pseudo operation (Sec. 3.9.4).

As an example, the following labels are valid global symbols.

```
<SYMPHØNY
<Block1
<abcdE
```

Whereas, the labels below are valid local symbols:

```
>GATØ
>FRØNT
>A2345
```

In contrast, the following are invalid symbols.

FIRST	does not begin with < or >
<Ab/5	contains a non-alphanumeric character
>Relativity	too many characters

3.4 SELF-DEFINING TERM

A word of one or more numeric characters in the range 0-7 is used to represent an octal value. If a decimal point precedes the numeric character string, the character string is taken to be in decimal and all the characters 0-9 are allowed. In either case, no other characters may appear including a leading < or >, if the word is to be treated as a self-defining octal or decimal value.

3.5 LOCATION COUNTER REFERENCE

The special symbol * represents the current value of the location counter; that is, the memory address where the current instruction or data word will be located.

3.6 EXPRESSIONS

An expression is an arithmetic combination of symbols, location counter references and self-defining terms. The only arithmetic operators permitted are + and -, which may be used only as binary operators. No other punctuation characters may appear in an expression.

Expressions may be used anywhere symbol references are permitted.

Example: *+5,<HERE->THERE

3.7 DATA TYPES

In addition to octal self-defining terms, data words may be defined by specifying a symbol reference, a character string, a pair of octal or symbolic half-word values, or a set of four quarter-word values. In addition, the two-bit flag field of each data word may be specified explicitly.

3.7.1 Full-Words

A 16-bit word is specified by entering an octal self-defining term or symbol reference giving the desired value. The range of octal values permitted is 0 to 177777_8.

Examples: 103751, <TERM, >P1-*+8

3.7.2 Half-Words

Two eight-bit values to be combined as one 16-bit word are specified by using the half-word option. This option is specified by entering H and two octal self-defining terms or symbol references giving the values desired. The range of octal values permitted is 0 to 377_8 . The value specifications are separated from the H and from each other by one or more spaces or SKIP.

Example: H <LR 37, H 5 >WS+2, H 105 0

3.7.3 Character Strings

A set of half-word ASCII characters preceded by a half-word character count may be specified using the character option. This option is specified by entering ' followed by the characters whose codes are to be compiled. The first non-blank character following the ' will be the first character included. All subsequent characters to the end of the statement, but not including the carriage return, will be included. However, if the first non-blank character is a ', it will not be included in the compiled string, although all subsequent characters will.

Examples:
```
' TWO WORDS
'ONE
'' LEADING BLANKS
```

3.7.4 Quarter-Words

Four 4-bit quantities can be specified for compilation into one 16-bit word using this option. The quantities are specified in octal or by symbol reference, with values from 0 - 17 . One space or SKIP is used to separate the quantities. A V must precede the data values and be separated from them by one space or SKIP .

Examples: V 10 3 2 7, V 17 0 5 0

3.7.5 Flag Specification

The two-bit flag field of each memory word is specified in addition to the 16-bit data field by entering # followed immediately by the desired octal flag value, which may be 0, 1, 2 or 3. Any flag specification must precede the data values for that word but follow the data type specification (H or V). Flags cannot be specified with character string words.

Examples: H #2 <STR >WS, #3 37, V # 1 5 3 2 0, #1 *+5

3.7.6 Repeated Data Words

Multiple copies of a data word, including flags, may be specified by entering DR followed by the repeat count in octal, the flag specification (0 flags if omitted) and the octal data value. As usual, the values must be separated by either space or SKIP.

Example: DR 36 0, DR 100 #1 177777, DR 10 #3 >ØNE

3.8 STATEMENT FORMAT

The System Assembler recognizestwo types of statements, Data Statements and Microinstruction Statements.

Data Statements are used to specify macroinstructions as well as data.

3.8.1 Data Statements

A Data Statement consists of an optional label, zero or more data specifications and an optional comment field. The label, if it occurs, must be

the first word of the statement and it must be a valid symbol. A symbol so used is assigned the current location counter value as its value and may be used to reference the data. If no label is to be used, the first character of the statement should be either space or (SKIP) . The data specifications are separated from the label, if any, by one or more spaces or skips. Any of the data types given in the preceding section may be used for a data specification. If more than one specification is made in one data statement, either a comma, a semi-colon or a colon must be used to separate each field from its successor. Spaces or (SKIP) may be used to separate fields within a data specification, or preceding a data specification. A character string data specification may only occur as the last data specification in a statement. The comment field occurs last in a data statement, and is optional. It begins with a full quote (") character. All subsequent characters in the statement are ignored. Comments may not be used with character string data specifications.

3.8.2 Macroinstruction Statements

Macroinstruction statements are processed by the System Assembler as data statements. The operator is specified by a symbol reference in the first data specification field. If any operands are required, the operator specification is followed by a comma, semi-colon or colon separator, then the operands, also specified as data. A label in a macro language statement can be referred to in <GOTO, <CALL and similar macroinstructions.

3.8.3 Microinstruction Statements (Instruction Mode)

The System Assembler may also be used to enter microinstructions for direct execution by the MP-32 processor. The definition of the microinstruction set in given in TMA4. Microinstruction specifications are accepted by the assembler only when in instruction mode, which is entered by a statement consisting of I (see section 3.9.1). The location counter is adjusted to be even. The microinstruction fields are specified in octal or by symbol reference. They are given in the order T, M, J, ØP, DCBA. If M=3, the ØP and DCBA fields are combined into a single CA field. Each instruction mode statement begins with an optional label field, a required but otherwise ignored IA field, and the instruction fields given above. All subsequent fields are ignored and may be used for comments. The IA field is intended to allow listing of the instruction pad address of those instructions which will be executed from instruction pad.

No data specification statements are permitted in instruction mode except for full-word octal values, which are given by entering Ø followed by the optional flag specification and the data value specified in octal or by symbol reference.

To terminate instruction mode and resume recognition of data statements, enter a statement consisting of D (see 3.9.2).

3.8.4 Comment Statements

Any statement which begins with a full quote character (") will be ignored by the System Assembler, and may be used for additional annotation of a program.

3.9 SYSTEM ASSEMBLER PSEUDO OPERATIONS

In addition to data statements and microinstruction statements, the System Assembler recognizes statements which are instructions to the assembler, rather than specifications of code to be produced. These statements are called assembler Pseudo Operations. They provide facilities to control the mode of instruction execution, alter the normal sequencing of the location counter, and define new symbols. These pseudo-operations must appear as the first word after the label field in the statement.

3.9.1 I - Enter Instruction Mode

Subsequent statements will be interpreted as microinstruction statements, (see TMA4).

3.9.2 D - Enter Data Mode

Subsequent statements will be interpreted as data specification statements. This is the standard mode and is established automatically at the beginning of processing each text document.

3.9.3 ORG - Establish New Location Counter Value

The ORG pseudo-operation must be followed by a symbol reference or octal value. The location counter is set to the value specified. Any symbols used in an ORG pseudo operation statement, if not defined in the apriori system (see section 3.10.2), must be defined by its use as a label in a statement processed before the ORG statement.

```
Examples:  ØRG  * + 100

           ØRG  >old

           ØRG  50000
```

3.9.4 E - Equate Symbol Value

The equate operator requires two operands, which are separated from each other and the E by spaces or SKIP. The first operand is a symbol whose value is to be assigned. The second operand is an octal self-defining term or a symbol reference providing the value. Both local and global symbols may be defined using the Equate pseudo operation. However, global symbols defined in this way are not accessible in documents processed prior to the document containing their definition statement.

```
Example:  E  <Newname     <Oldname-5
```

3.9.5 $ - Terminate Processing of a Document

The $ pseudo-operation marks the last statement to be processed in a text document. Any statements following the $ statement, as well as the remainder of the $ statement, are not processed by the System Assembler and may be used for comments. If a $ statement does not appear in a text document, the assembler prints the diagnostic message "$ missing" and continues processing with the next document.

3.9.6 INCLUDE - Include an Array Processor Process Module

The INCLUDE pseudo-operation must be followed by the name of a document in the current program library. This document must be an AP processor module (the output of APLINK). The location counter is adjusted to an even value, if necessary. The name of the AP process document is entered in the global symbol table, together with the location counter value. The length of the AP process and its load point in AP program store are written in the first doubleword. The AP code from the document is written starting with the second doubleword. The copy of the AP process created is in a format for use by the LOADPS macro instruction to actually load the AP program store as needed during execution. The amount of memory required for this pseudo operation is one doubleword for each AP instruction in the process plus one additional doubleword for the length and starting address in PS. It creates a data array in MP memory, and should be coded with other data statements.

Example: INCLUDE MYAP "<MYAP can be used to reference this.

3.10 USING THE SYSTEM ASSEMBLER

The Signal Assembler is designed to assemble multiple source documents to produce a MOS-image program document. It also produces a table containing the names and values of all global symbols defined or known in the assembly. This global symbol table may be displayed for use in understanding the program by using the Library System Display command. The global symbol table may also be used in subsequent assemblies of other systems to provide an apriori set of known symbols representing systems which will be co-resident with the system being assembled. In particular, the locations of the primitive microinstruction routines which define the resident macroinstruction set is made known to the assembler in this way.

3.10.1 System Definition Document

In order to specify the set of text documents which are to be assembled and to set the base address at which the system will be loaded for execution, a system definition document must be created in the text library containing the text documents. This definition document has the following format.

First statement: (SYST) remainder of statement ignored

Second statement: B space or (SKIP) system base.

The system base value must be specified as an octal number.

The remaining statements contain one text document name per statement, preceded by spaces or a (SKIP). The last statement has $ at the beginning to indicate the end of the document.

Example:

```
(SYST)          "A three document system
 B (SKIP)       140000
(SKIP)          MATH
(SKIP)          FILTER2
(SKIP)          SUM
$
```

3.10.2 Apriori System

If desired, the global symbol definitions of a previously assembled system may be made available for reference in the system being assembled. Normally, the facility is used in order to make available the locations of the resident system macro definitions. It can also be used to make possible uni-directional communication with any separately assembled system. Naturally, if this communication is to be valid, the apriori system must in fact be resident when the reference is executed. The effect of apriori system references in the system assembler is cumulative. That is, if the apriori system of the current assembly was itself assembled with an apriori system, then the global symbols defined in both prior systems will be available during the current assembly and all three systems' symbols will be in the global symbol table output by the assembler. The apriori system to be used in assembling a given system must be a document in the same program library as the output program document.

3.10.3 Invoking the System Assembler

The system assembler is implemented as a system operator which may be initiated from the library mode by the following call:

(OP) ASM_ (SYST) definition-doc[; apriori system]: system-doc (/)

Definition-doc is the system definition document described above; it must exist in the current text library.

Apriori system is an existing system document in the current program library.

System-doc is the desired output document; which must be declared in the current program library.

Example: (LIBE) source (/) establishes current text library

(LIBE) pgm (/) establishes current program library

(OP) ASM_ (SYST) FUNCTIONS;SYSONE:FUNCTS (/)
performs the assembly

3.10.4 Assembler Output

The system assembler produces a system program document containing two parts. The first part is a table giving the names and definitions of all global symbols defined in the assembled system or its apriori system. The second is a MOS image module containing the assembled code in a form ready for execution. In addition, the base and length of the MOS-image module are a part of the system program document.

Informational and error messages are displayed on the console screen during the assembly. A listing of these messages is given in Section 3.11.

3.11 ASSEMBLER ERROR MESSAGES

1. IS NØT A CØMPILER ØPTIØN

   The key SYST was not present in the command invoking the assembler. The assembly is aborted.

2. (systname) IS NØT A VALID ITEM

   The SYST definition document name is not contained in the directory. The assembly is aborted.

3. (systname) IS NØT A VALID SYSTEM

   a) The first word on line one in the definition document is not SYST.

   b) The base value is not properly specified on line two of the definition document. The assembly is aborted.

4. NØT ENØUGH MEMØRY

   Insufficient memory space was available to execute the assembly. The assembly is aborted.

5. (document name) IS UNDEFINED

   The document named was not found in the current text library. The assembly is aborted.

6. NØ CURRENT LIBRARY

   Either the current text or program library was not declared. The assembly is aborted.

7. (apriori syst name) does not exist

   The named apriori system does not exist in the current program library. The assembly is aborted.

8. INPUT SYNTAX ERRØR - COMP ABØRTED

   In the input command, a colon did not precede the destination program name. The assembly is aborted.

9. (pgm name) IS UNDEFINED

   The program name is not in the current program library. The assembly is aborted.

The following error messages have the form:

ERRØR IN MACRØ (name) ØN PAGE (#)LINE(#)(error)

When an error occurs in a line, the line is printed and the word which was being processed is bracketed by carats (^).

10. NØ DØLLAR SIGN

    The last line in the document does not contain a $ in column one. Processing will continue as though a $ were present.

11. HEADER BUFFER ØVERFLOW

    The number of input documents exceeds the assembler's capacity. The assembly is aborted.

12. LINE

    A blank or meaningless line was encountered in instruction mode. The line is ignored and processing continues with the next line.

13. MØDE

The current mode was not recognized. This message should not occur.

14. WØRD

No word was found when a word was expected due to a previous Ø keyword. The rest of the line is ignored and processing continues with the next line.

15. DATA

A Syntax error occurred in a data statement. On this pass further processing of the line is terminated and processing continues with the next line.

16. INSTRUCTIØN

A Syntax error occurred in instruction mode. The rest of the line is ignored and processing continues with the next line. This may result in improper instruction alignment.

17. EXPRESSIØN

a) The expression was not complete.

b) The octal value specified was too large for the field.

c) An expression began with + or -. The remainder of the line is ignored and processing continues with the next line.

18. LABEL UNDEFINED

a) A label was not found in the symbol table. Processing of the line continues.

19. TERMINATØR

A data specification was not terminated by SPACE, SKIP, CØMMA, SEMI-CØLØN, CØLØN or ( ). The rest of the line is ignored and processing continues with the next line.

20. FLAG

The flag field of a data word was incorrectly specified. The remainder of the line is ignored and processing continues with the next line.

21. STATEMENT

A data type keyword was not followed by any data specification. The rest of the line is ignored and processing continues with the next line.

22. HALFWØRD

The separation between the two halfword specifications was not SPACE or SKIP. The rest of the line is ignored and processing continues with the next line.

23. LABEL

a. The word does not exist.

b. In an equate statement the symbol being defined is missing or invalid.

c. The symbol is a duplicate symbol.

d. In an equate statement, the defining expression is missing. The rest of the line is ignored and processing continues with the next line.

24. VECTOR

Less than four field value expressions were given in a quarter-word data specification. The rest of the line is ignored and processing continues with the next line.

25. ØRG

a. The octal value specified was too large.

b. No value expression was given. The rest of the line is ignored and processing continues with the next line.

26. GLOBAL PASS

All source text documents have been located and the assembly is beginning the first pass through all source documents.

27. FINAL PASS

The first pass through all documents has been completed and the final pass is beginning.

28. CMP DONE. SYSTEM LENGTH IS nnn. SYSTEM STARTS ON TRACK n.

The assembly is completed. The total system length and the base track where the program was written are given.

Chapter 4. MACRO LANGUAGE PROGRAMMER'S GUIDE

Once compiled, macro language programs are normally executed as background jobs. The background execution facility of the system provides for selection of a job for execution, loading it from disk into main memory, swapping of temporarily interrupted jobs onto disk so new jobs may be executed and restoration and resumption of interrupted jobs. In order for a program to be able to execute properly as a background job, certain conventions of program structure must be followed. Special macro instructions are also needed in order to communicate with the background job processor. This chapter will discuss each of these areas.

4.1 BACKGROUND JOBS

A background job is a program which competes for processing time in the MP-32 independent from Library, Edit or Math System service as well as other background jobs. It is written as a macro language program and compiled using the System Assembler. A properly prepared program may be loaded from the current program library for execution as a background job using the (PGM) operation in the Library System. It may also be scheduled from another background job using the JOB subroutine (see Appendix).

Background jobs are executed one at a time, with each job having the use of up to $52,736_{10}$ 8-bit words of MOS for program and data. Once initiated, a background job is allowed full use of the processor until it either finishes, must wait for console input or other independent activity, or it voluntarily relinquishes the processor. If the job is not completed, the contents of the MOS it is using are copied onto disk for restoration when it once again gets processor service.

Each job has associated with it a priority between 0 and 255_{10}. Any time the processor is ready for reassignment, interactive Library and Edit System processing is given first priority. If no such processing is needed, the highest priority background job is selected. That job is then loaded into MOS and initiated or resumed.

Background jobs are not precluded from interaction with the user, but some care must be exercised if more than one background job is to be executed at a time. All job communication is with the terminal whose number is given in data pad cell 74. Output to the display screen of the terminal is accomplished by using the system output macro instructions and subroutines. Input from the terminal is possible using the GETKEY, GETINTGR, GETOCTAL or GETNAME macro instructions to obtain input from the current input statement (typically, the statement in which the job was started through the (PGM) operator). Once the carriage return terminating the statement has been received, further input is possible only by first requesting the next statement using the GETSTMNT subroutine, then the input macro instructions. The GETSTMNT subroutine will allow other jobs to receive service if no input statement has been received from the terminal.

4.2 PROGRAM STRUCTURE FOR BACKGROUND JOBS

Each background job must have associated with it a 64-word data pad save area. This save area must be at the beginning of a disk resident job, and its address is the job address for resident jobs. The first word of the save area contains the entry point address of the job. Words 1 to 73_8 and 77_8 will be loaded into data pad cells 1 - 73_8 and 77_8 when the job is executed. Data pad 0 will contain a parameter provided when the job was submitted. Word 77 must point at a push-down stack area on a doubleword boundary. The first word of this area must have flag 2 on. The last doubleword of the area must have flag 1 on the left word. The right word of this doubleword is not used. A minimum of 3,+ 2 words for each level of subroutine call are required for this pushdown stack area. Normally, 13_8 words are adequate. The assembly language specification of the save area and pushdown stack might typically look like this:

```
     >START                          "Beginning address for execution
     DR 76 0                         "Data pad cells 1-76
       *+1                           "Pushdown stack base
   #2 0, _DR 11 0, #1 0              "Pushdown stack
>START
     <MODE3                          "First executed instruction
     <SETBBND,>LAST,>R
          .
          .
          .
>R   <RETURN
>LAST
```

The job itself is structured as a subroutine, or nested set of subroutines. It should begin with a mode setting macroinstruction such as MODE3 shown above. This establishes the mode of interpretation of subsequent macro instruction statements (see description of INTERNAL in Appendix for an example of a different mode). When the job is finished, it returns to the job scheduler using the RETURN macroinstruction. Alternatively, a job may terminate by using the PULLOUT macro, which will return to the job scheduler regardless of the level of nested subroutine from which it is called.

Once a job receives processor service, it will continue to have exclusive use of the processor (except for interrupt service) until it finishes or voluntarily releases the processor. For jobs which require more than a second or two of processor time, if operating in a situation where other jobs may desire service, it is recommended that the BREAK macro be used to allow service to higher priority activities. When a higher priority activity is given service the job being pre-empted is written onto disk. When its turn for execution comes up, it is read back into MOS and processing resumes at the point where it left off. In order for the system to know how much information must be written onto disk, the job must use the SETBBND macro instruction (see Appendix). This instruction establishes the upper limit for the information to be saved. It also checks this upper limit to be sure the job is not encroaching on the upper memory communication region (BKLIMIT). It is recommended that SETBBND be used at the beginning of the job and any time the limits of the space used are changed significantly.

4.3 PROCESSING MODE CONTROL

The processing mode of the system at any time defines the method of sequencing from one operation to the next and specifies the instruction and data pad environment of the mode. The mode is determined by the contents of data pad cell 75_8, which is used in the following ways:

1. Pad 75 is assumed to contain the address of a primitive micro-instruction program which, when executed, will establish instruction pad and data pad as necessary.

2. Whenever an interrupt routine must use instruction pad and/or data pad, it will save data pad in the system data pad save area. When it is finished, it will restore data pad to its old contents, then execute the program whose address is in data pad 75 as a means of resuming the interrupted activity.

3. In macro sequencing modes, the contents of both pad 75 and 76 are saved in a push down stack whenever a subroutine is called. When a RETURN (or RETURNTO) is executed, the contents of pad 75 are compared with the value saved in the top stack entry. If they differ, the program whose address is given in the stacked pad 75 is executed after restoring pad cells 75 and 76.

A macro mode is normally established by executing a mode setting macro instruction, which will load the necessary programs and data and set data pad 75. Each subroutine must begin by establishing its own mode, both to insure correct sequencing of its own instructions and to allow correct restoration of the calling program's mode.

Two standard macro-instruction modes are provided: MODE3 provides basic sequencing and interrupt detection; INTERNAL provides a special set of fast register arithmetic instructions as well as support of most macro-instructions and interrupt detection. Other modes may be developed by a user using micro-instruction sequences to optimize his processing for particular applications.

APPENDIX: MACRO INSTRUCTIONS

Operand Specification

In general, there are six types of operands: actual data values, base addresses for blocks of memory, descriptors of information, data pad (register) addresses, statement labels and addresses where values are to be returned. The appropriate interpretation for each op-code-operand field is found in the instruction descriptions which follow this introduction.

Many macro instructions permit their operands to be specified indirectly, that is, by giving the location where the operand is located. Such indirection is indicated by the use of non-zero flags with the operand word. The general interpretation given to these flags is:

#0 (or no flags)	The operand is directly specified.
#1	The operand is located at the memory address specified by this word
#2	The operand is located in the data pad cell (register) whose PA is given in the right half of this word.
#3	The operand is located at the memory address specified in the data pad cell whose PA is given in the right half of this word (indirect through pad).

When a macroinstruction returns an operand the use of flags is slightly altered.

#0 (or no flags)	This word contains the memory address where the result is to be stored.
#1	The memory word whose address is given in this word contains the memory address where the result is to be stored.

The meanings of #2 and #3 are unchanged. In the descriptions which follow, such operands are marked by a superscript(*).

There are some macroinstructions which do not permit indirect specification of their arguments. These operands are marked by a superscript(@).

Any other special uses of flags are given in the individual instructions (see Internal Mode).

Operand Field Format Description Conventions

In addition to the conventions noted in Chapter 1 of this manual, the following conventions are observed in describing the general format of the operand field of the macro assembler commands.

PA	data pad address
n	positive octal integer
e-exit,label	statement label
H	half word
A	memory address

Non-Resident Macro Instructions

Some of the macro instructions described are not included as part of the resident nucleus. These macros are identified by a ** after the macro name. It is necessary to include the document containing their definition in any system in which they will be used. Copies of the documents may be obtained from the shared text library MACROS. The documents needed are listed in parenthesis after the macro operand list in the operations index.

SYMBOL TABLE COMMANDS

		Section
LABEL	n, value, e-exit@	2.1
READLABEL	A(corr)*, corr value*, e-exit@	2.2
UNLABEL	n, e-exit@	2.3
NXTNAME	H PA(name) PA(pointers), e-exit@	2.4
OCTALF**	H n PA, e exit@	2.5
DECIMALF**	H n PA, e-exit@	2.6

DATA TRANSFER COMMANDS

LOADPD	H n PA, A	3.1
STOREPD	H n PA, A	3.2
MOVE	A(from), A(to), n	3.3
RMOVE	A(from), A(to), n	3.4
LDBYTE	H code PA(to), A(from) Code: 0 left→left 1 left→right 10 right→left 11 right→right	3.5

SINGLE WORD ARITHMETIC AND LOGIC

MOVEWD	a, b	Sets b=a	4.1
INCBYWD	a, b	Sets b=a+b	4.2
DECBYWD	a, b	Sets b=b-a	4.3
MULTBYWD	H a b@, c@	(MULTIBYWD)	4.4
DIVBYWD	H a b@, c@		4.5
NEGPD**	a@	Sets a=-a	4.6
BIC**	mask, a		4.7
BIS**	mask, a		4.8

MODE OPERATIONS

MODE3		5.1
INTERNAL		5.2
LEAVEINT		5.3
H #2 LR	PA(to), a	5.4.1
H (#2 / #3) STR	PA(from), A(to)*	5.4.2

Note: * = destination argument @ = flags ignored ** = nonresident macro

Rev.A 4/74
Rev.B 6/75
Rev.C 10/75

.....continued		Section
H{#2}AR (#3)	PA(a), H [flag] PA(b) PA(a+B)@	5.4.3
H{#2}SR (#3)	PA(a), H [flag] PA(b) PA(a-b)@	5.4.4
H{#2}INCR (#3)	PA(a), H [#1] incr., PA(a+incr.)@	5.4.5
EXECUTION CONTROL		
GOTO	label@	6.1
GOTOR	PA@	6.1
IFEQ IFNE IFLT IFGT	a, b, label@	6.2
SCAN SCANLT	a, table (SCAN) (SCANLT)	6.3
BIT**	a, b, label@	6.4
SUBROUTINE EXECUTION CONTROL		
CALL	Subroutine-name@ [, parameter,...,parameter]	7.1
RETURN		7.2
RETURNTO	label@	7.3
SUBROUTINE ARGUMENT EXCHANGE		
GETSARG	a*	8.1
GETMARG	n, a*	8.2
PUTSARG	value	8.3
DISC INPUT/OUTPUT SUBROUTINES		
CALL GETDIR,	disk-desc., buffer-address	9.1
CALL READHDR,	A(doc-name), A(doc-directory) buffer desc., e-exit@	9.2
CALL READSEG,	A(doc-header), doc. segment #, buffer-desc., e-exit@	9.3
CALL SEEK,	disk-desc.	9.4

Note: * = destination argument @ = flags ignored ** = Nonresident MACRØ

.....continued			Section
CALL WRITE[F],	disk-address, buffer-desc.		9.5
CALL READ[F],	disk-desc, buffer-desc. , e-exit@		9.6
CALL WRITEP[F],	disk-desc., buffer-desc.		9.7
CALL READREC,	disk-desc., buffer-desc. , e-exit@		9.8
CALL ALLOCATE,	allocation-desc., buffer-desc. , track number*		9.9
CALL RELEASE,	allocation-desc., buffer-desc. , track number		9.10
	KEYBOARD ARGUMENT FETCHING		
GETNAME	H PA(terminator) PA(name)		10.1
GETOCTAL	H PA(terminator) PA(n)		10.2
GETINTGR	H PA(terminator) PA(n)		10.3
GETKEY	next single key*		10.4
MDECKR			10.5
	DISPLAY SCREEN OUTPUT		
BLDOCT	H PA (char. string) n, value		11.1
TYPE	Character string		11.2
TYPEWD	Word pointer list		11.3
CRETURN			11.4
SPACE			11.5
SKIP			11.6
[UP / DOWN / OVER / BACK]**	# positions		11.7
TYPRSET			11.8
ERASE			11.9
POINT**	X, Y	(LINE)	11.10
LINE**	X, Y	(LINE)	11.11
CALL DECOUT	value	(DECOUT)	11.12

Note: * = destination argument @ = flags ignored ** = Nonresident macro

FLAG OPTIONS			
EXPNDF**	H reg PA(A(value))@ note: flag→reg+1	(FLAGOPS)	Section 12.1
MERGEF**	H PA(Address) reg@ note: flag←reg	(FLAGOPS)	12.2
BACKGROUND JOB PROCESSING			
CALL, JOB	j-desc, c-exit, parameter, priority		13.1
SETBBND	limit address, e-exit@		13.2
BREAK			13.3
BWAIT			13.4
CLRBWAIT	A(job element)	(CLRBWAIT)	13.5
CALL GETSTMNT			13.6
CLRWAIT			13.7
PULLOUT			13.8
GETITRK			13.9
ARRAY PROCESSOR SUPPORT**			
LØADPS			14.1.1
APBASIC			14.1.2
APFLØAT			14.1.3
APFFT			14.1.4
TIEAP		(APFLØAT)	14.2.0.1
UNTIEAP		(APFLØAT)	14.2.0.2
STACKAP	A(data)@ or #2 PA(A(data))@	(APFLØAT)	14.2.1
LØADAP	A(data)@ or #2 PA(A(data))@	(APFLØAT)	14.2.2
STØREAP	A(data)@ or #2 PA(A(data))@	(APFLØAT)	14.2.3
ADDAP		(APFLØAT)	14.2.4
SUBTAP		(APFLØAT)	14.2.5
MULTAP		(APFLØAT)	14.2.6
DIVAP		(APFLØAT)	14.2.7

Note: * = destination argument @ = flags ignored
** = nonresident MACRØ

continued.....

SQAP		(APFLØAT)	14.2.8
SQRTAP		(APFLØAT)	14.2.9
NEGAP		(APFLØAT)	14.2.10
ABSAP		(APFLØAT)	14.2.11
SGNAP		(APFLØAT)	14.2.12
INVAP		(APFLØAT)	14.2.13
LØGAP		(APFLØAT)	14.2.14
EXPAP		(APFLØAT)	14.2.15
SINAP		(APFLØAT)	14.2.16
CØSAP		(APFLØAT)	14.2.17
ATANAP		(APFLØAT)	14.2.18
PUSHAP		(APFLØAT)	14.2.19
PULLAP		(APFLØAT)	14.2.20
FIXAP		(APFLØAT)	14.2.21
FLØATAP		(APFLØAT)	14.2.22
WAITAP		(APFLØAT)	14.2.23
MBYPWR	n@ or #2 PA(n)	(MBYPWR, APBASIC, APFLØAT)	14.3.1
CALL FPØUT	Number	(MBYPWR, APBASIC, APFLØAT)	14.4.1
CALL DISPLAY	A(Desc.), CODE, A(scale)	(LINE, APBASIC, APFLØAT)	14.4.2
CALL PRINT	A(Desc.)	(FPØUT)	14.4.3
SETDPA	address	(APBASIC)	14.5.1
SETMA	address	(APBASIC)	14.5.2
SETPSA	address	(APBASIC)	14.5.4
WRTAPMD	A(data), length	(APBASIC)	14.5.6

Note: · = destination argument @ = flags ignored ** = nonresident MACRØ

continued.....

WRTAPDP	A(data), length	(APBASIC)	14.5.7
LØADDPL	A(data), length	(APFFT)	14.5.8
WRTAPSP	Spvalue, spa	(APBASIC)	14.5.9
RDAPMD	A(buffer), length	(APBASIC)	14.5.10
RDAPDP	A(buffer), length	(APBASIC)	14.5.11
RDAPDR	A(buffer), length	(APBASIC)	14.5.12
RDAPSP	spa, A(buffer)	(APBASIC)	14.5.13
CALL FFTC	$\log_4$(blksize), log (buffer len),A(buffer)	(FFTR,APBASIC, APFFT)	14.6.1
CALL FFTR	$\log_4$(blksize), log (buffer len),A(buffer)	(FFTR,APBASIC, APFFT)	14.6.2
CALL IFTC	$\log_4$(blksize), log (buffer len),A(buffer)	(FFTR,APBASIC, APFFT)	14.6.3
CALL STORET	A(buffer), length, DS*	(FFTR,APBASIC, APFFT,APFLØAT)	14.7.4
CALL CONVOLVE	A(kernel), length, A(data), length	(CONVOLVE, APBASIC,APFFT)	14.7.1
CALL FILTER	A(forward kernel), length, A(feedback kernel), length, A(data), length		14.7.2

Note: * = destination argument @ = flags ignored ** = nonresident MACRØ

Rev. B, 6/75
Rev. A, 4/74

2. SYMBOL TABLE COMMANDS

These commands allow the user to define and remove labels as part of a symbol table, search a symbol table for a given label, and perform an ordered search through an entire symbol table. The symbol table format assumed for these commands is:

Word 0: Displacement to last used word in the symbol table (current length -1)

Word 1: Maximum possible length -10

Words $2-77_8$: Displacement from symbol table base to first entry which begins with a specific character, where the character is 0-9, A-Z and a-z respectively.

Each entry has the format:

Word 0: Displacement to next entry (in collating sequence) or 0 if last entry beginning with a character.

Words 1 - K/2: Characters 2-K of the symbol, packed two to a word, with 0 in the right half of the last word if necessary. K must be nine or less.

The word following word K/2 has its flag 1 on to mark the end of the symbol. It is also the first data word associated with the symbol.

The LABEL, READLABEL, and UNLABEL commands assume that the user has previously set data pad cells 0-5 as follows:

Data Pad Cell 0 = memory address of desired symbol table

1-5 = character string giving desired label. Must conform to standard regulations for statement labels and variable names; i.e., 1 to 9 alphanumeric characters, first character non-numeric. The left half of data pad cell 1 contains a count of the number of characters in the label. There must be at least one character in the label.

All the symbol table commands result in destruction of the contents of data pad cells 1 and 6.

2.1 Add Label to Symbol Table

Op Code	Operand
LABEL	n, value, e-exit@

This command adds the label in data pad cells 1-5 to the symbol table whose address is stored in data pad cell 0. The length (in words) of the area allocated for this label is specified by 'n'. The first word of the area allocated is set to the value indicated by 'value'. The address of the first word of the area allocated is returned in data pad cell 1.

If the label already exists in the symbol table, execution control is returned to the statement specified by 'e-exit', and data pad cell 1 is set to the address of the first word of the existing area. If the symbol table is full, an error message is displayed on the screen and data pad cell 1 is set to 0.

For example:

```
<LABEL,10,15,>DBLDEFSYM
```

This command would cause the symbol table whose address is stored in data pad cell 0 to be searched for the label stored in data pad cells 1-5. If the symbol already exists in the symbol table, control is returned to memory location DBLDEFSYM, a user-defined routine handling double-defined symbols, say, and data pad cell 1 is set to the address of the first word of the currently existing area.

If the label is not already in the symbol table, a 10-word area is allocated with this label, the first word of the area is set to 15, and the address of the first word of the area is returned in data pad cell 1.

2.2 Search for Label in Symbol Table

Op Code	Operand
READLABEL	A(corr)*, corr value,* e-exit@

This command searches the symbol table whose address is stored in data pad cell 0 for the label stored in data pad cells 1-5. If the label is not found, execution control is returned to the statement specified by 'e-exit.'

If the label is found, the address of the first word of the area allocated for this label is stored in 'A(corr)*'; the contents of the first word is stored in 'corr value*'.

For example:

```
<READLABEL,>A1,>V1,>UNDEFSYM
```

This command would cause the symbol table whose address is stored in data pad cell 0 to be searched for the label stored in data pad cells 1-5. If found, the address and contents of the first word of the area with the given label are stored in memory locations A1 and V1, respectively.

If not found, control is returned to address UNDEFSYM, a user routine for dealing with undefined symbols.

2.3 Remove Label from Symbol Table

Op Code	Operand
UNLABEL	n, e-exit@

This command searches the symbol table whose address is stored in data pad cell 0 for the label stored in data pad cells 1-5. If found, the label is removed from the symbol table and the number of words specified by 'n' are deleted from the area.

If the label is not found, control is returned to the statement specified by 'e-exit.'

For example:

```
< UNLABEL,5, >ERROREXIT
```

The above command would cause the symbol table whose address is stored in data pad cell 0 to be searched for the label in data pad cells 1-5. If found, the label is removed, and 5 words of the area are deleted. If not found, control is returned to location ERROREXIT.

2.4 Ordered Search of Symbol Table

Op Code	Operand
NXTNAME	H PA(name) PA(pointers), e-exit@

This command allows the user to perform an ordered search through a symbol table. The data pad cell indicated by 'PA(pointers)' contains the address of the symbol table to be searched. The next two consecutive pad cells should both be initialized to 2; these locations are used as working areas by the NXTNAME macro.

Each time NXTNAME is executed, the first-word address of a labelled entry in the symbol table is stored in the location specified by 'PA(name)'. The corresponding label is stored in consecutive pad locations beginning with 'PA(name)'+1.

The labels are brought up ordered by a collating sequence as follows: numbers, upper-case alphabetic, lower-case alphabetic.

When no labels remain in the table, control is transferred to the location specified by 'e-exit.'

For example, suppose a symbol table contains the following symbols: AVAL, AlLOC, STORLOC, FERN, F1, F3, Fern. The user might code a loop to examine the contents of the symbol table whose address is stored in data pad cell 15 as follows:

```
            <LOADPD, H 2 16, >TWO
>LOOP1      <NXTNAME, H 20 15, >ENDSRCH
            <IFEQ, #3 20, 0, >LOOP1      "IGNORE IF ENTRY 0"
                :
                :
              (code processing non-zero entries)
            <GOTO, >LOOP1
>TWO            2, 2
>ENDSRCH        ...
```

The symbols would be stored one at a time in data pad cells 21-25 in the following order: AlLOC, AVAL, F1, F3, FERN, Fern, STORLOC.

Data pad location 20 would contain the address of the first word of each of the areas. Control would be transferred to location ENDSRCH at the end of the search.

2.5 Build Octal Number from Character String**(DECØCT)

Op Code	Operand
ØCTALF	H n PA, e-exit

This command builds an octal number from a character string which resides in pad cells 1-5, and places it in the pad cell specified by the PA in the argument. If there are more than "n" bits in the result or if there is a non-numeric character in the character string, control will be passed to the error-exit.

For example:

Suppose pad cells 1-3 contained the character string (from the command NXTNAME for instance) "7301," and the following commands were executed:

<ØCTALF, H 16 10, >ERRØR

The result would be the value 7301_8 in pad cell 10.

2.6 Build Decimal Number from Character String** (DECØCT)

Op Code	Operand
DECIMALF	H n PA, e-exit

This command builds a decimal number from the character string in pad cells 1-5, and places it in the pad cell indicated by the PA in the argument. If there are more than "n" bits in the result or if there is a non-numeric character in the character string, control will be passed to the error exit.

For example:

Suppose pad cells 1-3 contained the character string "7301," and the following commands were executed:

<DECIMALF, H 16 10, >ERRØR

The result would be the value 16205_8 in pad cell 10.

3. DATA TRANSFER COMMANDS

These commands allow the user to transfer data within memory and the data pad.

3.1 Load Pad

Op Code	Operand
LOADPD	H n PA, A

This command loads 'n' words into consecutive data pad cells, beginning with the 'PA' location given, from the address indicated by 'A'.

Example:

```
<LOADPD,H 5 17, >DOG
```

This command would load five words, beginning with the word at memory location DOG, into data pad cells 17_8 through 23_8.

3.2 Store Pad

Op Code	Operand
STOREPD	H n PA, A

This command stores 'n' words into consecutive locations, beginning with the location given by 'A', from data pad, beginning with the 'PA' location given.

Example:

```
<STOREPD,H 3 21, #2 55
```

This command stores the contents of data pad cells 21, 22, and 23 into the memory location whose address is stored in data pad cell 55.

3.3 Move Data in MOS

Op Code	Operand
MOVE	A(from), A(to), n

This operation moves 'n' words, one at a time, beginning with the word at the location specified by 'A(from)' into consecutive locations beginning with the location specified by 'A(to)'.

The words "one at a time" are significant when the areas specified overlap. For example, the command MOVE 0, 2, 4 would have the following effect:

Cell Location	Initial Contents	1st Word Moved	2nd Word Moved	3rd Word Moved	4th Word Moved
0	000001_8	000001_8	000001_8	000001_8	000001_8
1	000002_8	000002_8	000002_8	000002_8	000002_8
2	000003_8	000001_8	000001_8	000001_8	000001_8
3	000004_8	000004_8	000002_8	000002_8	000002_8
4	000005_8	000005_8	000005_8	000001_8	000001_8
5	000006_8	000006_8	000006_8	000001_8	000002_8

The MOVE command issued on overlapping areas thus allows the user to fill large areas with some value or set of values.

3.4 Reverse Move

Op Code	Operand
RMOVE	A(from), A(to), n

This command functions in the same manner as the MOVE, except that movement begins with the last location of the areas specified and proceeds to the first location, thus eliminating the "fill" result on some overlapping areas. For example, suppose that a RMOVE 0, 2, 4 command had been given in the previous example:

Cell Location	Initial Contents	1st Word Moved	2nd Word Moved	3rd Word Moved	4th Word Moved
0	000001_8	000001_8	000001_8	000001_8	000001_8
1	000002_8	000002_8	000002_8	000002_8	000002_8
2	000003_8	000003_8	000003_8	000003_8	000001_8
3	000004_8	000004_8	000004_8	000002_8	000002_8
4	000005_8	000005_8	000003_8	000003_8	000003_8
5	000006_8	000004_8	000004_8	000004_8	000004_8

3.5 Load Byte

Op Code	Operand
LDBYTE	H code PA(to), a

This command loads one 8-bit byte of the data specified by 'a' into one byte of the designated 'PA'address according to the code specified:

Code	Result
0	left byte of a into left byte of pad
1	left byte of a into right byte of pad
10	right byte of a into left byte of pad
11	right byte of a into right byte of pad

For example, suppose that memory location MASK contains 114522_8 and that data pad location 17 contains 177777_8. Sample LDBYTE commands and their effect are illustrated below.

Command	Result in Data Pad Cell 17
<LDBYTE,H 0 17, #1 >MASK	1001100111111111_2
<LDBYTE,H 1 17, #1 >MASK	1111111110011001_2
<LDBYTE,H 10 17, #1 >MASK	0101001011111111_2
<LDBYTE,H 11 17, #1 >MASK	1111111101010010_2

4. SINGLE WORD ARITHMETIC

These commands allow the user to perform move and arithmetic operations on single-word operands in data pad or memory. If the result word (b) is in memory, its flags will not be changed.

4.1 Move Word

Op Code	Operand
MOVEWD	a, b

This command moves one word of data from the location specified by 'a' into the location specified by 'b'. Note that b is not a destination argument.

For example:

```
<MOVEWD,#2 2, #1 >DOG
```

The above command transfers the contents of data pad cell 2 into memory location DOG.

4.2 Increase by Word

Op Code	Operand
INCBYWD	a, b

This command adds the values indicated by 'a' and 'b' and stores the result in the location specified by 'b'.
For example, suppose memory locations A and B contain 010101 and 000032 , respectively. The command:

```
<INCBYWD,#1 A, #1 B
```

would have the following result:

Contents of A = 010101_8
Contents of B = 010133_8

4.3 Decrease by Word

Op Code	Operand
DECBYWD	a, b

This command subtracts (2's complement arithmetic) the value indicated by 'a' from the value indicated by 'b' and stores the result in the location specified by 'b'.

For example, suppose that memory locations Q and R contain 000003_8 and 000005_8, respectively The command:

```
<DECBYWD,#1 Q, #1 R
```

would have the following result:

Contents of Q = 000003_8
Contents of R = 000002_8

4.4 Multiply by Word

Op Code	Operand
MULTBYWD	H PA(a) PA(b)@, PA(c)@

This command multiplies the values 'a' and 'b' and stores the result in the location specified by 'c'. The result will be correct only if a and b are positive integers and $a*b<32767_{10}$. For example, suppose pad locations D and F contain 000100_8 and 000032_8 respectively. The command:

<MULTBYWD, H >D >F, >G

would have the following result:

Contents of D = 000100_8
Contents of F = 000032_8
Contents of G = 3200_8

4.5 Divide by Word

Op Code	Operand
DDIVBYWD	H PA(a) PA(b)@, PA(c)@

This command divides the value b into the value a and stores the resulting quotient in the location specified by 'c' and the remainder in location c+1. For example, suppose pad locations L and M contain 000002_8 and 000007_8, respectively. The command:

<DDIVBYWD, H >M >L >K

would have the following result:

Contents of L = 000002_8
Contents of M = 000007_8
Contents of K = 000003_8
Contents of K+1 = 000001_8

4.6 Negate Pad** (APFLØAT)

<NEGPD, a

Negates pad cell "a" and places the result in pad cell "a".

For example:

Suppose pad cell L contained 10_8. The command:

<NEGPD, L

would change pad cell L to contain 177770_8, or minus 10 octal.

4.7 Clear a Bit** (LØGICAL2)

<BIC, mask, arg

Clears each bit in the argument that corresponds to a set bit in the mask.

For example:

Suppose pad cell L contained 304_8. Then the command:

<BIC, 600, #2 L

Would result in pad cell L containing the value 104_8.

4.8 Set a Bit** (LØGICAL2)

<BIS, mask, arg

Sets each bit in the argument that corresponds to a set bit in the mask.

For example:

Suppose pad cell L contained 304_8. Then the command:

<BIS, 600, #2 L

would result in pad cell L containing the value 704_8.

5. MODE OPERATIONS

5.1 Establish Macro Mode

Op Code
MODE3

This statement is used to establish an initial mode of operation during the execution of a subroutine. Either MODE3 or INTERNAL should be the first instruction in a subroutine.

5.2 Establish Internal Mode

Op Code	Operand
INTERNAL	

This command establishes internal mode, causing the internal mode program to be loaded into instruction pad. The INTERNAL command may be used to reestablish internal mode after using an incompatible macro (GETMARG, BLDOCT, NXTNAME).

5.3 Leave Internal Mode

Op Code	Operand
LEAVEINT	

This command restores the standard macro execution mode; it should be used to terminate internal mode before issuance of GETMARG, BLDOCT, or NXTNAME commands. It is equivalent to MODE3.

5.4 Internal Mode Operations

The SIGNAL System Macro Assembler provides a special set of macros which allow the user to operate in what is called internal mode. Basically, when the user establishes internal mode (by giving the INTERNAL command, below), a small primitive program is loaded into instruction pad, using cells $7\text{-}57_8$. This program allows the user to specify high-speed load, store, add, subtract, and increment operations on the data pad and memory. These operations, described in detail below, may be interspersed with any macro commands except those which alter the contents of instruction pad cells $7\text{-}57_8$. These are GETMARG, BLDOCT, and NXTNAME.

The following operations may be specified once internal mode is established.

5.4.1 Load Register

Op Code	Operand
H #2 LR	PA(to), a

This command loads the specified pad cell with the data indicated by 'a'.

For example:

```
H #2 <LR 33, #1 >GAD
```

The above command would load data pad cell 33 with the contents of memory location GAD.

5.4.2 Store Register

Op Code	Operand
H {#2 / #3} STR	PA(from),A(to)*

This command stores the contents of the specified pad cell (or if the #3 flag option is specified, the memory location whose address is contained in the specified pad cell) into the location specified by 'A(to)*'.

For example, suppose data pad cell 55 contains 010000_8. The command:

```
H #2 <STR 55, >ACON
```

would store the value 010000 into memory location ACON. The command:

```
H #3 <STR 55, >ACON
```

would store the contents of memory location 010000_8 into memory location ACON.

5.4.3 Add Register

Op Code	Operand
H {#2 / #3} AR	PA(a), H [Flag] PA(b) PA(a+b)

This command adds the values indicated by 'PA(a)' & 'PA(b)' and stores the result in the location indicated by 'PA(a+b)'. The flag options may be used to specify that the associated pad location contains the address of the memory location to be used in the operation.

For example, suppose that data pad cell 17 contains the value 013567_8, data pad cell 13 contains 000001_8, and memory location 013567_8 contains 000002_8. Several illustrative commands and their results are shown below:

Command	Result
H #2 <AR 17, H 13 15	013570_8 (in data pad cell 15)
H #3 <AR 17, H 13 15	000003_8 (in data pad cell 15)
H #3 <AR 17, H #3 17 13	000004_8 (in memory location 000001_8)

5.4.4 Subtract Register

Op Code	Operand
H {#2 / #3} SR	PA(a), H [flag] PA(b) PA(a-b)

This command subtracts the value indicated by 'PA(b)' from the value indicated by 'PA(a)' and stores the result in the location indicated by 'PA(a-b).' The flag options may be used to specify that the associated pad location contains the address of the memory location to be used in the operation.

Two's complement arithmetic is used in the subtraction operation. For example, suppose that data pad cell 12 contains the value 010000_8 and that data pad cell 16 contains the value 001000_8 . The command:

H #2 <SR 12, H 16 31

would result in data pad cell 31 containing the value 007000_8.

5.4.5 Increment Register

Op Code	Operand
H>#2 INCR #3	PA(a), H [#1] incr., PA(a+incr.)

This command computes the eight-bit immediate increment to the value indicated by 'PA(a)' and stores the result in 'PA(a+inc.)'. The increment may be specified as an octal value, $0<incr<377_8$. Increments of 200_8 to 377_8 represent negative numbers obtained by extending the leftmost bit left eight places.

The flag options may be used to specify that the associated pad location contains the address of the memory location to be used in the operation.

6. EXECUTION CONTROL

These commands allow the user to alter the sequence of execution.

6.1 Unconditional Transfer

GOTO

Op Code	Operand
GOTO	label@

This command transfers control to the statement with the given label. For example:

```
                    .
                    .
                    .
                <GOTO,>ROLLO
>LOAD3          <LOADPD,H 3 17, >STORLOC
                <INCBYWD,#2 17, 1
>ROLLO          <MOVEWD, #1 J, #1 K
                    .
                    .
                    .
```

The GOTO ROLLO command would transfer control to the MOVEWD #1 J, #1 K command, bypassing the LOADPD and INCBYWD commands.

GOTOR

Op Code	Operand
GOTOR	PA@

The GOTOR (go to register) command transfers control to the statement whose memory address is contained in the specified location in the data 'PA'. For example, suppose that data pad cell 13 contains 000454_8. The command:

<GOTOR, 13

would transfer control to memory location 000454_8.

6.2 Conditional Transfer

There are three types of conditional transfer commands, IF, SCAN, and BIT. An IF command results in a single comparison of two quantities, a SCAN command results in the comparison of a given quantity against a table of quantities, and the BIT command results in the comparison of only certain bits of two quantities.

IF

Op Code	Operand
IFEQ IFNE IFLT IFGT	a, b, label@

If the values indicated by 'a' and 'b' satisfy the indicated conditions an IF command transfers control to the statement with the specified 'label.' Otherwise, control passes to the next command in the sequence.

For example:

<IFLT, 2, #2 31, >RTN1

The above command transfers control to the statement labeled RTN1 if integer 2 is less than data pad location 31.

6.3 SCAN

Op Code	Operand
SCAN SCANLT	a, table

A SCAN command specifies that the value indicated by 'a' by compared with a given 'table' of values. If the value equals (SCAN) or is less than (SCANLT) a table value, control is transferred to the corresponding address. If the condition is not met for any table entry, execution proceeds with the next command in sequence. There must be at least two entries in a scan table and the last entry must begin with #1. When 'table' is not immediately preceded by a flag (#1) the table to be used is contained in line. Otherwise 'table' is a table address. The table format is as follows:

table value ; exit label

⋮ ⋮

#1 table value$_k$; exit label$_K$

For example:

```
<SCANLT, #2 >TERM
 57; >FMTERR
 60; >DESCEXIT
  :      :
#1  277; >FMTERR
```

The above command transfers control to the statement labeled FMTERR if the content of register TERM is less than 57. If the content of TERM is not less than 57 the next entry in the table (which is 60) is compared with the content of TERM. Again, if this is less than 60 control is transferred to the statement labeled DESCEXIT. Otherwise, the next entry in the table is compared with the contents of TERM, etc. The last entry of the table is preceded by a flag (#1). If the content of TERM is not less than 277 control passes to the next command in the sequence.

6.4 BIT** (LØGICAL2)

Op Code	Operand
BIT	arg1, arg2, label@

This command uses arg1 as a mask and tests the bits in arg2 according to the bits which are on in arg1. Control is passed to the label only if every bit which is on in arg1 is off in arg2. Otherwise the next maxro instruction in sequence will be executed.

For example:

Suppose memory location >MASK contains 22_8 and pad cell >L contains 20_8. Then the command:

```
<BIT, #1 >MASK, #2 >L, >NØTØFF
```

would result in execution control passing to the statement labeled >NØTØFF.

**Non-resident MACRØ

7. SUBROUTINE EXECUTION CONTROL

These commands allow the user to transfer execution control between subroutines. The subroutines may be those defined within the current system or they may be system subroutines, such as the disc input/output subroutines described in Sec. 9.

Subroutines may be nested to any level as long as the number of levels does not exceed the depth of the "push-down stack". Execution control is maintained by the system via a"push-down stack," as described below.

Data values may be passed between subroutines via the CALL command and the subroutine argument exchange commands described in Sec. 8. For nested subroutines, data values must be passed and retrieved at each level; that is, if subroutine A calls subroutine B which calls subroutine C, and data is to be passed from A to C, B must actively retrieve the data from A and pass it on to C.

7.1 Call Subroutine

Op Code	Operand
CALL	subroutine-name@[,parameter$_1$, ..., parameter$_n$]

This command transfers control to the named subroutine. A return address pointing to the first command following the CALL is added to the top of the subroutine control push-down stack mentioned above, along with the current execution mode.*

* Special execution modes may be defined using the INTERNAL and MODE3 commands (Sec. 5).

Parameters may be passed between the calling routine and the subroutine. There are two means of achieving this; either an argument may be passed to the called subroutine or an argument may be used to specify where the subroutine is to return data. These arguments may be specified indirectly by use of flags as in macro instruction operands.

7.2 Return to Calling Routine

Op Code	Operand
RETURN	

This statement is used to return control to a calling routine. The most current address and execution mode are retrieved from the top of the subroutine control push-down stack mentioned above, eliminating it from the stack. The specified execution mode is re-established, and control is passed to the retrieved return address.

7.3 Return to Specific Address

Op Code	Operand
RETURNTO	label@

This statement allows the user to replace the current return address with one of his own choosing. The most current address and execution mode are retrieved from the top of the subroutine control push-down stack mentioned above, eliminating it from the stack. The specified execution mode is re-established, but control passes to the memory location specified by 'label'.

8. SUBROUTINE ARGUMENT EXCHANGE

These commands allow the user to accept values from and transfer values to a calling subroutine. They are issued from within the called subroutine, and reference arguments specified in the calling program's CALL statement. Each one of the commands below refers to one argument in the parameter list of the calling program's CALL statement; successive commands refer to successive arguments, as will be illustrated in the examples below.

8.1 Get Single-Word Argument

Op Code	Operand
GETSARG	a*

This command obtains a single-word value from the immediate calling program and transfers it to the data pad or memory location specified by 'a*'.

Example:

Calling Program	Called Program
.	
.	
.	
<CALL, <ROUTINE1,5,#1 >VAL	<ROUTINE1
	<GETSARG, >MABEL
	<GETSARG, #1 >MORRIS

The first GETSARG in the called subroutine obtains the literal value 5 and stores it in location MABEL. The second GETSARG obtains the value stored in memory location VAL and stores it in the location whose address is stored in memory location MORRIS. Note that the arguments used in the called subroutine are used to specify where the data is to go.

8.2 Get Multiple-Word Argument

Op Code	Operand
GETMARG	n, a*

This command obtains an n word argument value from the calling program and transfers them to consecutive locations, beginning with the data pad or memory location specified by 'a '.

The n words are obtained beginning with the word specified by an argument in the parameter list of the CALL statement of the calling subroutine.

Example:

Calling Program	Called Program
	<RTNL
.	.
.	.
<CALL <RTNL, #1 >ADR, #2 15	.
	<GETMARG 5, >LOC1
	.
	<GETMARG 2, >LOC2

The first GETMARG in subroutine RTNL obtains 5 consecutive words beginning with memory location ADR and stores them in 5 consecutive memory locations, beginning with memory location LOC1. The second GETMARG obtains the values stored in data pad locations 15 and 16 and stores them in two consecutive memory locations, beginning with LOC2.

GETMARG cannot be used in internal mode (see Sec. 5).

8.3 Put Single-Word Argument

Op Code	Operand
PUTSARG	value

This command stores the value indicated at a location specified in the calling subroutine's CALL statement.

Example:

Calling Program	Called Program
:	<DOIT
<CALL,<DOIT, #1 >LOC1,#2 15,>STOR	:
	<GETSARG, >PLACE1
	<GETMARG, 2, >PLACE2
	:
	<PUTSARG, #1 >DONE

The GETSARG command retrieves the value stored in location LOC1 and stores it in memory location PLACE1. The GETMARG command obtains the values stored in data pad locations 15 and 16 and stores them in two consecutive locations beginning with memory location PLACE2. The PUTSARG command stores the value in memory location DONE into memory location STOR.

9. DISC INPUT/OUTPUT SUBROUTINES

These disk I/O subroutines are provided as part of the Macro Assembler; they must be called via a CALL command.

Certain types of arguments recur in the formats of these subroutine calls, as follows:

disk-descriptor — A argument defining the location of a specific track on disk. The first word of the location specified by the argument contains the pack number of the disk pack containing the desired track. The word immediately following this location contains the desired track number.

For example, suppose that location DADDR contains the value 3589_{10}, and that location DADDR+1 contains the value 5_{10}. Then #1 DADDR used as a disk argument would refer to track 5 of pack 3589.

In the READREC command (Sec. 5.9.8), a third word is used, containing the record number on the specified track which is to be read.

buffer-descriptor — an argument defining an area in memory into which data is to be read or from which data is to be written on disk. The first word of the location specified by the argument contains the address of the buffer area; the word immediately following this location contains the length of the area in words.

For example, suppose that location BUFAD contains the value 3577_8 and that location BUFAD+1 contains the value 3200_{10}. Then #1 BUFAD used as a buffer-descriptor argument would refer to the area 3200 words in length beginning at memory location 3577_8.

allocation-descriptor — an argument used by the ALLOCATE and RELEASE subroutines, which allocate and release disk storage. The first word of the location specified by the argument contains the pack number where the tracks are to be allocated or released. The word immediately following this location specifies the number of tracks to be allocated or released. The third word contains a special "allocation code," which normally can range from $0\text{-}11_{10}$. This code must be specified in both the ALLOCATE and RELEASE subroutine calls; it protects the allocated tracks from inadvertent release.

For example, suppose that location ALC contains the value 1722_{10}, ALC +1 contains the value 7, and ALC + 2 contains the value 10. Then #1 ALC used as an allocation-descriptor would refer to 7 tracks on pack 1722 with allocation code 10.

label — The address of a sub-program which will be executed under conditions specified with each subroutine. These sub-programs are executed as part of the sub-routine; that is, a RETURN in one will return control to the statement after the original sub-routine call. If an alternate location is to be transferred to, the RETURNTO label operation should be used in the sub-program.

9.1 Get Directory

Format

```
CALL GETDIR, disk desc., buffer-address
```

This subroutine reads a document directory from disk, on the pack and track specified by disk-descriptor, into memory beginning at the location specified by the buffer-descriptor. An entire track of 3200_{10} flagged words (18 bits/word) is read into this area.

This subroutine sets data pad cell 0 to the address of the first word of the area into which the data is read.

For example:

```
<CALL, <GETDIR,#1 >DIRADDR,>DIREC
```

The above command would read the 3200-word directory located on disk on the pack and track indicated by the two words at DIRADDR and DIRADDR + 1. The directory is read into 3200 18-bit words beginning at location DIREC.

9.2 Read Header

CALL READHDR A(doc-name), A(doc-directory), buffer-descriptor,e-exit@

This subroutine reads the "header" of the document, whose name is stored in the locations beginning with the location specified by "A(doc-name)'. The left-half word of the location specified by 'A(doc-name)' contains the number of characters in the name; the characters themselves follow (as would be the result of a GETNAME command, see Sec. 5.10.1).

The address of the directory where the document name is listed is specified by 'A(doc-directory)'.

The buffer-descriptor gives the location and length of the area into which the header is to be read.

If the indicated document name is not contained in the directory, control transfers to the 'e-exit' statement.

This command sets data pad cell 0 to the address of the directory; data pad cell 1 to the address of the header; data pad cells 2-6 are destroyed during the subroutine's processing.

The format of document headers is as follows:

Doubleword No.	Word 1		Word 2
1	Number of segments in the document		Document type indicator 0 = text 4or5 = program 2 = data
2	Record no. where segment 1 is stored (8 bits)	Track no. where segment 1 is stored (12 bits)	Length of segment 1 (12 bits)
3	Record no. where segment 2 is stored (8 bits)	Track no. where segment 2 is stored (12 bits)	Length of segment 2 (12 bits)
	⋮	⋮	⋮
n+1	Record no. where segment n is stored (8 bits)	Track no. where segment n is stored (12 bits)	Length of segment n (12 bits)

For example:

<CALL, < READHDR>DOCNAME, #1 >DIREC, #1 >HDRADR,>ERROR

The above command would cause the document directory whose address is stored in location DIREC to be searched for the document name stored in location DOCNAME and following words according to the conventions prescribed above. If the document name is listed in the directory, the header for the document is read from disk and stored in memory in the area whose beginning address and length are stored in location HDRADR and HDRADR + 1. If the document name is not in the directory, control transfers to the statement labelled ERROR.

9.3 Read Segment

Format

CALL READSEG A(doc-header), doc-segment#, buffer-desc, e-exit@

This subroutine reads a segment* of the document whose header is stored in memory locations beginning with the one specified by 'A(doc-header)' (see READHDR, Sec. 9.2). The number of the segment to be read is indicated by 'doc-segment#.'

The buffer-descriptor specifies the location and length of the area into which the segment is to be read.

If the length of the segment exceeds the length of the buffer area, or the segment number specified is greater than the number of segments, control is transferred to the address indicated by 'e-exit.'

For example:

<CALL, <READSEG, #2 0, 5, #1 >SEGADR, >ERROR

The above command would transfer the 5th segment of the document whose header is stored beginning at the memory location whose address is stored in data pad cell 0 to be read into the buffer area whose address and length are stored in locations SEGADR and SEGADR + 1. If the header indicates that there are fewer than five segments, control transfers to the statement labelled ERROR.

9.4 Seek Track

Format

CALL SEEK, disk-desc.

This subroutine initiates positioning of the disk drive on which the pack specified by the first word of disk-descriptor to the track specified by the second word of discriptor.

Since positioning can occur simultaneously with execution of macro processor instructions, this subroutine returns control after the seek is initiated; no indication is given of completion of positioning. A SEEK is not necessary before issuing other disk requests.

*In text documents, a segment corresponds to a page.

9.5 Write Track

Format

CALL WRITE [F], disk-desc, buffer-desc

This subroutine writes the contents of the buffer area defined by buffer-descriptor onto disk at the disk location specified by disk-descriptor.

The F option indicates that the write is to be performed in flagged format; i.e., transfer of 18-bit words--16 bits plus the two associated flag bits. If the F option is not specified, unflagged 16-bit words are transferred.

The buffer length must be less than or equal to track capacity (3200_{10} flagged or 3600_{10} unflagged words).

Examples:

```
<CALL,<WRITE,#1 >DADDR, #1 >BUFADR

<CALL,<WRITEF, #1 >DISCAD, #1 >BUFFER
```

9.6 Read Track

Format

CALL READ[F], disk-desc., buffer-desc., e-exit@

This subroutine reads the entire track specified by disk-descriptor into memory in the buffer area specified by buffer-descriptor.

The F option indicates that the data exists on disk in flagged format (18-bit words) and is to be read as such into memory. If the F option is not specified, the data is assumed to exist on disk in unflagged format (16-bit words) and is to be read as such into memory. If the track was not written in the format in which the read requested, a DISK FORMAT ERROR will occur. An error message will be printed and control will be transferred to the system.

If the length of the data on the track exceeds the buffer length, the first portion of the track is read into the available buffer area; the data remaining on the track is not read, and control passes to the location specified by 'e-exit'.

Examples:

```
<CALL, <READ, #1 >DISCAD, #1 >BUFAD, >OVRFLOW

<CALL, <READF, #1 >discad, #1 >bufad, >error
```

9.7 Write with Protection Check

Format

CALL WRITEP[F], disc-desc., buffer-desc.

This subroutine functions in the same manner as WRITE[F] except that the track will be read back and compared word for word with the buffer contents. If they do not match exactly, a fatal error occurs.

9.8 Read Record

Format

CALL READREC, disc-desc., buffer-desc., e-exit

This subroutine reads the specified record of the given track (as specified by 'disc-desc.', in this case three words long--see preceding description of disc-descriptor arguments) into the area specified by 'buffer-desc'. In the record length exceeds the buffer length, control is transferred to the statement with the label 'e-exit.'

READREC may be executed only for those tracks written in record format, as follows:

RECORD FORMAT FOR TRACK

Word No.	Contents
1	n = number of records on track
2	location of next available word on track
3	displacement to record 1 (words to be skipped + 1)
4	length of record 1, in words
5	displacement to record 2 (words to be skipped + 1)
6	length of record 2, in words
⋮	⋮
2n+1	displacement to record n (words to be skipped + 1)
2n+2	length of record n, in words
2n+3	Record 1
⋮	Record 2
⋮	⋮
⋮	Record n
	(Unused portion of track, if any)

9.9 Allocate Tracks

Format

CALL ALLOCATE, allocation- desc., buffer-desc., track number*

This subroutine allocates a number of tracks on a disk pack under a given allocation code, as specified by the allocation-descriptor argument. (See discussion of allocation-descriptor arguments at the beginning of Sec.. 9).

The buffer-descriptor gives the address and length of a buffer area to be used by the ALLOCATE subroutine; the length of the buffer area must be at least 2100 words.

The address of the first track allocated is returned in the argument; if no space is available, the argument will be set to -1.

For example:

```
<CALL ,<ALLOCATE, #1 >ALCAD, #1 >BUFAD, >TRAKLOC
```

9.10 Release Tracks

Format

CALL RELEASE, allocation-desc, buffer-desc, track number

This subroutine releases previously allocated tracks. The allocation code in the allocation-argument must be the same as was used to allocate the given tracks.

The buffer-address gives the address and length of a buffer area to be used by the RELEASE subroutine; the length of the buffer area must be at least 2100_{10} words.

The address of the first track to be released must be specified in the third argument.

10. KEYBOARD ARGUMENT FETCHING

These commands allow the user to accept input from the keyboard during program execution.

10.1 Get Name

Op Code	Operand
GETNAME	H PA(terminator) PA(name)

This command accepts a "name" -- up to nine alphanumeric characters-- from the keyboard on the current input statement. The name is terminated when a nonalphanumeric character is encountered. The terminating character is placed in data pad cell'PA(terminator).'

The character string comprising the name is also stored in data pad. The left byte of cell 'PA(name)' is set to the total number of characters in the string. The name is stored in consecutive bytes, beginning with the right byte of cell 'PA(name)'. If no alphanumeric characters are encountered, PA(name) will be set to 0. If the current input statement was exhausted, the message'EOS ERROR' will be printed on the screen.

For example, suppose the user's program contains the command:

```
    .
    .
    .
<GETNAME,H 17 18
    .
    .
    .
```

If the current input line contained the characters:

PROG$

the result would be:

	Octal Value		Character	
Data Pad Cell	Left Byte	Right Byte	Left Byte	Right Byte
17	000	044	0	$
18	004	120	004_8 (no. of characters)	P
19	122	117	R	0
20	107	000	G	x

10.2 Get Octal

Op Code	Operand
GETOCTAL	H PA(terminator) PA(n)

This command accepts an octal number n ($100000 \leq n \leq 77777$) from the keyboard on the current line. The number is terminated when a non-octal character is encountered. The terminating character is placed in data pad cell 'PA (terminator)'.

The octal number is translated from a character string to a binary value and stored in data pad cell 'PA(n)'. If no octal characters are entered before a non-octal character is encountered, the value 0 is entered in 'PA(n)'. If the current statement was exhausted, the message 'EOS ERROR' will be printed on the screen.

For example, suppose the user's program contains the command:

```
    .
    .
    .
<GETOCTAL, H 31 30
    .
    .
    .
```

If the input line contains:

327C.

The results in data pad are as follows:

Data Pad Cell	Octal Value
30	000327
31	000103 (character "C")

10.3 Get Decimal Integer

Op Code	Operand
GETINTGR	H PA(terminator) PA(n)

This command accepts a decimal integer number n ($0 \leq n \leq 32767_{10}$) from the keyboard on the current statement. The number is terminated when a non-numeric character is encountered. The terminating character is placed in data pad cell 'PA(terminator)'.

The decimal number is translated from a character string to a binary value and stored in data pad cell 'PA(n)'. If no decimal characters are entered before a non-numeric character is encountered, the value 0 is entered in 'PA(n)'. If the current statement was exhausted, the message 'EOS ERROR' will be printed on the screen.

For example, suppose the user's program contains the command:

```
      .
      .
      .
<GETINTGR,H 31 35
      .
      .
      .
```

If the input line contained:

```
19N
```

The results in data pad are as follows:

Data Pad Cell	Octal Value
31	000116 (character "N")
35	000023

10.4 Get Key

Op Code	Operand
GETKEY	next single key*

This command obtains the next single value from the current statement and transfers it to the memory location specified by 'next single key*'. If all keys in the current statement have been processed, a carriage return key code (57_8) is returned. For example:

```
<GETKEY, #2 KEY
```

The above command obtains the next input value and stores it in pad cell location KEY.

10.5 MDECKR

The next input key pointer for the current statement is adjusted backwards by one position. This allows reprocessing of the last key input. The key input is indeterminate if an attempt is made to reposition the input pointer before the beginning of the statement.

11. DISPLAY SCREEN OUTPUT

These commands allow the user to form displays for the screen during program execution.

11.1 Build Octal Characters

Op Code	Operand
BLDOCT	H PA(character string) n, value

This command translates the binary value indicated by value to a string of octal character codes beginning at the data pad location specified.

The n operand specifies the maximum number of octal digits to be converted. A total of (n + 3)/2 data pad cells are used. One leading blank is stored in the right byte of the first word. The characters are assembled from right to left; blanks suppress leading zeros.

For example, suppose the memory location CHARLES contains the value 013562_8. Various commands and their result in data pad are illustrated below.

Command	Data Pad Location	Character String Left Byte	Character String Right Byte	
<BLDOCT ,H 20 6, #1 >CHARLES	20	7	40	
	21	40	61	
	22	63	65	
	23	66	62	
<BLDOCT ,H 20 3, #1 >CHARLES	20	4	40	(characters
	21	65	66	assembled
	22	62	0	right to left
				to total of
<BLDOCT ,H 20 11, #1>CHARLES	20	12	40	n)
	21	40	40	
	22	40	40	(characters
	23	61	63	assembled
	24	65	66	right to left,
	25	62	0	total of n,
				blank fill)

11.2 Display Character String Coded

Op Code	Operand
TYPE	'character string

This command causes the character string indicated by 'character string' to be displayed on the screen, beginning at the current position of the cursor. The character string may consist of up to 255 characters from the gray keys on the keyboard, including spaces. Character string must be preceded by an apostrophe, or if leading blanks are to be typed, by two apostrophies. Can be used, together with BLDOCT to display a number in octal.

For example: <TYPE, #2 20 would display any of the numbers in the previous examples.

11.3 Display Character String from Memory

Op Code	Operand
TYPEWD	word pointer list

This command displays characters stored in character string format in memory on the screen beginning at the current position of the cursor. The location indicated by 'word pointer list' contains an integer value m giving the total number of character strings to be displayed. The m consecutive words following this location are the addresses of separate character strings which will be displayed, one after another.

For example, suppose memory location FREEN contains the value 000003_8, and that consecutive locations following FREEN contain FIRST, SECOND, THIRD. The command <TYPEWD, #1 >FREEN would cause the 3 messages at locations FIRST, SECOND and THIRD to be printed on the screen at the current position of the cursor eg. if the contents of these locations were:

```
FIRST     Library ␣ (↵)
SECOND    DATA ␣ (↵)
THIRD     is undefined (↵)
```

␣ denotes a space.
The message displayed on the screen would be:

Library DATA is undefined

11.4 Carriage Return

Op Code	Operand
CRETURN	

This command positions the cursor at the beginning of the next line.

11.5 Space

Op Code	Operand
SPACE	

This command positions the cursor one space further on from its current position.

11.6 Skip

Op Code	Operand
SKIP	

This command positions the cursor at the next tab stop (as indicated by the FORMAT command, Sec. 5.11.10.) An attempt to SKIP past the last tab stop will cause an automatic CRETURN.

11.7 Position Cursor**

Op Code	Operand
UP DOWN OVER BACK	# positions

This command positions the cursor up, down, right or left the specified number of lines or character positions.

11.8 Reset Cursor

Op Code	Operand
TYPRSET	

This command positions the cursor at the first position on the top line of the display screen.

11.9 Erase

Op Code	Operand
ERASE	

This command erases the display screen and positions the cursor to the first position at the top line of the screen.

**Non-resident MACRØ

11.10 Set Starting Point for Line** (LINE)

Op Code	Operand
POINT	x-value, y-value

This command initializes the line position pointer to the specified x, y position on the screen. No line is drawn with this command. X-value and y-value are given in raster coordinates:

$$-3200 \leq x < 3200$$
$$-4095 \leq y \leq 4095.$$

11.11 Draw a Straight Line** (LINE)

Op Code	Operand
LINE	x-value, y-value

This command connects the last point (or endpoint of the previous line) and the point specified by a straight line. The x and y values are given in raster coordinates:

$$-3200 \leq x < 3200$$
$$-4095 \leq y \leq 4095$$

11.12 Display Decimal Integer Subroutine

Format

CALL DECOUT, a

The value of a as a 2's complement signed decimal integer is displayed on the screen in a seven-character field. The first character is a blank, the second is a minus sign (-) if the number is negative, or another blank. The last five characters will contain the decimal value, right adjusted with leading zeros placed by blanks.

** Non-resident Macro

12. FLAG OPTIONS

These instructions allow the user to either separate or merge flags and their associated data.

12.1 Expand Flags**

Op Code	Operand
EXPNDF	H reg PA(A(value))

This command transfers a flagged word of data from a memory location, whose address is given in data pad, to a pair of consecutive data pad cells such that the data is stored in 'reg' and the flag value in 'reg'+1.

For example, suppose DESCBASE contains the word #1 309. Then the command.

<EXPNDF,H >WS >DESCBASE

results in the transfer of 1 into WS+1 and 309 into WS.

12.2 Merge Flags**

Op Code	Operand
MERGEF	H reg PA(Address)

This command assumes that the flags are stored in 'reg' and the data in 'reg'+1. It merges the flags and data and stores them in a memory location whose address is given in data pad.

For example, suppose 1 is stored in WS and 309 in WS+1 then the command

<MERGEF, H >WS >DESCBASE

merges #1 with 309 and stores them in memory location DESCBASE.

13. MACRO INSTRUCTION AND SUBROUTINES RELATED TO BACKGROUND JOB PROCESSING

13.1. Add a Job to the Job Queue

Format

CALL, JOB, j-desc, c-exit, parameter, priority

If a job queue element is available, it is initiated with the information provided in the parameter and inserted in the job queue in priority order. The address of the job queue element is returned in PD 0. If no job queue element is available, PD 0 will be zero.
The parameters are:

a) j-desc - a two-word descriptor of the job. For disk-based programs, it is the starting track number and the total program length. For MOS resident programs, the first word is zero, the second word is the base address of the program's save area in MOS.

b) c-exit - the address of the completion exit routine, which is a subroutine which will be called when the job completes.

c) parameter - a 16-bit value which will be placed in data pad cell 0 when the job is executed.

d) priority - a number between 0 and 255 which represents the scheduling priority of the job. Each time background service is to be started, the job on the job queue with the largest priority number will be selected for service. In case of jobs with equal priority, service will be on a first-come, first served basis.

13.2. Establish Limit of Background Area Used

SETBBND, limit address, e-exit

The BKBND value is set to the limit address specified, provided that this address does not exceed BKLIMIT. If it does, control is transferred to the e-exit address. If a job is swapped out, all memory between BBASE and BKBND will be written on disk and restored to MOS when the job is resumed. SETBBND must be used before ERASE, BREAK or CALL GETSTMNT. Excessively large values for the limit address will cause increased swapping time.

13.3. Allow Pre-emption of Processor Service

BREAK

This instruction is used to allow other background jobs with higher priority or interactive requests to receive service. The current job is interrupted, and if any interactive service is required, it is given. When all interactive requests have been serviced, the highest priority job will be given service. BREAK should be used at regular intervals within any pre-emptable job. SETBBND must be used at some time prior to BREAK in order to establish the swap area limit.

13.4. Wait for Completion of Some External Event

BWAIT

The current job is transferred from the job queue to the wait stack. This job will receive no further processor service until some other process (background, interactive or interrupt) moves the job back onto the job queue. This instruction may be used to synchronize two separate jobs, providing that both know the address of the waiting job's job element.

13.5. Restore a Waiting Job to the Job Queue**

CLRBWAIT, A(job element)

The job whose job element address is given is transferred from the wait stack to the job queue, restoring it to eligibility for processor service according to its priority.

13.6. Obtain the Next Input Statement from an Interactive Terminal

CALL, GETSTMNT

The current input statement from the associated terminal is purged. If another statement has been input, control is returned and the statement may be processed using the normal input macro instructions. If no statements are available, the job is placed in the wait state until a statement is entered. If more than one job requests input from the same input terminal at the same time, an error message is printed.

13.7. Release Associated Terminal for Independent Interactive Service

CLRWAIT

When a job is scheduled as a result of either the PGM or OP operations in the interactive library mode, further interactive service for that terminal is blocked to permit the background job to interact with the user. If no such interaction is needed, the job may use the CLRWAIT macro to permit interactive service for that terminal to be overlapped with the job processing. If this is done, the job should not attempt to process input statements from the terminal.

13.8. Terminate Job

Op Code	Operands
PULLOUT	

The job is terminated, control transfers to the job scheduler which calls the completion routine specified at job submission and then re-schedules the processor.

13.9 Fetch Station's Interactive Track

GETITRK

Once a job is in execution, the only guarantee that the submitting station's interactive track is the current one is to use the GETITRK macro. Only one macro instruction is allowed to fetch or change any information on that interactive track, since another station's interactive track can be read in any time an interrupt can be processed, which is normally between any two macro instructions. GETITRK, on exit, bypasses this interrupt check in order to ensure that the interactive track which was fetched remains in MØS for at least one macro instruction.

For example:

To change the skip table being used for system displays, the user could use the following macro instruction sequence:

```
<GETITRK
<MOVEWD, >SKIPTBL, #1 <STYPETAB+2
```

The skip table used for typing would then be the table at memory address >SKIPTBL.

13.10 Test for the Presence of an Input Statement

IFSTMNT, label

The IFSTMNT macro allows a job to test for the presence of an input statement without entering wait state in the case that a statement is not currently available. If a statement is available, control will be passed to the macro program at the location specified by the label, where the GETSTMNT subroutine may be called with the knowledge that the job will not be placed in the wait state. If not, execution control will pass to the macro instruction following the IFSTMNT instruction.

For example:

To wait for an input statement without entering wait state, the following code could be used:

```
>CHK        <IFSTMNT, >GØTØNE
            <GØTØ, >CHK
>GØTØNE     <CALL, <GETSTMNT
```

14. Array Processor Support Macros

Three sets of array processor support macro instructions are available in the shared text library MACRØS, along with a set of subroutines which use some of them. These sets are divided into two basic sets of instructions, the first of which provides for use of the array processor as a floating point arithmetic unit, with the second set providing fixed point array transforms.

14.1 Loading AP Program Store (RAM)

Before either class of macro instructions can be used, the array processor's program store must be loaded with the appropriate sets of routines. Currently there are three standard sets of AP routines, one of which is presumed present by both of the other two. One macro instruction is provided with which to assure that the correct sets of AP routines have been loaded.

14.1.1 LØADPS -- Load AP Program Store

```
<LØADPS, Argument
```

Assumes the assembled and linked program store load is present in MØS at the address specified by the argument. Takes the first word at that address as the length (in 32-bit words) and the second word as the starting Program Store Address of the load and places the instructions in the AP Program Store. The INCLUDE pseudo op is used to create the proper MOS data.

14.1.2 APBASIC -- Common AP support for Floating and Fixed Point Operations

```
<LØADPS, <APBASIC
```

Places the common support routines in the AP program store as they are assumed by the floating point and fixed point AP operations.

14.1.3 APFLØAT -- Floating Point AP Operations

```
<LØADPS, <APFLØAT
```

Loads the routines for floating point operations into the AP program store.

14.1.4 APFFT -- Fixed Point AP Operations

```
<LOADPS, <APFFT
```

Loads the routines for fixed point transforms into the AP program store.

14.2 Floating Point Operations

The array processor functions as a stack oriented processor for floating point operations. Each operator acts on one or more operands from the top of the stack, replacing them by the result of the operation. In addition, operands may be transferred from the macroprocessor to the top of the stack, the top stack element may be transferred to MP MOS, the top stack element may be copied on top of itself, and the top stack element may be removed.

Sequencing Requirements - In order to transmit data and commands to the array processor, a communication path between MP-MOS and the AP must be established. The LOADAP, STACKAP and TIEAP commands all establish this path. Once the communication path is established, no macro instructions except array processor commands may be used until the path is broken. The STOREAP and UNTIEAP commands break the communication path. The AP commands listed in sections 14.2.4-14.3.1, as well as STOREAP, may be used only after the communication path is established. Hence, each sequence of AP macro instructions must have the following order:

A. Establish communication path with LOADAP, STACKAP or TIEAP

B. AP macro instructions, including LOADAP or STACKAP if desired

C. STOREAP or UNTIEAP.

Data Transfer Instructions

14.2.0.1 TIEAP The communication path between the MP32A and the AP is established, permitting use of AP macro instructions.

14.2.0.2 UNTIEAP The communication path between the MP32A and the AP is broken, permitting use of non-AP macro instructions.

14.2.1. STACKAP

```
STACKAP, { A(data)@
         { #2 PA (A(data))@
```

One 32 bit doubleword is transferred from MP-MOS at the address given to the top of the AP stack. All old elements already in the stack are pushed down one place. The address given must be even.

14.2.2. LOADAP

```
LOADAP { A(data)@
       { #2 PA(A(data))@
```

One 32 bit doubleword is transferred from MP-MOS at the address given to the top of the AP stack, replacing the old top stack element. All other elements in the stack retain their old positions. The data must be located in an even-odd pair of 16 bit MOS words.

14.2.3. STOREAP

```
STOREAP { Address@
        { #2 PA(Address)@
```

The element currently at the top of the AP stack is copied into MP-MOS memory at the doubleword address given. The stack contents do not change.

Data Manipulation Instructions - In the following descriptions, X represents the top of stack element and Y represents the second element on the stack.

14.2.4. ADDAP X+Y is placed on top of the stack. Both X and Y are removed from the stack.

14.2.5 SUBTAP Y-X is placed on top of the stack. Both X and Y are removed from the stack.

14.2.6 MULTAP X*Y is placed on top of the stack. Both X and Y are removed from the stack.

14.2.7 DIVAP Y/X is placed on top of the stack. Both X and Y are removed from the stack.

14.2.8 SQAP X*X replaces X.

14.2.9 SQRTAP $X^{1/2}$ replaces X if $X \geq 0$. 0 replaces X if X<0.

14.2.10 NEGAP -X replaces X.

14.2.11 ABSAP $|X|$ replaces X.

14.2.12 SGNAP SGN(X) replaces X where SGN(X) = $\left\{\begin{array}{l} 1 \text{ if } X>0 \\ 0 \text{ if } X=0 \\ -1 \text{ if } X<0 \end{array}\right\}$

14.2.13 INVAP 1/X replaces X.

14.2.14 LOGAP Natural logarithm ($|X|$) replaces X if $X \neq 0$.

14.2.15 EXPAP Exponential (X) replaces X.

14.2.16 SINAP Sine (X) replaces X.

14.2.17 COSAP Cosine (X) replaces X

14.2.18 ATANAP Arctan (X) replaces X.

14.2.19 PUSHAP X is copied on top of itself, leaving two copies of X as the top two elements on the stack.

14.2.20 PULLAP X is removed from the stack, leaving Y as the top stack element.

14.2.21 FIXAP X is converted from floating point to 32 bit 2's complement integer. If $|X| \geq 2^{23}$, the result is indeterminate. Fraction bits are truncated.

14.2.22 FLOATAP The rightmost 24 bits of X are assumed to represent a 2's complement integer. This integer is converted to floating point.

14.2.23 WAITAP Processing is held up until an AP interrupt occurs, at which time normal macro sequencing is continued.

Data Transfer and Manipulation Operations

14.3.1. MBYPWR

MBYPWR { n@ / #2 PA(n)@ }

X is multiplied by 10^n, where n may be positive or negative. The final product replaces X on the top of the stack.

14.4 Subroutines Using AP Macro Instructions

These subroutines are available as documents in the text library MACROS. Each communicates with the Array Processor as necessary for arithmetic support. They assume the programs for floating point operations are loaded in AP-PS.

14.4.1. FPOUT

```
CALL FPOUT, Number
```

The 32-bit floating point number specified is printed on the interactive console display screen at the current character output position. The format used is:

$$_[-]N.NNNNN[E\{\pm\}NN]$$

Where N represents a decimal digit and the portions shown in brackets are replaced by blanks when they do not apply. The exponent does not apply if it is 0.

14.4.2. DISPLAY - Curvilinear Display

```
CALL DISPLAY A(Desc.),CODE,A(Scale)
```

An array whose description is given is plotted on the interactive console display screen. Either real or complex, fixed or floating point data can be plotted using this subroutine. Real data is plotted at equal spaced abscissa values. Complex data is plotted using the real portion for the abscissa values and the imaginary portion for the ordinate values.

The descriptor whose address is the first parameter is four words long. These four words are:

1) H 4 Type where Type is: 111_8 for real, fixed point
   143_8 for complex fixed point
   122_8 for real, floating point
   103_8 for complex, floating point.

2) Base Address of array
3) Number of points in array
4) Address after last word in array

Fixed point data requires a 16-bit word per value; floating point data requires a 32-bit doubleword per value. Two values are required for each complex point, these occur as consecutive values, real followed by imaginary. Real data requires one value per point.

The CODE is ignored at present except for complex arrays in which case a code of 0 is a request for a complex display, a code of 1 is a request for a real display of the real portion of each point, a code of 2 is a request for a real display of the imaginary portion of each point.

The scale is always a 32-bit floating point doubleword representing the desired maximum ordinate value. This value must be greater than any value which will be plotted on the ordinate axis. For complex displays, the maximum abscissa value is approximately .75 of the scale value, due to the smaller horizontal dimension of the display screen. This, .75 times the scale should be greater than any value which will be plotted on the abscissa axis. Fixed point numbers are treated as integers for the purposes of scaling.

Nonresident macros required: FPØUT, LINE, APBASIC, APFLØAT

```
Example: <CALL,<DISPLAY,>AD, 0,>SC "complex fixed point
         <CALL,<DISPLAY,>AD, 2,>SC "real display of imaginary part

     >AD   H 4 143, >A,2000,>A+4000
      I
      D    "This gives doubleword alignment
     >SC   H 7 100,0   "Scale is 64=(2**7)*.5
```

14.4.3 PRINT

CALL PRINT A(Desc.)

The elements of an array whose descriptor is given are typed on the interactive console display screen. Either real or complex, fixed or floating point data can be printed using this subroutine. As many elements will be printed as are in the array or as will fit on the screen, whichever is less.

Nonresident macros required: FPØUT

For example: <CALL,<PRINT, >AD

```
>AD H 4 143, >A, 2000, A+4000
```

This will cause the first approximately 200 points of the complex fixed point array at location ">A" to be printed on the display screen, under normal formatting.

14.5 Fixed Point Operations

The array processor performs the following fixed point transforms on 16-bit data:

1) Fast Fourier Transform (FFT)
2) Inverse Fourier Transform (IFT)
3) Convolution
4) Recursive Filtering

The transform operations are restricted to operate on arrays containing exactly a power of two number of points, up to 4096. The FFT can transform either real or complex input data. The input for the IFT is always complex. Convolution and recursive filtering operate on real data only.

Fixed Point Support Operations

All the following assume the AP program store (PS) has been loaded with the fixed point routines. All MP memory addresses must be even.

14.5.1 Set Data Pad Address Register*

SETDPA, address

The data pad address register is set to the address given. DPA is used to address both data pad and program store for writing only.

14.5.2 Set Memory Address Register

SETMA, address

The memory address register is set to the address given. Normally, MA is set to one less than the first address where data is to be written, but to the exact address when data is to be read.

14.5.3 Set Data Read Only Memory Address Register

The data ROM address register is set to the address given.

14.5.4 Set Program Source Address Register*

SETPSA, address

The program source address register is set to the address given. This initiate array processor execution of the routine which starts at the specified address.

14.5.5 Load Program Source with a Special Routine*

WRTAPPS, A(program), length

*These macros do not require any routines to be loaded in PS.

Successive doublewords are transferred from memory at the address given to AP program source at the address given to AP program source at the address in DPA, incrementing DPA by one and CA by 2, until the length specified has been transferred.

14.5.6 Load Array Processor Memory

WRTAPMD, A(data), length

Doublewords are transferred from MP memory at the address given to AP memory at the address in MA+1 for the length (in doublewords) indicated.

14.5.7 Load Array Processor Data Pad

WRTAPDP, A(data) length

Doublewords are transferred from MP memory at the address given to data pad at the address in DPA for the length (in doublewords) indicated.

14.5.8 Load Array Processor Data Pad Left Halfwords

LOADDPL, A(data) length

Successive doublewords are transferred from MP memory at the address given to two data pad cells, beginning at DPA, and decrementing DPA as each word is stored.

14.5.9 Load Array Processor S-Pad Register

WRTAPSP, Spvalue, spa

The right twelve bits of the spvalue given are loaded into the s-pad register whose address is given.

14.5.10 Read Contents of Array Processor Memory

RDAPMD, A(buffer), length

The specified number of doublewords are transferred from AP memory, beginning with the current value of MA, to the buffer in MP memory specified.

14.5.11 Read Contents of Array Processor Data Pad

RDAPDP, A(buffer), length

The specified number of data pad doublewords are transferred into MP memory at the address indicated. Transfer begins with the data pad cell whose address is in DPA.

14.5.12 Read Array Processor Data Read Only Memory

RDAPDR, A(buffer), length

The specified number of doublewords are transferred from AP data ROM, starting at the address in DRA, to the MP-memory address specified. S-pad 0

must contain a 1 for this operation to function correctly.

14.5.13 Read Current Value of S-Pad Cell

RDAPSP, spa, A(buffer)

The rightmost 8 bits of the specified S pad cell are transferred to the leftmost 8 bits of the doubleword MP memory location specified.

14.6 Fourier Transform Subroutines

Each of these subroutines transfers a data array from MP memory, initiates the transform in the array processor, and returns. MP activities which do not require the results of the transform may then be executed in parallel with the transform. When the results are needed, the subroutine STORET is used to transfer the output of the transform, when ready, back to MP memory. Any attempts to initiate new transforms before the completion of the previous transform will be ignored.

These subroutines all load AP-PS with the fixed point routines themselves, and STORET restores the floating point routines when it has finished.

14.6.1 Initiate Complex Fast Fourier Transform

CALL, FFTC, $\log_2$ (blocksize), $\log_4$(buffer length), A(buffer)

The blocksize of the array to be transformed may be 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 or 4096 complex points. Only the $\log_2$ is specified in the subroutine call (2, 3, 4, 5, 6, 7, 10, 11, 12, 13, 14_8 respectively). The transform algorithm used requires that a power of four number of points be transformed, the possible buffer sizes are 4, 16, 64, 256, 1024 and 4096. Only the $\log_4$ is specified in the subroutine call (1, 2, 3, 4, 5, 6 respectively). When the blocksize is less than the buffer size, several transforms are calculated simultaneously. Thus, one 4096 point transform, 2-2048 point transforms 4-1024 point transforms, etc. could be performed at one time. The input data is arranged in one MP memory buffer, with the first array preceeding the second which preceeds the third, etc.

14.6.2 Initiate Real Fast Fourier Transform

CALL, FFTR, $\log_2$(blocksize), $\log_4$(buffer length), A(buffer)

All parameters are the same as for the complex FFT, but each doubleword now contains one point from each of two input data arrays which are to be transformed simultaneously. Thus, 2-4096 point real input arrays may be transformed at the same time. One array will occupy the real part of each doubleword (the even words), the other will occupy the imaginary parts (the odd words). The transform output will be complex, with the output for the real array followed by that for the imaginary component array. In each array only the first blocksize/2 output values will be returned, the remaining values can be computed if needed from the identity

$$F^N_{N-K}(s) = F^N_K(s)^* \quad \text{where the * indicates complex conjugation.}$$

If more than two real arrays are to be transformed at once, the additional arrays must be interleaved pairwise in subsequent blocks.

14.6.3 Initiate Inverse Fourier Transform

CALL, IFTC, $\log_2$(blocksize), $\log_4$(buffer length), A(buffer)

All parameters are the same as for FFTC, but the inverse transform is calculated.

14.6.4 Return Result of Fourier Transform

CALL, STORET, A(buffer), length, DS

When the last intiated transform is completed, the AP memory buffer will be transferred to MP memory at the address given. The length given is the number of doublewords to be transferred. A binary scale factor is returned to DS, which may be a memory address or a data pad cell. This factor will be a non-negative integer giving the power of two by which each point in the transformed output must be multiplied to obtain the result on the same scale as the input data.

14.7 Convolution and Filtering Subroutines

These subroutines transfer the kernels and data vectors to the array processor, initiate the operation, and return. MP operations may then be executed in parallel with the AP operation. When the result of these operations is needed, the subroutine STORET is used as described above. In this case, however, the value of DS returned is meaningless.

14.7.1 Initiate Convolution

CALL, CONVOLVE, A(kernel), length of kernel, A(data), length of data

The specified kernel, which must be less than 32 points long, is transferred into the left half of data pad. The data vector is transferred into AP memory. The convolution is initiated in the array processor. Both kernel and data must be located on doubleword boundaries. The convolution algorithm performed is:

$$K*f(j) = \sum_{i=o}^{\ell} K_{\ell-i} \cdot f_{j+i} \qquad j=0,\ldots,m-\ell$$

where the kernel K has $\ell+1$ points and the vector f has m points ($m \leq 8192$).

Note that the result has only $m-\ell+1$ points.

14.7.2 Initiate Recursive Filtering

CALL, FILTER, A(forward kernel), length of forward kernel, A(feedback kernel), length of feedback kernel, A(data), length of data.

APPENDIX 8

Microprogramming Manual

CHI

SIGNAL SYSTEM

TM41

SOFTWARE

TM44 MP-32 MICROPROGRAMMING MANUAL

CONTENTS:

1. INTRODUCTION
2. INTERNAL STRUCTURE
3. INSTRUCTIONS
4. FIELD DEFINITIONS BY MODE #
5. TIMING
6. CONVENTIONS FOR MACRO-INSTRUCTION DEFINITIONS
7. CHECKOUT OF MICRO-INSTRUCTIONS
8. EXAMPLES OF MICRO-INSTRUCTIONS

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	7/11/74	p. 2-7 Appendix A, pp. 2,4,15,17
B	9/6/74	Appendix A, pp. 13 and 13a
C	10/23/75	2-1,2-2,2-3,2-7,2-10,2-11,3-1,4-1,5-1, 2-4,7-7,7-8,Appendix A & B

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29107

© 1973
by Culler/Harrison, Inc.

CONTENTS

		Page
Chapter 1.	INTRODUCTION	1-1
Chapter 2.	INTERNAL STRUCTURE	2-1
	2-1 Scope	2-1
	2-3 Input/Output System	2-10
	2-4 Dimension and Address Convention	2-11
Chapter 3.	INSTRUCTIONS	3-1
	3-1 Format	3-1
	3-2 Field Definition	3-1
	3-3 Instruction Execution	3-2
	3-4 Execution of Instructions Directly from MOS Memory	3-3
	3-5 Instruction Subsets	3-4
Chapter 4.	FIELD DEFINITION BY MODE NUMBER	4-1
	4-1 Introduction	4-1
	4-2 Mode 0	4-1
	4-3 Mode 1	4-1
	4-4 Mode 2	4-1
	4-5 Mode 3	4-1
Chapter 5.	TIMING CONSIDERATIONS	5-1
	5-1 Cycle Times	5-1
	5-2 Execution Times	5-1
	5-3 Occurrences at TC=0 Only	5-1
	5-4 Operations with MOS	5-1
Chapter 6.	CONVENTIONS FOR MACRO INSTRUCTION DEFINITIONS	6-1
	6-1 Scope	6-1
	6-2 Definition	6-1
	6-3 Macro Operation Sequencing	6-1
	6-4 Instruction	6-2
	6-5 Interrupt Service Routines	6-3
	6-6 Accessing Macro Instruction Arguments	6-3
	6-7 Data Pad Usage	6-3
Chapter 7.	CHECKOUT OF PRIMITIVES	7-1
	7-1 Control Panel	7-1
	7-2 Primitive Checkout	7-7

Chapter 8. EXAMPLES OF MICRO-INSTRUCTIONS 8-1

8-1 Scope 8-1
8-2 Fetch a Macro-Instruction Argument 8-1
8-3 Scan a Table in Data Pad for a Match with A1 8-3
8-4 Data Pad and Adder Illustration 8-4
8-5 Interrupt Service Routine 8-5

TABLES: 2.1 Adder Summary 2-4
3.1 Instruction Subsets 3-5
4.1 Mode 0 Field Definitions 4-2
4.2 Mode 2 Field Definitions 4-2
7.1 Control Panel Controls 7-3
7.2 Control Panel Indicators 7-5

FIGURES: 2.1 Macro Processor Block Diag. 2-2
6.1 Operations Sequencer 6-2
7.1 Control Panel 7-2
8.1 Examples of Micro-Instructions 8-1
8.2 SEARCH 8-3
8.3 Edit Insert Setup 8-4
8.4 CLRAPBUSY 8-5

APPENDIX A: MP Micro Instruction Summary
APPENDIX B: MICRO Operation Definitions

PREFACE

This document contains the information necessary for programming the MP-32A in the primitive micro-instruction language executed directly by the hardware. Primitive micro-programs may be used to define new macro-instructions for the macroprogramming language or otherwise enhance the power and processing speed of the signal system for a particular application.

Chapter 1 gives an introduction to the MP-32A processor. Chapter 2 describes in more detail the internal structure, programmable components, and micro-operations of the MP-32A.

Chapters 3, 4 and 5 together with the appendix, describe the instruction set. The macro-language structure, and the conventions which must be followed in preparing new macro instructions, are covered in Chapter 6. Chapter 7 describes the use of the control panel facilities for checkout of new primitive micro-programs.

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California

Publication No. 29107
© 1973
by Culler/Harrison, Inc.

Rev. C

Chapter 1. INTRODUCTION

The MP-32A is a powerful processing unit which serves as the central processor and control unit for the CHI Signal System. It obtains very high effective execution speed by a combination of fast execution of individual operations and a parallel operations structure permitting the specification of four or more separate operations in a single micro-instruction. A hierarchy of storage media under explicit control of the micro-programmer, together with an interleaved main memory, permit instruction and data fetches to keep up with the operation execution.

This manual contains the information necessary to write micro-instruction sequences and incorporate these sequences as new macro instructions in the Signal system macro language.

For information about the macro language itself, see TMA3, the MP-32 Macro Programming Manual. A more detailed and technical description of the MP-32A hardware may be found in TMB2, the MP-32A Macroprocessor Reference Manual. It is recommended that the user be familiar with the Macro Programming Manual before reading this manual.

Chapter 2. INTERNAL STRUCTURE

2.1 SCOPE

This chapter describes the internal structure of the Macroprocessor with respect to those components which appear as operands in the instruction set. These consist of three types:

a. Memory
b. Arithmetic
c. Registers

The various programmable components and their interconnections are shown in Figure 2.1. A description of the I/O components is given in 2.3.

2.2.1 Memory

Three types of memory unit are used to provide a heirarchy of data and instruction storage media. These memories are:

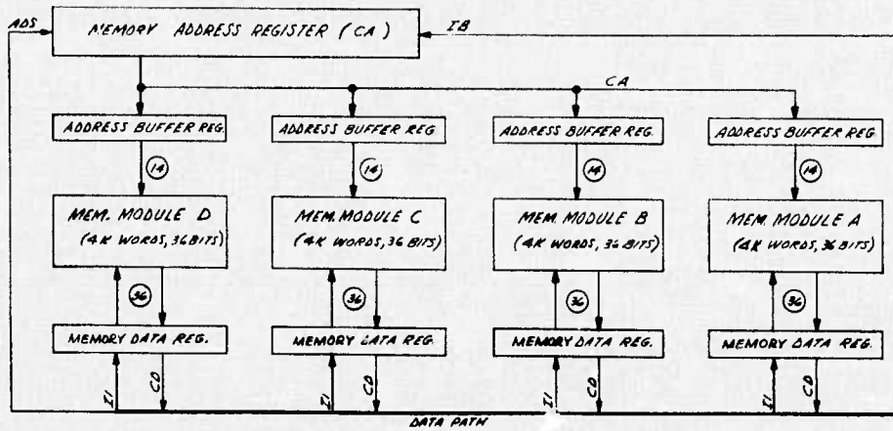
a. Instruction Pad
b. Data Pad
c. Main Memory (MOS)

2.2.1.1 Instruction Pad. This memory contains 64 28-bit words with a cycle time of 35 nanoseconds. It is used to hold a set of instructions. Any word in instruction pad may be specified as the successor to any other instruction, regardless of that instruction's location. Sequential execution is not necessary in instruction pad. Instruction execution from instruction pad makes all MOS memory cycles available for data accesses. In addition, all conditional branch instructions must have an instruction in instruction pad as their successor if the branch is taken. Instruction pad may not be examined with another instruction, it also cannot, with one exception, be altered. Two special load macro pad instructions are provided to transfer instructions from either MOS memory or data pad to instruction pad.

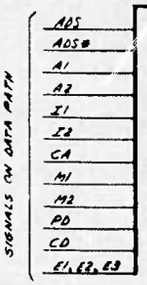
2.2.1.2 Data Pad. This memory also has a cycle time of 35 nanoseconds and contains 64 16-bit words. It is used for data or inactive instruction storage. It provides a fast, random-access scratchpad memory for the arithmetic unit. Blocks of data pad words may be read from or written into MOS memory with a single instruction.

2.2.1.3 Main Memory (MOS). The main memory is MOS with an access and cycle time of 667 nanoseconds. This memory consists of a minimum of 16K 36-bit words to a maximum of 32K 36-bit words. It is addressed in units of 18-bit words. MOS memory may be used for either instructions or data. Instructions may be executed sequentially out of MOS as long as they do not need to access main memory for data. MOS memory is interleaved four ways and may be operated in sequential mode for fetching where sequential words are accessed in anticipation of need, to permit an effective access time of 167 ns. Thus, it is possible to load all of data pad from MOS in 6 μs, to load all of instruction pad in 11.3μs, or execute sequentially from MOS at one instruction per clock time.

MEMORY SYSTEM D-21001-100-2

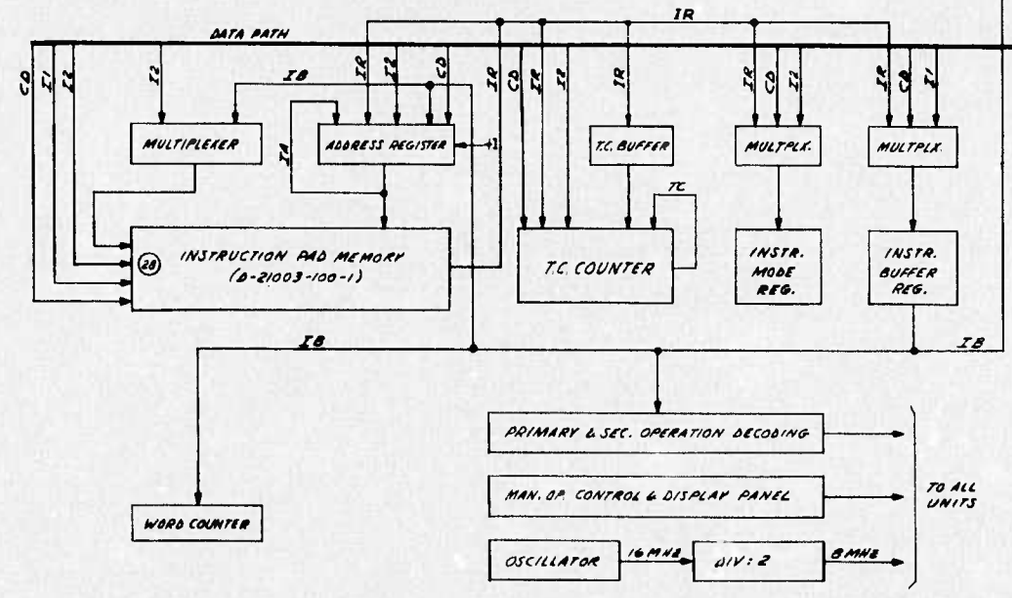


ARITHMETIC UNIT

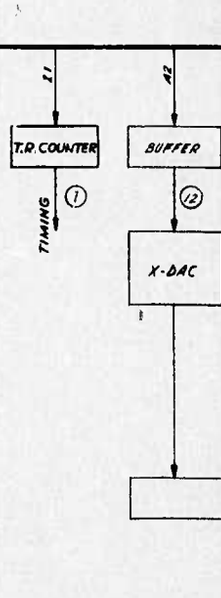


IB

TIMING & CONTROL D-21001-100-3

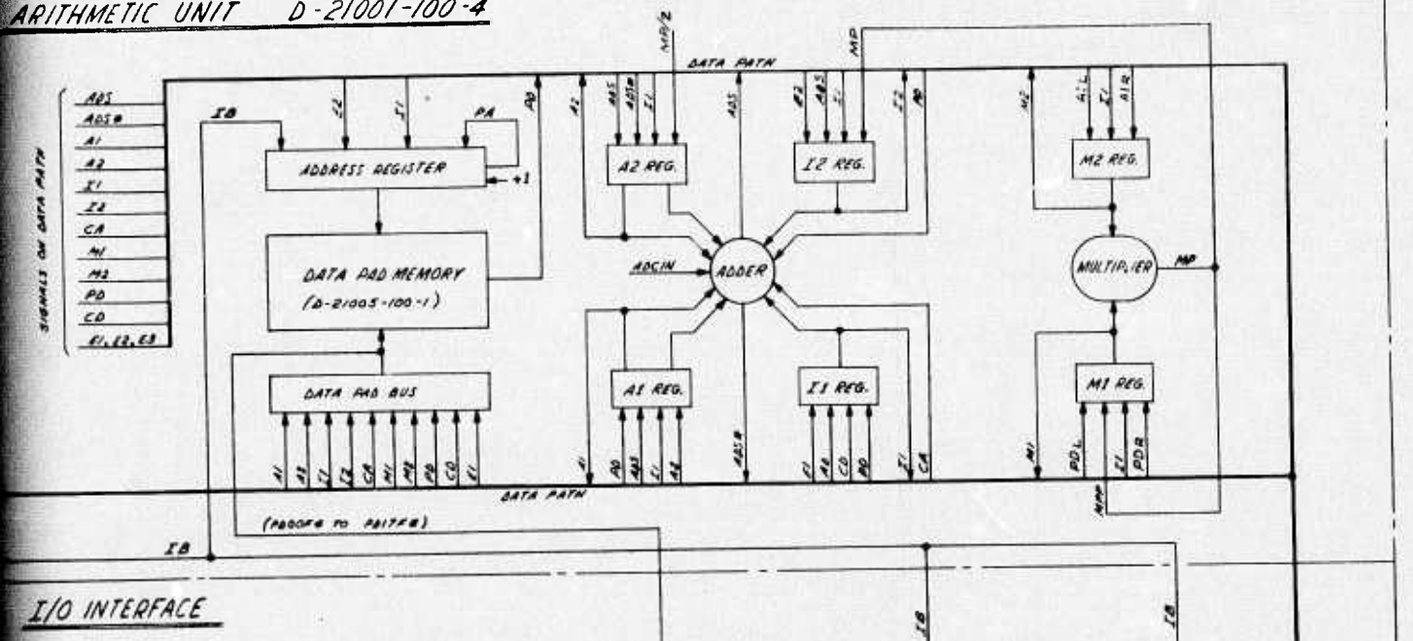


I/O INTERFACE

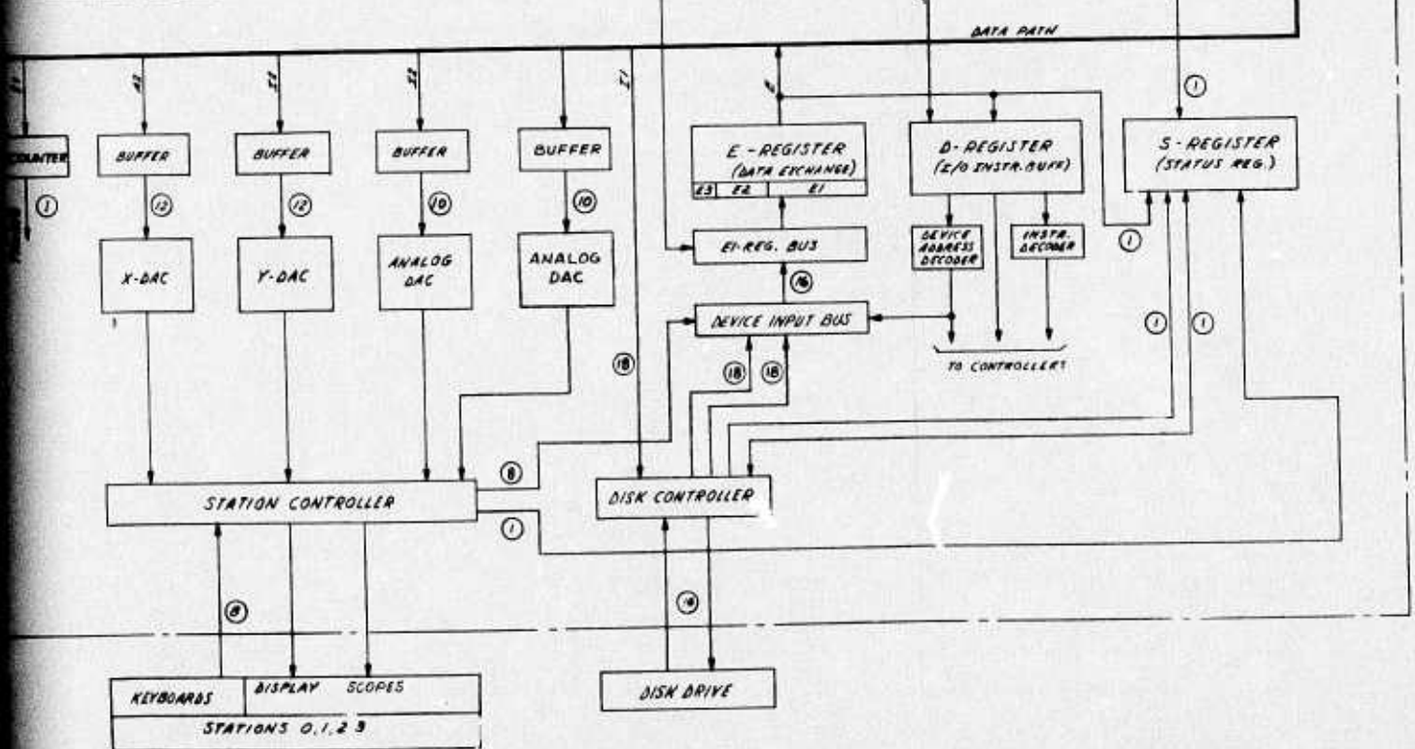


DATE	BY	REVISION RECORD	AUTH.	DR.	CK.

ARITHMETIC UNIT D-21001-100-4



I/O INTERFACE



TOLERANCES UNLESS OTHERWISE SPECIFIED	CHI CULLER-HARRISON INC.		
DECIMAL ±	MP32A	SCALE	DRAWN BY Cold.
FRACTIONAL ±	TITLE MACRO-PROCESSOR BLOCK DIAG.		
ANGULAR ±	DATE 5/30/73	DRAWING NUMBER D-21001-100-1	

Memory access is explicitly controlled by the micro level programmer. A MOS memory cycle is initiated whenever the output of the adder is directed to CA, or a MODE 3 instruction with T(10)=1 is executed. If the memory module which is to be accessed is busy with a previous request, an automatic pause in instruction execution by the Macroprocessor occurs until it is available. If the write flip/flop is set, either simultaneously with the transfer into CA or prior to it, the memory cycle will be a write of the contents of I1 register and flags (F(0) and F(10)). In this case, I1 register and flags are copied into CD on the next clock after CA is loaded and the write flip/flop is cleared; thus, I1 may be loaded with new values immediately.

For memory read cycles the MOS memory may be operated in two modes. In the standard random access mode, when CA is loaded a read cycle is initiated. After 667 nanoseconds, CD will be valid and may be transferred to I1 register and its flags; an attempt to transfer CD into I1 before CD is valid will again cause an automatic pause condition. In sequential mode, memory cycles are initiated in advance for words in all memory modules, and CA is allowed to run six words (3 doublewords) ahead of the word being accessed. Once sequential mode is initiated, memory words can be transferred at a rate of one doubleword in 1 cycle. This maximum rate is directly useable only in doubleword data transfers including MOS to instruction pad, MOS to data pad, and MOS to instruction buffer. Even for single word transfers, however, this mode allows four times faster accessing than would otherwise be possible. Memory sequential mode is entered whenever CA is loaded in an instruction with T(13)=1. This applies to MODE 3as well as the adder transfers. At this time, a pause occurs until all memory modules are not busy. Then, on the next three clocks, after initiating the first memory module addressed, CA is incremented by 2 and a new memory fetch is initiated. This incrementing does not use the adder and goes on in parallel with normal instruction execution. At the end of this start up process, all four memory modules are busy fetching or have data ready. A secondary address register, called CR, keeps track of which memory module is currently being accessed. A request for transfer of CD into I1 (or a load pad type instruction) will fetch from the module indicated by CR. Each time CA is incremented (by either 1 or 2), CR is also incremented. Memory sequential mode is terminated by a transfer of the adder into CA (other than CA+1 or CA+2) or by a load pad type instruction (OP CODE 14) with T(13)=0. As long as sequential mode is in effect, MOS memory is not available to DMA channels. Sequential mode can be used for data transfers to an attached AP-90 array processor, however, memory control for such transfers is provided by the MP-32A.

Sequential mode is used only for memory fetch requests. Writing into memory is automatically interleaved for block STORE operations (OP CODE 14).

2.2.2 Arithmetic

The arithmetic components of the Macroprocessor are the adder and the multiplier.

a. Adder

The adder is a 16-bit, twos complement unit capable of performing logical functions and tests as well as addition or subtraction in one clock time. There are three possible register inputs for each adder operand plus 0

and all ones (-1). The output can be directed to one or more of five possible registers and the complement output to one register. An adder operation is completed in one clock time. Table 2.1 summarizes the adder inputs, operations and destinations. The possible micro operations using the adder are given in Appendix A, Tables 6 and 7.

Table 2.1 Adder Summary

A Inputs	B-Inputs	Destinations	Operations
A1	A2	A1	A+B
I1	I2	A2	A+B+1
$\overline{CA}$	$\overline{PD}$	I1	A∧B(logical)
$\overline{A1}$	$\overline{A2}$	I2	A=B(logical)
0	0	$\overline{CA}$	
-1	-1	$\overline{A2}$	

b. Multiplier

The multiplier is an 8x8 bit unit producing a full 16-bit product in three clock times. Unlike the adder, it operates on sign-magnitude quantities, and operates continuously, transforming the quantities in its input registers, M1 and M2 to produce a product in MP. The multiplier output, MP, can be used in three ways:

i) Directly, as a 16-bit magnitude

ii) Shifted right one place, for conversion to 2's complement or 16-bit sign magnitude

iii) As an 8-bit plus sign quantity, for input to a second multiplication.

2.2.3 Registers

The Macroprocessor has six registers which are used directly in conjunction with the adder for most arithmetic and logic operations. There are also several auxilliary registers, an address register, instruction buffer registers and multiplier input and output registers.

2.2.3.1 Registers Used with the Adder. The six registers which can be used as inputs to the adder are A1, A2, I1, I2, PD and CA, their capabilities are:

a. A1 Register - 16 bits

Input to: A side of Adder (also complemented)
M2 register
Data Pad

Loaded from: Adder
I1 and A2 registers
Data Pad

Shifts: Left
May be linked with I1 register for double length shifts

This register, together with A2, has maximum arithmetic usage. It may be used for adder operations with A2, I2 and PD registers.

b. A2 Register - 16 bits

Input to: Y side of Adder (also complemented)
Data Pad
X DAC
A1 register

Loaded from: Adder
Adder complemented
I1 register
Multiplier Product right shifted

Shifts: Right, arithmetic or logical
May be linked with ~~I2~~ register for double length shifts.

This register, together with A1, has maximum arithmetic usage. It may be used for adder operations with A1, I1 and CA registers.

c. I1 Register - 16 bits

Input to: X side of adder
A1, A2 and I2 registers
M1 and M2 registers
Instruction buffer register
MOS memory
I/O controllers
Data Pad address register (PA)
Data Pad

Loaded from: Adder
MOS memory (CD)
Data Pad
E1 register

Shifts: Left
May be linked with A1 register for double length shifts
May be linked with flag register for double length rotation

This register is the main connection between the arithmetic and logic section of the Macroprocessor and the rest of the computer system. All MOS memory accesses by the macroprocessor use the I1 register, as do most input/output activities.

d. I2 Register - 16 bits

Input to:
- Y side of adder
- Data Pad
- Y DAC
- Analog DAC
- E2 register

Loaded from:
- Adder
- Multiplier Product
- I1 register
- E2 register
- E3 register (panel load only)

Shifts:
- Right, arithmetic or logical
- May be linked with A2 for double length shifts

e. PD - Data Pad Output - 16 bits

PD is not truely a register, but represents the contents of the data pad cell whose address was in the data pad address register (PA) at the beginning of the current clock time. When used as a destination, PD is the input bus to data pad and is written at the address in PA at the beginning of the last clock time of an instruction. If PD is specified as both a source and destination in the same instruction, the source data will not be received correctly at the end of the instruction.

Input to:
- Y side of adder
- I1 and A1 registers
- M1 register
- E1 register

Loaded from:
- A1, A2, I1, I2, CA registers
- E1, E2, M1, M2, S registers
- MOS memory

f. CA - Memory Address Register - 16 bits

This register is primarily used to hold the address of the MOS memory word being read or written. Whenever it is the adder destination, a memory cycle is initiated one clock after the end of the instruction. Furthermore, the actual transfer into CA occurs only on the last clock of an instruction. However, it is possible to load CA directly from the instruction without initiating a memory cycle, and CA may be used as an adder input at any time. CA may be incremented by two without using the adder.

Input to:
- X side of adder
- Data pad

Loaded from:
- Adder (causes memory initiate)
- Instruction Buffer

2.2.3.2 Other Arithmetic Registers. Two additional registers, M1 and M2, hold the input data for the multiplier. The flag register is a collection of bits which are used in some arithmetic operations. Other condition registers are OF, CF and MS.

a. M1 register - 8 bits

Input to: Multiplier (continuously)
Data Pad

Loaded from: Data Pad (left or right half)
Multiply product (left half)
I1 register (right half)

b. M2 register - 8 bits

Input to: Multiplier (continuously)
Data Pad

Loaded from: A1 register (left or right half)
E1 register (right half reverse ordered)
I1 register (right half)

The product of the 8-bit magnitudes in M1 and M2 is available as a 16-bit product in 500 nsecs.

c. Flag register - 16 bits

Each of the eight registers, A1, A2, I1, I2, M1, M2, PD, HALT has 2 flag bits called F1 and F2 associated with it. These bits do not participate in parallel transfers from register to register, nor are they inputs to the adder or multiplier. They are used in the following ways:

i) A1F1 and A2F1 hold the sign of registers A1 and A2 after a convert to sign-magnitude operation. They determine the sign during a convert to two's complement operation.

ii) M1F1 and M2F1 are exclusive-OR'ed to produce MS - the multiplier product sign.

iii) I1F1 and I1F2 holds bits 16 and 17 of a word read from memory after memory data (CD) is transferred to I1 register. They provide the values for these memory bits during memory write operations.

iv) Any flag register bit can be set, reset and tested programmatically.

v) The flag register can be interchanged with I1 register sequentially by a circular shift operation.

d. One-bit registers ØF, CF and MS

CF is set by the carry out of the most significant bit of the adder during any adder operation except tests. It may be added to the sum of A1 and A2 during a special adder operation for multiple precision arithmetic.

ØF is set when the adder output exceeds 16-bits.

MS is the exclusive-OR of M1F1 and M2F1, and is the sign of a 16 or 15 bit product.

CF, ØF and MS may all be tested programmatically.

2.2.3.3 Auxilliary and Input/Output Registers.

a. E1 - register - 16 bits

Input to:
- I1 Register
- M2 Register (right half, bits are reverse ordered)
- Data Pad

Loaded from:
- Input Data Bus (DI)
- Data Pad Input Bus (this path permits transfer of data destined for data pad to E1 instead)
- Control Panel Switches

E1 is used primarily to hold data being input to the processor. It may also be used for temporary data storage and the left and right halves may be interchanged.

b. E2 register - 8 bits

Input to:
- I2 register
- PA register
- Data Pad (right halfword)

Loaded from:
- I2 register
- PA register
- Instruction buffer
- Control Panel Switches

E2 is used for temporary saving the value of PA and introducing halfword constants from an instruction.

c. E3 register - 4 bits

This register contains four bits which may be individually set from the control panel switches even when the processor is running. They are individually testable programmatically, but cannot be altered. They also serve as input for the left four bits of the instruction pad and the two flag bits of each register and MOS memory during manual control panel load operations.

d. Data Input Bus (DI) - 16 bits

Input to: E1 register

Loaded from: Input device controller which is addressed by bits 3-7 of D register.

This bus is used for transfer of data from external device controllers to the processor. If no device is selected (D(7)=0) it may be used as a source of 0. Two clock times are required to transfer DI to E1.

e. D register - 8 bits

Loaded from: Instruction Buffer

This register is used to select one device controller to have access to the DI bus and I/Ø data ready bit. The D register contains three fields; a one-bit device communication bit which is 1 when the remaining bits are valid, a four-bit device address, and a three bit command which is interpreted by the addressed controller. Device addresses and commands for each device are given in Appendix A, tables 11-14.

f. S register - 8 bits

This register contains eight service request (interrupt) bits. Each bit is associated with a particular input/output device controller. That controller may set its bit. The bits may be programmatically cleared, set and tested. The entire register can be tested for all 0 or not all 0 in a single clock time and a sequential serial test of all bits in S is also possible. S-bit assignments are given in Appendix A, table 11.

Input to: Data Pad (right halfword)

Loaded from: E1 Register (right halfword)
Device Controller (single bit set)
Instruction Mode (single bit)

2.2.3.4 Data Pad Address Register (PA) - 6 bits. This register holds the address of the data pad cell which is currently accessible. Whenever PA is changed, data may be written or read from the new cell at the next clock time. References to data pad during the same clock time as PA is changed will access the old current pad cell. PA may be incremented or decremented by 1 or 8 without using the adder.

Input to: E2 register bits 0-5

Loaded from: E2 register bits 0-5
I1 register bits 0-5
Instruction buffer bits 0-5

2.2.3.5 Instruction Registers.

a. Instruction Buffer (IB) - 16 bits

This register holds the right 16 bits of the current instruction, including the OP, D, C, B and A fields.

Input to:
- Pad Address Register (PA)
- Instruction Address Register (IA)
- E2 Register
- Instruction Register Bits 20-25
- Word Counter (WC)

Loaded from:
- Instruction Register Bits 0-17
- I1 register
- Memory data register (CD) Bits 0-17

b. Instruction Address Register (IA) - 6 bits

This register holds the address of the current cell being used in Instruction Pad. It is normally the J field of the instruction being executed. It may be incremented as the result of test and link jump instructions.

Loaded from:
- Instruction Register Bits 20-25
- I2 register Bits 0-5
- Memory data register (CD) bits 20-25
- Instruction Buffer Register bits 0-5

c. Instruction Register (IR) - 28 bits

This is the data register for instruction pad. Its contents are the location addressed by IA.

Input to:
- Instruction Buffer Register
- Instruction Address Register
- Instruction Mode Register
- TC Counter
- TC Buffer Register

Loaded from:
- Memory data register (CD)
- I2 and I1 registers
- Instruction Buffer Register bits 0 - 5 → 20 - 25

2.3 INPUT/OUTPUT SYSTEM

The MP-32A supports input/output devices through controllers integrated into the processor cabinet. The data transfer path for all devices (except the AP-90) is a common data input (DI) bus for input and the I1 register (and in some cases A2 and I2) for output. A common data ready bit (I/O DRDY) is used to synchronize data transfer. A device is selected by placing its address in the D register along with a three-bit command. The device address and commands for each device are given in Appendix A, Table 11-15.

The command structure is extended for devices such as disk drives by using command two to indicate that the data in I1 is a 16-bit command. D(7), the left-most bit of the D register, is set to 1 when a command is given, and is cleared either at the end of the I/O instruction, or at a later point by either the addressed device, to indicate that it has accepted the command, or by the Macro-processor, when further communication with the device is not needed. As long as D(7) is set, the addressed device controller may set or clear I/ØDRDY and load data onto the DI Bus. An attached AP-90 array processor must be addressed with D(7) set in order for either data transfer to and from MOS or a SET APPSA command to function.

A device controller, whether addressed or not, may set its service request bit in the S register. For devices with addresses 10_8 or higher, this bit will be cleared automatically when the requesting devices status is input to the E1 register. Multiple data transfers involving different device controllers can be carried out simultaneously by using the S bit associated with each device to synchronize data requests. The requesting device can then be addressed and its data input or output, depending on the operation.

2.4 DIMENSION AND ADDRESS CONVENTION

Component dimensions are in decimal but component addresses are in octal with the least significant bit designated by 0. Bit number designators always follow the name of the register to which they apply. These designators are usually enclosed by parenthesis. The designation R following the name of a 16-bit register refers to bits 0-7 of that register (the right half); the designation L refers to bits 10-17 (the left half). For 36-bit components R refers to bits 0-17 and their two flags; L refers to bits 20-37 and their flags.

Chapter 3. INSTRUCTIONS

3.1 FORMAT

The 28 bits used to specify an instruction are grouped into eight FIELDS as shown below:

33 32 31 30	27 26	25 24 23 22 21 20	17 16 15 14	13 12 11	10 7 6	5 4 3	2 1 0
13 12 11 10	7 6	5 4 3 2 1 0	17 16 15 14	13 12 11	10 7 6	5 4 3	2 1 0
T FIELD	MODE	J FIELD	OP CODE	D FIELD	C FIELD	B FIELD	A FIELD

The upper set of numbers describes the logical octal numbering of the positions. The lower set of numbers represents the numbering as they appear on the control panel. In particular, the right hand side (numbers 0 through 17) represent operational control whereas the left hand side (numbers 0 through 13) represent program control.

3.2 FIELD DEFINITION

The definition and general use of the eight fields is as follows:

a. The T-FIELD is generally a repeat number and is decremented at each clocktime (167 ns) during execution until it is zero. In general, the instruction is performed one more time than shown in the Repeat Number. See section 4.3 for a explanation of instruction execution.

There is a 4-bit repeat number for OP CODES 15-17 and a 3-bit repeat number for all other OP CODES. In the latter instructions the most significant bit of the T field, T(13), is used to control the entering or terminating of sequential mode in certain memory instructions. In the mode 3 instruction, the T field is not used as a repeat number but instead has the following meaning:

T(10)=1 Initiate memory
T(11)=1 Set memory write control
T(12)=1 Enter memory execute mode
=0 Terminate memory execute mode
T(13)=1 Enter sequential mode
=0 Terminate sequential mode

b. MODE is the instruction mode and may specify the overall meaning of the fields OP-CODE, D-FIELD, C-FIELD, B-FIELD, A-FIELD. There are four possible instruction modes:

i. MODE 3 is used to set the memory address register (CA). Bits 0-17 of the instruction give the desired CA value.

ii. MODE 2 instructions generally set the data pad address register.

iii. MODES 0 and 1 provide basic operations as defined for the particular operation type specified in the OP CODE field.

c. J-FIELD is the instruction address of normal successor instruction.

d. OP-CODE is the operation type. Together with the instruction MODE, this field specifies the meaning of the D, C, B and A fields.

e. D,C,B,A are parallel operations. Each of these three-bit fields may permit the selection of one out of eight possible operations as defined by MODE and OP CODE.

3.3 INSTRUCTION EXECUTION

Generally, one instruction is executed each clock time when the Macroprocessor is in run mode. During the clock time, all micro-operations specified in the instruction are executed in parallel. Each operates on the information present at the beginning of the clock period and modifies its specified destinations at the end of the clock period. Thus, one micro-operation may be modifying the contents of a register at the same time that another micro-operation in the same instruction is using the old contents of that register. However, any combination of two micro-operations which both modify the same register in the same instruction is not permitted. If a non-zero T field causes an instruction to be executed repeatedly, any changes made during one clock period will be reflected in the input data for the next clock period. There are a few exceptions to this repeated parallel execution procedure:

a. CA is loaded only at the end of the last clock period in an instruction.

b. PD is loaded only during the last clock period in an instruction, and may not be relied on as a source when it is being loaded.

c. All tests are performed only during the last clock period. If the test is passed, an additional clock time is required before the next instruction is executed, but no parallel micro-operations are executed then.

To illustrate the effects of this parallel execution with varying repeat numbers, consider the following examples:

Example 1: MOVE I1:A2 INC I1 MOVE A2:A1

During each clock period that this instruction is executed, 1 is added to the contents of I1. At the same time, the old contents of I1 are transferred to A2 and the old contents of A2 are transferred to A1. If the initial values of I1, A2 and A1 are 0, 100 and -5 respectively, the effects of different T fields on the final results would be:

T = 0 I1 = 1, A2 = 0, A1 = 100
T = 1 I1 = 2, A2 = 1, A1 = 0
T = 2 I1 = 3, A2 = 2, A1 = 1
T = 7 I1 = 8, A2 = 7, A1 = 6

Example 2: INC PA TEST PD = A1 J = 10

In this instruction, the data pad location referenced by PD is determined by the contents of PA at the beginning of each clock period. Hence, if PA is initially 0, data pad cell 0 would be compared with A1 if T = 0, data pad cell 1 would be compared with A1 if T = 1, etc. The final value of PA would be one greater than the address of the cell tested. If the cell tested did equal A1, then the instruction at IA 11 would be executed next, otherwise, the instruction at IA 10 would be next.

Example 3: INC CA MOVE CA:PD DEC PA

Since CA is only loaded at the end of the final clock period, the value of CA moved into data pad would be its value at the beginning of the instruction, no matter what the T field was. Only one data pad cell would be loaded, the cell whose address was in PA at the beginning of the final clock period. The final PA value would be one less than the address of the cell loaded.

3.4 EXECUTION OF INSTRUCTIONS DIRECTLY FROM MOS MEMORY

As mentioned earlier, instructions may be executed directly from MOS memory, without needing to copy them into instruction pad. MOS memory execution mode is entered by executing a MODE 3 instruction with T(13)=1 or by executing certain OP CODE 14 instruction types. Once in memory execution mode, instructions are automatically fetched directly from the memory data register for execution. The memory address register (CA) is incremented automatically by 2 as this fetch is made in preparation for the next instruction. The IA register (and the J field) are not used in this mode for instruction fetching; CA is used instead. Any instruction may be executed in memory execution mode; however, if it loads CA, the successor instruction will be taken from the new memory location referenced by CA. References to data in memory are, therefore, limited to locations intermixed with the instructions such as the following:

```
Instruction 1:  MOVE CD:I1  INC CA by 2
Data word 1:     357
Data word 2:  (ignored)
Instruction 2:  LSL I1(7).
```

Instructions in memory must be located on even address boundaries.

Once memory execution mode is entered, instructions will be fetched directly from memory until one of the following events occurs:

a. The instruction being executed has a non-zero J-field and is not an OP CODE 17 type.

b. A MODE 3 instruction is executed with T(13)=0.

c. An OPCODE 14, load macro block instruction is executed with MODE not 1.

d. A test micro-operation in the instruction being executed is passed.

e. An OPCODE 17, MODE 1 instruction is executed.

3.5 INSTRUCTION SUBSETS

The Macroprocessor instruction set may be classified into nine instruction subsets. Each such subset corresponds to given values of the MODE and OP-CODE and includes instructions making generally the same use of the remaining fields. A list of the instruction subsets as well as the type of operations permitted in instructions in each subset are given in Table 3.1. Appendix A contains a detailed breakdown of the micro-operations available in each subset and the octal field values used to encode them. A dictionary giving the definition of the micro-operation notation used in Appendix A will be found in Appendix B.

Table 3.1 Instruction Subsets....Page 1 of 3

Instr. Subset Name	T	Mode	Op Code$_8$	Allowable Operations/Single Instruction
Single Bit Decode	0-7	0	0	Register Operations and Special Functions and Special Multiply
Adder Succinct Adder	0-7	0	1-5	1 of 7 Adder Operations and Load of 3 registers, or special functions in lieu of loading registers.
	0-7	1	1-5	All mode-0 operations with adder output to memory address register (CA)
	0-7	2	1-4	1 of 7 adder operations, and load of 1 register, and set data pad address register (PA)
Adder Full Adder	0-7	0	6-13	1 of 63 adder operations, and load of 2 registers or special functions in lieu of loading registers
	0-7	1	6-13	All mode-0 operations with adder output to memory address register (CA)
	0-7	2 2	6 7,10,11,12	1 of 63 adder operations and set data pad address register (PA) 1 of 50 adder operations with the adder destination selected by op code and set pad address register (PA)
	0-7	2	13	1 of 50 adder operations, adder outputs to memory address register and initiate memory write cycle.

Table 3.1 Instruction Subsets....Page 2 of 3

Instr. Subset Name	T	Mode	Op Code$_8$	Allowable Operations/Single Instruction
Input/Output and Control	Variable	0	14	a) Block data transfer between memories b) Indirect execute or enter memory execute mode c) Set E2 register value d) Input/Output control - pulsed e) Generate delays.
		1	14	Same as mode 0 for b,c,e Remain in memory execute mode for a Continuing Input/Output Control.
Bit Select Conditionals	0	0	15	Test a "0" in single bit of Flag Register, or A2 Register, or I1 Register or E1 Register. Condition met: NIA=Jump Address=(B&A) Fields. Condition not met: NIA address in normal sequence (J-Field)
	0	1	15	Same as mode-0 except a test for "1" is performed. Condition not met: NIA (J-Field) Condition Met: NIA=Jump Address =(B&A Fields)
	Variable	2	15	Sequential test for a "1" in each bit in Flag Register, or A2 Register, or I1 Register, or E1 Register, Condition met: NIA=Jump Address=(B&A) Fields. Condition not met: NIA=next in sequence (J-Field)

Table 3.1 Instruction Subsets....Page 3 of 3

Instr. Subset Name	T	Mode	Op Code$_8$	Allowable Operations/Single Instruction
Logical Operations	Variable	0	16	Logical Operations: (1) Single condition tests resulting in program jumps or a continuation of normal instruction sequence. (2) Logical actions (sets, clears, shifts, transfer & swaps). (3) Single condition tests in combination with logical actions.
	Variable	1	16	Same as mode-0, except in general under mode 1, tests for equality become tests for inequality and instructions which set a bit=0, set it=1.
	Variable	2	16	Sequential test for a "1" in each bit of the S Register. Condition met: NIA = Jump Address (B&A Fields). Condition not met: NIA (J-Field)
Memory Controls		3	NA	Load Memory Address Register with the contents of the Instruction Buffer Register
Linkage Operations	1	0	17	Writes return address in the J Field of the instruction at location J specified by the D & C fields. Executes J+1 as next instruction.
	1	1	17	Same as mode 0 except that the next instruction to be executed is defined by the Jump Address (B&A Fields)
	1	2	17	Same as mode 0 except that in addition, PA is set = B&A Fields
Set AP PSA	1	2	5	Set array processor PSA to contents of IB(0-13).

Chapter 4. FIELD DEFINITION BY MODE NUMBER

4.1 INTRODUCTION

To adequately illustrate the rich variety of operations and combinations of operations available within the instruction set, the instructions are first grouped according to MODE; second (within a MODE) according to OP CODE; third (within an OP CODE) according to DCBA-FIELD definitions; and last according to the particular transformation resulting from a given numerical value within a FIELD.

4.2 MODE 0

Table 4.1 gives the field definitions for each OP CODE when operating in MODE 0. Table 3.1 gave the general type of operations which can be performed for specific OP CODES and the number of clocks necessary to complete the operations.

4.3 MODE 1

The MODE 1 instructions are the same as MODE 0 but with the following modifications:

OP CODE	Modification
1,2...13	Transfers the adder output to CA
14	Several exceptions--see Table 3-1
15	Reverses the consequences of MODE 0
16	Reverses the consequences for several instructions-see Table 3-1
17	Same as MODE 0 except that the immediate successor is defined by the JUMP ADDRESS

4.4 MODE 2

Table 4.2 gives field definitions for each OP CODE when in MODE 2. In this mode the D and C fields generally specify the inputs to the adder and the OP CODE specifies the destination of the output. The B and A fields contain the address of data pad (PA).

OP CODE 0 has no MODE 2. OP CODE 15 is a scan test of the register specified by the two most significant bits of the D field. OP CODE 16, D=7, C=5 is a scan test of the service request register, S. The scan test allows sequential testing of all bits in a register, starting at the highest order bit (bit 7), and exiting upon finding a 1. OP CODE 17 is the same as in MODE 0 except that PA is set in the B and A fields. OP CODE 5 is used to set an associated AP-120 array processor's PSA.

4.5 MODE 3

There is only one operation in MODE 3: SET CA equal to the value of the octal number in the OP,D,C,B,A fields. The T field is used for memory controls.

Table 4.1 MODE 0 Field Definitions

OP CODE	D FIELD	C FIELD	B FIELD	A FIELD
0	SPECIAL PARALLEL INSTRUCTIONS			
1	ADDER ROW (C FIELD=0 in ADDER OPERATIONS TABLE)	E1, E2	PA	I2
2		E1, E2	A2	PD
3		M1	M2	A1
4		M1	M2	I2
5		E1, E2	I1	PD
6	ADDER ROW	ADDER COLUMN	PA	PD
7			PA	A1
10			A2	A1
11			I1	A1
12			A2	I2
13			I1	I2
14	LOAD TYPE		# WORDS - 1	
15	BIT TEST		JUMP ADDRESS	
16	LOGIC TYPE		SPECIFIER OR JUMP ADDRESS	
17	RETURN ADDRESS		NOT USED	

Table 4.2 MODE 2 Field Definitions

OP CODE	D FIELD	C FIELD	B FIELD A FIELD
1	ADDER ROW (C FIELD=0 in ADDER OPERATIONS TABLE)	E	SET PA
2		PD	
3		PDI→E1	
4		M1	
5	AP PROGRAM SOURCE ADDRESS		
6	ADDER		
7	ADDER → A1		
10	ADDER → A2		
11	ADDER → I1		
12	ADDER → I2		
13	ADDER→CA, SET WRT		
14	LOAD TYPE		# WORDS-1
14	EXECUTE		SET PA
15	REG.	NO USED	JUMP ADDRESS
16	7	5	JUMP ADDRESS
17	RETURN ADDRESS		SET PA

Chapter 5. TIMING CONSIDERATIONS

5.1 CYCLE TIMES

a. MOS = 667 ns but interleaved four way for 167 ns sequential access time

b. Processor - 167 ns.

5.2 EXECUTION TIMES

a. Tests: If BA→IA, add 167 ns to execution time
If J+1→IA, add 167 ns to execution time

b. Multiply: Wait 500 ns between →M1→M2 and MP→

c. Wait 167 ns extra, when DI→EI

5.3 OCCURRENCES AT TC = 0 ONLY

a. All tests

b. →PD, if OP CODE ≠ 0

c. →CA

d. Link Jump: Return address is written

e. Set and clear of S bits

5.4 OPERATIONS WITH MOS

a. →CA, MODE 3 (regardless of T-field, MODE 3 execution time = 167 ns)

1. T10=1, Memory initiated,
2. T11=1, set WRT (single)
3. T12=1, enter memory execute mode
4. T13=1, set sequential mode

b. Storing into MOS

1. If you wish to store into MOS, SET WRT before or simultaneously with →CA.

c. In the halted state; any →CA, clears WRT.

Chapter 6. CONVENTIONS FOR MACRO INSTRUCTION DEFINITIONS

6.1 SCOPE

This section contains the information necessary to prepare primitive micro-instruction programs for use as macro instructions in the Signal Macro Language. A description of the standard macro instructions and the macro language assembler is given in TMA3, MP-32 MACRO-PROGRAMMING MANUAL.

6.2 DEFINITION

A sequence of micro instructions which performs a well defined operation is called a primitive. A primitive may be used as a macro-operation provided it conforms to the conventions of the macro language for operand fetching, use of mnemonics and termination. Primitives also may be used as subroutines, using the link jump micro instruction, and sequences of primtives are used in the signal system as facility controllers.

6.3 MACRO OPERATION SEQUENCING

The basic facility used to provide for sequencing from one macro instruction to the next, and to allow recognition of service requests to interrupt normal sequencing, is the operations sequencer. The operations sequencer is a small primitive which is normally resident in instruction pad cells 0 to 6. It makes use of data pad cell 76 as an instruction counter to locate the next macro instruction for execution. The operations sequencer fetches one word from MOS memory at the address given in cell 76. Cell 76 is then incremented by 1. The word fetched is the MOS address of the primitive which defines the macro instruction to be executed. This address is loaded into CA, sequential memory fetch mode is initiated, and memory execute mode is entered. The primitive may fetch its arguments by using the address in cell 76, incrementing it by 1 for each word fetched. Standard primitive subroutines may be used for this argument fetching. When the primitive is finished, it terminates by returning to instruction pad cell 3, which is part of the operations sequencer. The operations sequencer then tests the S register for any service requests. If the S register is not zero, the interrupt decoding routine is executed to provide interrupt service. If the S register is zero, the operation sequencer fetches the next macro instruction from MOS at the address given in cell 76 and the process is repeated.

At this point, it should be helpful to examine the operations sequencer primitive, the code is given in figure 6.1. Beginning with instruction address 5, the contents of cell 76 are incremented and transferred to the I2 register. The I2 register is then transferred back to cell 76; at the same time the decremented contents of I2 are loaded into CA, initiating a read cycle. In instruction 0, executed next, the memory data word is transferred through the I1 register into CA, and memory sequential mode is initiated. In instruction 2, execution begins in the primitive directly from MOS memory. If the primitive is not to execute entirely out of MOS, it will have a load macro pad instruction, which when exected will transfer one or more instructions from MOS into instruction pad, then either continue executing from MOS or execute the last instruction loaded, depending on the MODE of the load macro instruction. When the primitive is complete it transfers to instruction address 3. In this instruction, a test for

S≠0 is made. If this test is passed, the instruction at IA 4 is executed setting CA to the address of the interrupt decode routine and initiating sequential memory access mode and memory execute mode. If S=0, we are back to IA 5, ready to fetch the next macro instruction. Note that the instruction at IA 1 has not been used. Its function is to provide a common location where primitive subroutines may exit. Its J field is altered by a link jump instruction when the primitive subroutine is called.

Figure 6.1 Operations Sequencer

IA	T	M	J	OP	DCBA	FUNCTION
0	11	0	2	11	6230	MOVE CD:I1:CA SEQ=1
1	0	0	1	0	0000	NØ-ØP
2	10	0	0	14	7400	ENTER MEMORY EXECUTE MODE
3	0	1	5	16	1004	IF S≠0 GOTO 4 OTW 5
4	15	3	0		INTDC	EXEC SEQ (INTDC)
5	1	2	6	12	3376	PA=76 INC PD:I2
6	0	1	0	6	2404	MOVE I2:PD DEC I2:CA GOTO 0

6.4 INSTRUCTION PAD USAGE

Except for the words occupied by the operations sequencer, all of instruction pad is available for use by primitives. Additional words of instruction pad are often used to hold resident primitives which are part of an execution environment called a mode. A mode permits more efficient processing by keeping resident commonly used primitives. It also may alter the normal operations sequencing by interpreting macro instruction op codes as other than MOS memory addresses. For example, INTERNAL mode (described in TM#1) uses one of the flag bits in the Op-code word of each macro instruction it interprets to cause the use of the left half of the op code as an instruction pad address of a resident primitive routine. Such resident mode defining primitives use up to 50_8 additional words of instruction pad, leaving words 57_8 - 77_8 of instruction pad for general usage by primitives which co-exist with them. Most standard system macro instruction definition primitives meet this restriction.

There are several techniques which may be used when a primitive must use more instruction pad than is available. The most common technique is to partition the primitive into sub-primitives, each of which ends with a EXEC SEQ operation to initiate the next sub-primitive from MOS memory. Alternatively, a subroutine organization may be used to permit saving of instruction pad words. In the event that more instruction pad must be used than is available under the constrictions of the current mode, the primitive may make use of any or all of instruction pad by restoring it when it finishes. To do this, each mode defining primitive must load data pad cell 75 with the address of a primitive which will restore those words of instruction pad it uses. To restore these words, a primitive needs only to terminate by loading CA from PD(75), entering memory sequential access mode, and transferring to IA 2 to enter memory execute mode, this may be done with one instruction (PA=75 MOVE PD:CA SEQ=1 T=1). If instruction pad cells 0-6 have been used, the primitive OPSEQ may be executed to restore these words as well as the mode (EXEC SEQ(OPSEQ)).

6.5 INTERRUPT SERVICE ROUTINES

When a service request is recognized by the operations sequencer, the interrupt decoding primitive is executed. This program transfers control to the interrupt routine whose address is associated with the service request bit. Interrupt service routines may use instruction pad cells 70_8 - 77_8 and data pad cells 70_8 - 73_8. If they use any other instruction pad cells, they must restore the mode as described in section 6.4. If they need to use data pad, they must first save it by executing the primitive subroutine STORPD, which saves a copy of all 64 data pad cells in a system provided save area and exits to IA 1. At the end of the interrupt service, data pad is restored and instruction pad re-established for the mode by executing RSTRT. The interrupt service routine is responsible for clearing its service request bit in the S register before exiting.

6.6 ACCESSING MACRO INSTRUCTION ARGUMENTS

Standard conventions of the Signal Macro Language permit the macro level programmer to specify the arguments for most standard macro instructions directly, in specified data pad cells, or indirectly, with the address of the argument given in line or in a data pad cell. The type of argument specification is indicated by using the two flag bits associated with each word in MOS. A primitive subroutine may be used by the micro-level programmer to fetch arguments following the same conventions.

To fetch an argument, the subroutine GETARG is executed with a return link in IA 1. GETARG uses instruction pad 70_8 - 77_8 and registers A1, A2 and I1; registers I2 and E1 are not used. The argument value will be returned in register A1; register A2 will contain the address +1 where the value was located, if it was located in MOS. If the argument was located in data pad, PA will contain the data pad address. The location counter in PD(76_8) will have been incremented by 1. The A1 register flags (F(1) and F(11)) will be set to the flags of the original argument. Register A2 can be used to access additional consecutive argument values for multiple word arguments or to update an argument value. If the argument is located in line, and is more than one word long, it will be necessary to increment PD(76) to point after the last word. If more than one argument is to be fetched, GETARG can be called repeatedly.

6.7 DATA PAD USAGE

Data pad cells 70_8 - 77_8 are used by the signal macro language for fixed purposes:

Cells 70-73 are work space within a primitive

Cell 74 contains the station number of the current interactive console

Cell 75 contains the address of the current mode defining primitive

Cell 76 contains the address of the next macro-instruction to be executed or the address of the next argument of the current macro instruction

Cell 77 contains the address of the top of the push-down stack used for subroutine cells. The top entry on the stack is the contents of cell 76 of the immediate calling program.

Chapter VII. CHECKOUT OF PRIMITIVES

7.1 CONTROL PANEL

The Control Panel is a useful tool for the checkout of primitive coding. The panel incorporates a single 3-position toggle switch in combination with a number of pushbutton switches and indicator lights. The switches and indicator lights are logically arranged and identified with descriptive labels as shown in Figure 7.1.

The panel provides an operator the capability of:

a. Displaying contents of all registers
b. Manual Load -- data or instructions
c. Initiating the dead-start Disk or Host Loaders (manual or automatic operation)
d. Single instruction execution
e. Breakpoint operations

The panel controls and indicators, along with the function of each, are shown in Tables 7.1 and 7.2.

7.1.1 Register Display

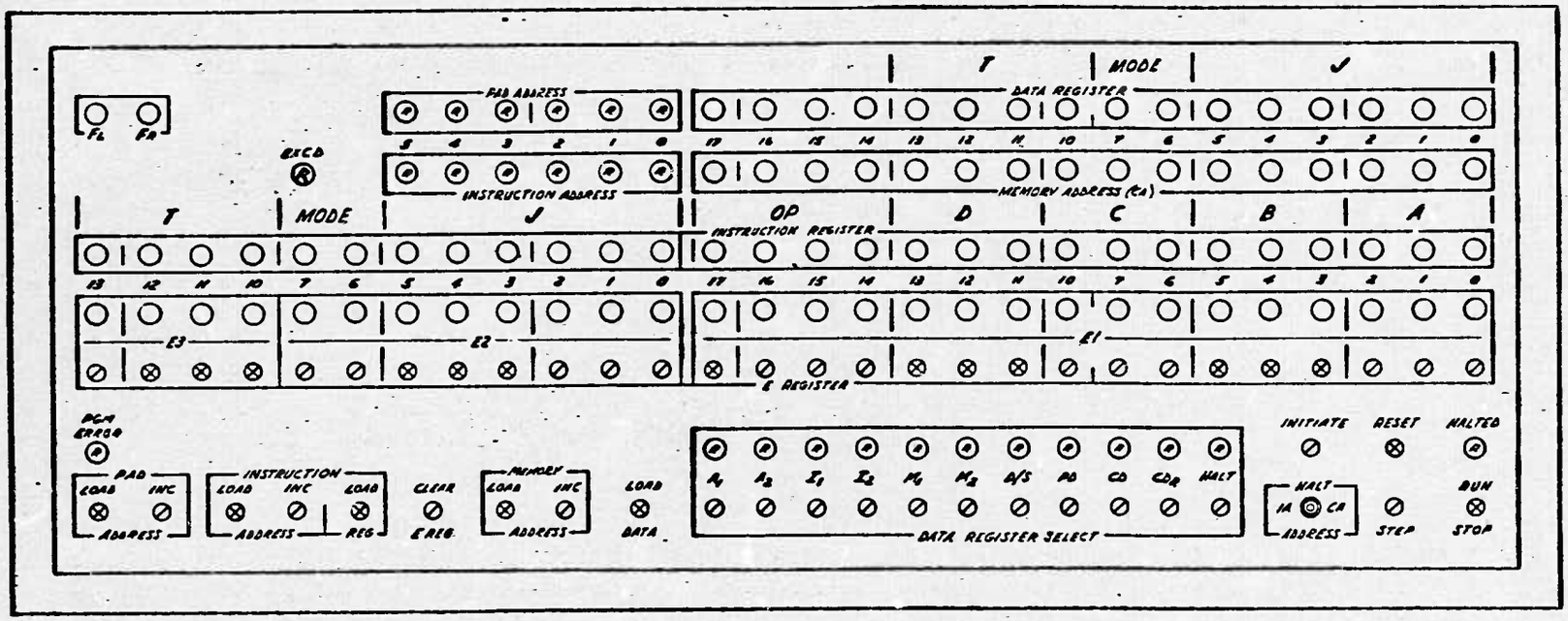
The Macroprocessor must be in the halted state (HALTED indicator lit) for the display of most registers. The data pad address (PA), instruction address (IA), memory address (CA), E1, E2, E3 and current instruction pad output (IR) registers each have their own set of indicators and are displayed whenever the Macroprocessor is halted. To display any of the 12 registers whose names are given in the data register select box, press the momentary contact button under its name. The corresponding indicator will light and the contents of that register, and the two flag bits associated with it, will be displayed in the data register indicators and the F_L and F_R indicators at the top of the panel.

There are two special cases, the D/S and CD_R registers. When D/S is selected, the 8-bit D register is shown in the left half of the data register and the 8-bit S register is shown in the right half of the data register indicators. The F_L indicator displays the setting of the I/O DRDY flip/flop. The F_R indicator displays the setting of the APDRDY flip/flop. When CD_R is selected, the right half of the doubleword addressed by CA(1-17) is displayed, independent of the low order bit of CA. The F_L indicator is lit if the memory is in sequential mode, in which case the CD and CD_R displayed correspond to CA-6. The F_R indicator displays the setting of the memory write flip/flop.

The PAD ADDRESS INC, INSTRUCTION ADDRESS INC and MEMORY ADDRESS INC. buttons may be used to increment these registers by one each time they are pushed.

7.1.2 Manual Load of Registers

The macroprocessor must be halted for any register except E3 to be loaded from the panel. The switches immediately beneath the E registers are used to



WHITE INDICATOR LAMP

RED INDICATOR LAMP

BLUE INDICATOR LAMP

BLACK PUSH BUTTON SW.

WHITE PUSH BUTTON SW.

3 POSITION TOGGLE SW.

Figure 7.1 Control Panel

Table 7.1. Control Panel Controls

Panel Designation	Switches	Function
PAD ADDRESS	LOAD	Transfer the contents of E2 to Pad Address Register and I2
	INC	Pad Address + 1 to Pad Address Register
INSTRUCTION ADDRESS	LOAD	Transfers the contents of E2 to Instruction Address Register & I2
	INC	Instruction Address + 1 to Instruction Address Register
INSTRUCTION REG	LOAD	Transfers contents of E to Instruction Register, I2 and I1
MEMORY ADDRESS	LOAD	Transfers the contents of E1 to MOS Address Register and I1
	INC	MOS Address + 1 to MOS Address Register
E REGISTER		Change state of corresponding E-bit
CLEAR E REG		Replace information in E with zeros
DATA REGISTER SELECT		Each switch (A1, A2, I1, I2, M1, M2, D/S, PD, CD, CDR, HALT) selects the corresponding register to be displayed in the Data Register.
LOAD DATA		Transfers the contents of the E1 Register to the register selected by the Data Register select controls; may also set I1
HALT ADDRESS Switch	IA Position	Breakpoint on Instruction Address: Computer stops on address set in Halt Register
	CA Position	Breakpoint on MOS Address: Computer stops in address set in Halt Register
	CENTER Position	Halt Register may be used for program SYNC.

Table 7.1. Continued

Panel Designation	Switches	Function
RUN-STOP		Starts and stops the computer
INITIATE		Sets bootstrap mode
RESET		Clears all registers in the computer except: (1) Halt, (2) Instruction, and (3) Data Select
STEP		Executes a single instruction each time the switch is depressed.

Table 7.2 Control Panel Indicators

Panel Designation	Function
PAD ADDRESS	Displays current address of Data Pad Memory
INSTRUCTION ADDRESS	Displays current address of Instruction Pad Memory
MEMORY ADDRESS (CA)	Displays current address of MOS Memory
INSTRUCTION REGISTER	Displays contents of Instruction Register
E REGISTER	Displays contents of E Register
DATA REGISTER	Displays the contents of the Register selected by the Data Register Select control
F_L	Displays contents of left most flag bit of the register selected by the Data Register Select control. When the data register selected is CD_R, F_L lights if the program is in sequential mode. When the data register selected in D/S, F_L lights if IØDRDY=1.
F_R	Displays the contents of the right most flag bit of the register selected by the Data Register Select control. When the data register selected in CDR, F_R lights if the MOS memory is performing a write operation. When the data register selected is D/S, F_R lights if APDRDY=
PGM ERROR	Not presently used.
EXCD	Indicates that the program is executing out of the MOS memory.
HALTED	Indicates that the machine is stopped
DATA REGISTER SELECT	Indicates which of eleven registers has been selected to be displayed in the Data Register Indicators. (When D/S is selected, both registers are displayed -- D in the eight highest bits and S in the lowest eight bits.)

set or clear individual bits of these registers; or the CLEAR E REG switch can be used to clear all bits. The contents of the E registers can be selectively transferred to other registers by pressing the appropriate LOAD switch.

a. PAD ADDRESS LOAD transfers the rightmost 6 bits of E2 into PA.

b. INSTRUCTION ADDRESS LOAD transfers E2 into I2 and then transfers I2(0-5) into IA.

c. INSTRUCTION REG LOAD transfers E3 into the T field, E2 into I2 and then into the MODE and J fields and E1 into I1 and then into the OP, DCBA fields of IR.

d. MEMORY ADDRESS LOAD transfers E1 into I1 and then transfers I1 into CA, initiating a memory read cycle at that address.

e. LOAD DATA transfers E1 to the register selected with data register select switches. At the same time, E3(13) is transferred to the bit displayed in F_L and E3(12) is transferred to the bit displayed in F_R. The transfer uses I1 unless the register selected is D/S or HALT. If CD or CD_R is selected, a memory write cycle at CA is initiated.

7.1.3 Initiating the Dead Start Disk or HOST Loaders

A read-only memory is provided which holds two short primitives which may be used to initiate the system from either a disk drive or a host computer.

a. DISK LOADER

To initiate from the lowest numbered disk drive which is ready; from cylinder 0, head 0, the following three steps are sufficient:

i) Press RESET
ii) Press INITIATE
iii) Press RUN.

If it is desired to load from a different disk drive, enter the drive number (0-7) in I1 (10-12) after pressing RESET, then continue as above.

b. HOST LOADER

To initiate from another computer, the sequence used is:

i) Press RESET
ii) Press INITIATE
iii) Press INSTRUCTION ADDRESS INC.
iv) Press RUN.

The Macroprocessor will wait for an interrupt from the host computer.

7.1.4 Single Instruction Execution

Whenever the Macroprocessor is in the halted state, pressing the STEP switch causes one instruction to be executed. If that instruction is a LOAD MACRO or EXECUTE instruction, the single step includes the execution of the last instruction loaded or target instruction. The Macroprocessor remains in the halted state.

7.1.5 Breakpoint Operations

The HALT register, together with the three-position HALT ADDRESS switch, can be used to cause the Macroprocessor to enter the halted state when a match between the HALT register and either IA or CA occurs. In order for a comparison to cause a halt, the HALT ADDRESS switch must be in either the IA, for IA match halts, or CA, for CA match halts, position. No halt will take place if the switch is in the center position. This switch may be activated while the Macroprocessor is running.

An IA halt will occur when a match occurs between the low order 6 bits of the HALT register and the IA register. This halt will occur even if the instruction at this address is not executed because a test operation caused a transfer to an alternate instruction. The halt takes place before the instruction is executed.

A CA halt will occur when a match occurs between all 16 bits of CA and the HALT register. A CA halt will not occur if the match takes place during a load block type (OP CODE 14) instruction. If the memory is in sequential mode, the data in CD when the halt occurs is in memory at CA-6. To halt before an instruction is executed in memory execute sequential mode, the halt address should be six greater than the actual CA of the instruction.

7.2 PRIMITIVE CHECKOUT

The control panel features described above allow the "stepping through" of primitives during program checkout when necessary. Any register can be displayed or altered directly, and the HALT register can be used to quickly skip over program steps which are irrelevant to the checkout. This section offers several suggestions and warnings about checkout at the MP-32 panel.

a. Bits of the E3 register can be tested programmatically, but only set or cleared from the panel. These four switches are useful in at least two ways.

   i) To halt at a particular point. By testing for an E3 bit set, and looping whenever the test is satisfied, the primitive to be tested can cause the Macroprocessor to spin at the point where detailed panel checkout is needed. By halting the Macroprocessor and clearing the E3 bit, the primitive can then be stepped through. This is often the most practical way to halt when a particular primitive is loaded into instruction pad for execution, since CA halts do not function during the load.

ii) To conditionally execute sections of a program which are used for checkout, such as printing of intermediate results.

b. When stepping through input/output programs, consideration must be given to critical timing sequences which must be observed with some devices. For example, disk drives must receive data within about six microseconds of the time they are ready for it, or an over-run condition will occur. Usually, these sections should be skipped over in run mode by setting an IA halt after the time-critical part.

c. When loading any register from the panel, one must remember that other registers may be altered. In particular, the E registers are always changed and register I1 is altered for most panel loads. (See section 7.1 for details.)

d. Memory sequential mode cannot be directly altered from the panel without pushing RESET. To set CA and enter sequential mode or sequential execute mode, it is necessary to manually enter a MODE 3 SET CA instruction in instruction pad and execute it in step mode.

e. When it is desired to execute one macro-instruction per step, set the halt register to 5 and set IA halt. When the RUN button is pushed, one macro instruction will be executed. If IA=4 at the halt, then an interrupt is about to be serviced; pushing the RUN button again will service the interrupt only.

Chapter 8. EXAMPLES OF MICRO-INSTRUCTIONS

8.1 SCOPE

This chapter contains several examples of micro-programs. Examples of primitives used as macro-instruction definitions, primitive subroutines and interrupt response routines are included. Most of the examples are taken from the Signal system software. It is appropriate to note here that the programming techniques illustrated here by no means represent the only way, or even necessarily the best way, to code the particular function. It is hoped that they will provide useful examples of methods which have been used.

8.2 FETCH A MACRO-INSTRUCTION ARGUMENT

This primitive subroutine uses the flags on the next word in the macro-instruction stream to indicate the location of the argument:

#0 argument is given directly
#1 argument is in memory at address given
#2 argument is in data pad at address given
#3 argument is in memory at the location given in data pad at address given.

The first argument word is returned in A1 register. The address of the second word of the argument is in A2, the PA used, if any, is in PA register. The subroutine exits to IA1.

Figure 8.1 GETARG Subroutine

Ref	IA	T	M	J	OP	DCBA	FUNCTION
1	C	1	2	0	10	3376	PA=76 INC PD:A2
2	C	0	0	70	14	1407	LOAD MACRO 10:70
4	70	1	0	71	11	6233	MOVE CD:I1:A1 MOVE I1:CA
5	71	0	0	72	16	4301	MOVE I1F:A1F
6	72	0	0	73	15	1074	IF I1F2=0 SKIP
7	73	1	1	74	7	3231	MOVE I1:PA MOVE PD:CA,A1
8	74	0	0	75	15	0001	IF I1F1=0 GOTO 1
9	75	0	0	76	10	4310	INC A1;A2
10	76	1	0	1	11	0033	MOVE CD:I1:A1 GOTO1
3	77	0	1	70	6	0402	DEC A2:CA MOVE A2:PD GOTO70

1. The subroutine is entered with an EXEC SEQ (GETARG)instruction. The first step sets PA to 76 during the first clock; the incremental contents of the old current data pad cell are also transferred to A2, but will be replaced. During the second clock, the contents of pad cell 76 are incremented by 1 in the adder, the output of the adder is gated in register A2.

2. The remainder of the subroutine is loaded into instruction (macro) pad, addresses 70-77. At the conclusion of this load, the last instruction loaded, at IA77, is executed.

3. The contents of register A2 are decremented by 1 in the adder and transferred to CA, initiating a read cycle at the end of the instruction. The contents of A2 are transferred back to data pad cell 76. The effect of instructions 1 and 3 is to increment the contents of pad 76 while transferring the old value into CA. The successor to this instruction is at IA 70.

4. This instruction includes a request to transfer CD into I1 register. Since memory was initiated in the previous instruction, the data is not ready immediately and a pause of three clock times duration will occur. After this pause, the now valid contents of CD will be transferred to I1; at the same time, the old contents of I1 will be transferred to A1; no transfer of I1 into CA takes place in this clocktime since it is not the last clock in the instruction. During the second clock-time, CD is again transferred to I1; the contents of I1, which were setup in the first clock, are transferred to A1; they are also transferred, through the adder, into CA, initiating a read cycle to fetch an indirect (#1) argument, if needed.

5. The flags of the direct argument are transferred to the A1 register flags, where they are saved for use by the calling routine.

6. The left of the two flags is tested to see if the direct argument specifies a data pad cell. If not, instruction pad 74 is executed next, skipping step 7.

7. If the left flag was on, the direct argument is a PA. This instruction sets PA with the value specified by transferring I1 into PA during the first clock. At the same time, the contents of data pad 76 are transferred through the adder to A1, but this transfer is negated during the next clock. During the second clock, the contents of I1 are again transferred to PA, and the contents of PD at the new PA are tranferred through the adder to A1 and, at the end of the clock, to CA. No pause is required, since four clocks have elapsed since the last time memory was initiated.

8. At this point, the data being fetched is the desired argument if the right flag is on (#1 or #3). Otherwise, the argument is in the A1 register. The right I1 flag is tested, if it is off, the subroutine exits to IA 1. Otherwise, steps 9 and 10 are executed.

9. The contents of register A1 are incremented by one in the adder, the adder output is transferred to A2. This instruction is necessary only for the #3 case, to establish the address of possible additional words of argument in register A2.

10. The memory word is transferred from CD to I1 during the first clock; a pause may be necessary here if the data is not yet valid. During the second clock, this data is transferred from I1 into A1. The subroutine then exits to IA 1.

8.3 SCAN A TABLE IN DATA PAD FOR A MATCH WITH A1

This primitive segment searches sequentially through data pad for the first word which is equal to A1. It then uses the PA value where the match occurred to index a table whose address is in PD(0). The table entry is assumed to be the address of the next macro instruction.

Figure 8.2 SEARCH

Ref	IA	T	M	J	OP	DCBA	FUNCTION
1	C	0	2	70	14	1403	PA=0 LOAD MACRO 4:70
3	70	0	0	70	7	3510	IF A1=PD GOTO 71 OTW SPIN INC PA
4	71	0	0	72	13	2427	MOVE PA:E2 DEC I2:I1
5	72	1	1	0	12	6401	MOVE E2:I2 ADD I1,I2:CA GOTO 0
2	73	0	2	70	12	3201	MOVE PD:I2 PA=1 GOTO 70

1. This primitive is assumed to be entered by an EXEC SEQ(SEARCH). The first instruction loads the following four instructions into instruction pad cells 70 to 73, it also sets PA=0 (MODE 2). The last instruction loaded is then executed. This primitive uses instruction pad because the test in step 3 is to be its own successor in case of failure. It also must have its test passed successor in instruction pad.

2. The contents of data pad 0 are transferred through the adder to register I2. At the same time, PA is set to 1. The next instruction is fetched from IA 70.

3. This instruction compares the current data pad cell and register A1. If their contents are equal, the instruction at IA 71 is executed, otherwise, the instruction is repeated. In either case, PA is incremented by 1, so that if the instruction is repeated, the next data pad cell is compared. Note that this instruction would cause the processor to spin until manually stopped if no data pad cell matched A1.

4. When a match is found, PA has a value one greater than that of the matching cell. To compensate for that, one is subtracted from the contents of register I2 (the base address) as it is transferred to register I1. At the same time, PA is transferred to register E2.

5. During the first clock register E2 is transferred to I2. In the second clock time, registers I1 and I2 are added together and the adder output is transferred to CA, initating a read cycle, at the end of the instruction. The next instruction is taken from IA 0, which is part of the operations sequencer. It transfers the memory data through I1 into CA and then enters CORE EXEC SEQ mode from the instruction in IA 2. A listing of the operations sequencer may be found in Figure 6.1.

8.4 DATA PAD AND ADDER ILLUSTRATION

This primitive micro-program illustrates some aspects of use of data pad and the adder.

Figure 8.3 Edit Insert Setup

Ref	IA	T	M	J	OP	DCBA	FUNCTION
1	C	1	2	0	7	3362	PA=62 INC PD:A1
2	C	1	2	0	10	3261	PA=61 MOVE PD:A2
3	C	0	0	26	14	1407	LOAD MACRO 10:26
5	26	1	2	27	2	0755	PA=55 MOVE CA:PD
6	27	1	1	30	11	6030	MOVE CD:I1 ADD I1,A2:CA
7	30	1	2	31	2	0754	PA=54 MOVE CA:PD
8	31	1	2	32	7	3263	PA=63 MOVE PD:A1
9	32	1	2	33	11	3054	PA=54 ADD A1,PD:I1
10	33	0	2	34	6	6250	MOVE I1:CA PA=50
11	34	1	1	40	5	6037	MOVE CD:I1 MOVE CA:PD ADD I1,A2:CA GOTO 40
4	35	0	1	26	6	4000	ADD A1,A2:CA GOTO 26

1. Instruction is executed out of memory. Sets PA=62 in the first clock. Increments the contents of Pad 62 into register A1 in the second clock.

2. Moves the contents of Pad 61 into register A2.

3. Loads the next 8 instructions from Cd into Macro Pad at IA 26 and executes the last one loaded, IA 36.

4. Add the contents of registers A1 and A2 and place the sum in register CA, thereby addressing a new memory cell, then execute IA 26.

5. Moves the contents of the register CA into Pad cell 55 and executes the instruction at IA 27 next.

6. Reads memory into register I1 at the location where CA pointed when the instruction began, and the adder adds the contents of register I1 when the instruction began to the contents of register A2 during the first clock of the instruction. Since the adder is only moved into CA on the last clock of an instruction, the output of the first add does not go anywhere. During the next clock time the new contents of I1 are added to the contents of A2 and the sum is moved into CA, thus addressing a new memory cell. Instruction 30 is executed next.

7. Moves the contents of register CA into pad cell 54 and executes instruction 31.

8. Moves the contents of pad 63 into register A1 and executes instruction 32.

9. During the first clock-time the adder forms the sum of register A1 and Pad cell 63 and takes that sum into register I1 while changing PA to 54. During the second clock-time the sum of A1 and Pad cell 54 is formed and moved into register I1. Execution continues at IA 33.

10. The contents of register I1 is moved into register CA, thus addressing a new cell of memory, while PA is set to 50. Successor is IA 34.

11. During the first clock the sum of registers I1 and A2 is formed while memory is read and its contents placed in register I1. CA does not go into Pad until the beginning of the second clock time. After the old value of CA has been taken into Pad, the sum of the new I1 and register A2 is moved into register CA and the successor instruction, IA 40, is executed.

8.5 INTERRUPT SERVICE ROUTINE

This primitive is executed when a service request is recognized from an attached AP-120 array processor. It treats the interrupt as a signal that the array processor has complete a transform, and clears a global variable which is used to indicate the state of the array processor to macro-processor programs.

Figure 8.4 CLRAPBUSY

Ref	IA	T	M	J	OP	DCBA	FUNCTION
1	C	0	1	0	16	4403	S(3)=0
2	C	0	0	0	11	1120	MOVE 0:I1
3	C	3	3	3		APBUSY	WRITE (APBUSY) GOTO 3

1. The service request bit for the array processor is cleared.

2. The data to be written in APBUSY is put in I1.

3. The contents of I1 are written into memory at address APBUSY. The micro-program must leave core execute mode at this point. It does so by transferring to IA 3, where if no further service requests exist, the next macro instruction will be processed.

APPENDIX A: MP MICRO INSTRUCTION SUMMARY FOR S:1/2

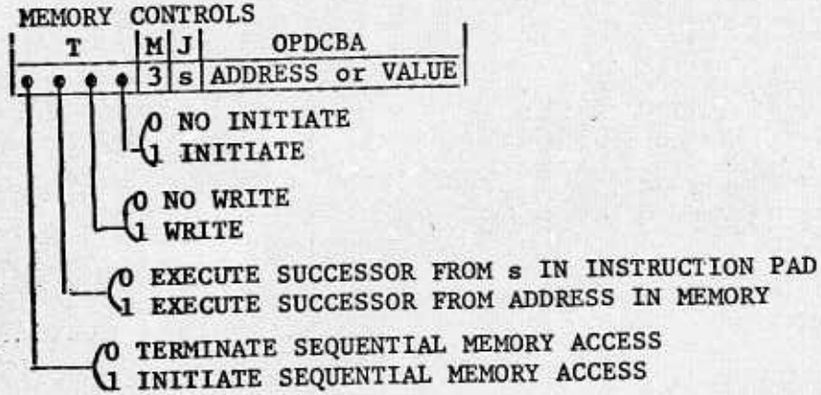
TABLE OF CONTENTS

SECTION	TITLE	FUNCTION	PAGE
1	MODE 3	Memory Controls	1
2	OP 0	Single Bit Decode Instructions	2
3	OP 1,2,3,4,5	Succinct Adder Instructions	3
4	OP 6,7,10,11,12,13	Full Adder Instructions	5
5	OP 14	I/O and Control Instructions	8
6	OP 5,14,16	AP-Commands	14
7	OP 15	Bit Select Conditionals	15
8	OP 16	Logical Operations	16
9	OP 17	Linkage Operations	18

TABLE 1

CA-COMMANDS

MEMORY CONTROLS



****
* MODE 3 *
****

FUNCTION	T	M	J	OPDCBA	NOTES
CA=(VALUE)	0	3	s	VALUE	Load immediate
READ(ADDRESS)	1	3	s	ADDRESS	Initiate memory access, read mode
READ SEQ(ADDRESS)	11	3	s	ADDRESS	Initiate sequential memory access; read mode
WRITE(ADDRESS)	3	3	s	ADDRESS	Write I1→CD at given address
EXEC(ADDRESS)	5	3	0	ADDRESS	Start memory execution
EXEC SEQ(ADDRESS)	15	3	0	ADDRESS	Start sequential memory execution

NOTE: Any CA - command that is not sequential will terminate a prior sequential mode

TABLE 2

TWELVEFOLD OPERATION

OP	D	C	B	A
0	SINGLE BIT DECODE			

****
* OP 0 *
* MODE 0 *
****

CODE	D	C	B	A
0	NOOP	NOOP	NOOP	NOOP
1	I2=0	I1=0	MOVE I1:A2:A1	A1=0
2	NOOP	FL=0	MULT PDR,A1R	A2=0
3	I2=0	(1) and (2)	(1) and (2)	(1) and (2)
4	NOOP	MOVE A1:I2 DEC PA	MOVE I2:PD	MOVE PD:I1
5	I2=0	(1) and (4)	(1) and (4)	(1) and (4)
6	NOOP	(2) and (4)	(2) and (4)	(2) and (4)
7	I2=0	(1), (2) and (4)	(1), (2) and (4)	(1), (2) and (4)

MOVE PD:REGS means MOVE PD:I1 MOVE I1:A2:A1 MOVE A1:I2
MOVE REGS:PD means MOVE I1:A2:A1 MOVE A1:I2 MOVE I2:PD

OP	D	C	B	A	FUNCTION
0	0	4	1	4	MOVE PD:REGS DEC PA
0	0	4	5	0	MOVE REGS:PD DEC PA

TABLE 3

FOURFOLD OPERATIONS

OP 1,2,3,4,5
MODE 0, 1

OP	D	C	B	A
1	ADDER	E	PA	I2
2	ADDER	E	A2	PD
3	ADDER	M1	M2	A1
4	ADDER	M1	M2	I2
5	ADDER	E	I1	PD

CODE	D: ADDER ADDOP	D: ADDER ADDFN	C: E	C: M1
0	NOOP	ADD A1,A2	NOOP	NOOP
1	SM/2c A2	NEG A2	MOVE I2:E2	MOVE PDL:M1
2	ADD A1,I2:I2	ADD A1,I2	MOVE PA:E2	MOVE MPL:SM1
3	ADD A1,PD:A1	ADD A1,PD	FLIP E1	MOVE I1R:M1
4	ADD A1,A2:A2	ADD A1,A2	(PDI):E1	MOVE PDR:SM1
5	SM/2c A1	NEG A1	MOVE DI:E1	SM1=0
6	ADD I1,A2:I1	ADD I1,A2	MOVE PDR:E1R	IF PA=0
7	ADD CA,A2:CA	ADD CA,A2	MOVE PD:E1	MOVE PDL:SM1

CODE	B: PA	B: A2	B: I1	B: M2
0	NOOP	NOOP	NOOP	NOOP
1	INC PA	(ADDFN):A2	MOVE E1:I1	MOVE A1L:SM2
2	DEC PA	(ADDFN)':A2	(ADDFN):I1	DECIM E1:M2
3	MOVE I1:PA	MOVE I1:A2	MOVE CD:I1	MOVE I1R:M2
4	MOVE E2:PA	A1:CR:CL:A2	MOVE PD:I1	MOVE A1R:SM2
5	EXCH E2:PA	WF=1	MOVE DI:E1:I1	SM2=0
6	DEC ØCT	DBLWF=1	MOVE E1R:I1	MOVE CR:CL:A2
7	INC ØCT	APDRDY=0	MOVE E1L:I1	MOVE A1:CR:CL

CODE	A: PD	A: A1	A: I2
0	NOOP	NOOP	NOOP
1	MOVE A1:PD	MOVE PD:A1	MOVE E2:I2
2	MOVE A2:PD	(ADDFN):A1	(ADDFN):I2
3	MOVE I1:PD	MOVE I1:A1	MOVE I1:I2
4	MOVE I2:PD	MOVE A2:A1	MP:I2,MS:A2E1
5	MOVE A1R:PDR	WC=1	WF=1
6	MOVE A1L:PDL	MOVE PDR:A1	MOVE CD:I1
7	MOVE CA:PD	SCAT PD:A1	MOVE PA:E2

Examples:

TABLE 4

THREEFOLD OPERATIONS

OP	D	C	BA
1	ADDOP	E	PA=BA
2	ADDOP	PD	PA=BA
3	ADDOP	E1	PA=BA
4	ADDOP	M1	PA=BA

```
**************
* OP 1,2,3,4 *
*            *
* MODE 2     *
*            *
* PA=BA      *
**************
```

CODE	D	C			
	ADDOP	E	PD	E1	M1
0	NOOP	MOVE CD:PD	MOVE DI:E1:PD	MOVE PD:E1	NOOP
1	SM/2c A2	MOVE I2:E2	MOVE A1:PD	MOVE A1:E1	MOVE PDL:M1
2	ADD A1,I2:I2	MOVE PA:E2	MOVE A2:PD	MOVE A2:E1	MOVE MPL:SM1
3	ADD A1,PD:A1	FLIP E1	MOVE I1:PD	MOVE I1:E1	MOVE I1R:M1
4	ADD A1,A2:A2	MOVE CD:I1	MOVE I2:PD	MOVE I2:E1	MOVE PDR:SM1
5	SM/2c A1	MOVE DI:E1	MOVE A1R:PDR	MOVE A1:E1	SM1=0
6	ADD I1,A2:I1	MOVE PDR:E1R	MOVE A1L:PDL	MOVE A1:E1	IF PA=0
7	ADD CA,A2:CA	MOVE PD:E1	MOVE CA:PD	MOVE CA:E1	MOVE PDL:SM1

TABLE 5

START AP OPERATION

OP	DCBA
5	PSA=DCBA

```
************
* OP 5     *
*          *
* MODE 2   *
*          *
* PSA=DCBA *
************
```

TABLE 6

THREEFOLD OPERATIONS

OP	DC	B	A
6	ADDOP	PA	PD
7	ADDOP	PA	A1
10	ADDOP	A2	A1
11	ADDOP	I1	A1
12	ADDOP	A2	I2
13	ADDOP	I1	I2

****
* OP 6,7,10,11,12,13 *
* MODE 0 *
* DC = ADDOP *
****

C

D	ADDOP 0	1	2	3	4	5	6	7
0	NOOP	AND A1,A2:A2	MOVE A2:CA	INC A2	DEC A2	IF A1=A2	IF A1<A2	IF A1>A2
1	SM/2c A2	CA=0	NEG A2:A2	2c/SM A2	SUBT I1,A2:A2	IF A1=0	SUBT A1,A2:A2	MOVE A2':A2
2	ADD A1,I2:I2	AND A1,I2:I2	MOVE I2:I1	INC I2	DEC I2	IF I2=0	IF A1<I2	IF A1>I2
3	ADD A1,PD:A1	AND A1,PD:A1	MOVE PD:CA	INC PD:A1	DEC PD:A1	IF A1=PD	IF A1<PD	IF A1>PD
4	ADD A1,A2:A2	AND A1,A2:A1	MOVE A1:A2	INC A1	DEC A1	IF A1R=A2R	ADD A1,A2,CF:A1	MOVE A2:I1
5	SM/2c A1	OR A1,A2:A2	NEG A1:A1	2c/SM A1	SUBT I2,A1:A1	EOR A1,A2:A2	SUBT A2,A1:A1	SUBT A2,A1:A2
6	ADD I1,A2:I1	AND I1,A2:I1	MOVE I1:CA	INC I1	ADD I1,I2:I1	IF A2=I1	IF A2<I1	IF A2>I1
7	ADD CA,A2:CA	AND CA,A2:CA	MOVE CA:I1	INC CA	DEC CA	IF A2=CA	IF A2<CA	IF A2>CA

(ADDOP=INC CA(2) M=1)

CODE	B PA	A2	I1	A PD	A1	I2
0	NOOP	NOOP	NOOP	NOOP	NOOP	NOOP
1	INC PA	See NOTE	MOVE E1:I1	MOVE A1:PD	MOVE PD:A1	MOVE E2:I2
2	DEC PA	See NOTE	See NOTE	MOVE A2:PD	See NOTE	See NOTE
3	MOVE I1:PA	MOVE I1:A2	MOVE CD:I1	MOVE I1:PD	MOVE I1:A1	MOVE I1:I2
4	MOVE E2:PA	A1:CR:CL:A2	MOVE PD:I1	MOVE I2:PD	MOVE A2:A1	MP:I2,MS:A2=1
5	EXCH E2:PA	WF=1	MOVE DI:E1:I1	MOVE A1R:PDR	WC=1	WF=1
6	DEC ØCT	DBLWF=1	MOVE E1R:I1	MOVE A1L:PDL	MOVE PDR:A1	MOVE CD:I1
7	INC ØCT	APDRDY=0	MOVE E1L:I1	MOVE CA:PD	SCAT PD:A1	MOVE PA:E2

NOTE: If any of the ADDFN operations;
(ADDFN):A2
(ADDFN):A2
(ADDFN):I1
(ADDFN):A1
(ADDFN):I2
are required, use TABLE 7 for correct instruction.

TABLE 7

THREEFOLD OPERATIONS

OP	DC	B	A
6	ADDFN	PA	PD
7	ADDFN	PA	A1
10	ADDFN	A2	A1
11	ADDFN	I1	A1
12	ADDFN	A2	I2
13	ADDFN	I1	I2

OP 6,7,10,11,12,13

MODE 0,1

DC = ADDFN

C

	ADDFN							
D	0	1	2	3	4	5	6	7
0	ADD A1,A2	AND A1,A2	MOVE A2	INC A2	DEC A2	EOR'A1,A2	SUBT' A1,A2	SUBT A2,A1
1	NEG A2	(REG)=0	NEG A2	NEG A2	SUBT I1,A2	MOVE A1'	SUBT A1,A2	MOVE A2
2	ADD A1,I2	AND A1,I2	MOVE I2	INC I2	DEC I2	MOVE I2'	SUBT' A1,I2	SUBT I2,A1
3	ADD A1,PD	AND A1,PD	MOVE PD	INC PD	DEC PD	EOR' A1,PD	SUBT' A1,PD	SUBT PD,A1
4	ADD A1,A2	AND A1,A2	MOVE A1	INC A1	DEC A1	EOR' A1,A2	ADD A1,A2,CF	MOVE A2
5	NEG A1	OR'A1,A2	NEG A1	NEG A1	SUBT I1,A2	EOR' A1,A2	SUBT A2,A1	SUBT A2,A1
6	ADD I1,A2	AND I1,A2	MOVE I1	INC I1	ADD I1,I2	EOR' I1,A2	SUBT'A2,I1	SUBT I1,A2
7	ADD CA,A2	AND CA,A2	MOVE CA	INC CA	DEC CA	EOR' CA,A2	SUBT'A2,CA	SUBT CA,A2

(ADDOP= INC CA(2) M=1)

CODE	B			A		
	PA	A2	I1	PD	A1	I2
0	NOOP	NOOP	NOOP	NOOP	NOOP	NOOP
1	INC PA	(ADDFN):A2	MOVE E1:I1	MOVE A1:PD	MOVE PD:A1	MOVE E2:I2
2	DEC PA	(ADDFN)':A2	(ADDFN):I1	MOVE A2:PD	(ADDFN):A1	(ADDFN):I2
3	MOVE I1:PA	MOVE I1:A2	MOVE CD:I1	MOVE I1:PD	MOVE I1:A1	MOVE I1:I2
4	MOVE E2:PA	A1:CR:CL:A2	MOVE PD:I1	MOVE I2:PD	MOVE A2:A1	MP:I2,MS:A2FL
5	EXCH E2:PA	WF=1	MOVE DI:E1:I1	MOVE A1R:PDR	WC=1	WF=1
6	DEC ØCT	DBLWF=1	MOVE E1R:I1	MOVE A1L:PDL	MOVE PDR:A1	MOVE CD:I1
7	INC ØCT	APDRDY=0	MOVE E1L:I1	MOVE CA:PD	SCAT PD:A1	MOVE PA:E2

TABLE 8

TWOFOLD OPERATIONS

OP	DC	BA
6	ADDOP	PA=BA
7	(ADDFN):A1	PA=BA
10	(ADDFN):A2	PA=BA
11	(ADDFN):I1	PA=BA
12	(ADDFN):I2	PA=BA
13	(ADDFN):CA WF=1	PA=BA

```
**********************
* OP 6,7,10,11,12,13 *
*                    *
* MODE 2             *
*                    *
* PA=BA              *
**********************
```

C

	ADDOP							
D	0	1	2	3	4	5	6	7
0	NOOP	AND A1,A2:A2	MOVE A2:CA	INC A2	DEC A2	IF A1=A2	IF A1<A2	IF A1>A2
1	SM/2c A2	CA=0	NEG A2:A2	2c/SM A2	SUBT I1,A2:A2	IF A1=0	SUBT A1,A2:A2	MOVE A2':A2
2	ADD A1,I2:I2	AND A1,I2:I2	MOVE I2:I1	INC I2	DEC I2	IF I2=0	IF A1<I2	IF A1>I2
3	ADD A1,PD:A1	AND A1,PD:A1	MOVE PD:CA	INC PD:A1	DEC PD:A1	IF A1=PD	IF A1<PD	IF A1>PD
4	ADD A1,A2:A2	AND A1,A2:A1	MOVE A1:A2	INC A1	DEC A1	IF A1R=A2R	ADD A1,A2,CF:A1	MOVE A2:I1
5	SM/2c A1	OR A1,A2:A2	NEG A1:A1	2c/SM A1	SUBT I2,A1:A1	EOR A1,A2:A2	SUBT A2,A1:A1	SUBT A2,A1:A2
6	ADD I1,A2:I1	AND I1,A2:I1	MOVE I1:CA	INC I1	ADD I1,I2:I1	IF A2=I1	IF A2<I1	IF A2>I1
7	ADD CA,A2:CA	AND CA,A2:CA	MOVE CA:I1	INC CA	DEC CA	IF A2=CA	IF A2<CA	IF A2>CA

C

	ADDFN							
D	0	1	2	3	4	5	6	7
0	ADD A1,A2	AND A1,A2	MOVE A2	INC A2	DEC A2	EOR'A1,A2	SUBT' A1,A2	SUBT A2,A1
1	NEG A2	(REG)=0	NEG A2	NEG A2	SUBT I1,A2	MOVE A1'	SUBT A1,A2	MOVE A2
2	ADD A1,I2	AND A1,I2	MOVE I2	INC I2	DEC I2	MOVE I2'	SUBT' A1,I2	SUBT I2,A1
3	ADD A1,PD	AND A1,PD	MOVE PD	INC PD	DEC PD	EOR' A1,PD	SUBT' A1,PD	SUBT PD,A1
4	ADD A1,A2	AND A1,A2	MOVE A1	INC A1	DEC A1	EOR' A1,A2	ADD A1,A2,CF	MOVE A2
5	NEG A1	OR'A1,A2	NEG A1	NEG A1	SUBT I2,A1	EOR' A1,A2	SUBT A2,A1	SUBT A2,A1
6	ADD I1,A2	AND I1,A2	MOVE I1	INC I1	ADD I1,I2	EOR' I1,A2	SUBT' A2,I1	SUBT I1,A2
7	ADD CA,A2	AND CA,A2	MOVE CA	INC CA	DEC CA	EOR' CA,A2	SUBT' A2,CA	SUBT CA,A2

TABLE 9

MICRO DATA BLOCKS

T	M	J	OP	D	C	BA
●	●	●	14	●	0	●

- BA+1 = NUMBER OF TRANSFERS
- TRANSFER SELECTION CODE
- SUCCESSOR (S)
- {0,1 PA NOT INITIATED / 2 PA=0} (m)
- EXECUTE TIME CONTROL

```
**************
* OP 14      *
*            *
* C = 0      *
*            *
* D = 1,2,3,4*
**************
```

FUNCTION	T	M	J	OP	D	C	BA	NOTES
INPUT n:PD	2	m	s	14	0	0	n-1	n 16 bit words DI→E1→PD
LOAD PAD (n)	0	m	s	14	1	0	n-1	n 32 bit words CD→PD;CA,PA even
STORE PAD (n)	1	m	s	14	2	0	n-1	n 32 bit words PD→CD;CA,PA even
LOAD AP (n)	0	m	s	14	3	0	n-1	n 32 bit words CD→DP;CA even
STORE AP (n)	0	m	s	14	4	0	n-1	n 32 bit words DP→CD;CA even

WAIT (k·n)

T	M	J	OP	D	CBA
k-1	m	s	14	5	n-1

(k·n)/8 μs wait time
n-1<400

```
*********
* OP 14 *
*       *
* D = 5 *
*********
```

E2 = VALUE

T	M	J	OP	D	CBA
0	0	s	14	6	Value

MODES 0,1,2 are equivalent

```
*********
* OP 14 *
*       *
* D = 6 *
*********
```

INPUT/OUTPUT

T	M	J	OP	D	CB	A
●	●	s	14	7	●	●

- COMMAND SELECTION CODE
- DEVICE NUMBER
- D(7)=M AFTER EXECUTION
- EXECUTION TIMING CONTROL

```
*********
* OP 14 *
*       *
* D = 7 *
*********
```

NOTE: Refer to TABLE for construction of I/O Commands.

TABLE 10

LOAD MACRO BLOCK

M = 0, successor = last instruction loaded
M = 1, successor = CD, CDR CEX=1
M = 2, PA=0, successor = last instruction loaded.

s = Initial IA s+n-1 = Last IA

n = number of instructions in block

FUNCTION	T	M	J	OP	DC	BA	NOTES
LOADE MACRO n:s	0	M	s	14	04	n-1	n copies of CD→instruction pad
LOADCD MACRO n:s	0	M	s	14	14	n-1	n instructions from CD→instruction pad
LOADPD MACRO n:s	1	M	s	14	24	n-1	n instructions from PD→instruction pad
LOADNULL MACRO n:s	k-1	M	s	14	34	n-1	instruction pad unchanged

k·n = execution time

OP 14
C = 4
D = 0,1,2,3
MODE 0,1,2

EXECUTE INSTRUCTIONS

FUNCTION	T	M	J	OP	DC	BA	NOTES
EXEC I2,I1	0	M	s	14	44	o	instruction = I2,I1
EXEC(s)L,I1	0	M	s	14	54	o	instruction = (s)L,I1
EXEC I2,(s)R	0	M	s	14	64	o	instruction = I2,(s)R
EXEC CD,CDR	0	M	s	14	74	o	instruction = CD,CDR CEX=1

M=0,1 BA=0
M=2 PA=BA

OP 14
C = 4
D = 4,5,6,7
MODE 0,1,2

TABLE 11

I/O COMMANDS

DEVICES	CODES
STATION	0,1,2,3
DISPLAY STATUS	4
DISK	10
ANALOG	12
HOST	13
TIMER	14
DMA	16
ARRAY PROCESSOR	17

SERVICE REQUEST BITS

S

7	6	5	4	3	2	1	0	
		OPEN			OPEN			KEYBOARD
								DISK POSITION
								ARRAY PROCESSOR
								TIMER
								DISK TRANSFER
								HOST

OP 14

D = 7

OP 14

DC = 70

STATION COMMANDS

FUNCTION	T	M	J	OP	DC	B	A	NOTES
INPUT KB	2	1	s	14	70	STN	0	KEY(STN)→DI IF DI(10)=1, no data ready
OUTPUT PT (A2,I2)	3	0	s	14	70	STN	1	A2→XDAC I2→YDAC DISPLAY STATUS=1 for 32 μs
ERASE	4	0	s	14	70	STN	2	DISPLAY STATUS=1 for 750 ms
NONSTORE MODE	3	0	s	14	70	STN	3	In NONSTORE MODE no storage results from display output
STORE MODE	3	0	s	14	70	STN	4	In STORE MODE storage results from display output
REQUEST LITE	3	0	s	14	70	STN	5	Station REQUEST light turns ON
LOCKOUT LITE	3	0	s	14	70	STN	6	Station REQUEST light turns OFF, LOCKOUT light turns ON
DISCONNECT LITES	3	0	s	14	70	STN	7	Station lights turn OFF

17 16 15 14 13 12 11 10 7 6 5 4 3 2 1 0

Not used

DISPLAY STATUS Station #3

" " " #2

DISPLAY STATUS Station #1

" " " #0

DISPLAY COMMANDS

FUNCTION	T	M	J	OP	DC	B	A	NOTES
DISPLAY STATUS	2	1	s	14	70	4	0	DISPLAY STATUS→DI If DI(STN)=0, station display unbusy
DISPLAY (A2,I2)	3	0	s	14	70	4	1	A2→XDAC I2→YDAC In SLEW MODE allow 2 μs per output
SLEW MODE	3	0	s	14	70	4	2	SLEW MODE begins SLEW MODE ends if no output for 32 μs

TABLE 12

DISK COMMANDS

FUNCTION	T	M	J	OP	DCB	A	NOTES
DISK READ	2	1	s	14	710	0	(when DATA→DI, IODRDY=1, S(6)=1)
DISK WRITE	2	1	s	14	730	0	IODRDY=1 (when I1→DATA, IODRDY=0, S(6)=1)
DISK STATUS	2	1	s	14	710	1	DISK STATUS→DI (when DI→E1, S(6)=0)
DISK EXEC I1	2	1	s	14	710	2	D(7)=1 (when I1→COMMAND, D(7)=0)
DISK EOB	4	0	s	14	710	3	IF READ, COMPARE CHECKSUM IF WRITE, STORE CHECKSUM } (when done, S(6)=1)
DISK RESET	4	0	s	14	710	4	RESTORE DRIVE to cylinder zero, RESET status
POSITION STATUS	2	1	s	14	710	5	POSITION STATUS→DI
SELECT DRIVE	0	1	s	14	710	7	DRIVE NUMBER = I1(12,11,10)

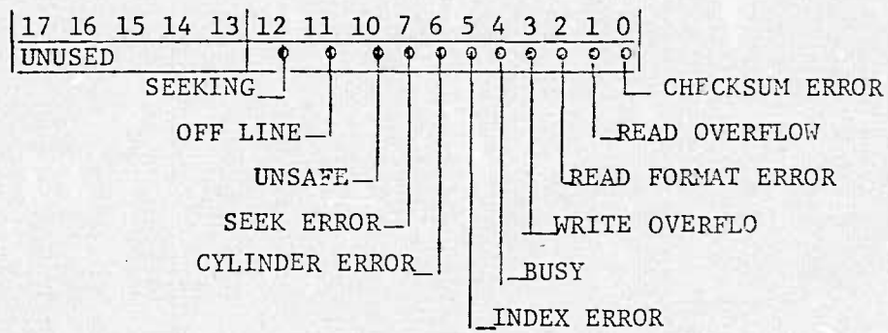
****
* OP 14 *
*   *
* DCB = 710 *
*   *
* OR 730 *
****

I1-COMMANDS

FUNCTION	I1
SEEK	CYLINDER NUMBER
READ18	10000 + HEAD NUMBER
WRITE18	20000 + HEAD NUMBER
READ16	50000 + HEAD NUMBER
WRITE16	60000 + HEAD NUMBER
READHDR	150000 + HEAD NUMBER
WRITEHDR	160000 + HEAD NUMBER

CYLINDER NUMBER <310
HEAD NUMBER <24

DISK STATUS



POSITION STATUS

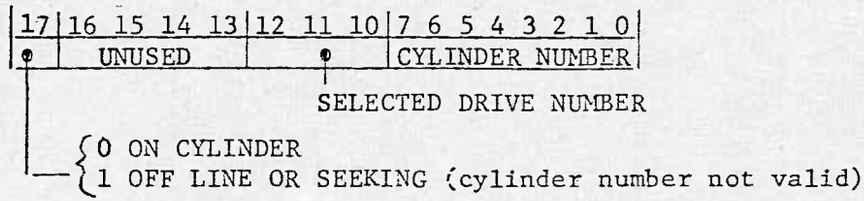


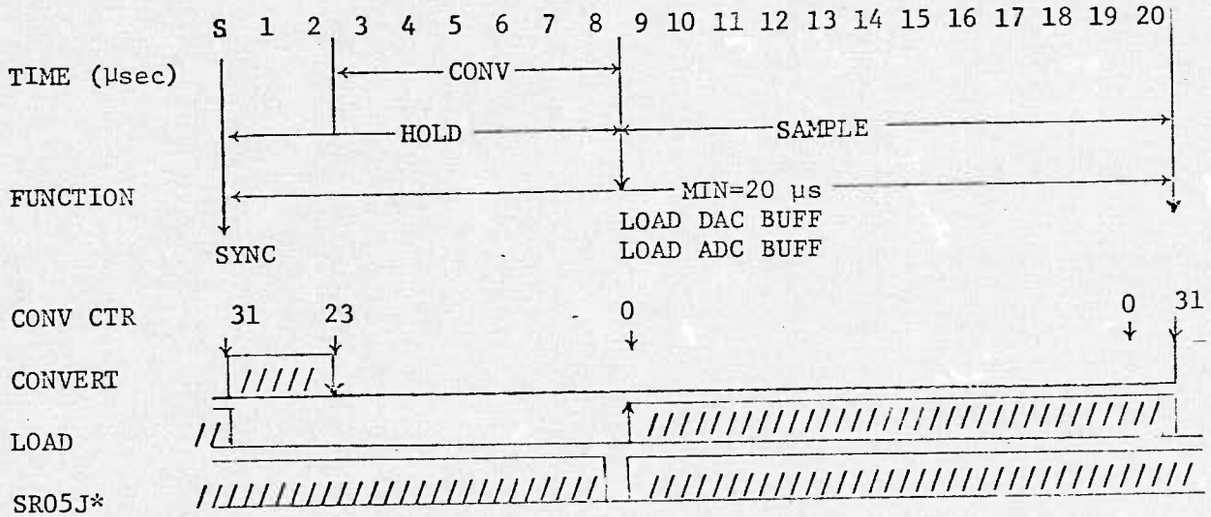
TABLE 13

ANALOG COMMANDS (SERIAL TWO)

****
* OP 14 *
* DCB=712 *
****

FUNCTION	T	M	J	OP	DCBA	NOTES
INPUT	2	1	S	14	7120	ADC(PA)→DI
INPUT & CLRS5	2	1	S	14	7121	ADC(PA)→DI 0→S5
OUTPUT	7	0	S	14	7122	PD→DAC(PA)
OUTPUT & CLRS5	7	0	S	14	7123	PD→DAC(PA) 0→S5

PA	PD-FUNCTION
0	ADC0
1	ADC1
2	ADC2
3	--
4	--
5	--
6	--
7	--
10	DAC0
11	--
12	--
13	SYNC COUNT-1
14	DCØ
15	
16	
17	



RESOLUTION OF CONVERT COUNTER IS 250 μs/count

TABLE 14

HOST COMMANDS

FUNCTION	T	M	J	OP	DCB	A	NOTES
READ HOST	2	1	s	14	713	0	HOST DATA→DI
WRITE HOST	2	1	s	14	733	0	I1→HOST DATA
EXEC HOST	2	1	s	14	713	2	I1→HOST COMMAND

```
**********
* OP 14  *
*        *
* DCB=713*
*        *
* OR 733 *
**********
```

TIMER COMMANDS

FUNCTION	T	M	J	OP	DCB	A	NOTES
READ TIMER	2	1	s	14	714	0	TIMERDATA→DI
TIMER SYNC	3	0	s	14	714	1	START Synchronization
DISABLE TIMER SYNC	3	0	s	14	714	2	END Synchronization
SET TIMER	4	0	s	14	714	3	I1→TIMER DATA

```
**********
* OP 14  *
*        *
* DCB=714*
**********
```

DMA COMMANDS

FUNCTION	T	M	J	OP	DCB	A	NOTES
READ MODCOMP STATUS	2	1		14	716	1	MODCOMP STATUS→DI
SEND CHI STATUS	3	0		14	716	2	I1→MODCOMP DATA INTERRUPT MODCOMP
START TRANSFER	2	1		14	716	3	Direction and transfer type determined by I1
SET WORD COUNT	3	0		14	716	4	I1→transfer counter
SET START ADDRESS	3	0		14	716	5	I1→starting address
READ WORD COUNT	2	1		14	716	6	Transfer counter→DI

```
**********
* OP 14  *
*        *
* DCB=716*
**********
```

TRANSFER CODE I_1 =
- 0 output 32 bit words
- 1 input 32 bit words
- 2 output 16 bit words
- 3 input 16 bit words

TABLE 15

AP-COMMANDS

FUNCTION	T	M	J	OP	DCBA	NOTES
CONNECT AP	2	1	s	14	7170	MP/AP directly coupled
DISCONNECT AP	0	0	s	14	7170	MP/AP operate independently
LOAD AP(n)	0	0	s	14	30n -1	n 32 bit words CD→DP;CA even
STORE AP(n)	0	0	s	14	40n -1	n 32 bit words DP→CD;CA even
PSA = ADDRESS	1	2	s	5	ADDRESS	START AP EXECUTION at DCBA
IF AC(22)=1	0	0	no	16	25yes	Test bit 22 of AP accumulator
IF AC(23)=1	0	0	no	16	26yes	Test bit 23 of AP accumulator
IF APDRDY=M	0	M	no	16	37yes	Test AP data ready
APDRDY=1	0	0	s	16	4700	Set AP data ready
IF S(3)=1	0	0	no	16	4403	Test for AP interrupt
APDRDY=0	0	1	s	16	4700	Clear AP data ready
IF APDRDY=0 APDRDY=1	0	0	no	16	77yes	Test AP data ready, set AP data ready
If APDRDY=1 APDRDY=0	0	1	no	16	77yes	Test AP data ready, clear AP data ready

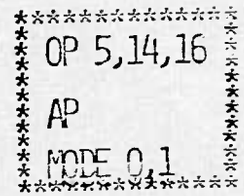
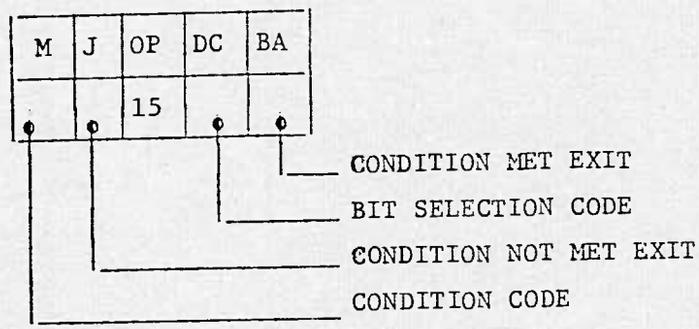


TABLE 16

BIT SELECTION CONDITIONALS



****
* OP 15 *
* MODE 0,1 *
* BIT TEST *
****

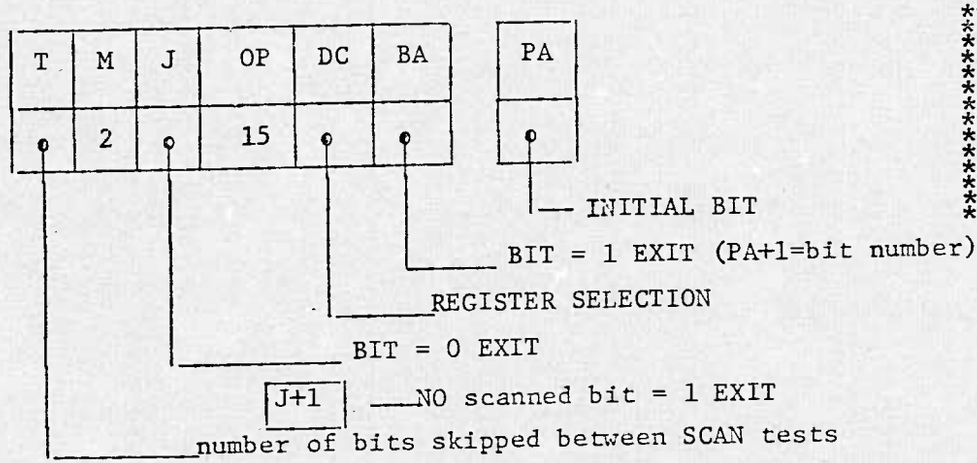
MODE 0 (CONDITION CODE = 0)

	D	C = 0	1	2	3	4	5	6	7
FR	0	IF I1F1=0	IF A1F1=0	IF A2F1=0	IF I2F1=0	IF M1F1=0	IF M2F1=0	IF PDF1=0	IF HF1=0
FL	1	IF I1F2=0	IF A1F2=0	IF A2F2=0	IF I2F2=0	IF M1F2=0	IF M2F2=0	IF PDF2=0	IF HF2=0
A2R	2	IF A2(0)=0	IF A2(1)=0	IF A2(2)=0	IF A2(3)=0	IF A2(4)=0	IF A2(5)=0	IF A2(6)=0	IF A2(7)=0
A2L	3	IF A2(10)=0	IF A2(11)=0	IF A2(12)=0	IF A2(13)=0	IF A2(14)=0	IF A2(15)=0	IF A2(16)=0	IF A2(17)=0
I1R	4	IF I1(0)=0	IF I1(1)=0	IF I1(2)=0	IF I1(3)=0	IF I1(4)=0	IF I1(5)=0	IF E1(6)=0	IF I1(7)=0
I1L	5	IF I1(10)=0	IF I1(11)=0	IF I1(12)=0	IF I1(13)=0	IF I1(14)=0	IF I1(15)=0	IF I1(16)=0	IF I1(17)=0
E1R	6	IF E1(0)=0	IF E1(1)=0	IF E1(2)=0	IF E1(3)=0	IF E1(4)=0	IF E1(5)=0	IF E1(6)=0	IF E1(7)=0
E1L	7	IF E1(10)=0	IF E1(11)=0	IF E1(12)=0	IF E1(13)=0	IF E1(14)=0	IF E1(15)=0	IF E1(16)=0	IF E1(17)=0

TABLE 17

SCAN FOR 1

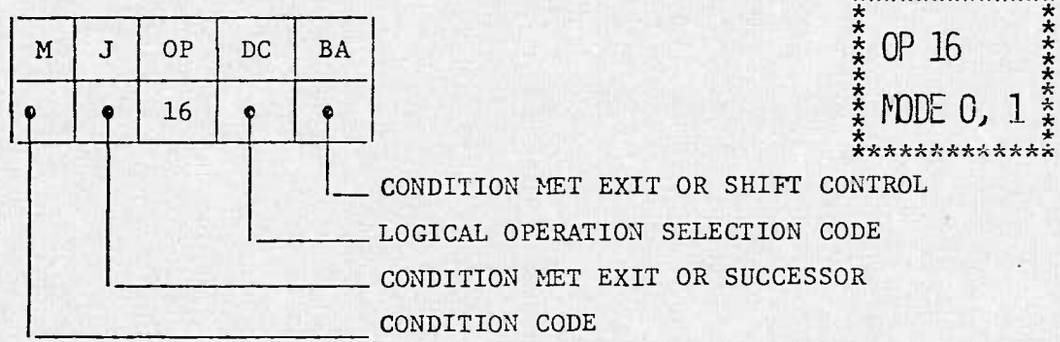
DC	REGISTER
00	F
20	A2
40	I1
60	E1



****
* OP 15 *
* MODE 2 *
* BIT SCAN *
****

TABLE 18

LOGICAL OPERATIONS



OP 16
MODE 0, 1

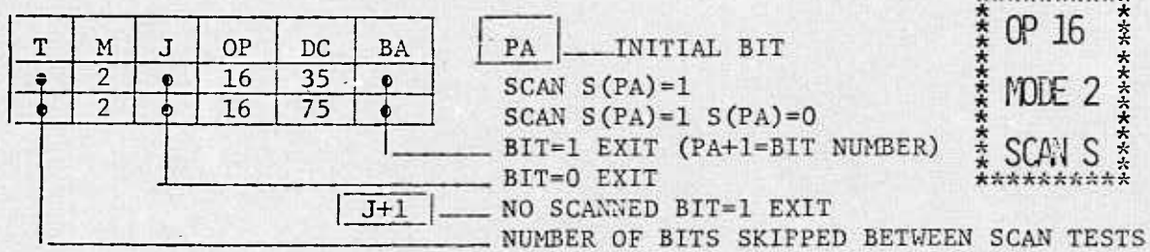
MODE 0

D \ C	0	1	2	3	4	5	6	7
0	IF OCT=0	IF OCT=1	IF OCT=2	IF OCT=3	IF OCT=4	IF OCT=5	IF OCT=6	IF OCT=7
1	IF S=0	IF E1≠0	IF PA=0	IF A2L=0	IF A1 NORM	IF I1 NORM	IF MS=0	IF A1(17)=0
2	NOOP	IF A1(10)=0	IF I2(17)=0	IF CF=0	IF OF=0	IF AC(22)=1	IF AC(23)=1	IF IODRDY=0
3	IF E3(10)=0	IF E3(11)=0	IF E3(12)=0	IF E3(13)=0	IF S(A)=0 GOTO J+1	IF S(PA)=0	IF D(7)=0	IF APDRDY=0
4	SHIFT(I)	SHIFT(II)	I1F=0	FLAGS(III)	S(A)=1	S(PA)=1	D(7)=1	APDRDY=1
5	NOOP	NOOP	NOOP	NOOP	IODRDY=1	F(BA)=1	MOVE E1R:S E1R=0	E1L=0
6	MOVE E1R:S	MOVE MP:I2 FLIP E1	NOOP	PD(IV)	E1=E2=0	NOOP	MERGE PDL,E1R:I1	MERGE E1L,PDR:I1
7	(3)	(3)	(3)	(3)	(3) and (4)	(3) and (4)	(3) and (4)	(3) and (4)

NOTES: 1. MODE 1 may be used to reverse the above conditionals
2. Refer to TABLES I, II, III, IV as indicated

TABLE 19

SCAN S



OP 16
MODE 2
SCAN S

TABLE I

REGISTER SHIFTS

T	M	J	OP	DC	B	A
•	0	•	16	40	•	•

A — SHIFT CODE FOR I1 or I2

B — SHIFT CODE FOR A1 or A2

J — SUCCESSOR

T — T+1 = NUMBER OF PLACES SHIFTED

CODE	B	A
0	NOOP	NOOP
1	LSL A1	LSL I1
2	ROT A1	ROT I1
3	LS A1	LS I1
	A1(0)=I1(17)	I1(0)=A1(17)
4	RSL A2	RSL I2
5	RSA A2	RSA I2
6	ROT A2	ROT I2
7	RS A2	RS I2
	A2(17)=I2(0)	I2(17)=A2(0)

OP 16
DC=40
SHIFT(I)

DOUBLE SHIFT

FUNCTION	OP	DCBA
DLSL A1,I1	16	4031
DLSL I1,A1	16	4013
DROT A1,I1	16	4033
DRSL A2,I2	16	4047
DRSL I2,A2	16	4074
DRSA A2,I2	16	4057
DRSA I2,A2	16	4075
DROT A2,I1	16	4077

TABLE II

FLAG SHIFTS

FUNCTION	OP	DCBA
LS A1 A1(0)=A1F1	16	4110
RS A2 A2(17)=CF	16	4140
LS I1 I1(0)=I1F1	16	4101
RS I2 I2(17)=I2F1	16	4104
ROT I1F	16	4102

OP 16
DC=41
SHIFT(II)

TABLE III

MOVE FLAGS

CODE	B	A
0	I1F	I1F
1	A1F	A1F
2	A2F	A2F
3	I2F	I2F
4	M1F	M1F
5	M2F	M2F
6	PDF	PDF
7	HF	HF

OP	DC	B	A
16	43	•	•

B — MOVE(B):I1F

A — MOVE I1F:(A)

OP 16
DC=43
FLAGS(III)

Examples:

FUNCTION	OP	DCBA
MOVE M1F:I1F	16	4340
MOVE I1F:A2F	16	4302
MOVE M1F:I1F:A2F	16	4342

TABLE IV

INTO PD

OP 16
DC=63
PD(IV)

FUNCTION	OP	DCBA
MOVE E2:PDR	16	6300
MOVE M1:PDR	16	6301
MOVE M2:PDR	16	6302
MOVE S:PDR	16	6303
MOVE CD:PD	16	6304
GATHER A2:PD	16	6305
MOVE E1:PD	16	6306
GATHER A1:PD	16	6307

TABLE 20

LINKAGE OPERATIONS

T	M	J	OP	DC	BA
1	0	•	17	•	00

SUBSTITUTION VALUE (V)

SUBSTITUTION ADDRESS (A)

```
**********
* OP 17  *
*       *
* MODE 0 *
**********
```

J(A) = V GOTO A+1 (IF CEX=0)
J(A) = V GOTO *+1 (IF CEX=1)

T	M	J	OP	DC	BA
1	1	•	17	•	•

SUCCESSOR (S)

SUBSTITUTION VALUE (V)

SUBSTITUTION ADDRESS (A)

```
**********
* OP 17  *
*       *
* MODE 1 *
**********
```

J(A) = V GOTO S

T	M	J	OP	DC	BA
1	2	•	17	•	•

PA VALUE (a)

SUBSTITUTION VALUE (V)

SUBSTITUTION ADDRESS (A)

```
**********
* OP 17  *
*       *
* MODE 2 *
**********
```

PA=a J(A)=V GOTO A+1 (IF CEX=0)
PA=a J(A)=V GOTO *+1 (IF CEX=1)

APPENDIX B

MICRO-OPERATION DEFINITIONS

Set CA Operations

CA = (VALUE)	Set CA register to given 16-bit value, without initiating a memory cycle.
READ(Address)	Set CA register to given 16-bit value, initiate a memory read cycle at that address.
READ SEQ(Address)	Set CA register to given 10-bit value, intiate memory reads for four consecutive doublewords, incrementing CA by 2 three times.
WRITE(Address)	Set CA register to given 16-bit value, initiate memory write cycle. I1 is written into CD at address.
EXEC(Address)	Set CA register to given 16-bit value, initiate read cycle. Set memory execute mode.
EXEC SEQ(Address)	Set CA register to given 16-bit value, initiate memory reads for four consecutive doublewords, incrementing CA by 2 three times. Set memory execute mode starting with first doubleword read.

Multiplier Micro-Operations

MOVE PDL:M1	PD(10-17)→M1, 0→M1F1
MOVE MPL:SM1	MP(10-17)→M1, MS→M1F1
MOVE I1R:M1	I1(0-7)→M1
MOVE PDR:SM1	PD(0-7)→M1, PD(17)→M1F1
MOVE PDL:SM1	PD(10-16)→M1(0-6), 0→M1(7),PD(17)→M1F1
SM1=0	0→M1, 0→M1F1
MULT PDR, A1R	PD(0-7)→M1, PD(17)→M[illegible], A1(0-7)→M2,A1(17)→M2F1
MOVE A1L;SM2	A1(10-17)→M2, A1F1→M2F[illegible]
DECIM E1:M2	E1(0-7)→M2(7-0)
MOVE I1R:M2	I1(0-7)→M2
MOVE A1R:SM2	A1(0-7)→M2, A1F1→M2F1
SM2=0	0→M2, 0→M2F1
MOVE MP:I2	MP→I2,MS→A2F1

Adder Micro-Operations

SM/2C A2	If A2F1=1, -A2→A2
SM/2C A1	If A1F1=1, -A1→A1
ADD	2's complement 16-bit SUM
SUBT	2's complement 16-bit DIFFERENCE
ADD	Bit by bit logical AND
OR	Bit by bit logical OR
OR'	Bit by bit logical NOT OR
NEG	2's complement negation
INC	2's complement add one
2c/SM A2	A2(17)→A2F1, IF A2(17)-1, -A2→A2
2c/SM A1	A1(17)→A1F1, IF A1(17)=1, -A1→A1
DEC	2's complement add minus one
IF	IF THE CONDITION SPECIFIED IS TRUE INCREMENT IA by 1.
EOR	Logical bit by bit exclusive or
'	Logical bit by bit negation
EOR'	Logical bit by bit NOT exclusive or
INC CA(2) M=1	FOR MODE=1 INC CA causes CA to be incremented by 2. Adder output is CA+1
NOOP	No adder operation specified in this field

Parallel Register Operations

reg=0	Clear named register
MOVE I1:A2:A1	MOVE A2→A1, MOVE I1→A2 each clock time
MOVE I2:E2	I2(0-7)→E2
MOVE PA:E2	PA→E2(0-5),0→E2(6-7)
FLIP E1	INTERCHANGE E1(10-17) and E1(0-7)
(PDI):E1	Transfer whatever is gated→PD in this instruction to E1 instead.
MOVE DI:E1	DI→E1, if D7=0, DI=0, T>0.
MOVE PDR:E1R	PD(0-7)→E1(0-7), E1(10-17) unchanged
EXCH E2:PA	PA→E2 and E2→PA
DEC OCT	PA-8→PA
INC OCT	PA+8→PA
(ADDFN)	Adder output as given by ADDFN Table
(ADDFN)'	Complemented adder output
WF=1	Set write request flip/flop, Next memory cycle will be a write
DBLWF=1	Set write 36-bit mode (AP-120 AC)
APDRDY=0	CLR AP-120 data ready bit, if connected
MOVE CD:I1	CD(0-17)→I1, CDF1→I1F1, CDF2→I1F2
MOVE DI:E1:L1	DI→E1, E1→I1, T>1
MOVE E1R→I1	E1(0-7)→I1(0-7), 0→I1(10-17)
MOVE E1L→I1	E1(10-17)→I1(10-17), 0→I1(0-7)
MOVE A1R→PDR	A1(0-7)→PD(0-7), PD(10-17) unchanged
MOVE A1L→PDL	A1(10-17)→PD(10-17),PD(0-7) unchanged
WC=1	0→I1F1, 0→I1F2, Set write request flip/flop
MOVE PDR:A1	0→A1(10-17), PD(0-7)→A1(0-7)
SCAT PD:A1	PD(0-16)→A1(0-16), 0→A1(17), PD(17)→A1F1
MOVE E2:I2	E2→I2(0-7), 0→I2(10-17)
MOVE I2:E2	I2(0-7)→E2
IF PA=0	If PA=0 INC IA by 1.
PSA=DCBA	Set AP-120 SA=DCBA field if D(3-7) = 36

APPENDIX 9

AP-90 Programming Manual

CHI

SIGNAL SYSTEM

TMA8

SOFTWARE

TMA8 - AP-90 PROGRAMMING MANUAL

JUNE 1, 1974

CONTENTS:

1. INTRODUCTION
2. PROGRAMMABLE COMPONENTS
3. INSTRUCTION SET
4. ASSEMBLER
5. LINK LOADER

CULLER-HARRISON INC.

REVISION LOG

REVISION	DATE	DESCRIPTION
A	7/14/75	pp. 4-1,4-2,4-3,4-6,4-7,4-8,4-10,4-11,4-12,4-14 5-1 thru 5-5
B	10/23/75	pp. 2-4,2-5,A-9,A-22

Address comments about this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication Number 29205

© 1973
by Culler/Harrison, Inc.

PREFACE

This document contains the information necessary for programming the AP-90 Array Processor in AP Assembly Language.

Chapter 1 gives a brief overview of the use of the AP-90. Chapter 2 describes the programmable components of the AP-90 and Chapter 3, along with Appendix A, gives the instruction set. The AP Assembler is described in Chapter 4. Chapter 5 lists the mnemonics used in the assembly language.

Address comments on this manual to:

Culler/Harrison, Inc.
150-A Aero Camino
Goleta, California 93017

Publication No. 29205
© 1973
by Culler/Harrison, Inc.

The logical design principles embodied in the AP-90 are protected under U.S. Patent No. 3,771,141.

Chapter 1. INTRODUCTION

The Culler/Harrison Array Processor, the AP-90, is a high-performance array/arithmetic processor intended to work in parallel with a host computer such as the Culler/Harrison MP-32 Macroprocessor.

Designed for signal processing applications such as speech research, sonar analysis and seismic studies, the MP-32 acts as a driver to the AP-90. Indeed, if the user has extreme system requirements for throughput, up to eight AP-90's may be driven by the MP-32 Macroprocessor.

The AP-90 has been designed to allow the host computer to request operations in two basic modes:

> Arithmetic Mode -- In this mode, the AP-90 is used as a peripheral arithmetic unit by the host computer; execution is tightly controlled by the host.
>
> Array Processing Mode -- In this mode, the host computer fills the data memory of the AP-90 with information to be processed, and turns control over to the AP-90. When finished, the AP-90 interrupts host processing and returns control.

The AP-90 hardware reflects these usage requirements with a unique design in which speed and flexibility are key elements. The AP-90 Assembler software provides a readily understandable mnemonic language for coding the machine instructions, along with a small but sufficient set of pseudo operations.

Subsequent chapters describe the programmable components of the AP-90 (Chapter 2), the instruction set (Chapter 3), and the Assembler and Assembler Design (Chapters 4 and 5).

Chapter 2. PROGRAMMABLE COMPONENTS

The programmable components of the AP-90 Array Processor are shown in Figure 1 and tabulated in Table 1.

2.1 PROGRAM STORAGE

The PS-ROM (Program Storage--Read-Only-Memory) consists of 512 to 1024 32-bit words (in increments of 256 words) containing hard-wired machine instructions organized into subroutines called operations. The elementary function operation set may reside in PS-ROM, along with the Fast Fourier Transform (FFT), Inverse FFT and other operations as defined by the customer.

The PS-RAM (Program Storage--Random-Access Memory) consists of 0, 256 or 512 32-bit words and is used for variable program storage. Both PS-ROM and PS-RAM are logically divided into 16-word pages. The 512-word Program Storage capacity is the minimum configuration, and can be expanded to 1536-word capacity.

The Command Buffer (CB) is the 32-bit register where instructions are decoded and executed. Instruction sequence is controlled through the Program Storage Address register (PSA), a 12-bit register which contains the address of the next instruction to be executed. PSA is usually incremented by one; however, the last 4 bits of PSA may be modifed by certain test instructions, allowing jumps to locations within the current page. Other instructions permit loading PSA with a full 12 bits, allowing jumps to any location in PS-ROM or PS-RAM.

The EXITS are 16 12-bit words which indicate addresses in PS-ROM or PS-RAM to which control is to be transferred. The EXITS are used as return addresses when a jump is made to a subroutine; return is made by these subroutines to the calling routine through these EXITS.

2.2 S-PAD, DATA PAD, AND ARITHMETIC REGISTERS

The Subroutine Parameters storage area, or S-Pad (SP), consists of 16 12-bit words of high-speed semiconductor random access memory. SP may be used as index registers for data memory address (SP can be added to MA), displacement registers for Data ROM (SP can be added to DRA), and tally registers (SP can be incremented, decremented, and tested for zero). SP is addressed directly from the instruction by specification of the SPA field (S-Pad Address).

Data Pad (DP) consists of 32 32-bit words of high-speed semiconductor random access memory, to be used as a readily accessible scratch pad. The low order 5 bits of the 12-bit Data Pad Address register (DPA) points to a word of the DP. One word of data pad may be used as one 32-bit floating point or integer value, or as two 16-bit fixed point values.

The 32-bit accumulator (AC) is the central register of the unit. AC acts as the accumulator with respect to DP, and the data input/output register with respect to the memory (MD), and host computer (HD).

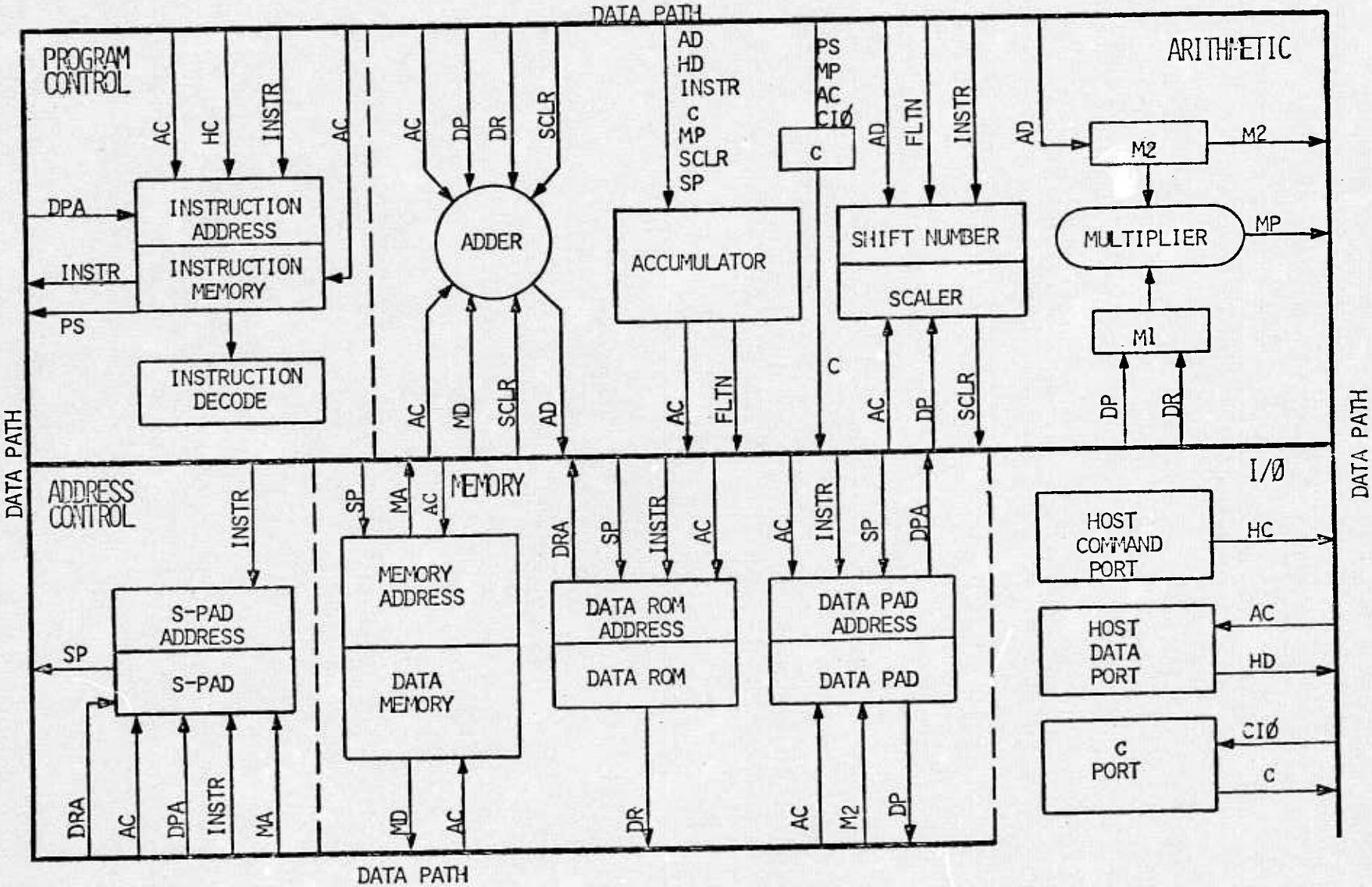


FIGURE 1 AP-90 BLOCK DIAGRAM (PROGRAMMABLE COMPONENTS)

Table 1. Programmable Components

Unit	Typical Usage	Type	Access/Cycle Time
AC	Accumulator for arithmetic operations	Bipolar-Schottky	--
AD	Fast adder	Bipolar-Schottky	--
C	Temporary Storage and Communication	Bipolar-Schottky	--
CB	Command buffer for instruction decoding and execution	Bipolar-Schottky	--
DP	Scratch pad	Bipolar	35ns
DR	Tables for transforms, filters, and fixed processes	Bipolar	60ns
EXITS	Settable exits for operations and subroutines	Bipolar	35ns
MD	Main memory, signal and transform buffer	MOS	500/500ns
MP	Multiplier	Bipolar	--
M1,M2	Input registers for Multiplier	Bipolar-Schottky	--
PS-RAM	Variable control programs and special operations	Bipolar	35ns
PS-ROM	Fixed micro programs and standard operations	Bipolar	60ns
SN,SCLR	Shift registers	Bipolar-Schottky	--
SP	Parameters and index registers for subroutines	Bipolar	35ns

The 32-bit communication register (C) is used in the arithmetic unit for temporary storage of data from AC and for transfer of the 32-bit product from the multiplier unit. It also provides a 16-bit communication path to the host computer.

The 32-bit adder (AD) is used primarily to combine DP and AC, and can be configured as two 16-bit adders for fixed point operations, as an 8-bit and a 24-bit adder for floating point operations or as a single 32-bit adder.

M1 and M2 are the multiplication input registers; the MP register performs the multiplication and holds the result. MP can form the 32-bit product of two 16-bit 2's complement numbers in 333 ns. It can form the 48-bit product of two 24-bit 2's complement numbers in 500 ns. In the second case, the rounded most significant half of the product is all that is retained. If a multiply is initiated in one instruction, the product should not be read until the second or third instruction after the initiation request.

SN and SCLR are special shift registers. SN holds an 8-bit shift number; the most significant bit gives the direction of the shift (0=right, 1=left) and the remaining bits give the magnitude of the shift. SCLR is the 27-bit register where the shift takes place (the three least significant bits are used for rounding). All shifts are end-off; on right shifts, the sign is extended.

2.3 DATA STORAGE

The Data Read-Only Memory (DR) consists of 2048, 3072 or 4096 32-bit words and is generally used to store tabular data for transforms, filters, fixed processes and to store diagnostics programs. DR is accessed through the 12-bit Data ROM Address Register (DRA). A read requires one clock for completion. This means that a DR value may be used by the next instruction after DRA is changed. Tables may be automatically indexed in modules of 128 or 256 words.

Data Memory (MD) consists of 4096 36-bit words of metal-oxide semiconductor (MOS) random access memory. MD is accessed through the 12-bit Memory Address register (MA). A read of MD is initiated whenever a new value is loaded into MA, unless a "no memory initiate" option is specified in the instruction, or a data write is requested from the accumulator. Both read and write operations require three clocks (500 ns) for completion. For a read, this means that an MD value may be retrieved and used no earlier than the third instruction after the read is initiated, and MA may not be changed during this period (unless the "no memory initiate" option is specified). An attempt to initiate a read or write operation less than three clocks after a previous memory initiate will cause an automatic pause in execution of subsequent instructions until the new memory initiate can take place.

2.4 DATA REPRESENTATIONS

Three data representations are used for data in the AP-90. These are 16-bit fixed point, 32-bit integer and 32-bit floating point. All forms use two's complement representation for negative numbers.

2.4.1 16-Bit Fixed Point

The number is treated as a fraction, with the least significant bit having a weight of 2^{-15}. Sixteen-bit fixed point numbers can be stored in either the left or right half of any 32-bit memory element. They may be added, subtracted or multiplied.

2.4.2 32-Bit Integer

An entire 32-bit word is used to represent at integer number. The weight of the least significant bit is one. Thirty-two bit integers are produced as the output of the integer multiply operation. They may be added together or subtracted from each other.

2.4.3 32-Bit Floating Point

The floating point representation is an 8-bit binary exponent followed by a 24-bit fractional mantissa. Both parts use two's complement representation for negative numbers. The least significant bit of the mantissa has a weighting of 2^{-23}. The exponent part is an integer representing the power of two by which the mantissa is multiplied to obtain the represented value. Floating point numbers are usually maintained in a normalized form. The normalized form has a mantissa with magnitude greater than or equal to one-half unless the number is identically zero, in which case the exponent is -128_{10}.

Chapter 3. INSTRUCTION SET

3.1 GENERAL DESCRIPTION

The instruction set for the AP-90 comprises fourteen basic instruction formats. Each of these formats offers one "primary" operation, plus up to six "secondary" operations which are all performed in parallel during one machine cycle except for the multiply, DR read, and MD read operations noted in Chapter 2. For each of the primary and secondary operations, the user may select from a variety of different manipulations (there are 15 different possibilities for multiplication, for example).*

A general description of the formats is given in Table 2; usually, the first operation listed for a format in this general description is regarded as the "primary" operation, although in some formats (such as 5 and 6) the distinction between primary and secondary operations is somewhat hazy. The format names are the assembly language format mnemonics.

Table 2. General Description of Formats

Format Name	Format #	Operations
SETDPR	A	1. Load right half of Data Pad with value given in instruction. 2. Various data transfer options.
SETREG1	B	1. Load a selected address register with value given in instruction. 2. Various data transfer options. 3. Initiate Data Memory cycle.
SETREG2	C	1. Load Program Storage or selected S-Pad register with contents of accumulator. 2. Various data transfer options. 3. Initiate Data Memory cycle. 4. Load Accumulator. 5. Perform fixed point add 6. Perform operation on Data Pad Address register.
SETTBL1	D	1. Access Data ROM for an entry in a table of type 1 (128 entries). 2. Various data transfer options. 3. Initiate Data Memory cycle.
SETTBL2	E	1. Access Data ROM for an entry in a table of type 2 (256 entries). 2. Various data transfer options. 3. Initiate Data Memory cycle.

*The computation of the total number of different 32-bit instructions which can be executed on the AP-90 is left as an exercise for the reader.

Format Name	Format #	Operations
SETSP	1	1. Load selected S-Pad register with value given in instruction. 2. Various data transfer options. 3. Initiate Data Memory cycle.
ACR→REG	2	1. Load selected register with contents of Accumulator. 2. Various shifts of selected S-pad register, other data transfer options. 3. Initiate Data Memory read cycle. 4. Load Accumulator. 5. Load Data Pad. 6. Perform fixed-point add. 7. Perform operation on Data Pad Address register.
ACR→SP	3	1. Transfer contents of Accumulator to selected S-Pad register. 2. Initiate Data Memory read cycle. 3. Load Accumulator. 4. Perform fixed-point add. 5. Perform operation on Data Pad Address register. 6. Initiate a 16 x 16 multiply.
SP→SP	4	1. Transfer data between selected S-Pad registers. 2. Various shifts of S-pad, other data transfers (7 options). 3. Initiate Data Memory read cycle. 4. Load Accumulator. 5. Load Data Pad. 6. Perform fixed-point add. 7. Perform operation on DPA.
TEST3	5	1. Perform conditional jumps within current 16 word page of Program Storage. 2. Initiate Data Memory read cycle. 3. Load Accumulator. 4. Load Data Pad. 5. Load the communication register. 6. Perform operation on Data Pad Address register.
SHIFT or TEST4 or FLAG	6	1. Shift Accumulator or current Data Pad word. 2. Special-purpose set/clear for FFT,IFT, host interrupt, etc. 3. Conditional jump within current page of Program Storage. 4. Load the communication register

Format Name	Format #	Operations
TESTO or FADD	7	1. Floating/fixed point add, including 32-bit option 2. Conditional jump within current page of Program Storage. 3. Various operations on Accumulator. 4. Load Data Pad. 5. Perform operation on Data Pad Address register. 6. Load the communication register
TEST1 or MULT	10	1. Floating/fixed point multiply. 2. Floating/fixed point add. 3. Conditional jump within current page of Program Storage. 4. Various data transfer operations. 5. Initiate Data Memory read cycle. 6. Load Accumulator. 7. Perform operation on Data Pad Address register.
TEST2	20	1. Conditional jump within current page of Program Storage. 2. Various data transfer operations. 3. Initiate Data Memory cycle. 4. Load Accumulator. 5. Load Data Pad. 6. Perform fixed-point add. 7. Perform operation on Data Pad Address register.

3.2 DETAILED DESCRIPTION

Detailed descriptions of the 14 different instruction formats are given in Appendix A. The format name and number are identified by headings on each page of each format description. Each description begins with the bit configuration of the instruction format. The symbol Ø indicates that a field is ignored. The variable fields of each format are described in the table which follows. The assembler code used to specify each field and the corresponding octal code are listed along with the resulting action and any restrictions which may apply to the field's specification. These restrictions are given in terms of values of other fields in the instruction. For example, the specification "XO≠3,4,5" in the Restrictions column would mean that that particular field value could not be specified when the octal code for the XO field in that instruction was 3, 4 or 5.

The following format conventions are used in the Resulting Action column. All programmable components are referred to by the abbreviations noted in Chapter 2. Parentheses around a symbol indicate "the contents of" a particular register or storage location. The symbol (AC) for example, refers to the contents of the Accumulator.

Subscripts on storage-area acronyms identify the particular word in the storage area to be used in an operation. DP_{DPA}, for example, would refer to

the word in Data Pad whose location is given by the current value of the Data Pad Address register. If a number is used as a subscript in an example discussion, it refers to a specific word; e.g., DP_{13} would refer to the 14th word in Data Pad. (Words are numbered in octal from 0 to N-1 in all storage areas where N is the total number of words in the storage area.)

Superscripts on a symbol refer to particular bit positions within a word or register. For example, AC^{0-15} would refer to bits 0 through 15 in the Accumulator. (Bits are numbered in decimal beginning with 0 as the least significant bit.)

In assembler code, the letters L and R generally means "left" and "right." For example, ACL would refer to the left portion of the Accumulator; this would mean AC^{16-31} for fixed point operations, and AC^{24-31} for floating point operations.

Arrows precede the destination in load and data transfer operations.

Chapter 4. ASSEMBLER

The AP Assembler consists of a translator which accepts a straight-forward mnemonic language and translates source code statements in this language to the AP machine instructions described in the previous chapter. The Assembler also offers a small set of pseudo operations allowing the user to direct certain aspects of the assembly process.

Some aspects of Assembly language coding are discussed in Appendix A; the basics of specifying AP instructions are noted and some examples of statements are shown. This chapter elaborates on the specification of instructions, and discusses the pseudo operations provided by the Assembler.

4.1 STATEMENT FORMAT

Two types of statement are recognized by the AP Assembler: Instruction Statements and Pseudo Operation Statements. Both types of statement are specified one statement per "line," where a "line" is defined in the particular system in use; e.g. as a series of one or more "words" terminated by the carriage return key in the SIGNAL edit system.

In general, a statement line appears as follows:

space or label or skip	Instruction-statement or Pseudo-operation-statement	"comments

4.1.1 Label

The AP assembler recognizes labels of one to nine alphanumeric characters. the first of which must be alphabetic. A symbol appearing in the label field is assigned the current program counter (PSA) value as its value. It may be referred to in JUMP, fetch or VALUE fields associated with the SETPSA, SETEXIT, CALLSUB or SETSP micro operations.

4.1.2 Instruction and Pseudo Op Statements

The AP Assembler translates each Instruction Statement to one 32-bit AP machine instruction to be executed in PS-RAM. Pseudo Operation Statements are directions to the AP Assembler itself; they are non-executable and are not translated to machine instructions.

Specific formatting rules for Instruction and Pseudo Operation Statements are discussed in Sections 4.2 and 4.3.

4.2 INSTRUCTION STATEMENTS

An instruction statement is coded as a format indicator followed by field indicators and the field value.

4.2.1 Format Indicators

The format indicator in an instruction statement tells the Assembler which of the fourteen types of instructions discussed in the previous chapter is desired. As such, it defines the combination of fields specifying operations which may be executed in parallel during one clock period.

Format indicators may be specified as:

format-name

or as: FMT=format-number

The format indicator must be the first non-blank word after the first word (label) in the line. It may have no spaces or SKIP characters within it.

The options available for specifying the format name and number are shown in the first two columns in Table 3. This table also illustrates the 32-bit machine instruction formats for each of the 14 types of instruction and the field names used in specifying field indicators as described in Sec. 4.2.2, below.

For example, the following format indicators would have the same effect:

```
FMT=6
SHIFT
FLAG
TEST4
```

4.2.2 Field Indicators

Following the format indicator, the user specifies valid field indicators for that format, and the values to be used for each field.

Generally, each field is defined by specifying:

field-name=value

where field-name is one of the alphanumeric names shown as field definitions in Table 3 and where the values which may be used for each field are as defined in the previous chapter and summarized in Table 4. The numeric values 0-7 shown in bit positions 27-29 and 30-31 of the instruction formats are fixed-field values supplied by the assembler and not field-names. The Ø values represent fields which have not yet been assigned; as such they are not field names. The "CODE" columns in Table 4 show the octal code substituted by the Assembler for the source code field name specified. For example, the field indicator X2 = AC→PS in a FMT=A instruction would cause the Assembler to provide the octal value 7 in bits 16-18 of the object code instruction.

Some fields listed in Table 3 have no corresponding values listed in Table 4. In these cases, the value is specified as either an octal number or a symbolic name. The format and limits of the specification of each of these fields are as follows:

Format	Field	Form	Range
SETDPR	VALUE	octal value or symbol**	0-177777*
SETREG1	VALUE	octal value or symbol**	0-7777
SETTBL1	VALUE	octal value only	0-37
SETTBL2	VALUE	octal value only	0-17
SETSP	VALUE	octal value or symbol**	0-7777
SHIFT	SHIFT	n must be octal	0-37
all	JUMP	symbol@ or *{+n / -n where n is octal	0-17
ALL	SPA,SPAR,SPAW	octal value only	0-17

* A symbol may assume values between 0 and 77777 only.

** Symbols defined in label fields are relocated when the program is linked for loading. It is not possible to use such symbols for DPR, DPA, MA or DRA value assignment. Symbols declared as GLOBAL and defined by EQU statements in the APLINK definition document may be used in these cases.

@ Any symbol used in a jump field must be defined as a label in the current assembly. *refers to the current value of the location counter (PSA).

Table 3. Field Definitions

FORMAT NAME	FORMAT NUMBER	31 30	29 28 27	26 25 24 23	22 21 20 19	18 17 16	15 14 13	12 11 10	9 8 7 6 5 4 3	2 1 0
SETDPR	A		0	DPR	SPA	X2	VALUE			
SETREG1	B		0	REG1	SPA	X2	MA	Ø	VALUE	
SETREG2	C		0	REG2	SPA	X2	MA	AC1	ADDER	DPA
SETTBL1	D		0	TBL1	SPA	X2	MA	Ø	VALUE	Ø
SETTBL2	E		0	TBL2	SPA	X2	MA	Ø	VALUE	Ø
SETSP	1		1	Ø	SPA	X2	MA	Ø	VALUE	
ACR→REG	2		2	REG	SPA	XO	MA	DP AC2	ADDER	DPA
ACR→SP	3		3	MULT	SPA	XO	MA	AC1	ADDER	DPA
SP→SP	4		4	SPAW	SPAR	XO	MA	DP AC2	ADDER	DPA
TEST3	5		5	JUMP	SPA	TEST3	MA	AC1	DP C Ø	DPA
TEST4 or SHIFT or FLAG	6		6	JUMP	Ø TEST4	F SC	STATUS	FLAG	Ø C SHIFT	
TESTO or FADD	7		7	JUMP	TESTO	F A	ADDI	ACO	DP C ADDIFN	DPA
TEST1 or MULT	10	1	TEST1	JUMP	MULT	F X1	MA	AC1	ADDER	DPA
TEST2	20	2	TEST2	JUMP	SPA	X2	MA	DP AC2	ADDER	DPA

Table 4. Field Selections

CODE	ADDER A,B INPUT	ADDERFN	DPA	MA	TEST1	TEST2	TEST3	TEST4	XO	X2
0	DP,AC	A-1	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP	NØ-ØP
1	DP',AC	A	INC	SP	AC→C	SP=0	JMP	JMP	AC→C,NØMI	NØMI,AC→C
2	AC,ACX	A+B	DEC	SP+1	DPALSC=7	SP≠0	AC22=1	DP22=1	SP→SP	WRT
3	0,ACX	A+B+1	DPA-8	SP-1	JMP	DPA≠7	AC23=1	DP23=1	MAFN→SP	MAFN→SP
4	DP,ACX	A-+B	SP	MA+1	DPA≠37	DPA≠37	DPA≠37	ADC23=1	MA→SP	MA→SP
5	ACL/DPR,ACR/0	A-B	DPA+SP	SP+8	HDRDY=0	HDRDY=0	HDRDY=0	MDAQ=1	SP*2→SP	DPA→SP
6	0,MD	A+-B	SWAPSP (DPA→SP)	SP+MA	IØDRDY=0	JMP	Ø	MDBQ=1	SP/4→SP	DPA+SP+1→DPA
7	DP'/DPR,ACL+1/ACR	A+1	WRT	DRA SP DP	IØDRDY=1	HDRDY=1	HDRDY=1	Ø	SP 8→SP	AC→PS

Field Selections (continued)

CODE	ACO	AC1	ADDI A,B INPUTS	ADDIFN FUNCTIONS	FLAG	MULT M1,M2	REG	TESTO
0	NØ-ØP	NØ-ØP	DP,AC	A-1	NØ-ØP	NØ-ØP	PSA	NØ-ØP
1	AD→AC	Ø	DP',AC	A	FTQ=1	M1,ADR	DPR	ØVFL=0
2	HD→AC	Ø	AC,AC	A+B	CLRFLGS	M1,ADL	DRA	ØVFR=0
3	C→AC	C	0,AC	A+B+1	IFTQ=1	DRM,ADR	MA	ADTEST=1
4	FP→AC	MP→ACL	SCLR,AC	Ø	FTP2Q=1	DPR,M2	DPA	JMP
5	RNDAC	AD→ACL	FPO,AC	A-B	FTP2Q=0	DPL,M2	SPA	AL=BL
6	FLØTAC	HD→ACL	FP1/2,AC	AL+BL+1,AR	SP→DRA	DRR,M2	NA	AL>BL
7	ADR→ACR	C→ACL	FP-1 ,AC	A+1	SPAX=11	DRL,M2	NA	AL<BL
10	ADL→ACL	MP→ACR	DP,MD		IØDRDY=0	DPR,ADR	CALLSUB	NA
11	SCLDPR→ACR	AD→ACR	DP',MD		IØDRDY=1	DPL,ADR	MP→C	NA
12	ADL→SN	HD→ACR	AC,MD		HDRDY=0	DRR,ADR	PS→C	NA
13	DSCL→SN	C→ACR	0,MD		HDRDY=1	DRL,ADR	NO-OP	NA
14	MP→ACR	MP→AC	Ø		SENDHI	DPR,ADL	Ø	NA
15	SP→SCL	AD→AC	DR,AC		MDOVF→MDS	DPL,ADL	LØADPS	AR=BR
16	SMAG→2C	HD→AC	Ø		DECIMATE	DRR,ADL	SCL→SP	AR>BR
17	2C→SMAG	C→AC	Ø		Ø	DRL,ADL	SETEXIT	AR<BR

CODE	AC2	DP	SC	X1
0	NØ-ØP	NØ-ØP	ACR	NØ-ØP
1	AD→AC	ACL→DPL	ACR,SN	MAFN→SP
2	HD→AC	ACR→DPR	DPR	MA→SP
3	C→AC	AC→DP	DPR,SN	M2→DPR

CODE	REG1
0	PSA
2	DRA
3	MA
4	DPA
10	CALLSUB
13	NØ-ØP
17	SETEXIT

CODE	REG2
11	MP→C
12	PS→C
15	LØADPS
14	NØ-ØP
16	SCL→SP

CODE	SHIFT
n	R:n
40+n	L:n

CODE	DPR
1	DPR

CODE	TBL1
6	TBL1

CODE	TBL2
7	TBL2

CODE	C
0	NO-OP
1	AC→C

CODE	A
0	NORMAL
1	DBLE

Field indicators may follow the format indicator in any order; each field indicator is recognized as a word and may have no imbedded blanks or SKIP characters. Field indicators may be separated from each other by spaces, SKIP characters or the semicolon (;) character.

If the user omits a field indicator which is valid for a particular format, that field is set to the "NØ-ØP" code for that field (see Table 4). The NØ-ØP code for the REG and REG1 fields is 13_8. For the REG2 field it is 13_8. For fields with no "NØ-ØP" code, the field value is set equal to 0.

For example, suppose the user wishes to set DPR to the value 13_8.

```
SETDPR   DPR=DPR  VALUE=13
```

In this example, the SPA and X2 fields, which are valid for this type of statement, have been omitted. The Assembler would form the following 32-bit machine instruction when translating this source statement:

31	26	22	18	15	0
0	1	0	0	13	

If the user specified:

```
SETDPR  DPR=DPR  VALUE=13  X2=DPA→SP  SPA=5
```

the Assembler would form the following 32-bit machine instruction:

31	26	22	18	15	0
0	1	5	5	13	

4.2.3 Short Form of Instruction

To eliminate the redundancy typified by the previous example, the AP Assembler allows shortening of the form of some instructions. This short form allows the user to specify the last example as simply:

```
SETDPR:13   X2=DPA→SP(5)
```

Table 5 shows the general format-indicator short form and the fields specifications so combined, along with an example of each, and a description of the action taken for each example. In the "Short Form" column of Table 5, underlined characters must be typed as shown. The lower-case characters of Table 5 give the name of the field (Table 3) for which the user may specify one of the values given in Table 4. These short forms can be used only as part of the instruction format-name specification. Only one of these forms may be used in an instruction.

Table 6 shows the short form field-indicators allowed by the AP Assembler. More than one of these forms may be used in the same instruction.

Examples of valid Assembly-language instructions are:

```
SETTBL1:3  X2=MA→SP(6)
ACR→REG:MA  ADDER=0,ACX:A+B  AC1=AD→AC
SP→SP  XO=SP(3)-8→SP(4)  AC2=AD→AC  ADDER=DP,AC:A-B
```

Table 5. Short Form Format Indicators

Short Form	Example	Action Taken
SETDPR:value	SETDPR:12	$12 \rightarrow DP_{DPA}^{0-15}$
SETREG1:reg1:value	SETREG1:PSA:13	13→PSA
SETREG2:reg2	SETREG2:LOADPS	$(AC) \rightarrow PS_{DPA}$
SETTBL1:value	SETTBL1:0	Defines first table of type 1 in DR memory
SETTBL2:value	SETTBL2:3	Defines fourth table of type 2 in DR memory
SETSP(spa):value	SETSP(5):13	$13 \rightarrow SP_5$
ACR→REG:reg	ACR→REG:PSA	$(AC^{0-11}) \rightarrow PSA$
ACR→SP(spa)	ACR→SP(12)	$(AC^{0-11}) \rightarrow SP_{12}$
SP(spa)→SP(spa)	SP(5)→SP(12)	$(SP_5) \rightarrow SP_{12}$
TEST3:test3	TEST3:JMP	Defines unconditional jump
SHIFT:shift	SHIFT:R:2	Shift register indicated by SC field right 2 bits
TESTO:testO	TESTO:AL=BL	Set ADTEST=1 if AL=BL
TEST1:test1	TEST1:DPA≠37	Jump if DPA≠37_8
TEST2:test2	TEST2:HDRDY=0	Jump if HDRDY=0
TEST4:test4	TEST4:JMP	Jump unconditionally
FADD:addi:addifn	FADD:DP,AC:A+B	Add DP and AC
MULT:mult	MULT:DPR,ADR	Set M1=DPR and M2=ADR
FLAG:flag	FLAG:ERROR=1	Set ERROR INDICATOR

Table 6. Short Field/Value Indicators

Short Form	Example
ADDER=a,b:adderfn	ADDER=DP,AC:A+B
ADDI=a,b:addifn	ADDI=AC,AC:A+B+1
SHIFT={R/L}:n	SHIFT=L:5
AC1={input:destination / input→destination}	AC1=MP:ACR AC1=MP→ACR
DPR=DPR:value	DPR=DPR:13
TBL1=TBL1:value	TBL1=TBL1:5
TBL2=TBL2:value	TBL2=TBL2:13
REG1=reg1:value	REG1=PSA:23
Wherever SP is specified in a field, parentheses may be used to enclose the address within SP to be used, eliminating the necessity to specify SPA, SPAR, or SPAW fields.	MAFN→SP(13) DPA+SP(3)+1→DPA SP(2)-8→SP(4) DRA+SP(17)→DRA

4.2.3.1 Extended Format Names - The following extended format names may be used in place of FMT=B or SETREG1 to provide both FMT and REG1 information to the assembler:

Extended Format	Equivalent
SETMA	SETREG1:MA
SETDPA	SETREG1:DPA
SETPSA	SETREG1:PSA
SETDRA	SETREG1:DRA
SETEXIT	SETREG1:SETEXIT
CALLSUB	SETREG1:CALLSUB

In each case, the VALUE field value may be concatenated onto the extended format indicator, e.g. CALLSUB:INVERSE.

4.2.4 Conflicting Field Values

As was noted in the previous chapter, some combinations of field values are not possible, even though the fields exist in the same instruction format. The assembler detects conflicting field values and produces an error message indicating the field names of the fields whose values conflict. Table 7 lists the possible conflicts.

Table 7. Field Value Conflicts

Formats	Conditions	Restrictions
B	REG1=MA,DPA	X2≠DPA+SP+1→DPA
B	REG1=DPA	MA=MA+1 only
B	REG1=DRA	MA≠DRA+SP→DRA
B	REG1=MA	No MA Field
C	REG2=SCL→SP	X2=NOMI,WRT or AC→PS only
D,E	SETTBL1,SETTBL2	MA≠DRA+SP→DRA
1	SETSP	X2=NOMI,WRT or AC→PS only
1,3		MA=MA+1 only
2	REG=SCL→SP	XO=NOMI only
2	REG=MA,DPA	XO=NOMI,MAFN→SP or MA→SP only DPA≠SP,DPA+SP or SWAPSP
2	REG=DRA	MA≠DRA+SP→DRA
2	REG=DPA	MA=MA+1 only
2	REG=MA	No MA field
2	REG=DPR	ADDER≠5,7;ADDER=0,1,4 only if ADR not used.
2,3,4	DPA=DPA+SP or SWAPSP	XO=NOMI only
2,3,4	DPA=SP	XO≠MAFN→SP or SP-8→SP
2,3,4,5,10,20	DPA=SP or SWAPSP	MA=SP or MA+1 only
2,3,4	MA=SP	XO≠SP-8→SP
2,3,4	MA=SP+1,SP-1,SP+8,MA+SP DRA+SP→DRA	XO=NOMI,MAFN→SP,MA→SP only
3	ACR→SP	XO=NOMI only

Table 7 continued.....

Formats	Conditions	Restrictions
4	DP≠0	ADDER≠0,1,4,5,7
5	DP≠0	MULT≠4,5,10,11,14,15
7	DP≠0	ADDI≠0,1,10,11
10	DPA=SP	X1≠MAFN→SP
10	DPA=DPA+SP or SWAPSP	X1≠MAFN→SP or MA→SP
10	X1=M2→DPR	ADDER≠5,7;ADDER=0,1,4 only if ADR not used.
10	X1=M2→DPR	MULT≠4,10,14
20	DPA=INC,DEC,DPA-8	X2≠DPA+SP+1→DPA
20	DPA=SP	X2≠MAFN→SP or DPA+SP+1→DPA
20	DPA=DPA+SP or SWAPSP	X2=NOMI,WRT or AC→PS only
20	DP≠0	ADDER≠0,1,4,5,7

4.2.5 Exceptions to the Normal Field Values

There are five cases when the MAFN→SP value for the X0,X1 or X2 field is replaced by a different function. These are enumerated below:

	Condition	Special Field Value
a.	FMT=ACR→REG, and REG=MA or DPA	ACR→SP
b.	FMT=SETREG1, and REG1=MA or DPA	VALUE→SP
c.	FMT=SETDPR	SP+VALUE→SP
d.	MA=MA+1 or NØ-ØP and FMT=SETTBL1,SETTBL2,SETREG2 or SETREG1 with REG1=DRA or PSA	SP+VALUE→SP
e.	MA=MA+1 or NØ-ØP and none of the above conditions true.	SP+ACR→SP

Note: The right 12 bits of the instruction are used for VALUE.

```
Example:  SETREG1:NØ-ØP     X2=SP+VALUE→SP(3)  VALUE=7    "Adds 7 to SP(3)
          SETMA:2000        X2=VALUE→SP(5)     "Sets MA & SP(5) to 2000
```

4.3 PSEUDO OPERATION STATEMENTS

4.3.1 Header Statement

The header statement is used to name the operation performed by the assembly module. It must appear first in the text document. The only field which should be coded is the label field. The operation name must occur in the label field, and may contain any characters or operator keys except space, SKIP, (,), ; or :. This operation name is presently not used, but must be present.

4.3.2 GLOBAL Statement

This statement is used to declare which symbolic variables defined or referenced in this module are to be available for reference or are defined in other modules. All symbolic names appearing as labels on an instruction statement are considered to be defined in the module. If the same names appear in a GLOBAL statement, they are treated as entry points, whose values will be made available to other modules. All other symbols appearing in a GLOBAL statement are considered to be external references, defined outside of this module. External references may be entry points in other AP programs or symbolic constants whose values are assigned in APLINK EQU statements.

Global statements, if they are used, must precede any instruction statements and follow immediately after the header statement. They should not be labeled.

4.3.3 END Statement

The END statement must be specified as the last statement in an assembly program; it indicates to the Assembler that the end of the assembly module has been reached. The format is simply:

```
END
```

The code END is recognized as the second word of a statement line and may have no imbedded blanks or SKIP characters.

4.3.4 Comments

Comments may be specified anywhere on a line except as the field value in an instruction. A comment is indicated by a quote symbol (") and indicates that the remainder of the line is to be ignored by the Assembler. They should be separated from the last field specification, if any, by spaces or SKIP.

However, the characters following the quote are stored as part of the text line to serve as documentation.

For example:

```
PgOLocO  SP(12)→SP(13)  "Exchange S-Pad values

"The following instructions determine the transpose of

"the matrix AMATRIX.
```

4.4 INVOKING THE ASSEMBLER IN THE "SIGNAL" SYSTEM

The Assembler translates one text document written in the Assembler language into machine instructions which are stored in a program library document. The assembler is invoked from LIBE mode by typing:

(OP) APASM textdoc:pgmdoc (↵)

The text document (textdoc) named must exist as a member of the current text library. The program document named must be declared as a member of the current program library. The translated machine instructions are placed in the named program document together with alignment, global symbols, and the length for the complete program module. If any errors are detected during the assembly, an appropriate error message (see Table 8) and the text line in which the error occurred will be displayed. Processing will generally continue in spite of errors with the field in error being filled with 0.

An assembled program module may be linked and optionally loaded into the PS-RAM by using the APLINK operator.

4.5 ASSEMBLER OUTPUT

The output of the AP assembler is made up of two parts. An AP program module, containing the AP instructions in relocateable format, the global symbol table, the possible starting positions within a program store page for this program and the program's length, is written in the current program library in the document specified. A listing is output on the console display with the following information:

1. For each error detected, an error message, the document containing the error, and the line containing the error.

2. The alignment word, an octal number with a one in each bit position corresponding to an allowable starting location for the program on a AP program store page.

3. The length of the program.

4. A Symbol Table, listing all symbols referenced or defined in the program. If a symbol was defined, its relative location within the program is given. If the symbol was referenced, but not defined, either GLOBAL, if it was declared as a GLOBAL symbol, or LABEL UNDEFINED, if it was not, is printed.

5. The document name given as the destination document is printed.

Table 8. AP Assembler Messages

1. AP ØPERATOR STORED AS DOCUMENT docname

   The AP assembler has completed its task. If no other messages appear, no errors were detected. A listing of the relative values of all symbols defined will be printed.

2. NØ ASSEMBLY

   The assembler was unable to complete its task. The immediate preceding message gives the cause.

3. NØ TEXT NAME GIVEN

   No source text document name was specified in the APASM command.

4. NØ TEXT LIBRARY SPECIFIED

   A current text library has not been declared. Use the |LIBE| operator to specify one.

5. NØ SUCH TEXT ITEM IN CURRENT LIBRARY

   One of the text documents named in the APASM command does not exist in the current text library.

6. NØ PGM NAME GIVEN

   No destination program document name was specified in the APASM command.

7. NØ PRØGRAM LIBRARY SPECIFIED

   A current program library has not been declared. Use the |LIBE| operator to specify one.

8. NØ SUCH PRØGRAM IN CURRENT LIBRARY

   The destination program document named in the APASM command has not been declared in the current program library. Use the |INSRT| operator to declare it in the current library.

9. PGM LIBRARY FULL

   No space is available on the system pack. Delete any unneeded temporary libraries and repeat the assembly.

10. END ØF TEXT ENCØUNTERED BEFØRE END STATEMENT

    No |END| pseudo-operator was found in a source text document being assembled. The assembler provides the missing |END| statement after all other statements in the document.

Each of the error messages below also includes a listing of the source document name and the line being processed when the error was detected.

11. PRØGRAM TØØ LØNG

    The length of the assembled program exceeds 255 words.

12. INVALID ØRG STATEMENT

    ØRG statements are not processed by the AP assembler.

13. INVALID FØRMAT

    The first field specification was not a valid format name.

14. fname, INVALID FIELD NAME

    'fname' is not a recognized field name.

15. fname, INVALID FØR FØRMAT

    'fname' is not a valid name for the format specified for this micro instruction.

16. fname, FIELD VALUE INVALID

    The field value specified for field 'fname' is not valid. See table 4 and section 4.2.5 for the permitted values for this field.

17. fname, ØCTAL FIELD VALUE INVALID

    Either the field 'fname' does not take on octal value or the value specified was out of range.

18. fname, FIELD HAS INVALID USE OF PARENTHESIS

    A parenthesis was used incorrectly in the specification of field 'fname.' Parentheses are used only for specifying S-PAD addresses in the form: SP(spa).

19. fname, FIELD HAS INVALID RIGHT PARENTHESIS

    A right parenthesis which was not preceded by a left parenthesis was encountered in the specification of field 'fname.'

20. fname, FIELD HAS INVALID USE OF CØLØN

    A colon was used to terminate a field value specified for 'fname' where a short form is not permitted. See section 4.2.3 for valid short forms.

21. fname ,fname ,..., FIELDS HAVE CØNFLICTING VALUES

    The combination of field values specified for the fields named is not possible due to conflicts in the usage of internal system elements.

22. →DP, DP usage CØNFLICT

    Data pad was specified as both a source and destination in the same instruction. This is permitted only when one half of a data pad cell is the source and the other half is the destination.

23. JUMP DISTANCE >17

    A jump field value was specified which required a jump of more than 17_8 instructions.

24. JUMP CANNOT BE RESOLVED....

    It is impossible to align the program on pages of PS in such a way that all jumps will be to locations on the same page as the instruction containing the jump.

25. DUPLICATE LABEL

    The same symbolic name was used to label more than one statement.

26. sname LABEL UNDEFINED

    The symbolic name, sname, was referenced but did not appear as a statement label or in a GLOBAL statement.

Chapter 5. AP PROCESS LINKER

The AP Process Linker (APLINK) is a simple editor which combines one or more AP programs to produce an AP process module ready to be loaded into the array processor program store and executed. The linking is controlled by a document specifying the set of AP programs to be combined, the location in AP-PS where they will reside, and the values of any symbolic constants used in the AP programs. The linker will collect the AP programs specified, plus any additional AP programs referenced in these programs, and create an AP process in the program library. The AP process may be combined with MP programs through the MP assembler INCLUDE pseudo operation.

5.1 INPUT

Input to the AP process linker is obtained from two sources. The process definition document is a member of the current text library. The AP programs are members of the current program library which have previously been output by the AP assembler described in Chapter 4.

5.2 OUTPUT

The output of APLINK includes the AP process document, written in the current program library member named, and the console display listing. The AP process document contains the composite global symbol table for the process and the object code itself, including the length and load address in AP program store. The listing includes error messages (listed in Table 9), all gaps created when successive AP programs cannot be loaded contiguously because of page alignment restrictions, a global symbol table describing the name and value of each global symbol, and the base and length of the process module.

5.3 THE PROCESS DEFINITION DOCUMENT

The AP process definition document describes the AP programs to be included in the process and their order, defines symbolic constants and specifies the location in program store where they are to reside.

5.3.1 Statement Format

Two types of information statements are used in process definition documents. These are the AP program statement and the control statement. In addition, comment statements may be freely used to annotate the document. Comment statements must have a " character in column one, the rest of the statement is ignored.

5.3.2 AP Program Statement

An AP program statement names an AP program document in the current program library which is to be included in the process. The document is loaded and edited to produce an image of its execution time form in program store. The name of the program document is the second "word" on the statement. All other information on this statement is ignored. Program documents cannot be named ORG, EQU or END.

5.3.3 Control Statements

A control statement has up to three fields. These fields are the label, operation and value fields. The label field must begin in column one. Fields are separated by spaces or SKIP. Three control operations are recognized: ORG, EQU and END.

5.3.3.1 Define the Location Counter Value - ORG: The ORG statement has no label field, the value field is an absolute octal value which is to be used as the new PSA base for subsequent AP programs. An ORG statement should be used as the first non-comment statement of a definition document and may be used any number of times to establish known starting locations for AP programs. Each PSA value so specified must be greater than or equal to the current PSA value resulting from previous ORG statements and any program documents processed.

5.3.3.2 Define a Symbolic Constant - EQU: The label field of an EQU statement contains a symbol of one to nine alphanumeric characters, the first of which must be alphabetic. This symbol is assigned the octal value given in the value field. The value can range from 0 to 77777_8. This value is used to resolve all global references to the symbol occurring in AP programs. If the same symbol appears in more than one EQU statement, the value used in resolving a reference in an AP program shall be that assigned in the last preceding EQU statement, or the first following EQU statement defining it, if there were none preceding.

5.3.3.3 Terminate the Process Definition - END: The END statement has no label or value field. No further statements are processed from the definition document. Instead, the global symbol table is searched in alphabetical order for any undefined global references. If one is found, the program library is searched for an AP program document with that name. If one is found, it is added to the process module being constructed. The search of the symbol table continues until all undefined symbols have been examined. The process is then written in the program library under the document name specified along with the global symbol table. The AP process linker then terminates.

5.4 INVOKING THE ARRAY PROCESSOR PROCESS LINKER

The AP process linker is invoked from LIBE mode by entering:

(OP) APLINK defndoc[:pgmdoc] (↵)

defndoc is the name of a document in the current test library containing the definition statements for the process. pgmdoc is the name of an entry in the current program library where the output process is to be stored. If no :pgmdoc is specified, the process will be loaded into the array processor program store, but not executed or otherwise saved. This permits manual initiation (using (OP) SETPSA) or control panel debugging.

5.5 USING THE AP PROCESS LINKER

APLINK is designed to permit the combination of general-purpose, parameterized AP programs into a complete AP process. If AP programs are written to accept all parameters as global symbols, the data definition document can be written as a higher level control program to order the sequencing from program to program. AP programs are assigned PS locations in the order they are specified in the program statements. When a program cannot start on the next available word, a SETPSA instruction is inserted in that open word pointing to the actual start of the program. Thus, if programs are written to "fall through" at their termination, the next program will always be started with its first instruction. Parameters, including counts in S-pad, DPA and MA values, and even positive data pad values, can be specified using global symbols whose values are given in the definition document with EQU statements. If more than one process is to be resident in program store at one time, references to programs in other processes are possible using EQU to resolve program locations.

5.6 EXAMPLE OF A DEFINITION DOCUMENT

"AN3 - This is part three of an analysis process

```
         ORG      1500   "This part starts at PSA=1500
FLOAT    EQU      1461   "Address of subroutine in another process
INVERSE  EQU      1156   "This is the inverse subroutine address
PPICK    EQU      1236   "Same for a third subroutine
IFM1     EQU      2677   "Starting MA-1 for input data
NZC      EQU      2677   "Address of common variable
IZM      EQU      140    "A fixed parameter for the process
         PITCH           "This uses IFM1, NZC, IZM and calls PPICK
POUT     EQU      2040   "Address of output buffer for PEAKFIT
         PEAKFIT         "Interpolates about peak found by PITCH
                         "Uses INVERSE and FLOAT
AOUT     EQU      2040   "Parameter for AOUT
         OUTDATA         "Standard interrupt MP and send data program
         END
```

Table 9. AP Process Linker Messages

1. n OPEN WORDS AT psa

The next AP program could not be started at the first available location in program source. A SETPSA *+n-1 instruction was inserted at this location, which is given as psa in the message. The actual load point of the new program is n-1 words after the SETPSA.

2. pgmname IS NOT AN AP ASSEMBLY MODULE

pgmname appears in a program statement of the definition document, but the document with that name in the current program library is not an AP assembly module or no document with that name exits.

3. NO TEXT NAME GIVEN

The source definition document name is not given in the APLINK statement.

4. NO SUCH TEXT ITEM IN CURRENT LIBRARY

The source definition document does not exist in the current text library.

5. PROGRAM TOO LONG

The AP process being defined exceeds 1798 words or PSA 7777_8.

6. DUPLICATE LABEL

A symbol is defined in the symbol table of an AP program which already has a value assigned to it.

7. ORG VALUE BAD

The value field of an ORG statement was missing or not an octal number greater than the current PSA and less than 7777_8.

8. EQUATE ERROR

No octal value was given.

9. END OF TEXT ENCOUNTERED BEFORE END STATEMENT

No END statement appeared in the definition document. A END statement is generated internally and processing continues with that statement.

10. NO PGM NAME GIVEN

The APLINK statement ended in : without a program document name.

11. NO SUCH PGN IN CURRENT LIBRARY

The current program library does not contain an entry for the destination AP process document.

12. NO TEXT LIBRARY SPECIFIED

The user has no current text library.

13. NO PGM LIBRARY SPECIFIED

The user has no current program library.

14. LINKAGE CANCELLED

The preceding error prevented continuation of the APLINK program. No output was written in the program library.

15. GLOBAL SYMBOL TABLE

The composite global symbol table, giving the symbolic name and final value of each global symbol, is listed.

16. symbol CANNOT BE FOUND!

The global symbol given was not resolved, and no program giving its definition could be found in the program library.

17. LINKAGE COMPLETE - BASE = psa, Length = nnn

The AP process has been created. Its base address in program source and its total length are given.

18. AP LOADED

No destination document was specified, so the AP process was loaded directly into the array processor program source memory.

APPENDIX A

FIELD SELECTION DESCRIPTION

31 27	23	19	16	15 0
0	DPR	SPA	X2	VALUE

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DPR	1		Indicates DP_{DPA}^{0-15} is to receive CB^{0-15}; i.e., Data Pad Right receives VALUE	
SPA	0-17	0-17	Indicates location in S-Pad to be used when X2-3, 4, 5, or 6.	
X2	0	NØ-ØP	or No operation or field omitted	
	1	AC→C	(AC)→C	
	2	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur, instead of data read from MD, for next instruction which executes a "set MA" operation.	
	3	VALUE+SP→SP	VALUE+$(SP)_{SPA}$→SP_{SPA}	
	4	MA→SP	(MA)→SP_{SPA}	
	5	DPA→SP	(DPA)→SP_{SPA}	
	6	DPA+SP+1→DPA	(DPA)+(SP_{SPA})+1→DPA	
	7	AC→PS	(AC)→PS_{DPA}: note Data Pad Address register points to location in PS.	
VALUE	0-177777	0-177777	Value to be stored into DP_{DPA}^{0-15}	

Example:

SETDPR X2=DPA+SP(13)+1→DPA VALUE=1000

The above instruction would cause the value 1000_8 to be stored into data pad at the address indicated by the "old" contents of DPA. The value would be stored in the right half of this location, i.e...., bits 0-15. In parallel with this operation, DPA would be updated to contain the sum of its "old" contents plus the contents of SP_{13_8} plus 1.

31 27	23	19	16	13	12	0
0	REG1	SPA	X2	MA	Ø	VALUE

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
REG1	0	PSA	$(CB^{0-11})\rightarrow PSA$; i.e., jump to VALUE address	
	2	DRA	$(CB^{0-11})\rightarrow DRA$; initiates DR read (see Sec. 2.3)	X2≠6
	3	MA	$(CB^{0-11})\rightarrow MA$; initiates MD read (see Sec. 2.3)	X2≠6 MA≠1-7
	4	DPA	$(CB^{0-11})\rightarrow DPA$	X2≠6 MA≠1,2,3, 5,6,7
	10	CALLSUB	$(PSA)+1\rightarrow EXIT_{DPA}$, $(CB^{0-11})\rightarrow PSA$; i.e., return jump	
	17	SETEXIT	$(CB^{0-11})\rightarrow EXIT_{DPA}$	
SPA	0-17	0-17	Indicates location in S-Pad to be set or used when X2=3,4,5 or MA=1,2,3,5,6,7	
X2	0	NØ-ØP or omit	No operation	
	1	AC→C	$(AC)\rightarrow C$	
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read is initiated.	
	2	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	
	3	VALUE+SP→SP	$(CB^{0-11})+(SP_{SPA})\rightarrow SP_{SPA}$	MA=0,4; REG1≠3,4
	3	VALUE→SP	$(CB^{0-11})\rightarrow SP_{SPA}$	REG1=3,4;MA=0,4
	3	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPA}, rather than MA or DRA	REG1≠3,4;MA≠0,4

Field	Octal Code	Assembler Code	Resulting Action/Indication		Restrictions
	4	MA→SP	$(MA)\rightarrow SP_{SPA}$		
	5	DPA→SP	$(DPA)\rightarrow SP_{SPA}$		
	6	DPA+SP+1→DPA	$(DPA)+(SP_{SPA})+1\rightarrow DPA$		REG1≠3,4
	7	AC→PS	$(AC)\rightarrow PS_{DPA}$: note that Data Pad Address register points to location in PS.		
MA	0	NØ-ØP or omit	No operation		
	1	SP	$(SP_{SPA})\rightarrow MA$		REG1≠3,4;
	2	SP+1	$(SP_{SPA})+1\rightarrow MA$	initiates MD read (see Sec.2.3)	REG1≠3,4
	3	SP-1	$(SP_{SPA})-1$ MA	unless X2=1,2 or 3	REG1≠3,4
	4	MA+1	$(MA)+1\rightarrow MA$		REG1≠3
	5	SP+8	$(SP_{SPA})+8\rightarrow MA$		REG1≠3,4
	6	SP+MA	$(SP_{SPA})+(MA)\rightarrow MA$		REG1≠3,4
	7	DRA+SP→DRA	$(DRA)+(SP_{SPA})\rightarrow DRA$	initiates DR read (see Sec. 2.3)	REG1≠2,3,4
VALUE	0-7777	0-7777	Value to be stored into register indicated by REG1 field		

Example:

SETREG1:PSA:31 X2=WRT MA=SP(15)

This instruction would set PSA=31, effecting a jump to PS_{31}. In parallel with this operation, a data write to MD would be initiated, causing the contents of the accumulator (AC) to be written to the MD address indicated by the contents of S-Pad location 15.

31 27	23	19	16	13	9	3	0
0	REG2	SPA	X2	MA	AC1	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
REG2	11	MP→C	$(MP^{0-31})\rightarrow C$	X2≠1
	12	PS→C	$(PS_{DPA})\rightarrow C$	X2≠1,7
	14	NØ-ØP	No Operation	
	15	LOADPS	$(AC)\rightarrow PS_{DPA}$	
	16	SCL→SP	$(AC^{24-31})\rightarrow SP_{SPA}^{0-7}$; i.e., the most significant 8 bits of AC (the scale field in a floating point value) is stored in the least significant 8 bits of a word in S-Pad	X2≠3,4,5,6
SPA	0-17	0-17	Indicates location in S-Pad to be set or used when REG2=16; X2=3,4,5,6; MA=1,2,3,5,6,7	
X2	0	NØ-ØP or omit	No operation	
	1	AC→C	$(AC)\rightarrow C$	REG2≠11
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated.	
	2	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applied to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	
	3	VALUE[1]+SP→SP	$(CB^{0-11})+(SP_{SPA})\rightarrow SP_{SPA}$	MA=4; REG2≠16
	3	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPA}, rather than MA or DRA	REG2≠16; MA≠1,4
	4	MA→SP	$(MA)\rightarrow SP_{SPA}$	REG2≠16
	5	DPA→SP	$(DPA)\rightarrow SP_{SPA}$	REG2≠16
	6	DPA+SP+1→DPA	$(DPA)+(SP_{SPA})+1\rightarrow DPA$	REG2≠16
	7	AC→PS	$(AC)\rightarrow PS_{DPA}$	REG2≠11

[1]VALUE is the contents of bits 11-0 of the instruction.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
MA	0	NØ-ØP or omit	No operation	
	1	SP	$(SP_{SPA})\rightarrow MA$	
	2	SP+1	$(SP_{SPA})+1\rightarrow MA$	
	3	SP-1	$(SP_{SPA})-1\rightarrow MA$	
	4	MA+1	$(MA)+1\rightarrow MA$	initiates MD read (see Sec. 2.3)
	5	SP+8	$(SP_{SPA})+8\rightarrow MA$	
	6	SP+MA	$(SP_{SPA})+(MA)\rightarrow MA$	
	7	DRA+SP→DRA	$(DRA)+(SP_{SPA})\rightarrow DRA$	initiates DR read (see Sec. 2.3)
AC1		General form is input→destination, where input =		
	0	MP	$(MP)^{15-30}$ destination	
	1	AD	(AD)→destination	
	2	HD	(HD--host computer data)→destination, CLR HDRDY	
	3	C	C destination, CLR IØDRDY	
		and where destination =		
	0	NØ-ØP	No transfer	
	1	ACL	$input\rightarrow AC^{16-31}$	
	2	ACR	$input\rightarrow AC^{0-15}$	
	3	AC	input→AC	
ADDER		General form is A-input,B-input: function, where A-input,B-input =		
	0	DP,AC	(DP_{DPA}),(AC) used in function defined	
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	0,(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA}),(AC)-cross used in function defined	
	5	ACL/DPR,ACR/0	(AC^{16-31}),(AC^{0-15}) used in function defined, result into AD^{16-31}; (DP_{DPA}^{0-15}), 0 used in function defined, result into AD^{0-15}.	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	6	0,MD	0,(MD_{MA}) used in function defined. This option is used to retrive memory data for which a read memory cycle was initiated at least four clocks earlier (500 ns).	
	7	DP'/DPR,ACL+1/ ACR	1's complement of (DP_{DPA}),(AC$^{16-31}$) +1 used in function defined, result in AD$^{16-31}$; (DP^{0-15}_{DPA}),(AC$^{0-15}$) used in function defined, result in AD$^{0-15}$.	
		and where function =		
	0	A-1	(A-input specified)-1→AD	
	1	A	(A-input specified)→AD	
	2	A+B	(A-input specified)+(B-input specified)→AD	
	3	A+B+1	(A-input specified)+(B-input specified)+1→AD	
	4	A-+B	(A-input$^{16-31}$)-(B-input$^{16-31}$)→AD$^{16-31}$ (A-input$^{0-15}$)+(B-input$^{0-15}$)→AD$^{0-15}$	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	(A-input$^{16-31}$)+(B-input$^{16-31}$)→AD$^{16-31}$ (A-input$^{0-15}$)-(B-input$^{0-15}$)→AD$^{0-15}$	
	7	A+1	(A-input)+1→AD	
DPA	0	NØ-ØP or omit	No operation	
	1	INC	(DPA)+1→DPA	X2≠6
	2	DEC	(DPA)-1→DPA	X2≠6
	3	DPA-8	(DPA)-8→DPA	X2≠6
	4	SP	(SP_{SPA})→DPA	MA≠2,3; 5-7 X2≠6
	5	DPA+SP	(DPA)+(SP_{SPA})→DPA	X2≠3-6
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	MA≠2,3,5,6,7; X2≠3-6

*(AC)-cross means the two 16-bit fields of AC are reversed; i.e., (AC$^{0-15}$)→B$^{16-31}$ and (AC$^{16-31}$)→B$^{0-15}$.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DPA	7	WRT	Indicates that a data write of $(AC) \rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

SETREG2:MP→C X2=MAFN→SP MA=SP(13)+1 ADDER=DP,AC:A

ACJ=AD→ACL DPA=INC

This instruction would cause the following operations:

$(MP)^{0-31} \rightarrow C$

$SP_{13}+1 \rightarrow SP_{13}$

$(DP_{DPA})^{16-31} \rightarrow AC^{16-31}$

$(DPA)+1 \rightarrow DPA$

These operations would be executed simultaneously.

31 27	23	19	16	13	12 7	0
0	TBL1	SPA	X2	MA	Ø	VALUE	Ø

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
TBL1	6		Indicates that table to be accessed in DR is of type TBL1, DRA is set to VALUE*2^7+(AC^{21-15})	
SPA	0-17	0-17	Indicates location in S-Pad to be set or used when X2=3,4,5,6 or MA=1,2,3,5,6,7	
X2	0	NO-OP or omit	No operation	
	1	NOMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated.	
	1	AC→C	(AC)→C	
	2	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	
	3	VALUE+SP→SP	(CB^{0-11})+(SP_{SPA})→ SP_{SPA}	MA=4
	3	MAFN→SP		MA≠4
	4	MA→SP	(MA)→SP_{SPA}	
	5	DPA→SP	(DPA)→SP_{SPA}	
	6	DPA+SP+1→DPA	(DPA)+(SP_{SPA})+1→DPA	
	7	AC→PS	(AC)→PS_{DPA}	
MA	0	NO-OP or omit	No operation	
	1	SP	(SP_{SPA})→MA	
	2	SP+1	(SP_{SPA})+1→MA	
	3	SP-1	(SP_{SPA})-1→MA initiates MD read (see Sec. 2.3)	
	4	MA+1	(MA)+1→MA	
	5	SP+8	(SP_{SPA})+8→MA	
	6	SP+MA	(SP_{SPA})+(MA)→MA	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
VALUE	0-17	0-17	Defines which table in DR is to be accessed; (AC^{21-15}) is used to index the table	

Example:

SETTBL1:15 X2=MAFN→SP(17) MA=SP+8

In this instruction table 15 in DR is to be accessed; the entry in the table is defined by (AC). In parallel with this operation, SP_{17} is set to its "old" value plus 8.

31 27	23	19	16	13	12	8	0
0	TBL2	SPA	X2	MA	Ø	VALUE	Ø

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
TBL2	7		Indicates that table to be accessed in DR is of type TBL2. DRA is set to $VALUE*2^8+(AC^{21-14})$	
SPA	0-17	0-17	Indicates location in S-Pad to be set or used when X2=3,4,5,6 or MA=1,2,3,5,6,7	
X2	0	NØ-ØP or omit	No operation	
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated.	
	1	AC→C	(AC)→C	
	2	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	
	3	VALUE+SP→SP	$(CB^{0-11})+(SP_{SPA})\rightarrow SP_{SPA}$	MA=4
	3	MAFN→SP		MA≠4
	4	MA→SP	$(MA)\rightarrow SP_{SPA}$	
	5	DPA→SP	$(DPA)\rightarrow SP_{SPA}$	
	6	DPA+SP+1→DPA	$(DPA)+(SP_{SPA})+1\rightarrow DPA$	
	7	AC→PS	$(AC)\rightarrow PS_{DPA}$	
MA	0	NØ-ØP or omit	No operation	
	1	SP	$(SP_{SPA})\rightarrow MA$	
	2	SP+1	$(SP_{SPA})+1\rightarrow MA$	
	3	SP-1	$(SP_{SPA})-1\rightarrow MA$ initiates MD read cycle (see Sec. 2.3)	
	4	MA+1	$(MA)+1\rightarrow MA$	
	5	SP+8	$(SP_{SPA})+8\rightarrow MA$	
	6	SP+MA	$(SP_{SPA})+MA\rightarrow(MA)$	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
VALUE	0-7	0-7	Defines which table in DR is to be accessed; (AC^{22-15}) is used to index the table	

Example:

SETTBL2 :7 X2= DPA→SP MA=SP→MA SPA=1

This instruction indicates that table 7 in TM is to be accessed; the entry in the table is defined by (AC). In parallel with this operation, a data read of MD is initiated from the address indicated by (SP_1), and (DPA) is stored in SP_1.

31 27	23	19	16	13	12	0
1	Ø	SPA	X2	MA	Ø	SPVALUE

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
SPA	$0\text{-}17_8$	$0\text{-}17_8$	SP address, indicating location in S-Pad to be set to SPVALUE	
X2	0	NO-OP or omit	No operation	
	1	NOMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated	
	1	AC→C	(AC)→C	
	2	WRT	Indicates that data write of (AC) $\rightarrow MD_{MA}$ is to occur instead of data read the next time MA is set.	
	7	AC→PS	$(AC)\rightarrow PS_{DPA}$	
MA	0	NO-OP or omit	No operation	
	4	MA+1	(MA)+1→MA -- initiates MD read cycle (see Sec. 2.3)	
SPVALUE	$0\text{-}7777_8$	$0\text{-}7777_8$	Value to be stored in SP at address indicated by SPA field.	

Example:

SETSP(12):310

This instruction would cause SP_{12} to be set to the value 310_8. Note that the X2 and MA fields are null; i.e., not used.

31 27	23	19	16	13	11	9	3	0
2	REG	SPA	XO	MA	DP	AC2	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
REG	In general, this field denotes the register to receive (AC^{0-n})			
	0	PSA	$(AC^{0-11})\rightarrow PSA$; i.e., unconditional jump	
	1	DPR	$(AC^{0-15})\rightarrow DP^{0-15}_{DPA}$	
	2	DRA	$(AC^{0-11})\rightarrow DRA$	MA≠7
	3	MA	$(AC^{0-11})\rightarrow MA$	MA≠1-7;XO≠2,5,6,7;DPA≠4,6
	4	DPA	$(AC^{0-11})\rightarrow DPA$	MA≠1,5-7;XO≠2,5,6,7;DPA≠1-6
	5	SPA	(AC^{0-3}) SPA*	
	10	CALLSUB	$(PSA)+1\rightarrow EXIT_{DPA}$, $(AC^{0-11})\rightarrow PSA$; i.e., return jump	
	11	MP→C	$(MP^{0-31})\rightarrow C$	XO≠1
	12	PS→C	$(PS_{DPA})\rightarrow C$	XO≠1
	13	NO-OP	No operation	
	15	LOADPS	$(AC)\rightarrow PS_{DPA}$	
	16	SCL→SP	$(AC^{24-31})\rightarrow SP_{SPA}$	XO≠3-7 DPA≠5
	17	SETEXIT	$(AC^{0-11})\rightarrow EXIT_{DPA}$	
SPA	$0-17_8$	$0-17_8$	SP address indication location in S-Pad to be set or used when REG=16, XO=3,4,5,6,7,MA=1,2,3,5,6,7,DPA=4,5,6	
XO	0	NO-OP or omit	No operation	
	1	AC→C	$(AC)\rightarrow C$	REG≠11,12
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated.	
	3	ACR+SP→SP	$(ACR)+(SP_{SPA})\rightarrow SP_{SPA}$	MA=4;DPA≠4-6; REG≠3-4,16
	3	ACR→SP	$(ACR)\rightarrow SP_{SPA}$	MA=4;DPA≠4-6; REG=3,4
	3	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPA}, rather than MA or DRA, and no read cycle is initiated.	REG≠3,4,16 DPA≠4-6;MA≠0,4

*SPA set to value of SPA field at start of instruction, changed to (AC^{0-3}) at end.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	4	MA→SP	(MA)→SP_{SPA}	REG≠3,4,16;DPA≠5,6
	5	SP*2→SP	1-bit left shift of SP_{SPA}	REG≠3,4,16;DPA≠4-6; MA=0 or 4
	6	SP/4→SP	2-bit right shift of SP_{SPA}	REG≠3,4,16;DPA≠5,6; MA=0 or 4
	7	SP-8→SP	(SP_{SPA})-8→SP_{SPA}	REG≠3,4,16;DPA≠4-6; MA=0 or 4
MA	0	NØ-ØP or omit	No operation	
	1	SP	(SP_{SPA})→MA	XO≠2,5,7;DPA≠5; REG≠3,4
	2	SP+1	(SP_{SPA})+1→MA	XO≠2,5-7,DPA≠4,6; REG≠3,4
	3	SP-1	(SP_{SPA})-1→MA initiates MD read cycle (see Sec. 2.3)	XO≠2,5-7;DPA≠4,6; REG≠3,4
	4	MA+1	(MA)+1→MA	REG≠3
	5	SP+8	(SP_{SPA})+8→MA	XO≠2,5-7;DPA≠4,6; REG≠3,4
	6	SP+MA	(SP_{SPA})+(MA)→MA	XO≠2,5-7;DPA≠4,6; REG≠3,4
	7	DRA+SP→DRA	(DRA)+(SP_{SPA})→DRA initiates DR read cycle (see Sec.2.3)	XO≠2,507;DPA≠4,6; REG≠2
AC2	0	NØ-ØP or omit	No operation	
	1	AD→AC		
	2	HD→AC	(HD--host computer data lines)→AC, CLR HDRDY	
	3	C→AC	(C)→AC, CLR IØDRDY	
DP	0	NØ-ØP or omit	No operation	
	1	ACL→DPL	$(AC^{16-31})\rightarrow DP^{16-31}_{DPA}$	
	2	ACR→DPR	$(AC^{0-15})\rightarrow DP^{0-15}_{DPA}$	
	3	AC→DP	$(AC)\rightarrow DP_{DPA}$	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
ADDER	General form is A-input,b-input: function, where A-input,B-input =			
	0	DP,AC	(DP_{DPA}),(AC) used in function defined	
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	0,(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA}),(AC)-cross used in function defined	
	5	ACL/DPR,ACR/0	(AC^{16-31}),(AC^{0-15}) used in function defined, result into AD^{16-31}; (DP^{0-15}_{DPA}),0 used in function defined, result into AD^{0-15}.	
	6	0,MD	0,(MD_{MA}) used in function defined. This option is used to retrieve memory data for which a read memory cycle was initiated at least four clocks earlier (500 ns).	
	7	DP'/DPR,ACL+1/ ACR	1's complement of (DP_{DPA}), (AC^{16-31}) +1 used in function defined, result in AD^{16-31}; (DP^{0-15}_{DPA}),(AC^{0-15}) used in function defined, result in AD^{0-15}.	
	and where function =			
	0	A-1	(A-input specified)-1→AD	
	1	A	(A-input specified)→AD	
	2	A+B	(A-input specified)+(B-input specified)→AD	
	3	A+B+1	(A-input specified)+(B-input specified)+1→AD	
	4	A-+B	$(A\text{-input}^{16-31})-(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})+(B\text{-input}^{0-15})\rightarrow AD^{0-15}$	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	$(A\text{-input}^{16-31})+(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})-(B\text{-input}^{0-15})\rightarrow AD^{0-15}$	
	7	A+1	(A-input)+1→AD	

*(AC)-cross means the two 16-bit fields of AC are reversed; i.e., $(AC^{0-15})\rightarrow B^{16-31}$ and $(AC^{16-31})\rightarrow B^{0-15}$.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DPA	0	NØ-ØP or omit	No operation	
	1	INC	(DPA)+1→DPA	REG≠4
	2	DEC	(DPA)-1→DPA	REG≠4
	3	DPA-8	(DPA)-8→DPA	REG≠4
	4	SP	(SP_{SPA})→DPA	MA≠2,3,4,6,7; XO≠3,7;REG≠4
	5	DPA+SP	(DPA)+(SP_{SPA})→DPA	XO≠2-7;REG≠4
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	MA≠2,3,5,6,7; XO≠2-7;REG≠4
	7	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

ACR→REG:DRA MA=MA+1 AC2=AD→AC ADDER=0,MD:A+B DPA=WRT

This instruction causes bits 0-11 of the "old" contents of AC to be transferred to DRA. In parallel with this operation, the memory data from a previous read is transferred to AC through the adder and a memory write operation is initiated to write this data into the successor memory address. SPA is set to 0.

31 27	23	19	16	13	9	3	0
3	MULT	SPA	XO	MA	AC1	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions

MULT This field indicates the two 16-bit values that are to be loaded into the M1 and M2 registers, initiating a multiply operation. The product will be available in the MP register after 250 nanoseconds. If M1 or M2 is specified as one of the values, the contents of that register are not changed. Two options are available for 16 x 16 multiplication. If a full 32-bit integer product is desired, the C register must be referenced as the source in the AC1 field of this instruction, and the product must be read into the C register after two clocks or more. If a rounded 16-bit product (fractional multiply) is desired, C must not be the source in the AC1 field and the product must be read into ACR after two clocks or more using the AC0 or AC1 fields of formats 2,3,5,7 or 10 with F=0 or not coded.

0	NØ-ØP or omit	No change in M1 or M2
1	M1,ADR	(M1)x(ACR)→MP
2	M1,ADL	(M1)x(ADL)→MP
3	DRM,ADR	(DR_{DRA}^{0-23})x(ADR)→MP*
4	DPR,M2	(DPR_{DPA})x(M2)→MP
5	DPL,M2	(DPL_{DPA})x(M2)→MP
6	DRR,M2	(DRR_{DRA})x(M2)→MP
7	DRL,M2	(DRL_{DRA})x(M2)→MP
10	DPR,ADR	(DPR_{DPA})x(ADR)→MP
11	DPL,ADR	(DPL_{DPA})x(ADR)→MP
12	DRR,ADR	(DRR_{DRA})x(ADR)→MP
13	DRL,ADR	(DRL_{DRA})x(ADR)→MP
14	DPR,ADL	(DPR_{DPA})x(ADL)→MP
15	DPL,ADL	(DPL_{DPA})x(ADL)→MP
16	DRR,ADL	(DRR_{DRA})x(ADL)→MP
17	DRL,ADL	(DRL_{DRA})x(ADL)→MP

*The DRM,ADR multiplication requires 375 ns, hence three clocks must pass before MP can be read.

31 27	23	19	16	13	9	3	0
3	MULT	SPA	XO	MA	ACl	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
SPA	$0\text{-}17_8$	$0\text{-}17_8$	SP address, indicating location in S-Pad to be set to (AC^{0-11})	
XO	0	NØ-ØP or omit	No operation	
	1	AC→C	(AC)→C	
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read cycle is initiated.	
MA	0	NØ-ØP or omit	No operation	
	4	MA+1	(MA)+1→MA	
ACl	General form is input→destination, where input =			
	0	MP	$(MP)^{15-30}$→destination	
	1	AD	(AD)→destination	
	2	HD	(HD)--host computer data)→destination, CLR HDRDY	
	3	C	(C)→destination, CLR IØDRDY	
	and where destination =			
	0	NØ-ØP	No transfer	
	1	ACL	input→AC^{16-31}	
	2	ACR	input→AC^{0-15}	
	3	AC	input→AC	
ADDER	General form is A-input,B-input:function, where A-input,B-input =			
	0	DP,AC	(DP_{DPA}),(AC) used in function defined	
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA})(AC)-cross* used in function defined	
	5	ACL/DPR,ACR/0	(AC^{16-31}),(AC^{0-15}) used in function defined, result into AD^{16-31}; (DP^{0-15}),0 used in function defined, result into AD^{0-15}	

*(AC)-cross means the two 16-bit fields of AC are reversed; i.e., (AC^{0-15}) →AC^{16-31} and→(AC^{16-31})→AC^{0-15}.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	6	0,MD	0,(MD_{MA}) used in function defined. This option is used to retrive memory data for which a read memory cycle was initiated at least four clocks (500 ns) earlier.	
	7	DP'/DPR, ACL+1/ACR	1's complement of (DP_{DPA}),(AC^{16-31})+1 used in function defined, result stored in AD^{16-31}; (DP_{DPA}^{0-15}),(AC^{0-15}) used in function defined, result stored in AD^{0-15}.	
and where function =				
	0	A-1	(A-input)-1→AD	
	1	A	(A-input)→AD	
	2	A+B	(A-input)+(B-input)→AD	
	3	A+B+1	(A-input)+(B-input)+1→AD	
	4	A-+B	($A\text{-input}^{16-31}$)-($B\text{-input}^{16-31}$)→AD^{16-31} ($A\text{-input}^{0-15}$)+($B\text{-input}^{0-15}$)→AD^{0-15}	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	($A\text{-input}^{16-31}$)+($B\text{-input}^{16-31}$)→AD^{16-31} ($A\text{-input}^{0-15}$)+($B\text{-input}^{0-15}$)→AD^{0-15}	
	7	A+1	(A-input)+1→AD	
DPA	0	NØ-ØP or omit	No operation	
	1	INC	(DPA)+1→DPA	
	2	DEC	(DPA-1→DPA	
	3	DPA-8	(DPA)-8→DPA	
	7	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

FMT=ACR→SP(5) ADDER=0,MD:A+B AC1=AD→AC

With this instruction, data from MD (for which an MD ready cycle was initiated at least 500 ns earlier) is transferred to the accumulator through the adder. The "old" contents of AC, bits 0-11, are transferred to SP register 5.

SP→SP/FORMAT 4

31 27	23	19	16	13	11	9	3	0
4	SPAW	SPAR	XO	MA	DP	AC2	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
SPAW	$0\text{-}17_8$	$0\text{-}17_8$	SP address where data is to be stored.	
SPAR	$0\text{-}17_8$	$0\text{-}17_8$	SP address of data to be fetched.	
XO	0	NO-OP or omit	No operation	
	1	AC→C	(AC)→C	
	1	NOMI	No memory initiates; i.e., if MA field is set, value is transferred to MA or DRA, but no memory read is initiated.	
	2	SP→SP	$(SP_{SPAR})\rightarrow SP_{SPAW}$	DPA≠5,6; MA≠2,3,5-7
	3	ACR+SP→SP	$(ACR)_{0-11}+(SP_{SPAR})\rightarrow SP_{SPAW}$	DPA≠4-6 MA=4 or 0
	3	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPAW}, rather than MA or DRA.	DPA≠4-6; MA≠4 or 0
	4	MA→SP	$(MA)\rightarrow SP_{SPAW}$	DPA≠5,6
	5	SP*2→SP	1-bit left shift (end-off) of $(SP_{SPAR})\rightarrow SP_{SPAW}$	DPA≠4-6 MA≠1-3,5-7
	6	SP/4→SP	2-bit right shift (end-off, sign extended) of $(SP_{SPAR})\rightarrow SP_{SPAW}$	DPA≠5,6 MA≠2,3,5-7
	7	SP-8→SP	$(SP_{SPAR})-8\rightarrow SP_{SPAW}$	DPA≠4-6 MA≠1-3,5-7
MA	0	NO-OP or omit	No operation	
	1	SP	$(SP_{SPAR})\rightarrow MA$	XO≠5,7
	2	SP+1	$(SP_{SPAR})+1\rightarrow MA$	DPA≠4,6; XO≠2,5-7
	3	SP-1	$(SP_{SPAR})-1\rightarrow MA$ Initiates MD read (see Sec. 2.3)	DPA≠4,6; XO=2,5-7
	4	MA+1	(MA)+1→MA	
	5	SP+8	$(SP_{SPAR})+8\rightarrow MA$	DPA≠4,6; XO≠2,5-7
	6	SP+MA	$(SP_{SPAR})+(MA)\rightarrow MA$	DPA≠4,6; XO≠2,5-7

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	7	DRA+SP→DRA	(DRA)+(SP_{SPAR})→DRA initiates DR read (see Sec. 2.3)	DPA≠4,6; XO≠2,5-7
AC2	0	NØ-ØP or omit	No operation	
	1	AD→AC	(AD)→AC.	
	2	HD→AC	(HD--host computer data lines)→AC^{0-31}, HDRDY=0	
	3	C→AC	(C)→AC^{0-31}, IØDRDY=0	
DP	0	NØ-ØP or omit	No operation	
	1	ACL→DPL	$(AC^{16-31})\rightarrow DP^{16-31}_{DPA}$	ADDER≠0,1,4,5,7
	2	ACR→DPR	$(AC^{0-15})\rightarrow DP^{0-15}_{DPA}$	ADDER≠0,1,4,5,7
	3	AC→DP	$(AC)\rightarrow DP_{DPA}$	ADDER≠0,1,4,5,7
ADDER	General form is A-input,B-input:function, where A-input,B-input=			
	0	DP,AC	(DP_{DPA}),(AC) used in function defined	DP=0
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	DP=0
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	0,(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA}),(AC)-cross* used in function defined	DP=0
	5	ACL/DPR,ACR/0	(AC^{16-31}),(AC^{0-15}) used in function defined, result into AD^{16-31}; (DP^{0-15}_{DPA}), 0 used in function defined, result into AD^{0-15}.	DP=0
	6	0,MD	0,(MD_{MA}) used in function defined. This option is used to retrieve memory data for which a read memory cycle was initiated at least four clocks (500 ns) earlier.	

*(AC)-cross means the two 16-bit fields of AC are reversed; i.e., $(AC^{0-15})\rightarrow AC^{16-31}$ and $(AC^{16-31})\rightarrow AC^{0-15}$.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	7	DPL'/DPR,ACL+1/ACR	1's complement of (DP_{DPA}), $(AC^{16-31})+1$ used in function defined, result stored in AD^{16-31}; (DP_{DPA}^{0-15}), (AC^{0-15}) used in function defined, result stored in AD^{0-15}.	DP=0
	and where function =			
	0	A-1	(A-input)-1→AD	
	1	A	(A-input)→AD	
	2	A+B	(A-input)+(B-input)→AD	
	3	A+B+1	(A-input)+(B-input)+1→AD	
	4	A-+B	For A,B input: $(A\text{-input}^{16-31})-(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})+(B\text{-input}^{0-15})\rightarrow AD^{0-15}$ For $A_1/A_2, B_1/B_2$ input: $(A_1)-(B_1)\rightarrow AD^{16-31}$ $(A_2)+(B_2)\rightarrow AD^{0-15}$	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	$(A\text{-input}^{16-31})+(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})-(B\text{-input}^{0-15})\rightarrow AD^{0-15}$ For $A_1/A_2, B_1/B_2$ input $(A_1)+(B_1)\rightarrow AD^{16-31}$ $(A_2)-(B_2)\rightarrow AD^{0-15}$	
	7	A+1	(A-input)+1→AD	
DPA	0	NO-OP or omit	No operation	
	1	INC	(DPA)+1→DPA	
	2	DEC	(DPA)-1→DPA	
	3	DPA-8	(DPA)-8→DPA	
	4	SP	$(SP_{SPAR})\rightarrow$DPA	MA≠2,3,5-7; XO≠ 5,
	5	DPA+SP	$(DPA)+(SP_{SPAW})\rightarrow$DPA	XO≠2-7
	6	SWAPSP	$(DPA) \rightarrow SP_{SPAW}$ $(SP_{SPAR})\rightarrow$DPA	MA≠2,3,5-7; XO≠2-7

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	7	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

```
FMT=SP→SP SPAW=13 SPAR=12  XO=MAFN→SP  MA=SP+8  ADDER=DP',AC:A+1
AC1=AD→AC  DPA=INC
```

This instruction causes $(SP_{12})+8\rightarrow SP_{13}$.

In parallel with this operation, the 1's complement of (DP_{DPA}) is added to 1, forming the 2's complement of (DP_{DPA}). The results of this add are stored in the accumulator. In addition, (DPA) are incremented by 1; the "old" contents of DPA are used in locating DP_{DPA} for the add operation, however.

31 27	23	19	16	13	9	7	6	3	0
5	JUMP	SPA	TEST3	MA	ACl	DP	C	Ø	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
JUMP	$0\text{-}17_8$	$0\text{-}17_8$	$(CB^{23-26})\rightarrow PSA^{0-3}$; i.e., jump to location within current page; if condition specified by TEST 3 field is satisfied.	
SPA	$0\text{-}17_8$	$0\text{-}17_8$	Indicates location in S-pad to be set or used when MA=1,2,3,5,6,7, or DPA=4,5,6.	
TEST3	0	NØ-ØP or omit	No jump	
	1	JMP	Unconditional jump	
	2	AC22=1	Jump if $(AC)^{22}=1$	
	3	AC23=1	Jump if $(AC)^{23}=1$	
	4	DPA≠37	Jump if (DPA)≠37	
	5	HDRDY=0	Jump if host data ready bit=0	
	7	HDRDY=1	Jump if host data ready bit=1	
MA	0	NØ-ØP or omit	No operation	
	1	SP	$(SP_{SPA})\rightarrow MA$	
	2	SP+1	$(SP_{SPA})+1\rightarrow MA$	DPA≠4,6
	3	SP-1	$(SP_{SPA})-1\rightarrow MA$ initiates MD read (see Sec. 2.3)	DPA≠4,6
	4	MA+1	$(MA)+1\rightarrow MA$	
	5	SP+8	$(SP_{SPA})+8\rightarrow MA$	DPA≠4,6
	6	SP+MA	$(SP_{SPA})+(MA)\rightarrow MA$	DPA≠4,6
	7	DRA+SP	$(DRA)+(SP_{SPA})\rightarrow DRA$ initiates DR read (see Sec. 2.3)	DPA≠4,6
ACl	General form is input→destination, where input =			
	0	MP	$(MP)^{15-30}$→destination	
	2	HD	(HD--host computer data lines)→destination, CLR HDRDY	
	3	C	(C)→destination, CLR IØDRDY	
	and where destination =			
	0	NO-OP	No operation	
	1	ACL	$input\rightarrow AC^{16-31}$	
	2	ACR	$input\rightarrow AC^{0-15}$	
	3	AC	input→AC	
DP	0	NØ-ØP or omit	No operation	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DP	1	ACL→DPL	$(AC^{16-31})\rightarrow DP^{16-31}_{DPA}$	
	2	ACR→DPR	$(AC^{0-15})\ DP^{0-15}_{DPA}$	
	3	AC→DP	(AC)→DP	
C	0	NØ-ØP or omit	No operation	
	1	AC→C	(AC)→C	
DPA	0	NO-OP or omit	No operation	
	1	INC	(DPA)+1→DPA	
	2	DEC	(DPA)-1→DPA	
	3	DPA-8	(DPA)-8→DPA	
	4	SP	$(SP_{SPA})\rightarrow DPA$	MA≠2,3,5-7
	5	DPA+SP	$(DPA)+(SP_{SPA})\rightarrow DPA$	
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	MA≠2,3,5-7
	7	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

```
TEST3: DPA≠37  JUMP=10  MA=MA+1
       DPA=WRT AC1=MP→ACR
```

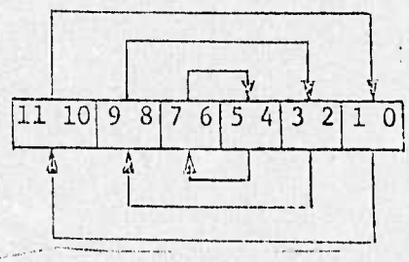
The above instruction would transfer the product from a previously initiated multiply into ACR. In parallel, the contents of MA would be incremented by 1. At the end of the instruction, a write cycle would be initiated to write AC into MD at the new MA address. Finally, if (DPA) was not 37_8, the successor instruction would be word 10_8 on the current page. If (DPA) was 37_8, the next instruction is taken in sequence.

SHIFT OR TEST4 OR FLAG/FORMAT 6

31 27	23	22	19	18	16	13	9	7	6	0
6	JUMP	Ø	TEST4	F	SC	STATUS	FLAG	Ø	C	SHIFT

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
JUMP	$0\text{-}17_8$	$0\text{-}17_8$	$(CB^{23-26})\rightarrow PSA^{0-3}$; i.e., jump to location within current page; if condition specified by TEST4 field is satisfied.	
TEST4	0	NØ-ØP or omit	No jump	
	1	JUMP	Unconditional jump	
	2	DP22=1	Jump if $(DP)^{22}_{DPA} = 1$	DPA must not have changed, or DPL loaded in previous instruction.
	3	DP23=1	Jump if $(DP^{23}_{DPA}) = 1$	
	4	ADC23=1	Jump if ADC23=1. ADC is set or cleared when adder is transferred to AC or TEST0=5,6,7,15,16,17 It is the carry from bit 23 of the adder.	
	5	MDAQ=1	Jump if MDAQ=1	
	6	MDBQ=1	Jump if MDBQ=1	
F	0-1	0-1	Floating point flag; if=1, indicates floating point operation; if=0, indicates fixed point operation.	
SC	0	ACR	Shift $(AC^{0-m})^*\rightarrow AC^{0-m}$ in direction and by amount indicated by SHIFT field. Causes $(CB^{0-4})\rightarrow SN^{0-4}$, $(CB^5)\rightarrow SN^7$.	
	1	ACR,SN	Shift $(AC^{0-m})^*\rightarrow AC^{0-m}$ in direction and by amount indicated by (SN). (See Sec.2.2)	
	2	DPR	Shift $(DP^{0-m})^*\rightarrow AC^{0-m}$ in direction and by amount indicated by SHIFT field. Causes $(CB^{0-4})\rightarrow SN^{0-4}$, $(CB^5)\rightarrow SN^7$, result (AC^{0-m}) and residue	
	3	DPR,SN	Shift $(DP^{0-m})^*\rightarrow AC^{0-m}$ in direction and by amount indicated by (SN), result (AC^{0-m}) and residue (See Sec.2.2)	
STATUS	0	NO-OP		
	3	ERROR3	Sets bit 3 in status register	
	4	ERROR4	Sets bit 4 in status register	
	5	ERROR5	Sets bit 5 in status register	
	6	ERROR6	Sets bit 6 in status register	
	7	NOTBUSY	Clears bit 7 in status register	

*m=15 if F field=0; m=23 if F field=1.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
FLAG	0	NO-OP or omit	No operation	
	1	FTQ=1	Initiates Fourier transform mode	
	2	CLRFLGS	Terminates Fourier tranform, Inverse Fourier Transform modes, Clears ERROR indicator	
	3	IFTQ=1	Initiates Inverse Fourier Transform Mode	
	4	FTP2Q=1	Initiates Fourier Transform Pass 2 mode	
	5	FTP2Q=0	Terminates Fourier Transform Pass 2 mode	
	6	SP→DRA	(SP_{SPA})→DRA; initiates DR read (see Sec. 2.3). SPA is not set by this instruction.	
	7	SPAX=11	Special purpose field used by FFT.	
	10	IØDRDY=0	Clears I/O Data Ready indicator;	
	11	IØDRDY=1	Sets I/O Data Ready indicator.	
	12	HDDRDY=0	Clears Host Data Ready indicator; signal to host computer that data the host has been retrieved.	
	13	HDDRDY=1	Sets Host Data Ready indicator; signal to host computer that data is ready for host computer to retrieve.	
	14	SENDHI	Send Host Interrupt. Causes hardware interrupt of host computer.	
	15	MDØVF→MDS	Transfer MD overflow bits (MDAQ, MDBQ) into MDS.	
	16	DECIMATE	$(MA^0)\leftrightarrow(MA^{10})$ $(MA^1)\leftrightarrow(MA^{11})$ $(MA^2)\leftrightarrow(MA^8)$ $(MA^3)\leftrightarrow(MA^9)$ $(MA^4)\leftrightarrow(MA^6)$ $(MA^5)\leftrightarrow(MA^7)$	
C	0	NØ-ØP	No-operation	
	1	AC→C	(AC)→C	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
SHIFT	0-37	n	Shift register indicated by SC field n bits to the right	
	40-77	n+40	Shift register indicated by SC field n bits to the left (Shifts are end-off; sign extended on right shift)	

Example:

SHIFT:R:2 F=1 SC=ACR

This instruction causes bits 0-23 of the accumulator to be shifted right two places. The sign bit (bit 23) is extended.

31 27	23	19	18	17	13	9	7	6	3	0
7	JUMP	TESTO	F	A	ADDI	ACO	DP	C	ADDIFN	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
JUMP	$0\text{-}17_8$	$0\text{-}17_8$	$(CB^{23-26})\rightarrow PSA^{0-3}$; i.e., jump to location within current page; if condition specified by TESTO is satisfied.	
TESTO	0	NØ-ØP or omit	No jump	
	1	ØVFL=0	Jump if overflow out of (AD^{31}); set or cleared when adder is transferred to AC or TESTO=5,6,7,15,16,17.	
	2	ØVFR=0	Jump if overflow out of (AD^{15}) or (AD^{23}), depending on F=0,F=1, respectively. Set or cleared when adder is transferred to AC or TESTO=5,6,7,15,16,17.	
	3	ADTEST=1	Jump if ADTEST=1. ADTEST is set as described below for TESTO-5,6,7,15,16,17. ADTEST is cleared when adder is transferred to AC.	
	4	JMP	Unconditional jump	
	5	AL=BL	if ADL=0, set ADTEST=1. No jump	
	6	AL>BL	If ADL≥0, set ADTEST=1*. No jump	
	7	AL<BL	If ADL<0, set ADTEST=1. No jump	
	15	AR=BR	If ADR=0, set ADTEST=1. No jump	
	16	AR>BR	If ADR≥0, set ADTEST=1*. No jump	
	17	AR<BR	If ADR<0, set ADTEST=1. No jump	
F	0-1	0-1	Floating point flag; if-1, indicates floating point operation; if=0, indicates fixed point operation.	
A	0	NORMAL or not coded	F field determines whether adder functions as 2 16-bit adders or one 8-bit and one 24-bit adder.	
	1	DBLE	Adder functions as a single 32-bit adder.	

*If TESTO=6 or 16, ADDIFNS A,A+B+1,A-B and A+1 become A-1,A+B,A-B-1 and A, respectively.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
ADDI	In general, specifies A,B inputs for ADDIFN field, or to be tested in TESTO field. General form is A,B-inputs: function, where A,B inputs are:			
	0	DP,AC	(DP_{DPA}),(AC) used as A,B in ADDIFN or TESTO	DP=0
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used as A,B in ADDIFN or TESTO.	DP=0
	2	AC,AC	(AC) used as both A,B input in ADDIFN or TESTO	
	3	0,AC	0,(AC) used as A,B in ADDIFN or TESTO.	
	4	SCLR,AC	(SCLR),(AC) used as A,B in ADDIFN or TESTO.	
	5	FPO,AC	Floating point zero,(AC) used as A,B in ADDIFN or TESTO.	
	6	FP1/2,AC	Floating point one-half,(AC) used as A,B in ADDIFN or TESTO.	
	7	FP-1,AC	FLoating point minus one,(AC) used as A,B in ADDIFN or TESTO.	
	10	DP,MD	(DP_{DPA}), (MD_{MA}) used as A,B in ADDIFN or TESTO.	DP=0
	11	DP',MD	1's complement of (DP_{DPA}),(MD_{MA}) used as A,B in ADDIFN or TESTO.	DP=0
	12	AC,MD	(AC),(MD_{MA}) used as A,B in ADDIFN or TESTO.	
	13	0,MD	0, (MD_{MA}) used as A,B in ADDIFN or TESTO.	
	15	DR,AC	(DR_{DRA}),(AC) used as A,B in ADDIFN or TESTO.	
ACO	0	NO-OP or omit	No operation	
	1	AD→AC	(AD)→AC.	
	2	HD→AC	(HD--host computer data lines)→AC^{0-31},clr. HDRDY	
	3	C→AC	(C)→AC^{0-31},CLR IØDRDY	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
ACO	4	FP→AC	Used during second step of a floating point add/subtract operation. Indicates that mantissas (bits 0-23) of A,B inputs are to be scaled according to the delta scale value in SN (see DSCL→SN option, below) so that add/subtract operation specified by ADDIFN will be meaningful. Result of scaled add or subtract stored in AC and residue.	ADDI=0,2,3,5, 6,7,15
	5	RNDAC	Round AC. Used to round the result of a floating point multiply or add when placed in AC. (AC^{23}) tested for 0 or 1, appropriate round for 2's complement mantissa is made.	
	6	FLØTAC	Normalizes & rounds floating point (AC);i.e., shifts (AC^{0-22}) such that $AC^{22} \neq AC^{23}$ and decrements AC^{31-24} by 1 for each place shifted. If OVFR=1, (AC^{0-23}) are shifted right one place, AC^{23} is inverted, and (AC^{31-24}) is incremented by 1.	ADDI must be AC,AC ADDFN must be A-1 or not coded.
	7	ADR→ACR	$(AD^{0-m}) \rightarrow AC^{0-m}$; where m=23 if F=1 =15 if F=0	
	10	ADL→ACL	(AD^{n-31}) AC^{n-31}, where n=24 if F=1 =16 if F=0	
	11	SCLDPR→ACR	(DP^{0-m}) shifted according to $SN \rightarrow AC^{0-m}$	
	12	ADL→SN	$\lvert AD^{31-24} \rvert \rightarrow SN^{5-0}$, SN^6= 1 if $(AD^{31-24}) < 0$	ADDIFN=1,5
	13	DSCL→SN	Delta scale to SN. $\lvert AD^{31-24} \rvert \rightarrow SN^{6-0}$ Used during first step of floating point add operation. Specified in conjunction with ADDIFN=A-B, this determines the difference in scale between quantities A and B and stores the sign-magnitude result in SN. A shift of the appropriate mantissa is initiated.	ADDIFN=1,5
	14	MP→ACR	$(MP^{m-2m}) \rightarrow (AC^{0-m})$; where m=23 if F=1 $(MP^{2m+1-2m+3})$, residue =15 if F=0	
	15	SP→SCL	$(SP^{0-7}_{SPA}) \rightarrow AC^{24-31}$	
	16	SMAG→2C	Converts (AC^{0-m}) from absolute value to 2's complement form; where m=23 if F=1 =15 if F=0. The sign is determined by the input sign to the last 2C→SMAG operation.	ADDI must be 0,AC ADDIFN must be A-B

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
ACO	17	2C→SMAG	Converts (AC^{0-m}) from 2's complement form to absolute value where m=23 if F=1, -15 if F=0	ADDI must be 0,AC ADDIFN must be A-B
DP	0	NØ-ØP or omit	No operation	
	1	ACL→DPL	$(AC^{16-31})\rightarrow DP^{16-31}_{DPA}$,	
	2	ACR→DPR	$(AC^{0-15})\rightarrow DP^{0-15}_{DPA}$	
	3	AC→DP	$(AC)\rightarrow DP_{DPA}$	
C	0	NØ-ØP or not coded	No operation	
	1	AC→C	(AC)→C	
ADDIFN	Specifies adder function for A,B inputs specified in ADDI			
	0	A-1	(A-input specified in ADDI)-1→AD	
	1	A	(A-input specified in ADDI)→AD	
	2	A+B	(A-input specified in ADDI)+(B-input specified in ADDI)→AD	
	3	A+B+1	(A-input specified in ADDI)+(B-input specified in ADDI)+1→AD	
	5	A-B	(A-input specified in ADDI)-(B-input specified in ADDI)→AD	
	6	AL+BL+1,AR	(A-input specified in ADDI,bits n-31)+(B-input specified in ADDI,bits n-31)+1→AD^{n-31} (A-input specified in ADDI, bits 0-m)→AD^{0-m} where n, m = 16, 15 if F=0 = 24, 23 if F=1	
	7	A+1	(A-input specified in ADDI)+1→AD	
DPA	0	NO-OP or omit	No operation	
	1	INC	(DPA)+1→DPA	
	2	DEC	(DPA)-1→DPA	
	3	DPA-8	(DPA)-8→DPA	
	4	SP	$(SP_{SPA})\rightarrow DPA$	
	5	DPA+SP	$(DPA)+(SP_{SPA})\rightarrow DPA$	
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	7	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur instead of data read from MD. Applies to next instruction in which MA is set.	

Examples:

Floating Point Add of DP,AC:

```
FADD:DP,AC:A-B   F=1  ACO=DSCL→SN
FADD:DP,AC:A+B   F=1  ACO=FP→AC
     FMT=TESTO   F=1  ACO=FLOTAC  ADDI=AC,AC ADDIFN=A-1
```

The first instruction causes the difference in scale of DP and AC to be stored in SN and shift the appropriate mantissa to be transferred into SCLR.

The second instruction completes the shift according to the value in SN, followed by the addition of the mantissas of DP and AC, the result being stored in AC^{0-23}. The exponent of DP and AC is simply transferred into AC^{24-31}.

The third instruction causes the normalization of AC.

31 30	27	23	19	18	16	13	9	3	0
1	TEST1	JUMP	MULT	F	X1	MA	AC1	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
TEST1	1	C→AC	(C)→AC	
	2	DPALSC=7	Jump if least significant octal character of DPA is 7; i.e., $DPA^{0-2}=111_2$	
	3	JMP	Unconditional jump	
	4	DPA≠37	Jump if DPA not equal to 37.	
	5	HDRDY=0	Jump if host data ready bit = 0***, Set HDRDY	
	6	IØDRDY=0	Jump if I/O data ready bit = 0***, Set IØDRDY	
	7	IØDRDY=1	Jump if I/O data ready bit = 1***	
JUMP	$0\text{-}17_8$	$0\text{-}17_8$	(CB^{23-26}) PSA^{0-3}; i.e., jump to location within current page; if condition specified by TEST1 field is satisfied.	
MULT			In general, this field indicates that two values are to be loaded into the M1 and M2 registers and multiplied, with the result appearing in the MP register. (In assembler code, when M1 or M2 are used, they refer to the current contents of M1 or M2; i.e., whatever was last loaded into M1 or M2 as a result of specifying a MULT field operation.) A 16-bit by 16-bit multiply (occurring when F=0) produces a 32-bit result in MP and requires 2 clocks.@ A 24-bit by 24-bit multiply (occurring when F=1) produces a 48-bit result in MP and requires 3 clocks.* The contents of MP may be retrieved into the accumulator for storage or other operations through the AC0 or AC1 fields of Formats 2,3,5,7,10.	
	0	NØ ØP or omit	No operation	
	1	M1,ADR	$(M1)x(AD^{0-m})\rightarrow MP$**	
	2	M1,ADL	$(M1)x(AD^{n-31})\rightarrow MP$	
	3	DRM,ADR	$(DR_{DRA}^{0-23})x(AD^{0-m})\rightarrow MP$	

*The DRM,ADR option specified with F=0 causes a 24-bit mantissa to be multiplied by a 16-bit value. This operation requires 3 clocks (375 ns).

**In the descriptions following, m=15 when F=0, m=23 when F=1; n-16 when F=0, n-24 when F=1.

***The test is repeated until the test condition is satisfied. Then the rest of the instruction is executed and the jump is performed.

@If an integer product is desired, C must be selected as the source register in the AC1 field and the MP→C operation used to retrieve the 32-bit result.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	4	DPR,M2	$(DP_{DPA}^{0-m}) \times (M2) \rightarrow MP$*	X1≠M2→DPR
	5	DPL,M2	$(DP_{DPA}^{0-31}) \times (M2) \rightarrow MP$	X1≠M2→DPR,if F=1
	6	DRR,M2	$(DR_{DRA}^{0-m}) \times (M2) \rightarrow MP$	
	7	DRL,M2	$(DR_{DRA}^{n-31}) \times (M2) \rightarrow MP$	
	10	DPR,ADR	$(DP_{DPA}^{0-m}) \times (AD^{0-m}) \rightarrow MP$	X1≠M2→DPR
	11	DPL,ADR	$(DP_{DPA}^{n-31}) \times (AD^{0-m}) \rightarrow MP$	X1≠M2→DPR,if F=1
	12	DRR,ADR	$(DR_{DRA}^{0-m}) \times (AD^{0-m}) \rightarrow MP$	
	13	DRL,ADR	$(DR_{DRA}^{n-31}) \times (AD^{0-m}) \rightarrow MP$	
	14	DPR,ADL	$(DP_{DPA}^{0-m}) \times (AD^{n-31}) \rightarrow MP$	X1≠M2→DPR
	15	DPL,ADL	$(DP_{DPA}^{n-31}) \times (AD^{n-31}) \rightarrow MP$	X1≠M2→DPR,if F=1
	16	DRR,ADL	$(DR_{DRA}^{0-m}) \times (AD^{n-31}) \rightarrow MP$	
	17	DRL,ADL	$(DR_{DRA}^{n-31}) \times (AD^{n-31}) \rightarrow MP$	
F	0-1	0-1	Floating point flag; if=1, indicates floating point operation; if=0, indicates fixed point operation	
X1	0	NO-OP or omit	No operation	
	1	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPA} instead of MA.	DPA≠4-6,MA=4
	1	ACR+SP→SP	$(ACR)+(SP_{SPA}) \rightarrow SP_{SPA}$	DPA≠4-6,MA=4
	2	MA→SP	$(MA) \rightarrow SP_{SPA}$	DPA≠5,6
	3	M2→DPR	$(M2) \rightarrow DP_{DPA}^{0-15}$	DPR not used as source
MA	0	NO-OP or omit	No operation	
	1	SP	$(SP_{SPA}) \rightarrow MA$	
	2	SP+1	$(SP_{SPA})+1 \rightarrow MA$ initiates MD read cycle (see Sec. 2.3)	DPA≠4,6
	3	SP-1	$(SP_{SPA})-1 \rightarrow MA$	DPA≠4,6
	4	MA+1	(MA)+1 MA	
	5	SP+8	$(SP_{SPA})+8$ MA	DPA≠4,6

*In the descriptions following, m=15 when F=0, m=23 when F=1; n=16 when F=0, n=24 when F=1.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	6	MA+SP	$(MA)+(SP_{SPA})\rightarrow MA$ (initiates MD read cycle (see Sec. 2.3))	DPA≠4,6
	7	DRA+SP→DRA	$(DPA)+(SP_{SPA})\rightarrow DRA$ (initiates DR read cycle (see Sec. 2.3))	DPA≠4,6
AC1	General form is input→destination, where input =			
	0	MP	$(MP^{m-2^m})\rightarrow$destination; where m=15 if F=0, m=23 if F=1	
	1	AD	(AD)→destination	
	2	HD	(HD--host computer data lines), clr HDRDY →destination	
	3	C	(C)→destination, clr IØDRDY	
	and where destination =			
	0	NO-OP	No operation	
	1	ACL	$input\rightarrow AC^{n-31}$; where n=16 if F=0, n=24 if F=1	IF AC1=MP, F=0
	2	ACR	$input\rightarrow AC^{0-m}$; where m=15 if F=0, m=23 if F=1	
	3	AC	input→AC	
ADDER	General form is A-input, B-input:function, where A-input,B-input =			
	0	DP,AC	(DP_{DPA}),(AC) used in function defined	X1≠M2→DPR **
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	X1≠M2→DPR **
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	0,(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA}),(AC)-cross* used in function defined	X1≠M2→DPR **
	5	ACL/DPR',ACR/0	(AC^{n-31});(AC^{0-m}) used in function defined, result into AD^{n-31};(DP^{0-m}_{DPA}), 0 used in function defined, result into AD^{0-m}, where n=16 for F=0, n=24 for F=1, and m=15 for F=0, m=23 for F=1.	X1≠M2→DPR **

*(AC)-cross means the two 16-bit fields of AC are reversed; i.e., $(AC^{0-15})\rightarrow B^{16-31}$ and $(AC^{16-31})\rightarrow B^{0-15}$

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	6	0,MD	$0,(MD_{MA})$ used in function defined. This option is used to retrieve memory data for which a real memory cycle was initiated at least four clocks (500 ns) earlier.	
	7	DPL'/DPR, ACL+1/ACR	1's complement of (DP_{DPA}),(AC^{n-31})+1 used in function defined, result stored in AD^{n-31}; (DP_{DPA}^{0-m}),(AC^{0-m}) used in function defined, result stored in AD^{0-m}, where n=16 for F=0, n=24 for F=1, and m=15 for F=0, m=23 for F=1.	X1≠M2→DPR**
	and where function =			
	0	A-1	(A-input)-1→AD	
	1	A	(A-input)→AD	
	2	A+B	(A-input)+(B-input)→AD	
	3	A+B+1	(A-input)+(B-input)+1→AD	
	4	A-+B	For A,B input: $(A\text{-input}^{n-31})-(B\text{-input}^{n-31})\rightarrow AD^{n-31}$ $(A\text{-input}^{0-m})+(B\text{-input}^{0-m})\rightarrow AD^{0-m}$ For A_1/A_2,B_1/B_2 input $(A_1)-(B_1)\rightarrow AD^{n-31}$ $(A_2)+(B_2)\rightarrow AD^{0-m}$ where n=16 for F=0, n-24 for F=1, and m=15 for F=0, m=23 for F=1.	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	For A,B input: $(A\text{-input}^{n-31})+(B\text{-input}^{n-31})\rightarrow AD^{n-31}$ $(A\text{-input}^{0-m})-(B\text{-input}^{0-m})\ AD^{0-m}$ For A_1/A_2,B_1/B_2 input: $(A_1)+(B_1)\rightarrow AD^{n-31}$ $(A_2)-(B_2)\rightarrow AD^{0-m}$ where n=16 for F=0,n=24 for F=1, and m=15 for F=0, m=23 for F=1.	
	7	A+1	(A-input)+1→AD	

**If MULT=3,6,7,12,13,16,17 (i.e., DR input to M1) then DP becomes DR in ADDER and restriction does not apply.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DPA	0	NO-OP or omit	No operation	
	1	INC	(DPA)+1→DPA	
	2	DEC	(DPA)-1→DPA	
	3	DPA-8	(DPA)-8→DPA	
	4	SP	(SP_{SPA})→DPA	MA≠2,3,5-7; X1≠1
	5	DPA+SP	(DPA)+(SP_{SPA})→DPA	X1≠1,2
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	MA≠2,3,5-7; X1≠1,2
	7	WRT	Indicates that a data write of (AC)→MD_{MA} is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

Floating point multiply of DP,AC:

```
MULT:DPR,ADR  F=1  ADDER=AC,ACX:A
    FMT=TEST0 F=1  ADDI=DP,AC:A+B  ACO=ADL→ACL
     FMT=7
    FMT=TEST1 F=1  AC1=MP:ACR
    FMT=TEST0 F=1  ACO=FLOTAC ADDI=AC,AC
```

In the first instruction, the ADDER field transfers (AC) to AD; a multiply operation is initiated for (DP^{0-23}) and the "new" (AD^{0-23}).

In the second instruction, the essential result is to add the exponents of (DP) and (AC) and transfer the resulting exponent to AC^{24-31}.

THE THIRD INSTRUCTION IS A FILLER TO ALLOW TIME FOR COMPLETION OF THE MULTIPLY.

The fourth and fifth instructions store the result of the multiply in AC^{0-23} and round the result. A new multiply could be started in the fourth instruction.

TEST2/FORMAT 20

31 30	27	23	19	16	13	11	9	3	0
2	TEST2	JUMP	SPA	X2	MA	DP	AC2	ADDER	DPA

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
TEST2	1	SP=0	Jump if $(SP_{SPA})=0$	SPA must be same as previous value**
	2	SP≠0	Jump if $(SP_{SPA})\neq 0$	SPA must be same as previous value
	3	DPA≠7	Jump if (DPA)≠7	
	4	DPA≠37	Jump if (DPA)≠37	
	5	HDRDY=0	Jump if host data ready bit=0*, Set HDRDY	
	6	JMP	Unconditional jump	
	7	HDRDY=1	Jump if host data ready bit=1*	
JUMP	$0\text{-}17_8$	$0\text{-}17_8$	$(CB^{23-26})\rightarrow PSA^{0-3}$; i.e., jump to location within current page; if condition specified by TEST4 is satisfied.	
SPA	$0\text{-}17_8$	$0\text{-}17_8$	Indicates location in S-Pad to be set or used when X2=3,4,5,6 or MA=1,2,3,5,6,7 or DPA=4,5,6, or TEST2=1,2	
X2	0	NØ-ØP or omit	No operation	
	1	NØMI	No memory initiate; i.e., if MA field is set, value is transferred to MA, but no memory read is initiated.	MA≠1
	1	AC→C	(AC)→C	
	2	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	
	3	ACR+SP→SP	$(ACR)+(SP_{SPA})\rightarrow(SP_{SPA})$	DPA≠4-6; MA=4
	3	MAFN→SP	Indicates that the function specified in the MA field is to be stored in SP_{SPA}, rather than MA or DRA	DPA≠4-6; MA≠4
	4	MA→SP	$(MA)\rightarrow SP_{SPA}$	DPA≠5,6
	5	DPA→SP	$(DPA)\rightarrow SP_{SPA}$	DPA≠5,6
	6	DPA+SP+1→DPA	$(DPA)+(SP_{SPA})+1\rightarrow DPA$	DPA≠1-6
	7	AC→PS	$(AC)\rightarrow PS_{DPA}$: note that Data Pad Address register points to location in PS.	

* The test is repeated until the test condition is satisfied. Then the rest of the instruction is executed and the jump is performed.

** The following restrictions apply to the previous instruction:

a) SP must not have been written

b) REG or REGI≠MA nor DPA.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
MA	0	NO-OP or omit	No operation	
	1	SP	$(SP_{SPA})\rightarrow MA$	-
	2	SP+1	$(SP_{SPA})+1\rightarrow MA$	DPA≠4,6
	3	SP-1	$(SP_{SPA})-1\rightarrow MA$ Initiates MD read (see Sec. 2.3)	DPA≠4,6
	4	MA+1	$(MA)+1\rightarrow MA$	
	5	SP+8	$(SP_{SPA})+8\rightarrow MA$	DPA≠4,6
	6	SP+MA	$(SP_{SPA})+(MA)\rightarrow MA$	DPA≠4,6
	7	DRA+SP	$(DRA)+(SP_{SPA})\rightarrow DRA$ (Initiates DR read (see Sec. 2.3)	DPA≠4,6
AC2	0	NO-OP or omit	No operation	
	1	AD→AC	(AD)→AC.	
	2	HD→AC	(HD--host computer data lines)→AC, HDRDY=0	
	3	C→AC	(C)→AC, IØDRDY=0	
DP	0	NO-OP or omit	No operation	
	1	ACL→DPL	$(AC^{16-31})\rightarrow DP^{16-31}_{DPA}$	
	2	ACR→DPR	$(AC^{0-15})\rightarrow DP^{0\ 15}_{DPA}$	
	3	AC→DP	$(AC)\rightarrow DP_{DPA}$	
ADDER	General form is A-input,B-input:function, where A-input,B-input =			
	0	DP,AC	(DP_{DPA}), (AC) used in function defined	DP=0
	1	DP',AC	1's complement of (DP_{DPA}),(AC) used in function defined	DP=0
	2	AC,ACX	(AC),(AC)-cross* used in function defined	
	3	0,ACX	0,(AC)-cross* used in function defined	
	4	DP,ACX	(DP_{DPA}), (AC)-cross* used in function defined	DP=0

*(AC)-cross means the two 16-bit fields of AC are reversed; ie., $(AC^{0-15})\rightarrow AC^{16-31}$, $(AC^{16-31})\rightarrow AC^{0-15}$.

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
	5	ACL/ACR,DPR/0	(AC^{16-31}),(AC^{0-15}) used in function defined, result into AD^{16-31};(DP^{0-15}_{DPA}) 0 used in function defined, result into AD^{0-15}.	DP=0
	6	0,MD	0,(MD_{MA}) used in function defined. This option is used to retrieve memory data for which a read memory cycle was initiated at least four clocks (500 ns) earlier.	
	7	DPL'/DPR, ACL+1/ACR	1's complement of (DP_{DPA}),(AC^{16-31}) +1 used in function defined, result stored in AD^{16-31};(DP^{0-15}_{DPA}),(AC^{0-15}) used in function defined, result stored in AD^{0-15}.	DP=0
and where function =				
	0	A-1	(A-input)-1→AD	
	1	A	(A-input)→AD	
	2	A+B	(A-input)+(B-input)→AD	
	3	A+B+1	(A-input)+(B-input)+1→AD	
	4	A-+B	For A,B, input: $(A\text{-input}^{16-31})-(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})+(B\text{-input}^{0-15})\rightarrow AD^{0-15}$ For $A_1,B_1;A_2,B_2$: $(A_1)-(B_1)\rightarrow AD^{16-31}$ $(A_2)+(B_2)\rightarrow AD^{0-15}$	
	5	A-B	(A-input)-(B-input)→AD	
	6	A+-B	For A,B input: $(A\text{-input}^{16-31})+(B\text{-input}^{16-31})\rightarrow AD^{16-31}$ $(A\text{-input}^{0-15})-(B\text{-input}^{0-15})\rightarrow AD^{0-15}$ For $A_1,B_1;A_2,B_2$: $(A_1)+(B_1)\rightarrow AD^{16-31}$ $(A_2)-(B_2)\rightarrow AD^{0-15}$	
	7	A+1	(A-input)+1→AD	

Field	Octal Code	Assembler Code	Resulting Action/Indication	Restrictions
DPA	0	NO-OP or omit	No operation	
	1	INC	(DPA)+1→DPA	X2≠6
	2	DEC	(DPA)-1→DPA	X2≠6
	3	DPA-8	(DPA)-8→DPA	X2≠6
	4	SP	(SP_{SPA})→DPA	MA≠2,3,5-7; X2≠6
	5	DPA+SP	(DPA)+(SP_{SPA})→DPA	X2≠3-6
	6	SWAPSP	(DPA) and (SP_{SPA}) interchanged	MA≠2,3,5-7; X2≠3-6
	7	WRT	Indicates that a data write of $(AC)\rightarrow MD_{MA}$ is to occur instead of data read from MD. Applies to this instruction if MA is set in this instruction; otherwise, applies to next instruction in which MA is set.	

Example:

```
FMT=TEST2;HDDRDY=1  JUMP=15  SPA=12  X2=DPA+SP+1→DPA  AC2=HD→AC
DP=AC→DP
```

With this instruction, when the host data-ready bit is equal to 1, the contents of AC are transferred to DP_{DPA}, the host input data is tranferred to AC, HDRDY is cleared, DPA is incremented by the contents of SP_{12}+1 and a jump to instruction 15 of the current page.