AD-A036 874 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394-VOL-8 NL

1 OF 3
AD
A036874

AD A036874

12

RADC-TR-76-394, Volume VIII (of eight)
Final Technical Report
January 1977

COMMUNICATIONS PROCESSOR SYSTEM

North Electric Company

DDC
MAR 14 1977
RECEIVED

Approved for public release;
distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

Because of the small size of each volume, volumes have been combined into the following reports: Volumes I - III, Volumes IV & V, Volumes VI & VII, and Volume VIII.

This report has been reviewed by the RADC Information Office (OI) and is releasable to the National Technical Information Service (NTIS). At NTIS it will be releasable to the general public, including foreign nations.

This report has been reviewed and approved for publication.

APPROVED: Daniel J. McAuliffe

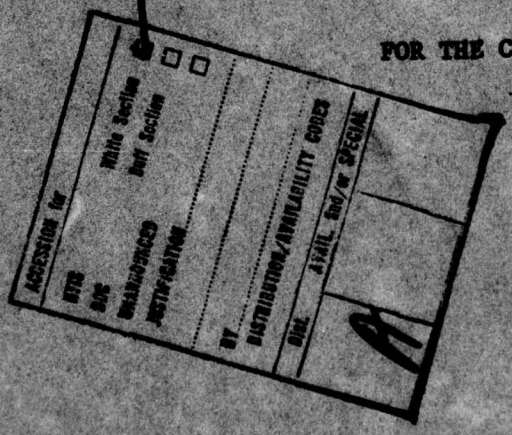
DANIEL J. McAULIFFE
Project Engineer

APPROVED: Fred I. Diamond

FRED I. DIAMOND
Technical Director
Communications & Control Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office



Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(19) REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (18) RADC-TR-76-394, Vol VIII (of eight)	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER (9)
4. TITLE (and Subtitle) (6) COMMUNICATIONS PROCESSOR SYSTEM.	5. TYPE OF REPORT & PERIOD COVERED Final Technical Report. June 1973 - March 1976.
	6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) (10) Mr. Kenneth Hagstrom, Dr. Boris Beizer, Data Systems Analysts	8. CONTRACT OR GRANT NUMBER(s) (15) F30602-73-C-0314
9. PERFORMING ORGANIZATION NAME AND ADDRESS North Electric Company, 553 South Market Street, Galion OH 44833	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) 62702F, 45191904 (17) 19
11. CONTROLLING OFFICE NAME AND ADDRESS (11) Rome Air Development Center (DCLT), Griffiss AFB NY 13441	12. REPORT DATE January 1977
	13. NUMBER OF PAGES 140 226 pp
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same	15. SECURITY CLASS. (of this report) UNCLASSIFIED
	15a. DECLASSIFICATION DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)
Same

18. SUPPLEMENTARY NOTES
RADC Project Engineer: Daniel J. McAuliffe (DCLT)
Because of the small size of each volume, volumes have been combined into the following reports: Volumes I - III, Volumes IV & V, Volumes VI & VIII, and Volume VIII.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
Communications Switching, Communications Processors, Processor Architecture, Circuit Switching, Message Switching, Packet Switching, Base Distribution

259 750

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)
This report covers the results of a study to develop a hardware architecture which will be the basis for a family of communications processors for applications processors for application in circuit, message, packet and base communications switch configurations. Over 23 switching equipments were investigated from which a functional baseline was defined for use in the subsequent studies for evolving an advanced Communications Processor System (CPS) architecture. These switches included circuit, message, and packet switching

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

bpp

equipments which were felt to be typical of traditional and advanced switching concepts. As an example, the AN/TTC-39, AUTODIN, ARPA Network were used as part of the circuit, message and packet switching baselines respectively. Air Force Base Communications studies were used as the baseline in that area.

From this baseline, a set of fifteen primitive functions were derived which represent the needed capabilities for any generally applied communications processor (CP).

The latest in the state-of-the-art in ADP technology was investigated to determine the best and most viable approach to the CPS. The goal being to develop a family of CP's which could be used to satisy switching needs for circuit, message, packet, base communications applications or in an integrated node of the future.

The results of the investigation were continually the subject of trade-offs through the use of an analytical modelling technique. The final outcome of this effort is a ten part specification detailing the performance requirements of each unit comprising the communications processor.

*This report is organized into eight volumes as follows: Volume I - Executive Summary; Volume II - Definition of Problem; Volume III - Modelling; Volume IV - CCC Architecture; Volume V - CPS Architecture; Volume VI - Software Generation; Volume VII - Appendices; and Volume VIII - CPS Central Processor Specification.

VOLUME VIII

SECTION I

GENERAL SPECIFICATION
FOR THE
CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE
CENTRAL PROCESSOR
OF THE
CENTRAL PROCESSING SYSTEM (CPS)

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	I-1
1.1	General	I-1
1.2	Background	I-1
1.3	Technical Assumptions	I-2
1.4	Specification Organization	I-2
2.0	Applicable Documents	I-3
2.1	Published Documents	I-3
2.1.1	Integrated Circuit/Message Switch Feasibility Model Development Final Report	I-3
2.1.2	Communications Processing System Study Final Report	I-3
2.2	CPS Program Documents	I-3
2.2.1	Prolegomenon to the Architecture of the Communications Switching Processor	I-3
2.2.2	Communications Processing System Study Interim Report	I-3
3.0	Central Processor (CP) Specification	I-3
3.1	General	I-3
3.2	General Characteristics of the CP Architecture	I-4
3.2.1	Units	I-4
3.2.1.1	Processing Unit (CPU)	I-6
3.2.1.2	Interrupt Control Unit (ICU)	I-6
3.2.1.3	General Purpose Channel Unit (CU)	I-6
3.2.1.4	Memory Unit (MU)	I-7
3.2.1.5	Memory-to-Memory Transfer Unit (MMTU)	I-7
3.2.1.6	System Monitor Unit (SMU)	I-7
3.2.1.7	System Clock Unit (SCU)	I-7

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.2.1.8	Bootstrap Unit (BU)	I-7
3.2.1.9	Performance Monitor Unit (PMU)	I-7
3.2.1.10	Matrix Unit (XU)	I-8
3.2.2	The Matrix	I-8
3.2.3	Maximum CP System	I-9
3.2.4	Inter-Unit Communication	I-9
3.3	General Features of Units	I-9
3.3.1	Control Cache Memory	I-9
3.3.2	Ports	I-10
3.3.3	Maximum Unit ID (MID)	I-10
3.3.4	Logical ID (LID)/Physical ID (PID)	I-10
3.3.5	ICU ID, Interrupts	I-10
3.3.6	Logical Independence	I-11
3.3.7	Instruction Stacking	I-11
3.3.8	Command Execution Speed	I-11
3.4	Priority Operation	I-11
3.5	The Unit as a Source	I-12
3.6	The Unit as a Destination	I-13
3.7	Generic Unit Micro-Commands	I-14
3.7.1	General	I-14
3.7.2	Command Formats and Modes	I-15
3.7.2.1	General	I-15
3.7.2.2	Person Mode	I-15
3.7.2.3	Chaining Mode	I-16
3.7.2.4	Indirect Mode	I-17
3.7.2.5	Command Length	I-17
3.7.2.6	Non-Descriptors	I-17
3.7.3	Single Character Commands: SS = 00	I-18
3.7.3.1	Transfers	I-18

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.7.3.2	Non-Transfers	I-18
3.7.4	Two Character Commands: SS = 01	I-20
3.7.4.1	Transfers	I-20
3.7.4.2	Non-Transfers	I-21
3.7.5	Three Character Commands: SS = 10	I-22
3.7.5.1	Transfers	I-22
3.7.5.2	Non-Transfers	I-22
3.7.6	Four Character Commands: SS = 11	I-22
3.7.6.1	Transfers	I-22
3.7.6.2	Non-Transfers	I-22
3.8	Input/Output (I/O) Operations and Inter-Unit (IU) Repertoire	I-23
3.8.1	General	I-23
3.8.2	I/O Commands and Channels	I-23
3.8.2.1	General	I-23
3.8.2.2	Overview of Command Operations	I-24
3.8.3	IU Command Modes	I-25
3.8.3.1	General	I-25
3.8.3.2	Termination Modes	I-27
3.8.3.2.1	General	I-28
3.8.3.2.2	Error Terminations	I-28
3.8.3.2.3	Interrupt Terminations	I-28
3.8.3.2.4	Condition Chaining Mode	I-29
3.8.3.2.5	Command Termination Mode	I-30
3.9	Controls and Consoles	I-30
3.9.1	General	I-30
3.9.2	Unit Alarms and Controls	I-30
3.9.2.1	Matrix Units	I-31
3.9.2.2	Memory Units	I-31
3.9.2.3	Channel Units	I-31

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.9.2.4	Memory-to-Memory Transfer Units	I-31
3.9.2.5	Processing Unit	I-31
3.9.2.6	System Monitor Unit	I-31
3.9.2.7	System Clock Unit	I-31
3.9.2.8	Bootstrap Unit	I-32
3.9.2.9	Performance Monitor Unit	I-32
3.10	Devices	I-32
3.10.1	General	I-32
3.10.2	Moving Head Disc	I-32
3.10.2.1	General Characteristics	I-32
3.10.2.2	Controller	I-33
3.10.3	VDU	I-34
3.10.3.1	VDU Characteristics	I-34
3.10.3.2	Controller	I-34
3.10.4	Keyboard/Printer	I-34
3.10.4.1	General Characteristics	I-34
3.10.4.2	Controller	I-34
3.10.5	Digital Line Termination Buffer	I-34
3.10.6	Low Speed Multiple Line Buffer	I-35
3.10.7	Scanner/Distributor	I-36
3.10.7.1	General Characteristics	I-36
3.10.7.2	Principles of Operation (Scanner Functions)	I-36
3.10.7.3	Controller Scan Functions	I-38
3.10.7.4	Output Functions	I-40
3.10.7.5	Applications	I-41
3.10.7.5.1	Circuit Switching	I-41
3.10.7.5.2	Message and Packet Switching	I-42
3.10.7.6	Technical Control Functions	I-42

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.11	Application Considerations	I-43
3.11.1	General	I-43
3.11.2	Typical Message Switch	I-43
3.11.2.1	Functional Configuration	I-43
3.11.2.2	Sacrifice Schedule	I-45
3.11.2.3	Physical Configuration	I-45
3.11.3	Packet Switch	I-47
3.11.3.1	Functional Configuration	I-47
3.11.3.2	Sacrifice Schedule	I-47
3.11.3.3	Physical Configuration	I-47
3.11.4	Circuit Switching	I-47
3.11.4.1	Functional Configuration	I-47
3.11.4.2	Physical Configuration of Large Circuit Switch	I-51
3.11.4.3	Configuration of Moderate Circuit Switch (600 Lines)	I-51
3.11.4.4	Very Small Circuit Switch	I-55
3.11.5	Technical Control System	I-55
3.11.6	Digital Concentrator	I-55

ABBREVIATIONS

BU	Bootstrap Unit
CP	Central Processor
CPS	Communications Processing System
CPU	Communications Processing Unit
CRC	Cyclic Redundant Code
CU	Channel Unit
DID	Destination Identity
DLE	Delete
FE	Front End
FIFO	First In-First Out
ICU	Interrupt Control Unit
ID	Identity
I/O	Input/Output
ITS	In-Transit Storage
I/U	Inter-Unit
LID	Logical Identity
MID	Maximum Identity
MMTU	Memory-to-Memory Transfer Unit
MP	Message Processor
MU	Memory Unit
PID	Physical Identity
PMU	Performance Monitor Unit
SCU	System Clock Unit
SID	Source Identity
SMU	System Monitor Unit
SOB	Start of Block
VDU	Visual Display Unit
XU	Matrix Unit

LIST OF FIGURES

FIGURE	TITLE	PAGE
I-1	Central Computing Complex Architectural Overview	I-5
I-2	Scanner Operating Principles	I-37
I-3	Functional Configuration of Typical Message Switch	I-44
I-4	Message Switch Physical Configuration	I-46
I-5	Functional Configuration of a Small Packet Switch	I-48
I-6	Small Packet Switch Physical Configuration	I-49
I-7	Functional Configuration of a Circuit Switch	I-50
I-8	Physical Configuration of a Circuit Switch	I-52
I-9	Moderate Sized Circuit Switch - Functional Configuration	I-53
I-10	Physical Configuration of Medium Circuit Switch	I-54
I-11	Unattended Circuit Switch	I-56
I-12	Physical Configuration - Small Circuit Switch	I-57
I-13	Single Thread Small Circuit Switch	I-58
I-14	Functional Control of Tech Control System	I-59

1.0 SCOPE

1.1 General

This document is the specification for a Central Processor (CP) which is specifically designed for Communications Processing System (CPS) applications. These CPS applications include: circuit switching, message switching, packet switching, digital multiplexers and concentrators. In this context, CP refers to a set of equipment which contains Communications Processor Units (CPU's), Input/Output (I/O) channels, performance monitor units, channel units, control panels, some form of data and program memory and a means to interconnect them. Therefore, this specification describes a family of computer controlled units which together comprise a CP and which can be equipped in various combinations and numbers to perform, efficiently, any of the CPS applications or any combination of them.

1.2 Background

The CPS Study Program (refer to 2.1.2 below) considered, as part of the study, a wide variety of existing and planned communications systems in all of the CPS applications mentioned above. This baseline study was performed to define the requirements of a CP which could be used efficiently in any of these applications. The systems studied included the entire gamut of communications systems from very small to very large in terms of lines, trunks, circuits, and terminals, from low traffic per inlet to very high traffic per inlet, and from individual systems to entire interconnected networks.

One of the major conclusions derived from this study was that any CP which could fit any of these applications efficiently must be highly flexible in terms of size (the number of units equipped) and capability (the types of units equipped) without changing the basic architecture.

A second major conclusion was that, for efficient operation, the ability to make maximum use of the system's resources must be inherent in the CP's architecture.

Of all of the existing systems studied, none remained static during its service life. Each system had gone (or is going) through modifications to its hardware or software or both to accommodate new or expanded requirements. This leads to a third major conclusion; the CP's architecture must facilitate adaptation to new or expanded requirements with a minimum disruption of service and, preferably, be amenable to these changes while "on-line".

A fourth major conclusion reached was that all existing and planned processor controlled systems featured a distributed control architecture and that a distributed control architecture would best satisfy the future CPS requirements.

This specification describes a CP which satisfies the requirements of all of these major conclusions.

1.3 Technical Assumptions

The entire CP concept described in this specification is based on the assumptions concerning Integrated Circuit (IC) technology contained in the CPS study Statement of Work.

It is assumed that an IC technology will (or does exist) which will result in gate densities on the order of 2000 to 5000 per 150 mil square chip with a quiescent power dissipation of 1 milliwatt per chip (200 nW per gate). It is also assumed that the on-chip gate delays of this technology will be on the order of 3 to 5 nanoseconds. The technology projections described in Volume V, Section 3 of the CPS Study Final Report indicate that these assumptions are not unrealistic (refer specifically to the I^2L technology description).

1.4 Specification Organization

This document, the CP specification, actually consists of a set of eleven specifications. A general CP specification and a specification for each of the ten unit types which comprise a CP. The specification titles and the sections in which they are contained are listed below:

Central Processor (CP) Specification	Section I
Communications Processor Unit (CPU) Specification	Section II
Interrupt Control Unit (ICU) Specification	Section III
Channel Unit (CU) Specification	Section IV
Memory Unit (MU) Specification	Section V
Memory-to-Memory Transfer Unit (MMTU) Specification	Section VI
System Monitor Unit (SMU) Specification	Section VII

System Clock Unit (SCU) Specification	Section VIII
Bootstrap Unit (BU) Specification	Section IX
Performance Monitor Unit (PMU) Specification	Section X
Matrix Unit (XU) Specification	Section XI

This CP specification is a preliminary specification of a Central Processor which satisfies the functional requirements derived in the CPS Study Program. It is not a hardware specification as such, although a set of hardware specifications could be derived from it and the information contained in the CPS Study Final Report.

2.0 APPLICABLE DOCUMENTS

2.1 Published Documents

2.1.1 Integrated Circuit/Message Switch Feasibility Model Development, Test and Evaluation

RADC-TR-72-27, Volumes I, II, III and IV

2.1.2 Communications Processor System Study Final Report

RADC-TR-76- , Volumes I thru VII

2.2 Program Documents

2.2.1 Prolegomenon to the Architecture of the Communications Switching Processor; January 25, 1974

DSA 292-TR-7

2.2.2 Communications Processor System Study Interim Report

3.0 CENTRAL PROCESSOR (CP) SPECIFICATION

3.1 General

The CP is a distributed system architecture composed of semi-autonomous units. Advantage has been taken of large scale integrated circuit technology to make the units "smart". Many of the capabilities normally associated with a CPU have been redistributed and allocated to other units.

3.2 General Characteristics of the CP Architecture

Figure I-1 is an overview of the architecture. The heart of the architecture is an interconnection matrix which allows all compatible units to communicate with one another. The architecture has the following characteristics:

(1) Modularity at the unit level.

(2) Replication as needed at the unit level.

(3) Automatic graceful degradation and recovery.

(4) Self-failure detection.

(5) Built-in diagnostic facilities and commands.

(6) Built-in monitoring capabilities.

3.2.1 Units

A unit is the basic interchangeable, replicable element of the architecture. Examples of units are:

(1) General purpose processing unit (CPU).

(2) Interrupt control unit (ICU).

(3) General purpose channel unit (CU).

(4) Memory Unit (MU).

(5) Memory-to-memory transfer unit (MMTU).

(6) System monitor unit (SMU).

(7) System clock and timing unit (SCU).

(8) Bootstrap unit (BU).

(9) Performance monitor unit (PMU).

(10) Matrix unit (XU).

All units have the following characteristics:

(1) Common interface with the matrix and with each other.

(2) Unit generic micro-instructions.

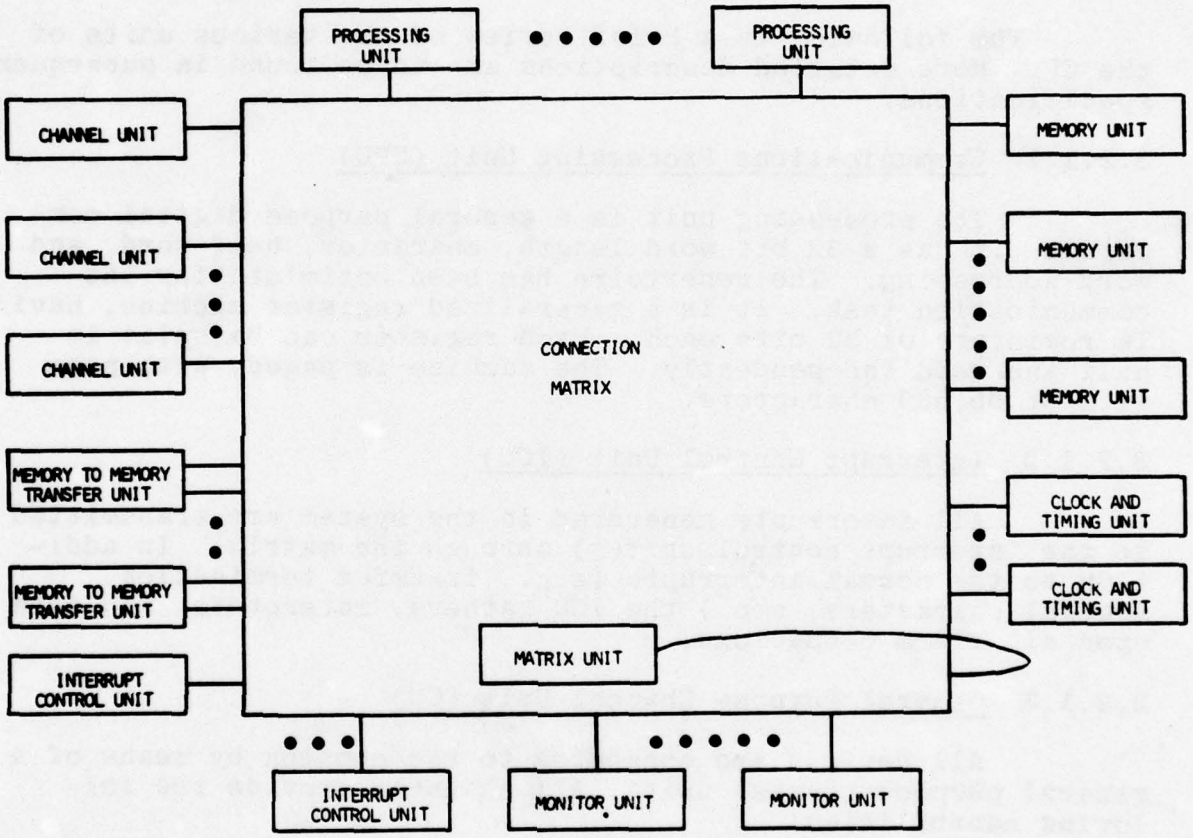


FIGURE I-1 CENTRAL COMPUTING COMPLEX ARCHITECTURAL OVERVIEW

(3) Dynamically modifiable logical identity which can differ from its physical identity.

(4) Interchangeability with any other unit of the same type.

(5) Self-failure and malfunction detection.

(6) Interconnection to other units via the matrix.

(7) Logically isolatable from other units.

(8) Power isolation from other units.

The following is a brief review of the various units of the CP. More detailed descriptions are to be found in subsequent specifications.

3.2.1.1 Communications Processing Unit (CPU)

The processing unit is a general purpose digital computer. It has a 32 bit word length, character, half-word, and word addressing. The repertoire has been optimized for the communication task. It is a generalized register machine, having 16 registers of 32 bits each. Each register can be split in half and used independently. The machine is paged, with page size of 65,536 characters.

3.2.1.2 Interrupt Control Unit (ICU)

All interrupts generated in the system are transmitted to the interrupt control unit(s) through the matrix. In addition to the normal interrupts (e.g., transfer termination, special characters, etc.) the ICU gathers, interprets, and acts upon all alarm conditions.

3.2.1.3 General Purpose Channel Unit (CU)

All devices are connected to the complex by means of a general purpose channel unit. All channels provide the following capabilities:

(1) Programmable speed

(2) Master or slave mode

(3) Direct memory transfers

(4) Chained I/O operation

(5) Full duplex operation

A channel may be used to terminate more than one physical device if a device controller of the proper type has been designed. The device side of the interface is a character interface. The matrix side of the interface is logically a word interface. The device side interface is universal. All controllers must be built to serve that interface.

3.2.1.4 Memory Unit (MU)

The memory unit is a module of 65,536 characters. From the programmer's point of view, word, half-word, character, and bit addressing are provided.

3.2.1.5 Memory-to-Memory Transfer Units (MMTU)

The memory-to-memory transfer unit is used to make memory-to-memory transfers and other transfers between passive units. Its operation is transparent to the programmer. Each MMTU takes up two unit numbers.

3.2.1.6 System Monitor Unit (SMU)

The system monitor unit is a watchdog unit over the CPU's. It expects signals from the CPU's on a periodic basis. Should an expected signal fail to arrive on time, the monitor unit will initiate corrective action, ranging from re-initializing the faulty CPU to throwing it out of the complex and reassigning a standby CPU to the task.

3.2.1.7 System Clock Unit (SCU)

The system clock unit provides a master timing source and time of day clock for the entire complex. It is addressable and programmable.

3.2.1.8 Bootstrap Unit (BU)

A unit which can, under operator control, perform the loading of various control memories and unit ID's as necessary to get the system into operation.

3.2.1.9 Performance Monitor Unit (PMU)

The performance monitor unit is a unit which is equipped with high impedance probes used to monitor activities within the complex. It is used to gather statistics on the complex's performance and internal matrix traffic. Its primary use is for system tuning although it can be helpful in software diagnosis.

3.2.1.10 Matrix Unit (XU)

The matrix is a distributed system containing a number of submatrices. Under normal operation, there is no need to explicitly access a submatrix or its controls. However, in the event of a submatrix failure, submatrix restoral, system bootstrap, or other conditions described elsewhere, it is necessary to force the state of the matrix and/or to query its status. For this reason, each submatrix control is also accessible as a unit through data paths that traverse the matrix itself.

3.2.2 The Matrix

The matrix is a distributed, modular structure with inherent graceful degradation capabilities. It is the means by which data and control paths are established between all units of the complex.

The connection duration is typically of the order of a few memory cycles to a few milliseconds, but can be modified every memory cycle if necessary. The operation of the matrix is for the most part transparent to the programmer. The matrix is the functional counterpart of the various busses found in previous system architectures. The matrix control is performed by a distributed control system and shares the graceful degradation capabilities of the matrix proper. Failure of part of the matrix may cause the system to slow down, but will not cause it to fail. Requests for connections are not directly under program control but are derived from the nature of the transfer or instruction being executed. The matrix is self-diagnosing and creates interrupts in the event of improper connection attempts.

The following conventions are used to establish a path through the matrix:

(1) A connection is normally initiated by an active unit.

(2) The initiating unit is called the "source unit". Its logical identity is called the "source identity" (SID).

(3) The receiving unit is called the "destination unit". Its logical identity is called the "destination identity" (DID).

(4) Every unit has a unique logical identity - else there has been a malfunction.

(5) Source and destination have nothing to do with the direction of data transfer - only with the identity of the unit that initiated the transfer. A passive unit, is normally a destination unit. An active unit may be either a source or a destination unit. Some specialized units may be only source units.

A complete description of the matrix and the signaling scheme used to establish paths through the matrix can be found in Section XI, the Matrix Unit Specification.

3.2.3 Maximum CP System

A complete CP system can consist of no more than 254 units. Units are categorized as being either active or passive. Memory and matrix units are passive. Most other units are active.

3.2.4 Inter-Unit Communication

Communication between units and control over units is obtained through the use of a unit level micro-instruction repertoire. There is a substantial amount of overlap between the unit repertoire and the I/O repertoire of the system; in many cases, the commands are indistinguishable. Unit level commands fall into two categories: unit generic commands and unit specific commands. Generic unit commands are identical for all units and can be applied to all units. The behavior of the unit under a generic command will be the same (except for some details) for all units. A given OPCODE in the generic repertoire can be expected to have the same action in all units. A unit specific command has an interpretation which is peculiar to the unit. The same OPCODE presented to some other unit could result in totally different actions. The unit generic repertoire will be discussed in this specification. The unit specific repertoire will be discussed in the specification pertaining to those units.

3.3 General Features of CP Units

3.3.1 Control Cache Memory

Every unit has a high speed cache memory used for priority control and instruction validation. The memory can range in size from 128 bits to 512 characters. The specific size is variable and the operation differs from unit to unit. As a minimum, the control memory specifies whether or not a command can be accepted from a given SID, and with what priority that command should be executed. The validation/priority field is from one to four bits in length. The code "0" always means that

a command cannot be accepted from the SID in question. The operation of the priority/validation memory consists of using the received SID as an index to the control memory and thereby fetching the priority/validation field. The control memory may contain other fields whose interpretation is peculiar to the unit.

3.3.2 Ports

A unit may have one or more ports into the matrix. A port is used to establish a path with some other unit. Only one port in each of two interconnected units is allowed to participate in the connection. Which ports are involved is arbitrary. Thus, if unit A has two ports, and unit B has four ports, only one of unit A's ports can be connected to one of unit B's ports. One and only one of a unit's ports can be designated as the primary port. The designation of the primary port is done by use of a micro-instruction. The primary port establishes tie-breaking in the event that two commands at the same priority are active simultaneously. Ports are numbered sequentially from 1 through N. The port numbering cannot be modified. Port priority is established (in decreasing order) by counting cyclically from the primary port. Thus, if a unit had four ports, and port #3 had been designated as the primary port, the port priority order would be 3, 4, 1, 2. Ports which are CLOSED are skipped in the priority ordering.

A port can have states which correspond to the states of a matrix link, that is: FREE, RESERVED, BUSY, LOCKED, ERROR, DEAD, as well as other intermediate transient states. Port states are partially controllable through the use of unit micro-instructions.

3.3.3 Maximum Unit ID (MID)

Every unit stores as a constant the numerically largest unit ID to which it is allowed to respond. This may correspond to the numerically largest ID in the system, but does not have to. The MID is changeable through the use of a unit generic micro-instruction.

3.3.4 Logical ID (LID)/Physical ID (PID)

Every unit, except XU's, store their own programmer modifiable LID, as well as their own physical ID. Physical ID's cannot be modified under program control. Physical ID's can be read through the use of a unit generic micro-instruction.

3.3.5 ICU ID, Interrupts

Every unit stores the logical ID of two other units, to

which it is to respond should an interrupt be required. Typically, these are the ICU and the back-up ICU, however, this is a usage convention and can be modified under program control. Interrupts may result from normal unit operation or from malfunctions. When these occur, the unit responds by transmitting an interrupt code along with pertinent interrupt related information. The interrupt code and the associated information is peculiar to the unit and may be interpreted differently from unit type to unit type. The primary and secondary ICU ID are changeable through the use of unit generic micro-instructions.

3.3.6 Logical Independence - Power Transient Protection, Malfunction Detection

Units are to the maximum extent feasible, electrically and logically independent of one another. Interfaces have been designed with safe logical values so that open line conditions will not impact the rest of the system. Some units have their own power transient protection (e.g., memory units). Illogical conditions are detected by the unit's logic where possible and result in the generation of an appropriate interrupt.

3.3.7 Instruction Stacking

Each unit is capable of storing one instruction per port. Some units may be capable of stacking more than one instruction per port. Instruction stack entries are not tied to the port as such. That is, a unit with 8 stack entries could stack three commands for one port, two for another, etc.

3.3.8 Command Execution Speed

While a given command such as an I/O command could entail many transfers over a long period of time, each such operation, when examined as a sequence of micro-instructions, takes place at the maximum matrix/unit speed. Thus, the transfer of characters for a channel operating at 100 characters per second, would be treated at the micro-instruction level, as 100 separate single character transfers every second.

3.4 Priority Operation

The following priority rules apply to all units:

(1) Unit (SID) priority as stored in the cache memory (if stored) is applied first.

(2) If there is no stored priority, or if the priorities for two SID's are the same, port order applies.

(3) If the unit is a single port unit (actually or effectively by virtue of ports being closed), or if otherwise command priority has not been resolved by the above rules, then unit specific priorities based on the OPCODE of the instruction are applied.

(4) If the priority has not been resolved through the use of rules 1 through 3 above, then FIFO applies.

(5) The exception to the above rules is the ABORT GLOBAL command which aborts all pending commands and thereby takes priority.

3.5 The Unit as a Source

A unit may issue a command directly as a result of its own operation, or indirectly as a result of the operation of some other unit. Furthermore, the command may have come from within itself, or may have been supplied externally - again through the action of some other unit. However, many steps of indirection are involved, ultimately the unit comes to the point where it will issue a command to some other unit. The following steps are involved in the unit's behavior as a command source:

1. Select Port

The highest priority (cyclic order from the primary port) free port is selected for the issuance of the command, unless a RESERVED or LOCKED port to the DID in question already exists.

2. The DID, SID, and OPCODE character(s) are transmitted sequentially without waiting for a validation in the form of an echo DID. It is (correctly) assumed that in most cases the command is valid and that it will get through the matrix. If the reflected DID does not match that which was transmitted, the path kill signal is sent, and the transmission of the OPCODE is curtailed. An interrupt is fabricated for transmission to the primary ICU. The interrupt contains: the SID, DID, OPCODE, PORT NUMBER, and the interrupt code "BAD DID REFLECTED". The interrupt is sent on a different port if possible.

3. Transmission of the rest of the command (SID and OPCODE) implies that someplace along the line, the destination unit will signal back its tentative acceptance of the command. If command is not accepted, from the point of view of the source, a PATH KILL signal will appear. In this case, the source unit will signal the fact by generating an interrupt with the same information as

before, except with a "COMMAND REJECTED" interrupt code. Mere failure to get through to the destination unit, as signalled by a BUSY, does not result in an interrupt for the typical unit level command.

The transmission of the OPCODE (which is of a known length) sets off a timer (actually a counter of character transmission periods) in the source units. The counter establishes a window within which the destination unit must respond with a LOCK signal. The window time includes the transmission delays of the matrix for the round trip. Should the LOCK signal, or BUSY - in general, a state change signal - fail to appear within the window, it is assumed by the source unit that the destination unit has failed to get all the characters of the command, or is expecting more characters than are forthcoming. The source unit will respond to this situation by the generation of an appropriate interrupt.

4. During the data transfers (if that is the nature of the operation) both the source and destination units establish windows based on the length of the transfer and monitor the duration thereof to see to it that all characters and no more than were expected are transferred. Recall that the direction of data transfer does not necessarily relate to which unit was a source or which unit was a destination.

5. The completion of the transfer or instruction execution is signalled by the destination unit. This results in changing the port state and the state of the link attached to the port to the RESERVED state, unless the path had been locked by a LOCK command.

A unit may initiate a command prior to the completion of commands on other ports. At any instance of time, however, there is only one micro-command active for a given port. The ability to issue simultaneous commands (on separate ports) is a peculiarity of the unit.

3.6 The Unit as a Destination

A unit's destination behavior is complementary to its behavior as a source. The following steps are entailed in the unit's behavior as a destination:

1. There is no port selection, except that which is implicit in the traffic pattern through the matrix. The connection attempt is made on a free port, or in the case of usurpation, on a busy, reserved, or locked port.

2. The received DID is compared, bit serially, by the destination unit. At the first instance that it has been determined that there is mismatch, the unit responds with the NO signal, which will have the effect of clearing that path. Note that the DID code could come into more than one port simultaneously. Once the DID has been validated and found to match, the destination unit sends it out as a reflected DID.

3. The SID is accepted and is used as the first part of command validation. At the first bit in which it is determined that the SID exceeds the MID stored in that unit, the unit responds with a command rejection signal back to the source and an interrupt to the ICU, containing the SID, DID, PORT NUMBER, and the code "SID REJECTED".

4. OPCODE validation proceeds as the OPCODE is received. Validation may require that more than one character of the OPCODE be examined. The validation rules are peculiar to the unit. An invalid OPCODE is rejected by a path KILL signal and a transmission of an interrupt to the ICU containing the SID, DID, and "OPCODE REJECTED" code in the interrupt.

   As with the source unit, the destination unit establishes a window for the receipt of the complete OPCODE. In the case of the destination unit, matrix delays do not have to be taken into account. If the OPCODE should be curtailed or extended (e.g., extra characters) an appropriate interrupt will be generated.

5. The completion of the transfer or the instruction execution is signalled by changing the state of the path.

6. If the same SID should appear simultaneously on two or more ports, the responding port is chosen in port order. The unused ports are given a path KILL signal.

3.7 Generic Unit Micro-Commands

3.7.1 General

This section is a discussion of the generic unit micro-commands; that is, those commands which apply to all units. It should be pointed out that the term "generic" can be occasionally misleading. It is not absolutely true that all generic commands in all modes can be issued and acted upon by all units. Some modes, for example, the indirect mode, can only apply to units which have or are memory units. In some cases, the interpretation and details of a "generic" command will differ from unit to unit - but the general action and intent will be the same.

Unit specific commands fall into the OPCODE structure of the generic commands and are discussed separately for each unit type. Each unit contains the logic to validate commands that it receives. When command validation fails, the destination unit (the one that interprets the command) generates an interrupt to the ICU.

3.7.2 Command Formats and Modes

3.7.2.1 General

A unit micro-command is a string of characters of variable length. The first character (after the SID and DID characters) is either a command or the descriptor of a command. Command descriptors are used to provide various modes of operation for commands. The first bit of the command is the D bit. If D = 0, the character is a command. If D = 1, the character is the descriptor of a command, which is to be found as the next character in the command stream. A descriptor can precede any valid command. The descriptor can specify a combination of three modes of operation: first person/third person, direct/indirect, chained/unchained. These terms are defined in the sequel. If a command is not preceded by a descriptor, it is in the first person, direct, unchained mode. The overwhelming majority of unit micro-instruction executions are in this mode. If the D bit of the command is 1, the next three bits specify the mode of the command as follows:

BIT 2 - First or third person mode

BIT 3 - Direct or indirect mode

BIT 4 - Chained or unchained mode

BITS 5, 6 - Command type descriptor
00 = generic unit micro

BITS 7, 8 - Termination code
00 = normal termination for generic unit micros.

3.7.2.2 Person Mode

Three different units are potentially involved in the execution of a command: the unit that initiates the command (first person - "I"), the unit that receives the command (second person - "YOU"), and possibly a unit that is the object of the command if different from the first two units (third person - "IT"). The forms that a command can take, then, are:

"I tell you to do such and such" (first and second person)

or

"I tell you to do such and such with it" (first, second, and third person).

In the first case, the source and destination are the participants in the command. In the latter case, the source directs the destination to become a source for the execution of a command with another destination.

We call these modes the "first person" and "third person" mode respectively. Generic unit commands and unit specific commands, when used as micro-instructions which form a part of a higher level command, are normally executed in the first person. When unit commands are used as part of I/O commands, they are generally used in the third person mode. Thus, a CPU issuing a command to a channel, directing it to transfer data to a memory is a clear example of the use of the third person mode. On the other hand, the same CPU requesting that a unit transfer its status to a CPU register is an example of a first person mode command.

A command given in the third person is identical to a first person command, except that immediately preceding the remaining characters of the command is the logical ID of the "third person" that is to be involved in the execution of the command. The destination unit stores the command and the UID of the third person unit, completes the command sequence with the source unit, frees the path, and thereafter, reissues the command as if it had originated with the destination unit. The destination unit therefore becomes the source unit for the reissuance of the command. The only change that occurs to the first character of the command is to change the person bit from the third person mode to the first person mode. Validation windows are adjusted to take the person into account.

3.7.2.3 Chaining Mode

This bit indicates that the command in question is part of a chain of commands. In addition to the other command parts, a unit ID (typically that of a memory) and an address (another two characters) is provided. The unit which executed a chained command retains the unit ID and the address. Upon completion of the present command, it fetches a new command from the indicated memory location. Chaining applies to either person modes. Chaining continues until an unchained command has been reached. Note that the act of chaining allows a source unit to execute

a string of separate commands without intervention by the unit which initiated the chain. Command validation and priorities for chained commands are established with respect to the unit from which the command is fetched and, except for the first command of the chain, not with respect to the unit that initiated the chain. Once the chain has been kicked off, the unit fetches the commands from the other (the memory) unit and treats it as if the memory unit had initiated the command. From the point of view of the memory unit, the fetch is accomplished by a normal fetch micro (see below). It is the receiving unit that does the interpretation of this data as a command rather than as data.

3.7.2.4 Indirect Mode

The indirect mode is like the chaining mode except that there is no command following the descriptor character, only the unit ID and the address at which the command is to be found. Indirect mode commands specify where the unit is to find the command which it is to execute. The format is:

DID	SID	DESCRIPTOR	UID	ADDRESS	ADDRESS

16 CHARACTER

3.7.2.5 Command Length

While the issuing unit always knows the command length at the time of issuance, in the case of indirect or chained mode commands, the command length is determined by a combination of the descriptor and the first character, or more, of the command. Where the command itself is not to be found in the issuing unit, the number of additional characters to be transferred is determined by the unit, and depending on the nature of the command, another micro-instruction may be executed without releasing the port, or controls are established which will cause the proper number of additional characters to be transferred.

3.7.2.6 Non-Descriptors

If the first character is not a descriptor, as indicated by a D bit = 0, it is a command and has the following format:

0 SS UUUUU

The S bits specify the number of characters to follow the given character, excluding those added by virtue of the descriptor and the modes therein. A micro-command, therefore, can consist of 1, 2, 3, or 4 characters, corresponding to S bit values of 00, 01, 10, and 11, respectively.

3.7.3 Single Character Commands: SS=00

These commands are the most frequently used micro-commands. They have been encoded into single character OPCODES in order to minimize matrix transfer times. The first bit of the OPCODE designates if the command is a transfer. There are five bits in the OPCODE. The first bit designates a transfer or a non-transfer. If a transfer, the remaining bits designate the size and direction of the transfer. If a non-transfer, the remaining bits designate which of 16 micro-instructions it is.

3.7.3.1 Transfers - DSSU 0001XNNN

These commands are all of the form:

Transmit 2 - transmit or receive N characters.

The "X" bit designates whether the direction is a transmit or a receive. The NNN bits designate the number of characters to be transferred, as a number from 1 to 8. The code 000 is by convention used to denote a transfer of 8 characters. This is the work-horse micro-level transfer command. It is used between units and memory for almost all purposes. The "transmit-2" portion of the command is interpreted by the destination unit (typically a memory) as an address. We use the generic notation X2XN or X2RN to designate variations on this command; where "R" means "receive" or "read". Examples of the use of this command are:

X2R1 - single character read by a CPU, or cycle stealing read of a single character by a channel.

X2R2 - memory half-word read, as might be used by a CPU in an indirect address operation.

X2R4 - full word read, as might be used by a CPU in an operand fetch, or a normal instruction fetch.

X2X1 - Cycle stealing of a character into memory.

X2X4 - word transfer from CPU to memory or channel to memory.

3.7.3.2 Non-Transfers - DSSU 0000 XXXX

Mnemonic	Name	Description
GOOF	GO OFF-LINE	The logical ID of the unit is set to zero. The unit is reset and an indicator light on the unit is lit. The unit cannot be brought back to on-line operation except by pressing a

Mnemonic	Name	Description
		physical reset button.
REST	RESET	The unit is reset to the quiescent state with respect to everything but the logical ID which is unchanged.
CLOS	CLOSE	The unit will refuse to respond to any command except an OPEN command. The state of the unit is not changed otherwise.
OPEN	OPEN	Reverses the action of the CLOSE command.
POWR	POWER DOWN	Same as close except that the unit enters a special power down safety state and re-opens itself at the conclusion of the power transient. The unit will not respond to any command as long as it believes the power to be inadequate.
RTST	RETURN STATUS	The unit responds by transmitting a unit specific status code.
XORO	XORO	Transmit nothing, receive nothing. A noop operation used primarily for diagnostic purposes.
LOCK	LOCK PATH	The unit keeps the path (on which the command was received) locked until unlocked or usurped.
UNLK	UNLOCK PATH	Unlocks a path on which it is received. Has no effect on a path already unlocked.
FINT	FORCE INTERRUPT	The unit responds by transmitting an interrupt to the ICU. Giving the usual interrupt related details (CODE, SID, DID, etc.).
FUST	FETCH UNIT STATE	The unit responds by transmitting a character corresponding to its internal control state. This is a diagnostic command.
LART	LOCAL ABORT	Aborts all commands pending for that SID.

Mnemonic	Name	Description
GART	GLOBAL ABORT	Aborts all commands pending.
INXT	INCREMENT STATE	This is a diagnostic command. The internal state counter of the unit is treated as a gray-code counter. Receipt of this command causes the count to be increased by 1, and the unit is allowed to perform exactly one state change. The state resulting from this operation is transmitted back to the requesting unit. This command must be preceded by a LOCK command.
QUEP	QUERY PID	The unit responds by transmitting its physical ID to the requesting unit.
XXXX	UNASSIGNED	

3.7.4 Two Character Commands: SS=01

3.7.4.1 Transfer Commands

The bits of this command are defined as follows:

0011IDXX NNNNNNNN

The D bit is defined as before. The S bits are "01". The fourth bit, whose value is 1, indicates that this is a transfer. The I bit indicates the direction of the transfer - source to destination or destination to source, and the D bit specifies if the information transferred is data or if the information is control information, such as the contents of the unit's cache memory. Bits 7 and 8 are reserved for variations in the interpretation of this command. The second character of the command specifies how many characters are in the transfer. Note that there is no address in this command and the source or destination unit (depending upon the direction of the transfer) must be able to interpret where the data in question is to come from or go to. Examples of the use of this command are:

XORN 001101XX - transmit 0, receive NNNNNNNN

XOXN 001100XX - transmit 0, transmit NNNNNNNN

The convention NNNNNNNN = 0 is used to indicate a transfer of 256 characters. This command is used for block transfers from channel to channel at full channel speed where an address does not have to be supplied.

SST1 Set State 1 00110000

SST2 Set State 2 00110001

SST3 Set State 3 00110010

SST4 Set State 4 00110011

Used to set up to 1024 characters (in increments of 256 characters) of a unit's cache memory (the source) to the specified destination. These commands are referred to as SET-STATE N commands.

FST1 Fetch State 1 00111000

FST2 Fetch State 2 00111001

FST3 Fetch State 3 00111010

FST4 Fetch State 4 00111011

Used to fetch up to 1024 characters into a unit's cache memory. These commands are referred to as FETCH STATE N commands.

3.7.4.2 Non-Transfers 00100XXXX NNNNNNNN

SLID SET LOGICAL ID - the logical ID of the unit is changed to NNNNNNNN.

SMID SET UNIT MAX - the maximum value of the LID that the subject unit will accept commands from is set at NNNNNNNN.

SICU SET ICU ID - the ID of the primary ICU for that unit is set by this command to the value NNNNNNNN.

SISU SET SECONDARY ICU ID - the ID of the secondary ICU for this unit is set by this command to value NNNNNNNN.

CLPN CLOSE PORT N - port N of the unit is shut down and will not be used again until opened. If the unit does not have N ports, the command is ignored and the fact is signalled by an interrupt. This command is used to patch around certain types of matrix failures. If closure of the port will cause the unit to be left without an

open port, the fact will be signalled by an interrupt and the unit will enter the CLOSED state.

OLPN OPEN PORT N - this reverses the action of the CLOSE PORT N command. This command can be received on any free port, whether OPEN or CLOSED.

SPNP SET PORT N TO PRIMARY - port NNNNNNNN is thereafter designated as the primary port.

STAT SET STATE - the unit state is set to that specified by the second character.

3.7.5 Three Character Commands: SS=10

3.7.5.1 Transfers

Three character transfer commands are analogous to the two character transfer commands, except that a single character code can be transmitted preceding the transfer. The interpretation is unit dependent.

3.7.5.2 Non-Transfers

CNID CONVERT NULL ID - if the logical ID of the destination unit is 00000000 and its physical ID matches that which is transmitted as the second character of the command, its logical ID is changed to that which is found in the third character. This is an important command for restoring failed units or for bootstrapping a system.

3.7.6 Four Character Commands: SS=11

3.7.6.1 Transfers

The same transfer modes as were found in the two character commands are provided. The additional two characters are now interpreted as a memory address. This command is used to transfer blocks of data to and from memory, and more important, to load and store the contents of cache memories, or registers for an interrupt.

3.7.6.2 Non-Transfers

The following commands are available in a four character version. It is assumed that in this case, that one of the

units is a memory or other unit that can make use of a 16 bit address field. The effect is to load or store the information in question from or to memory as appropriate:

RETURN STATUS (TO MEMORY), FETCH STATE (AND STORE), QUERY PID (AND STORE), SET LOGICAL ID (FROM MEMORY), SET UNIT MAX (FROM MEMORY), SET STATE (FROM MEMORY), INXMT (AND STORE), SET ICUID (FROM MEMORY), SET SECONDARY ICUID (FROM MEMORY).

3.8 Input/Output (I/O) Operations and Inter-Unit (IU) Repertoire

3.8.1 General

The I/U repertoire ("Inter-Unit") of the CP is an independent instruction set that is used to regulate and control the inter-actions of units. It is in no way tied to the operation of CPU's. In fact, I/O operations and inter-unit operations can be initiated, executed, and terminated without the intervention of a CPU, should that be desired. Part of the I/U repertoire has already been presented - the unit generic micro-commands. The I/U repertoire has four primary uses:

(1) Input/Output Operations

(2) Bootstrapping

(3) Recovery and Reconfiguration

(4) Micro-Level Control of Inter-Unit Operations

The major distinction between I/U micro-commands and I/U macro-commands (or just "commands") is the existence of a command descriptor character. An I/U macro is a specification for the automatic execution of a number of I/U micro-commands. For example, a command to transfer data from a channel to a memory would result in a series of micro-level X2X4 commands sufficient to accomplish the entire transfer. The control logic is part of the unit that executes the command. In general, the receiving unit responds only at the micro-command level and has no need to distinguish whether this is a separate micro-command or part of a longer sequence of micro-commands that was initiated by an I/U macro.

3.8.2 I/O Commands and Channels

3.8.2.1 General

The CP engages in two types of communications: internal and external. Internal communications are for the most part transparent to the programmer and involve inter-unit transfers as discussed in this section and in subsequent sections for

individual unit types. External communications are two devices which are not themselves units. All devices are attached to the CP by means of channels and all channels are identical, except for the number of ports. All input/output operations are performed through a hierarchy that consists of: unit, channel, device controller, and device. Philosophically, an I/O command consists of a string of characters as follows:

UNIT GENERIC COMMAND

UNIT SPECIFIC COMMAND = CHANNEL GENERIC COMMAND = I/U MACRO

DEVICE CONTROLLER COMMAND

DEVICE COMMAND

At each level, the subsequent part of the command string is considered to be data. Thus, the channel, considered as a unit, treats the channel generic command as data as well as the device controller command and device command portions. The channel, in its role as a specific kind of unit, interprets the channel generic portion of the command, and treats the device controller and device commands as data. The device controller (if any) interprets its portion of the command, and passes on the device specific command to the device, which does the final interpretation. Each level in the hierarchy then treats the portion of the command that is to be interpreted by the next level as if it were data. I/O commands in the CP are therefore all constructed commands. Only unit generic commands can be issued directly under the I/U repertoire. The reader, however, should not jump to the conclusion that each I/O operation will require the explicit construction of a four level hierarchy of commands. This has been a philosophical rather than substantive discussion of the I/O repertoire. In practice, a major part of the above hierarchy of commands is automatically fabricated by the channel (see Section VI).

3.8.2.2 Overview of Command Operation

It is desired to set up a transfer of ten blocks of 1024 characters each from a moving head disc unit to a number of 256 character dynamically allocated data blocks contained in a memory unit. It is also desirable to have a record, in memory, of the fact that each 256 character block and each 1024 character block has been successfully transferred. Furthermore, it is necessary to have an interrupt at the conclusion of the entire sequence.

The process would start with a device imperative macro, whose device specific data would contain the information: "seek track N". The termination code for this command would result in the chaining of the fact that the seek had been done, in a pre-

determined location, as part of the termination condition of the I/U macro. This first command would be command chained to a transfer I/U macro that would specify the memory locations into which the information was to be placed. The transfer command would also contain device specific information that would specify "Sector 17". The command would have been specified as a sequence of 256 character transfers and data chaining would be used. The termination sequence of each 256 character subtransfer would specify that the fact of conclusion of transfer of that 256 characters be placed in a specified location. The transfer rate in the case of a disc is controlled by the disc, therefore, the channel would be slave to it. Buffering in the channel would have been arranged to have transfers occurring in groups of 4 characters each. As each buffer full was acquired, the channel would execute a unit level micro-instruction to transfer the accumulated group of four characters into the specified memory locations. This would be done through the use of an X2X4 micro-instruction. The paths would normally be freed between each group of four transfers, and potentially, the channel would be free to accept other commands from other units. All commands, except an ABORT command, however, would be rejected until the entire transfer had been finished. Each new sector READ would be initiated by another chained I/U macro. The command chain would continue in this manner, until the last command of the sequence. This command would have a termination specification for an interrupt, rather than a chaining of the termination data. If at any time during the course of the command execution, the device sensed a condition that required attention on the part of the channel, or some additional control information, the flow of data would be interrupted and a control code would be transmitted to the channel, indicating if it were an error termination, a normal termination, or a special condition. Any of these could be followed by auxiliary data which is passed on to another unit and otherwise ignored by the channel.

3.8.3 IU Command Modes

3.8.3.1 General

The following must be distinguished in any I/U command:

(1) The identity of the unit that initiates the command.

(2) The identity of the unit that receives the command.

(3) The identity of the unit where the command is to be found.

(4) The identity(ies) of the unit(s) that will participate in the command - that is, the units that will actually execute the command.

(5) The manner in which the command will be terminated.

(6) The identity of the unit (if any) that will respond to the completion of the command.

These unit identities, in the typical system tend to be locked to one another, and there is no flexibility in their assignment. I/U command initiation in the CP, unlike most other systems, is not restricted to a CPU. Most active units can initiate I/U commands. Most units can receive unit specific commands, which in the case of a channel unit, turns out to be I/O commands. The command itself may exist in a memory unit, or may be built-in to another kind of unit, such as a bootstrap unit. On the other hand, the command may be "found" in the unit that initiates it. Identification of the participating units is the most standard part of the I/U command specification. Typically, the participants are a channel and a memory, or two channels. Only rarely is the participant a CPU. Finally, I/U commands require a termination specification. This is not always done by means of an interrupt. In some cases, no action is required upon completion. The action could range from an interrupt (in which case a CPU would be named), to merely placing the completion code in memory (in which case a memory unit is involved), or it could result in a recovery action which could involve the bootstrap unit or the system monitor unit.

If the actual I/U command is part of the command initiation sequence - that is, the initiating unit and the unit where the instruction is to be found are identical - the command is said to be in the direct mode. If the instruction is to be found in some other unit, the instruction is said to be in the indirect mode. Most I/U instructions are executed in the indirect mode.

The I/U command might be an individual command or it might be part of a chain of commands wherein the completion of a command automatically initiates the execution of the next command in the chain, until the end of the chain has been reached. In this case, command chaining is said to be used. In an analogous manner, if the command involves a data transfer, the data itself can be chained: successive blocks of data would be transferred, under the same command, until the end of the data chain had been reached. In this case, data chaining is said to be used. Data chaining can operate under command chaining, so that the initiation of a single I/U command can result in the setting of a chain of commands each of which transfers its own, variable length, chain of data.

In addition to command level chaining, and data chaining, it is possible to specify condition chaining. This provides the automatic ability to chain codes generated by the device in response to special conditions detected regarding the operation in progress. Examples of the use of condition chaining are: normal terminations, short block terminations, detection of special characters or character sequences, detection of any other condition which must be noted, but for which interrupts are not necessary.

The designation of the command mode, as to direct/indirect, chained/unchained, and first-person/third-person is provided by the appropriate three bits in the command descriptor. If the first character of the command is not a descriptor, then by definition, the command is generic unit micro-command in the first-person, direct, unchained mode. All generic unit commands described in this section are potentially usable as individual I/U commands.

One category of commands, the generic unit micro-commands, have already been defined. Three more categories of commands are now added: channel imperatives, device commands, and inter-unit transfers. The channel imperative commands are unit specific commands in which the unit is a channel. The same format, and indeed some of the same commands may apply to other units and will be discussed in that context. The designation of the command category is done by the command descriptor bits, 5 and 6. Third person, direct/indirect, and command chaining modes apply to all four categories of commands. The various command categories will be discussed in subsequent sections.

A command can be terminated in one of four ways:

(1) Do nothing.

(2) Interrupt.

(3) Chained condition code.

(4) Another command.

The selection of the command termination mode is made by the use of bits 7 and 8 of the descriptor. Generic unit micro-commands are normally terminated in the do-nothing mode. If they are issued without a descriptor, then per force there is a do-nothing termination. Command termination modes will be discussed in the next subsection.

3.8.3.2 Termination Modes

3.8.3.2.1 General

The termination modes provided are intended to allow the adaptability of the CP to a wide class of devices, many of which have yet to be defined or put into production. The termination mode of an I/U command is a specification of what is to occur inter-unit at the completion of the command, and in some cases, to allow a means to deposit auxiliary data into memory that describes data being transferred, but is not part of the transfer data proper. For some devices, the desirable mode of termination is an interrupt. For other devices, the placement of status data on queue is the best approach. The choice is in general arbitrary, depending upon the peculiarities of the device and the application.

3.8.3.2.2 Error Terminations

Whatever termination mode has been selected, every unit and every device that might be connected to a channel can detect malfunction conditions and/or errors in the specification or execution of a command. All such error conditions, when detected, will force the curtailment of the active I/U command(s) for that unit, the curtailment of data and command chaining, and the generation of an interrupt with an appropriate interrupt code, to the ICU. The error termination mode does not have to be specified. It will always occur and will override any other termination mode that has been specified in that command or one of its component parts.

From the point of view of a channel, a device error is whatever the device chooses to call an error. Consequently, the device or the device controller interface logic can be designed to provide a pseudo error termination which will result in an interrupt, for conditions which are operationally not errors at all. This is a matter of device design and software. The "device error" mode can be used in combination with other termination modes to provide a conditional branch in the program based on conditions detected by the device. Again, this is a matter of device design and application software.

3.8.3.2.3 Interrupt Termination

An interrupt termination is specified by setting bits 7 and 8 of the command descriptor to 01. The conclusion of the command will be an interrupt, containing the SID, a unit level interrupt code, and a variable number of device supplied termination data. The programmer does not have to specify how many characters this is. The interrupt will be properly constructed by the channel, in response to the interrupt data provided by the device.

3.8.3.2.4 Condition Chaining Mode

The chaining mode provides a means by which the termination information can be placed into a specified linked list of termination data. This can be examined by a CPU at leisure. Often, the information is required merely for bookkeeping purposes and a timely response is not needed. Condition chaining may occur at any time. The selection between condition termination or interrupt termination is made by the device. If chaining has not been specified, then the occurrence of a special condition by the device will result in an interrupt - e.g., it will be treated as an error interrupt.

Condition chaining is initiated by issuing at least one I/U command that contains the chaining data. The chaining data consists of 4 additional characters, consisting of the following:

Unit ID - the LID of the memory unit where the chained data is to be stored.

Link Address - the first address at which the address of the data area into which the condition data is to be stored is found. This must be in the same unit as the data.

Link Increment - a number from 1 to 256 which specifies how many character locations are to be added to the link address to get the next link address.

If the unit ID is 000_8, chaining is discontinued. There will be no link address or link increment. If the unit ID is 477_8, chaining is in effect and the link address is used to continue the chain. Should an I/U command be executed that employs the chaining mode, but does not contain the unit code 477_8, the unit code it does contain, as well as the link address and link increment will be used, effectively shifting the condition chain to another unit or another area of the specified unit. Since condition chaining is active until terminated by a 000 ID or by an explicit command to that effect, only the first command in a chain of commands has to specify that condition chaining be used. Subsequent commands can specify interrupt termination, command termination, etc.

Operationally, the link address is used to fetch the first address at which the condition data is to be placed. Thereafter, for each condition encountered, the link address is

incremented and the next address for the condition data is fetched out of the specified unit. Only one condition chaining can be established per unit at any given time. That is, condition data relating to input, output, or other conditions will be intermixed, in the order they were detected by the unit.

3.8.3.2.5 Command Termination Mode

The command termination mode is selected by setting bits 7 and 8 of the command descriptor to 11. The complete command is followed by another command, which must be in the indirect mode. That command can itself be chained. The command will be fetched by the unit and executed in the normal manner. However, the command chaining data will be kept for the original chain. This allows the termination of a command in a chain to initiate the execution of another chain of I/U commands. At the conclusion of the secondary chain of I/U commands, control will return back to the original chain. If the secondary or termination chain should also call for command termination, the topmost level will be lost. Caution is advised.

3.9 Controls and Consoles

3.9.1 General

There are no special consoles or control switches other than those which are associated with specific units, as described below. All other such functions are provided through a combination of terminal devices and software. The controls associated with the various units can be remotely located and brought to a central panel (for example, to a CPS Maintenance Position). All units have the following controls and/or indicators:

(1) Power on/off switch.

(2) Power on/off indicator.

(3) Reset button.

(4) Off-line button.

(5) On-line/off-line indicator.

(6) Major alarm indicator.

Specific unit types have additional controls as described below.

3.9.2 Unit Alarms and Controls

3.9.2.1 Matrix Units

No other indicators or controls.

3.9.2.2 Memory Units

No other indicators or controls.

3.9.2.3 Channel Units

(1) Device alarm.

(2) Device alarm reset.

(3) Four activity indicator lights corresponding to each direction of each interface. These lights glow in proportion to the character transfer rates in each direction of each interface.

3.9.2.4 Memory-Memory Transfer Unit

No other indicators or controls.

3.9.2.5 Communications Processing Unit

(1) State indicator lights for each program state and the trap state. Glow is proportional to time spent in each state.

(2) Halt/run light.

(3) Run/step switch.

(4) Step button.

3.9.2.6 System Monitor Unit

(1) Clock failure light, audible alarm, and acknowledge button.

(2) Unit failure light and acknowledge button for every logical unit tied to the SMU.

(3) Common alarm horn.

3.9.2.7 Clock Unit

(1) Loss of sync alarm, light, and acknowledge button.

(2) Time of day display.

(3) Reset button.

(4) Stop buttons (seconds, minutes, etc.).

(5) Hold button.

3.9.2.8 Bootstrap Unit

(1) Ten digit keyboard with decimal point for initiating bootstrap sequence.

(2) Bootstrap failure alarm, light and acknowledge button.

(3) Pause light.

3.9.2.9 Performance Monitor Unit

No other indicators or controls.

3.10 Devices

3.10.1 General

The term "device" is used here to identify system components which are not units, but which are needed in support of the communication tasks of the complex. Devices fall into two general categories: (1) general purpose computer devices which differ little from their commercial counterparts, and (2) specialized communication devices. The former need little in the way of explanation - the latter are discussed in greater detail.

All devices in the CP are terminated on one or more channels. A given device may service more than one line. In which case, the device is actually a controller of other lower level devices.

3.10.2 Moving Head Disc

3.10.2.1 General Characteristics

The CP moving head disc is an adapted commercial unit. It uses removable disc packs. The basic characteristics are:

Minimum seek time	- 5 to 7 milliseconds
Maximum seek time	- 50 to 70 milliseconds
Number of cylinders	- 100 to 512
Number of tracks/cylinders	- 10
Capacity	- 50 million characters

Sectors per track - 16 to 256

3.10.2.2 Controller

The controller is available in various configurations ranging from 1x1 to 4x8. The first digit refers to the number of channel connections and the second refers to the number of disc drives. Thus, a 2x4 controller is connected to 2 channels and has 4 drives. The controller has the following characteristics.

(1) Simultaneous transfers on all channels: e.g., a 2x4 can perform two simultaneous transfers on any two of the 4 drives.

(2) Independent seek on all connected drives.

(3) 512 character instruction/condition.

(4) One sector buffer for every drive.

(5) Variable physical/logical identities for all drives.

(6) Optional 2 and 4 way character level interlace.

(7) Positive seek commands to specified cylinder (i.e., SEEK CYLINDER NN, rather than SEEK NN CYLINDERS FROM PRESENT POSITION).

(8) Positive identification of innermost and outermost cylinder.

(9) 5% spare cylinders with transparent identification and allocation.

(10) Head position query instruction (i.e., positional data for cylinders, but not for sectors).

(11) Parity check and generate on all read/write operations.

(12) Re-read parity check on all write operations.

(13) Positive lockout until disc drive is up to speed.

(14) No program restrictions (i.e., there is no command sequence that will cause heads to crash).

(15) Logical and physical independence of controller sections (i.e., a 3x5 controller is effectively 3 controllers each of which can behave as 1x5

controllers).

3.10.3 Visual Display Unit (VDU)

3.10.3.1 VDU Characteristics

Screen Width	- 80 to 130 characters
Screen Height	- 25 to 50 lines
Buffers	- 3 pages screen storage, one line input and one line output storage.
Character Set	- ASCII plus extended communications functions.
Refresh Rate	- 30/second
Additional Features	- 3 color, light pen, cursor, vector mode (for graphics). Insertion into formats.
Keyboard (Optional)	- ASCII keyboard with communications functions.

3.10.3.2 Controller

Controllers are available in 1x1 to 4x16. Every VDU has a variable physical/logical identity. Up to 16 VDU's may be connected to one channel.

3.10.4 Keyboard/Printers

3.10.4.1 General Characteristics

(1) Pin-feed, fan-fold, multi-ply paper.

(2) 130 characters carriage width.

(3) 30 characters per second.

(4) ASCII keyboard with communication functions.

3.10.4.2 Controller

See VDU controller options.

3.10.5 Digital Line Termination Buffer

The digital line termination buffer device is used to terminate full-duplex synchronous and asynchronous data lines at a variety of speeds and codes. The line buffer is tied one to a channel. It has the following characteristics:

(1) Speed - variable from 150 baud to 50 kb.

(2) Codes - various 5, 6, 7 and 8 level codes programmable.

(3) Modes - full duplex, half duplex, or two simplex; synchronous and asynchronous; various polling procedures.

(4) Checking - programmable CRC polynomial on input and output. Per character parity compatible with code.

(5) Deletion and insertion of sync characters and DLE characters.

(6) Programmable significant character and character sequences of not more than 4 characters each. Option to select interrupt or condition chaining on detection.

(7) 4 character buffer on input and output.

(8) Multiple channel termination option (e.g., double homing to channels as an option).

3.10.6 Low Speed Multiple Line Buffer

The low speed multiple line buffer is used to terminate full duplex synchronous and asynchronous data lines at a variety of speeds and codes. The line buffer is tied at one to a channel. It has the following characteristics:

(1) Number of lines - 1 to 64.

(2) Sampling rate - 7, 11, 15 per bit (middle 3 bit concensus).

(3) Output distortion - less than .5%.

(4) Speed - variable from 45.5 baud to 300 baud.

(5) Codes - various 5, 6, 7 and 8 level synchronous codes, programmable per line.

(6) Modes - full duplex, half duplex, simplex, synchronous and asynchronous; various polling options.

(7) Checking - programmable CRC, parity check and generation as required by code.

(8) Deletion and insertion of sync and DLE characters.

(9) Programmable significant character detection as in the digital line termination buffer.

(10) 2 character buffer on input and output.

(11) Multiple channel option.

Characters arrive interlaced in memory as a pair of characters which identify the line number (logical) and the received character. Output characters are read out from a memory buffer where they have been stored as a pair (LINE#-CHARACTER).

3.10.7 Scanner/Distributor

3.10.7.1 General Characteristics

The scanner is a general purpose device intended to fulfill all scanning operations in message switching, circuit switching, and tech control. It is based on an exception scanning approach. That is, a scanning indication is provided only if a state change has been detected on a line. It is also used to distribute state change commands to lines. The following general characteristics apply:

(1) Number of lines per scanner - up to 4096.

(2) Scanning rate - programmable up to 1000/line/second.

(3) Line state - unipolar or bipolar - three levels detectable.

(4) Distribution rate - 1000 changes/line/second.

(5) Interfaces - interface by means of general purpose interface module(s), providing digital (binary) input and output on the CP side of the interface and line compatible levels on the line side of the interface.

3.10.7.2 Principles of Operation (Scanner Functions)

Figure I-2 shows the organization of the local scanner module. Each such module can terminate up to 16 lines. A line may use either unipolar or bipolar signaling. A state is represented as a two bit signal; with 00 indicating the ground state 01 and 10, the possible two other states if bipolar signaling is used. The state 11 indicates error conditions. The dotted

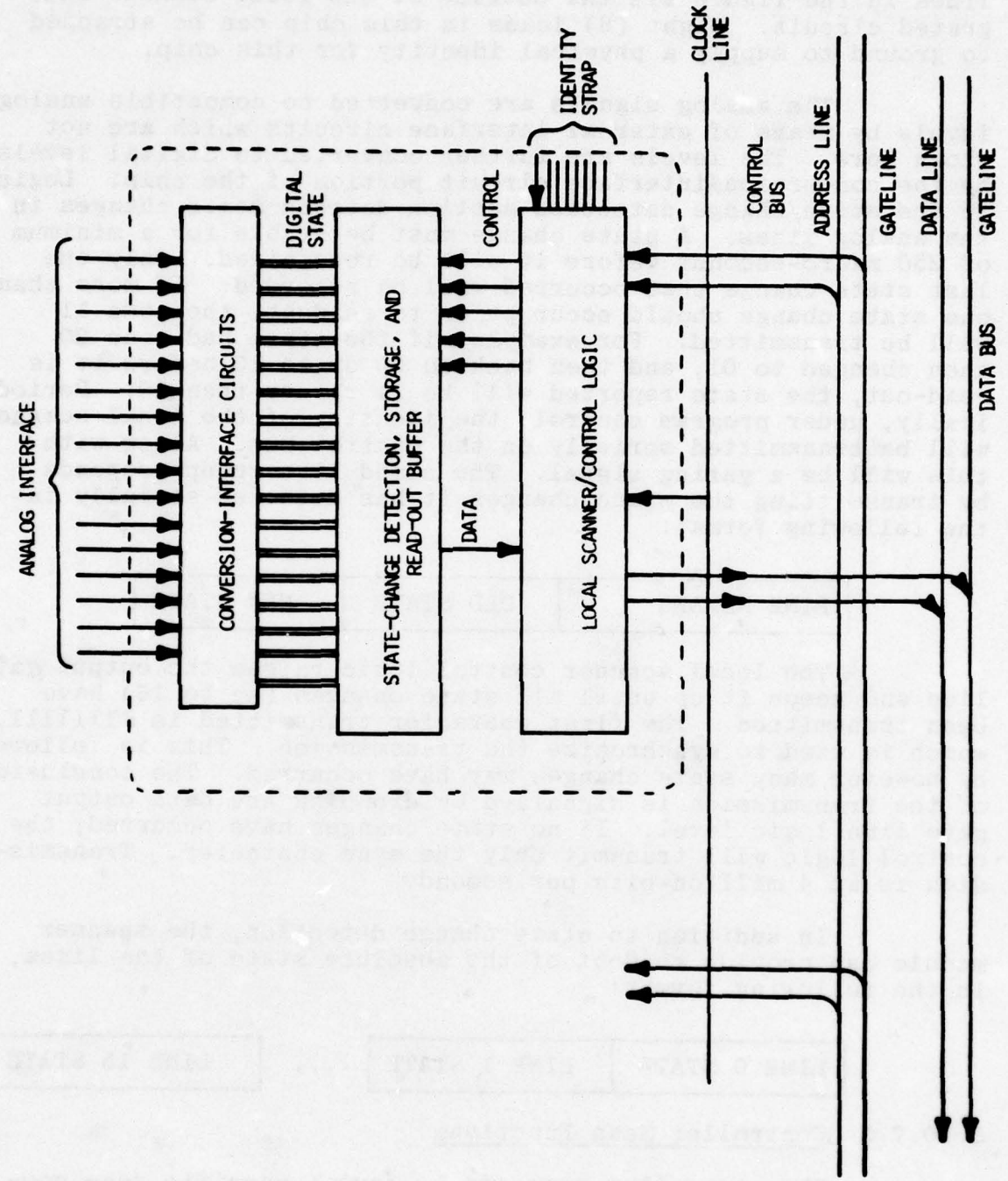


FIGURE I-2 SCANNER OPERATING PRINCIPLES

lines in the figure are the outline of the local scanner integrated circuit. Eight (8) leads in this chip can be strapped to ground to supply a physical identity for this chip.

The analog signals are converted to compatible analog levels by means of external interface circuits which are not shown here. The levels are further converted to digital levels by the conversion/interface circuit portion of the chip. Logic in the state/change detection section detects state changes in the analog lines. A state change must be stable for a minimum of 250 micro-seconds before it will be recognized. Only the last state change that occurred will be recorded. If more than one state change should occur prior to readout, the code 11 will be transmitted. For example, if the state had been 00 then changed to 01, and then back to 00 or to 10 before it is read-out, the state reported will be 11 rather than 10. Periodically, under program control, the identity of the local scanner will be transmitted serially on the control bus. Along with this will be a gating signal. The named line group responds by transmitting the state changes it has detected serially in the following format:

LINE NUMBER	OLD STATE	NEW STATE

The local scanner control logic raises the output gate line and keeps it up until all state changes (up to 16) have been transmitted. The first character transmitted is 11111111, which is used to synchronize the transmission. This is followed by however many state changes may have occurred. The conclusion of the transmission is signalled by dropping the data output gate line logic level. If no state changes have occurred, the control logic will transmit only the sync character. Transmission is at 4 million bits per second.

In addition to state change detection, the scanner module can provide readout of the absolute state of the lines, in the following format:

LINE 0 STATE	LINE 1 STATE	....	LINE 15 STATE

3.10.7.3 Controller Scan Functions

The controller responds to device specific scan commands of the following form:

SCAN LINE STATE CHANGES - scans all line groups starting with 000 and ending with 255 for state changes until either complete or

until the buffer is full. The buffer contents are transmitted to the channel when full. If buffer filling occurs prior to the scanning of the last implemented group, a condition will be signalled via the channel (either under condition chaining or interrupt as desired).

CONTINUE STATE CHANGE SCAN	- continue with previous state change scan if the scan had stopped because of a full buffer. The same as SCAN LINE STATE CHANGES if the previous scan had completed.
RESET SCAN	- resets the scan to 000 line group. No effect if already at 000.
SCAN STATE CHANGES GROUP$_0^{255}$	- scans the state changes of the indicated group only. The controller does not wait for a full buffer in this case but transmits the changes (if any) as soon as they are received.
SCAN STATES	- scans the current states of all lines up to a buffer full. Same behavior as state change command when buffer is full.
CONTINUE SCAN STATE	- analogous to CONTINUE STATE CHANGE SCAN for states rather than changes.
SCAN STATE GROUP$_0^{255}$	- reads out the current state of the indicated group.

The following formats are used with the various commands:

SCAN LINE STATE CHANGES:

LINE GROUP	LINE #	OLD STATE	NEW STATE
LINE GROUP	LINE #	OLD STATE	NEW STATE

.
.
.
.

LINE GROUP	LINE #	OLD STATE	NEW STATE

SCAN STATE CHANGE GROUP N:

LINE GROUP N			LINE #	OLD STATE	NEW STATE
LINE #	OLD STATE	NEW STATE			

LINE #	OLD STATE	NEW STATE	PAD CHARACTER

Possible pad character to fill out to even number of characters.

SCAN LINE STATES:

LINE GROUP N				STATE	STATE	STATE	STATE
STATE	STATE	STATE	STATE	STATE	STATE	STATE	STATE
STATE	STATE	STATE	STATE	LINE GROUP N+1			

Possible pad character to fill out to even number of characters.

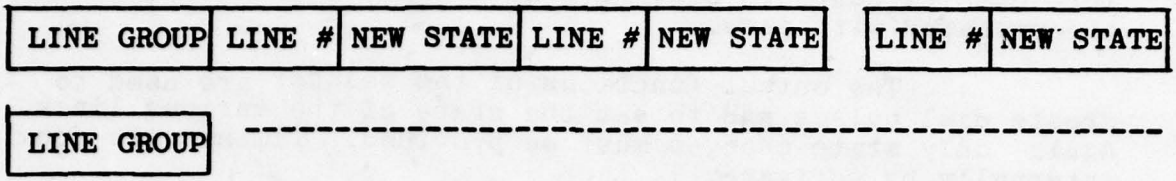
SCAN STATE GROUP N:

As above, with a pad character at end of group.

3.10.7.4 Output Functions

The line scanner/distributor operates in an output mode over a different set of control busses, independently of its scanning operation. On output, the distributor can be set

to change the state of specified lines to specified values. A 256 character command buffer is used for this purpose. The command format is:



Pad bits (0, 2, 4, or 6) are used to fill out each group to the nearest character. The entire command chain is padded out to the nearest character pair. Pad bits are not transmitted. Upon receipt of a full or partial buffer full of state change commands (as a device specific command via the channel) the controller serially transmits the commands on the data bus in the order they occur in the buffer. The proper line group responds upon detecting its own identity by raising a control line on a return bus. The controller then transmits the rest of the state change commands serially. An invalid state change command is not transmitted but is signalled by means of an interrupt (or condition chain entry) via the channel.

3.10.7.5 Applications

3.10.7.5.1 Circuit Switching

The scanner is primarily intended for use in line supervision functions in circuit switching, TELEX, or dial-up message switched trunks. General line supervision is programmed to scan all lines every millisecond. This is adequate for the highest normal circuit switched signaling speeds. Somewhat lower rates could be used with a slight complication in the software. For example, all lines are scanned every ten milliseconds. If there is at least one active line in a group of 16, that group is given 9 supplementary scans every ten milliseconds. Various strategies of this kind can be employed to reduce total number of executed scans. Since the scanning is effectively an exception report system, the fact that typically there are very few state changes assures us of having relatively few responses to a scan.

Supervision functions are programmed much in the way they would be in a micro-computer scanner. All timings are done by software. On-hook, off-hook and hook flash are done by programmed timing. Dial pulse capture and assembly can be done on the line side of the switching matrix using the scanner. The high resolution of the scanner allows all circuit switched dial pulse rates to be accommodated. DTMF receivers can be treated as a group of lines. The receiver itself is stripped of the

normal logic and simply reports state changes in the various tones. These are confirmed and converted to internal BCD representation by software. Other signaling speeds which require more complex controls over several signaling lines can also be accommodated with ease.

The output functions of the scanner are used to create dial pulses and to set the state of the various lines. Again, only state changes must be provided. Timings are done internally by software.

The following elements can be controlled or monitored by the scanner:

(1) Subscriber lines.

(2) Subscriber ring relay.

(3) Ringback bus.

(4) Tone bus.

(5) Hold bus (actually an unlatch).

(6) AC address signaling receiver lines.

(7) AC address signaling sender lines.

(8) Various special traffic attendant lines.

3.10.7.5.2 Message and Packet Switching Applications

In typical message switches, the application of a scanner falls into one of three categories: (1) applications which are really circuit switching functions, (2) applications which are really tech control functions, and (3) true message switching applications. The true message switching applications of the scanner are almost non-existent. While the scanner could be used to perform bit sampling and character assembly on low speed lines, if the number of such lines is large, the multiple line buffer device is a better choice for the purpose. A more likely use of the scanner in message switched applications would be in the implementation of message switching like functions that occur with low frequency in circuit switches.

3.10.7.6 Tech Control Functions

Most message and circuit switches have collocated technical control functions. Furthermore, there are large scale stand-alone tech control facilities. The scanner input capabilities can be used to advantage to detect and (subsequently by a CPU) process logical alarm conditions. Monitoring of several thousand test points is not a significant burden with the scanner. Conversely, the output functions can be used to

change the state of devices that are normally manipulated by a tech control facility, where such state changes are relatively simple on/off functions.

3.11 Application Considerations

3.11.1 General

This section shows various system configurations as constructed from the units of the CP. Each configuration is shown in functional and in physical form. The following systems are discussed: typical message switch, packet switch, circuit switch, tech control system, and digital concentrator. These applications are intended to illustrate the use of the CP in representative systems. They are not intended as final designs.

3.11.2 Typical Message Switch

3.11.2.1 Functional Configuration

Figure I-3 shows the functional configuration of a typical message switch in simplified form. Digital lines are terminated on a pair of low speed multiplexer-buffer devices if they are at 300 baud or less. Medium and high speed lines are terminated on high speed line devices at one per channel. Some of these are connected to more than one channel. Functionally, the communication lines are terminated on front-end CPU's. The front-end CPU's are also used to terminate the various traffic service VDU's, printers, and keyboards. These are ganged together in groups of two or three per channel.

The message processor is also the system executive processor. It carries the bulk of the memory. It is functionally closely coupled to the ICU, which uses its spare capacity to monitor line status condition and tech control alarms. The ICU is therefore connected to a set of scanners used for this purpose. The message processor also controls six disc devices. Two of these are used for in-transit message and historical storage. Two are used for message accounting, journalling and ledger data; one contains all system programs, and the last is a spare.

There is an off-line CPU with two memory units which is used for standby purposes. There is also a system service CPU which is functionally connected to the Bootstrap Unit, the PMU's and a system disc used to store test and diagnostic programs and other information needed to maintain the system.

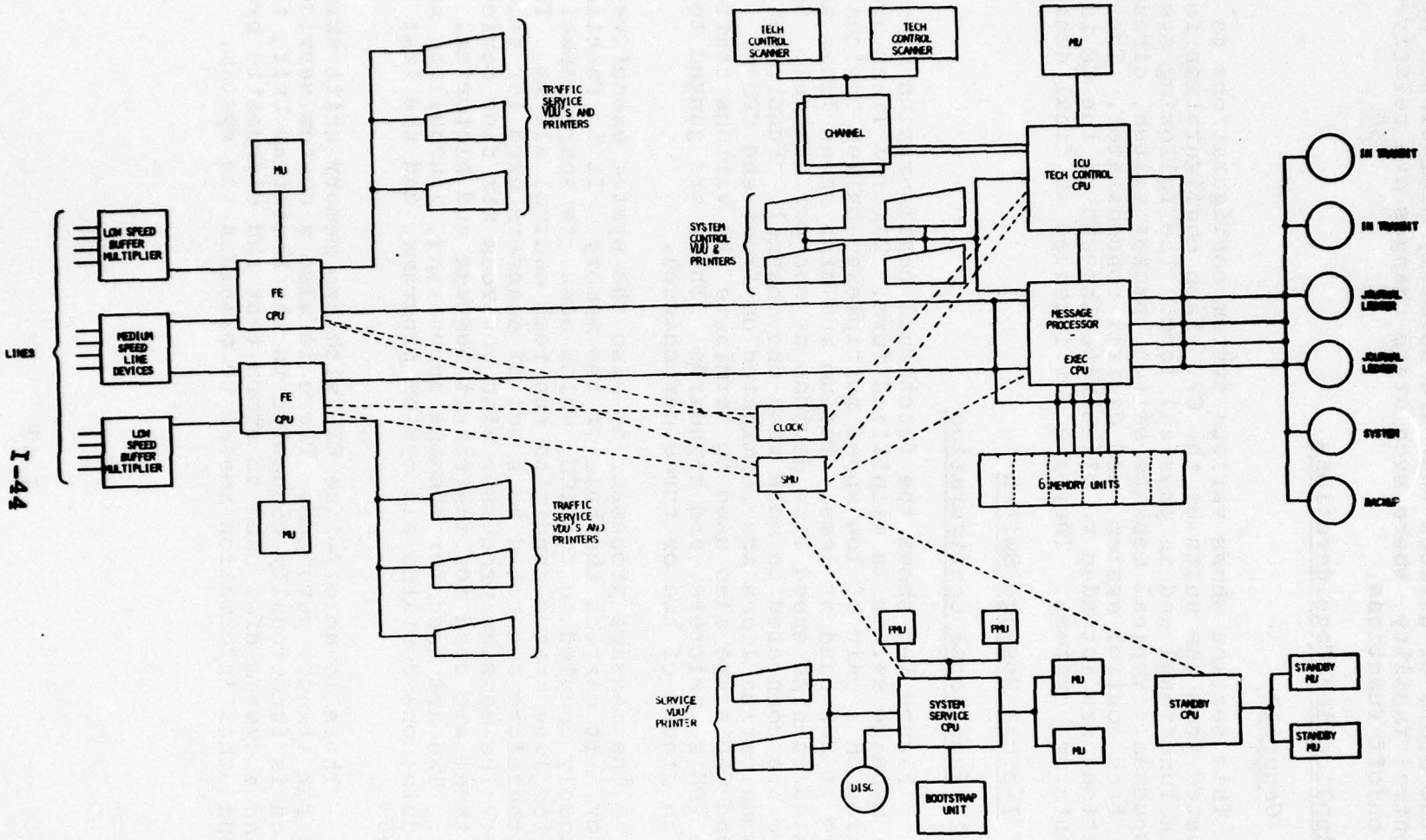


FIGURE 1-3 FUNCTIONAL CONFIGURATION OF TYPICAL MESSAGE SWITCH

3.11.2.2 Sacrifice Schedule

The following order is used to sacrifice functions in the event of various failure types:

CPU failures:

- Standby CPU
- First Front-end CPU
- ICU (ICU functions taken over by MP)
- System Service CPU

Memory Unit failures:

- First and Second Standby MU
- First System Service MU
- Two MP Buffer MU's
- One FE MU

Disc failures:

- Back-up Disc
- MP System Disc
- First Journal-Ledger Disc
- First ITS Disc
- System Service Disc

VDU/Printers/Keyboards:

- Traffic Service Positions - all but one
- System Control Positions - all but one
- Half of the System Service Positions
- Remainder of Traffic Service Positions

3.11.2.3 Physical Configuration

Figure I-4 shows the physical configuration of the message switch corresponding to the functional configuration of Figure I-3. There are a total of 63 active and 54 passive units. Most of the passive units are XU's. The passive unit stage, while constructed out of 4 x 12's is actually used as 3 x 12's. This side of the matrix could be reduced by not providing dual ports for the XU's. Connections are not totally symmetrical. Traffic patterns and the grading effect of primary port assignments should be used to advantage to bias higher traffic

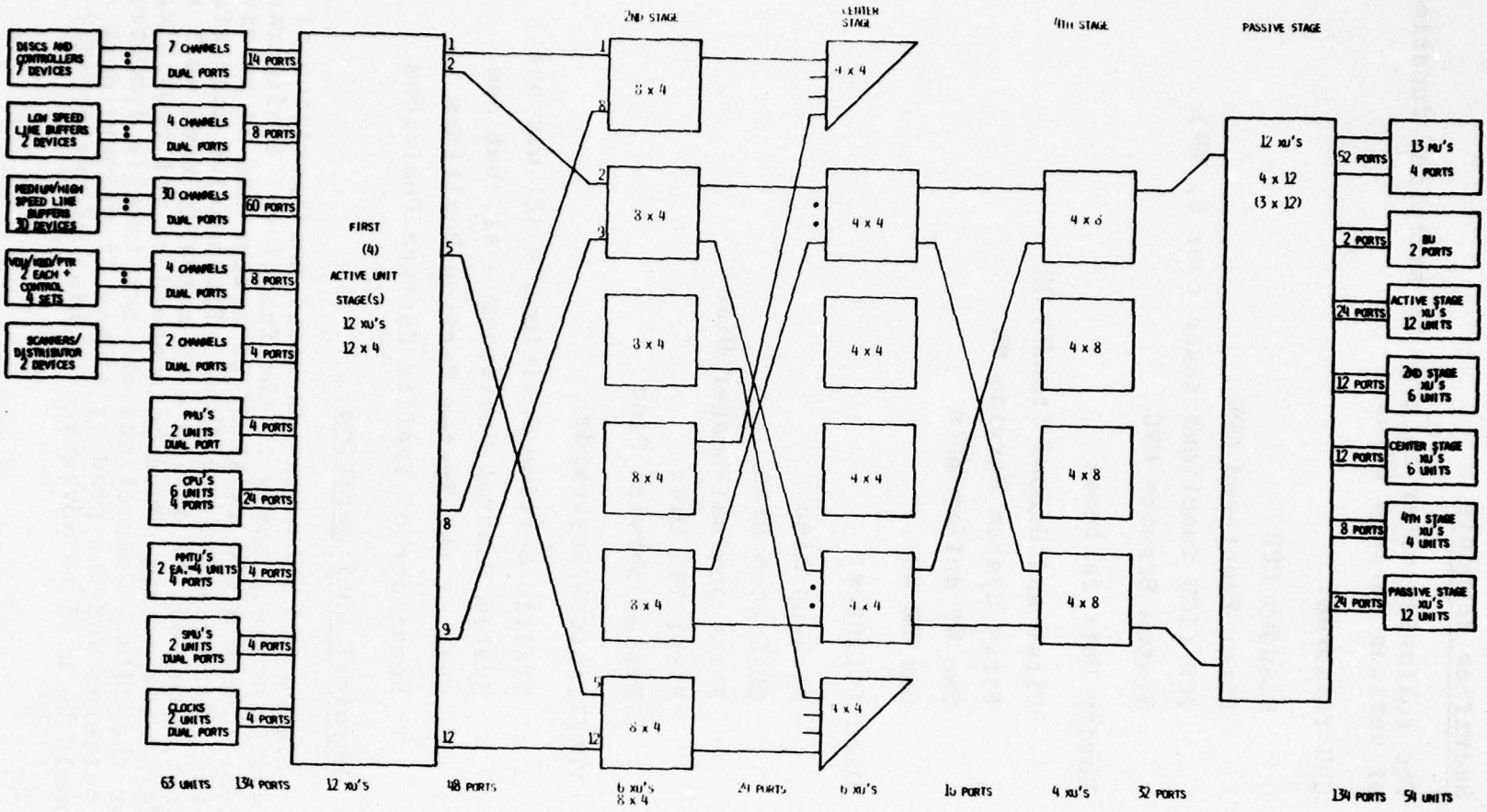


FIGURE 1-4 MESSAGE SWITCH PHYSICAL CONFIGURATION

connections to take advantage of the multiple paths that are available as a result of the asymmetries. The scanners are not connected to two channels since they do not perform primary functions in a message switching system.

3.11.3 Packet Switch

3.11.3.1 Functional Configuration

Figure I-5 shows the functional configuration of a small packet switch which handles only packet switched formats. There are no disc units, nor are there extensive facilities for tech control, traffic service and system control functions. Since the bulk of the processing load is in the FE CPU's, the MP CPU is relatively lightly loaded and can be used to implement the ICU functions as well. VDU's are used for system control and for maintenance functions.

3.11.3.2 Sacrifice Schedule

The sacrifice schedule is similar to that of the message switch. First the standby CPU is sacrificed, then a FE CPU, and the system service CPU (if necessary).

3.11.3.3 Physical Configuration

The physical configuration is shown in Figure I-6. Because of the relatively small number of units, it is possible to employ a three rather than a five stage matrix. The large triangular center stages are used to cross-connect channels for through packet traffic. That is, High-High packet traffic for which the system has no accountability can be cut-through once the processing has been done to determine which outgoing line is to be used. Condition chaining for SOB detection and packet header capture is used to allow the FE CPU's to validate each packet. Once this has been done, the rest of the packet does not have to come through memory. The buffering in the channel units and the line buffers allow ample time for this processing to take place.

3.11.4 Circuit Switching

3.11.4.1 Functional Configuration

Figure I-7 shows the functional configuration of a large (i.e., several thousand lines) circuit switching system based on the CP. This represents the upper end of circuit switching systems likely to be implemented using the CP. In many applications the various functional roles of the CPU's would be combined. The following combinations are likely:

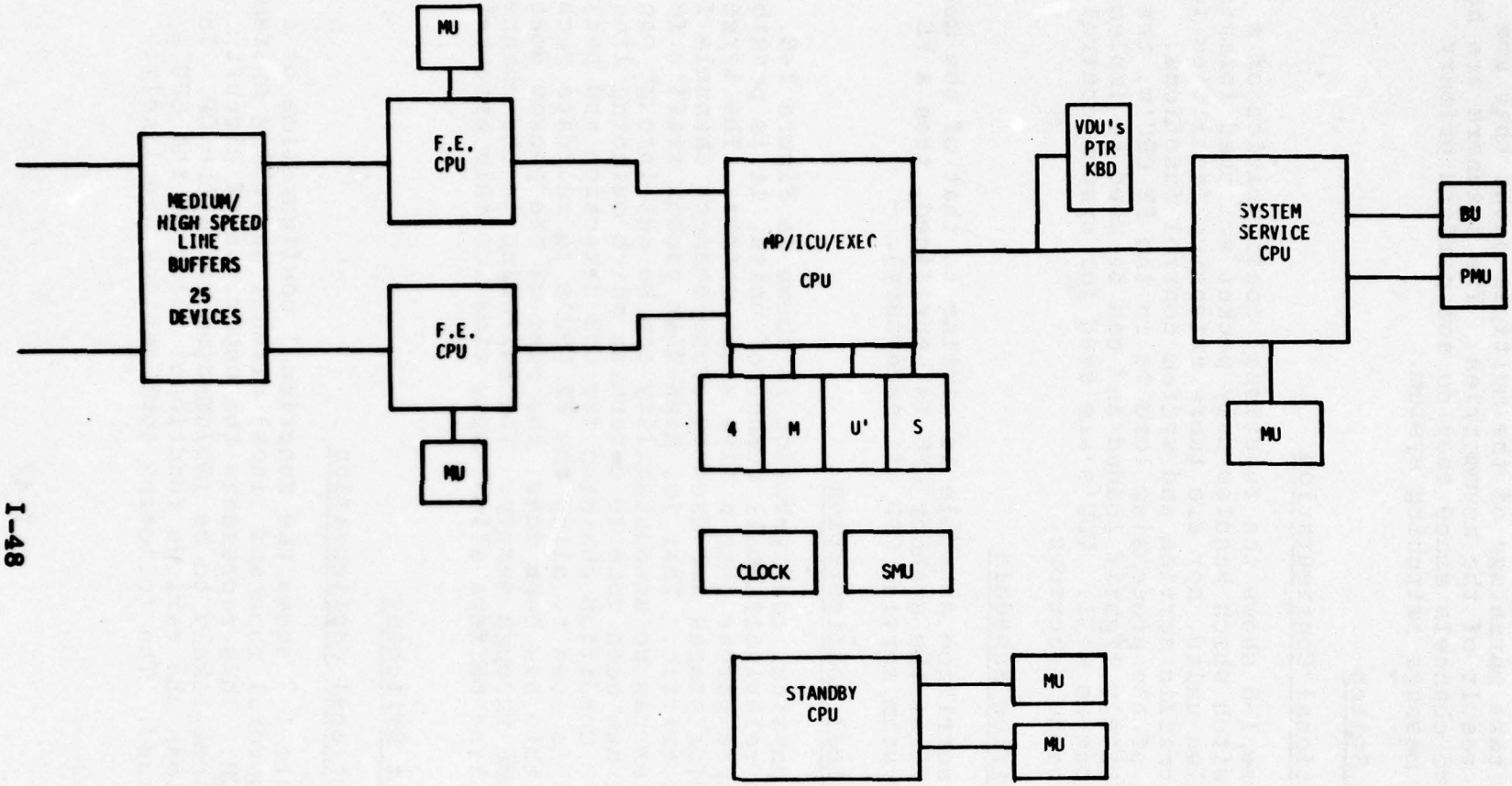


FIGURE I-5 FUNCTIONAL CONFIGURATION OF A SMALL PACKET SWITCH

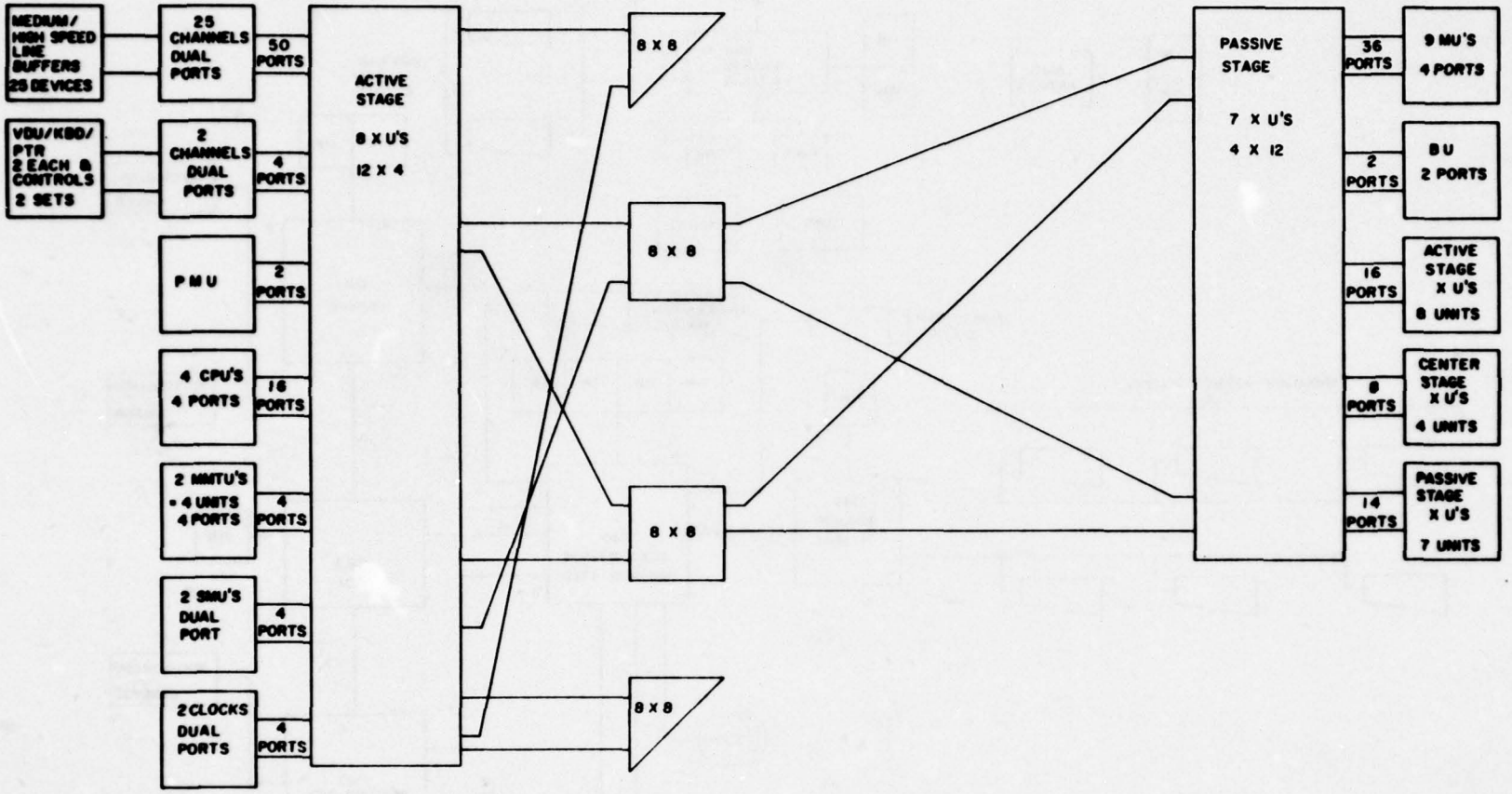


FIGURE I-6 SMALL PACKET SWITCH PHYSICAL CONFIGURATION

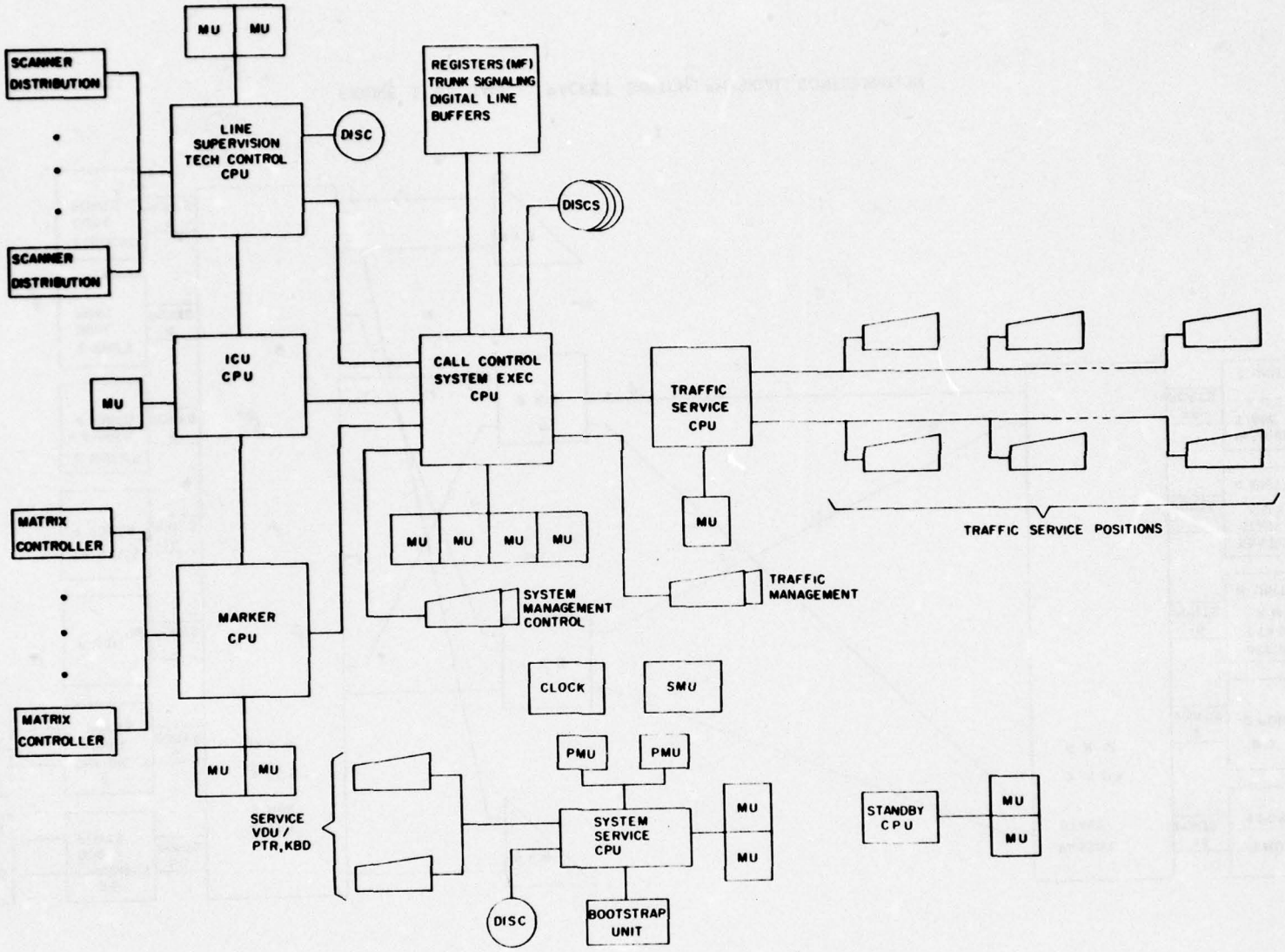


FIGURE I-7 FUNCTIONAL CONFIGURATION OF A CIRCUIT SWITCH

(1) Traffic Service with Call Control

(2) Marker with Call Control

(3) ICU with Marker

We have shown a large number of traffic service positions and a traffic service CPU in recognition of the fact that military switching may involve a greater degree of traffic service than would be typical in a commercial environment. The system service CPU and its associated configuration is essentially the same as that of the large message switch. Clock distribution lines and SMU connections are not shown for simplicity's sake. Also not shown are the multiple connections for the matrix controllers, and scanner/distributor.

The box labeled "Registers, trunk signaling, etc." represents all line terminating and register type devices. The "MF" device in particular represents SF, MF, DTMF, etc., or possibly a programmable unit that performs all such signaling functions. Frequency receiver/senders may actually be split into separate receivers and senders which individually contain very little logic. They would then be terminated on the scanner/distributor. This depends upon the number of such units employed and the details of the system design.

The digital line buffers and trunk signal devices are whatever are needed appropriate to the interfaces (both analog and digital) with which the system will have to contend.

3.11.4.2 Physical Configuration of Large Circuit Switch

Figure I-8 shows the physical configuration corresponding to the functional configurations shown in Figure I-7. The layout is similar to that which was used for the typical message switching system and the matrix is approximately the same size. Note that the scanners and (external) matrix controllers are each terminated in two channels, which in turn are dual ported.

3.11.4.3 Configuration of Moderate Circuit Switch (600 Lines)

Figure I-9 shows the functional configuration of a more moderate circuit switching system. Note that many of the functions have been combined and that tech control related processing has been virtually eliminated as a separate function. The number of traffic service positions have also been reduced significantly. The corresponding physical configuration is shown in Figure I-10. Note the substantial decrease in the size of the matrix. As with the small packet switch, we are again able to use a three stage matrix. Unlike packet switching, in which there can be considerable channel-to-channel communication, the circuit switching

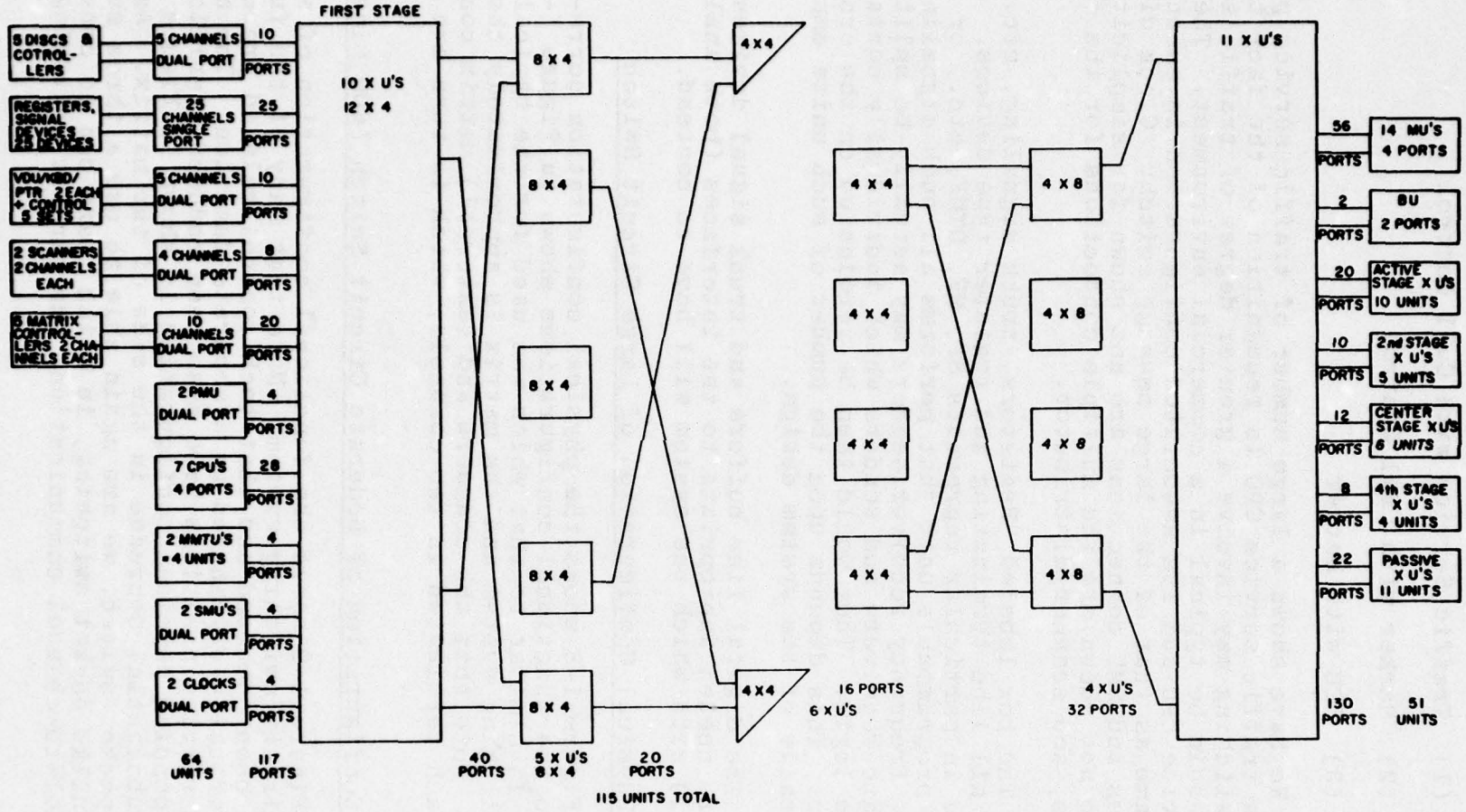


FIGURE I-8 PHYSICAL CONFIGURATION OF A CIRCUIT SWITCH

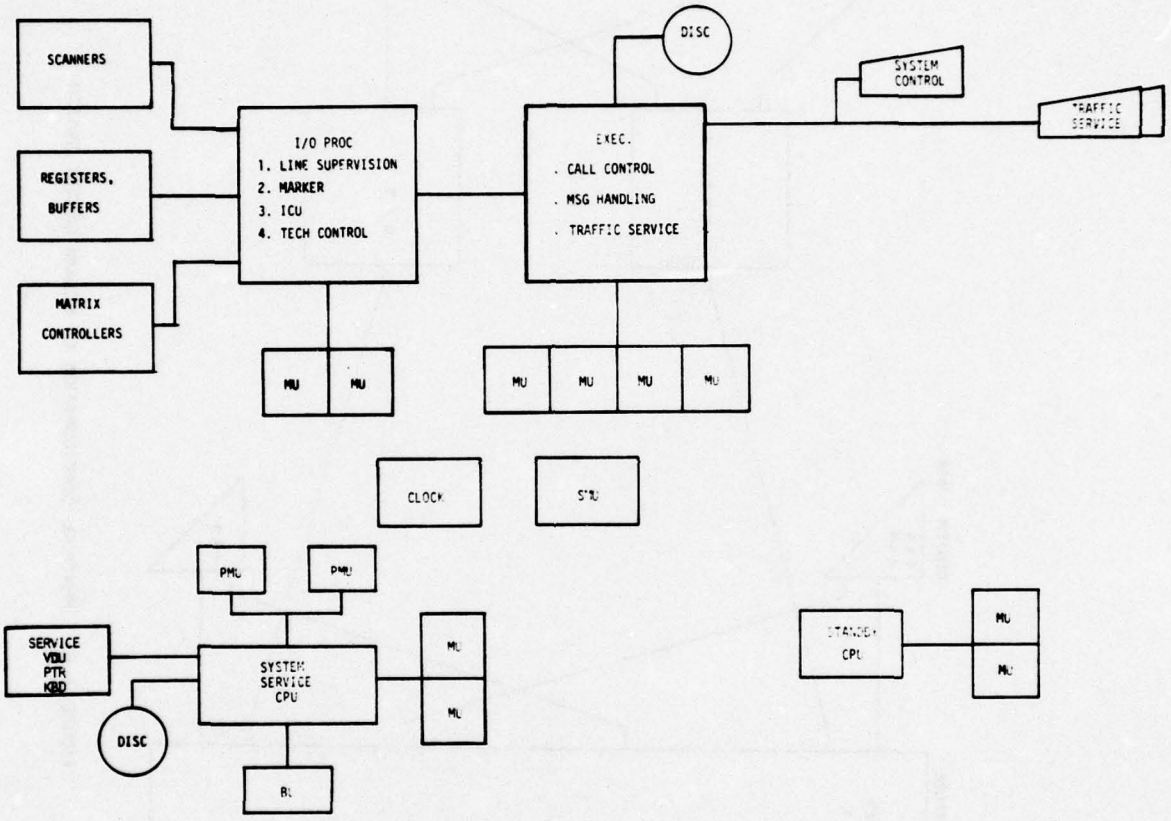


FIGURE I-9 MODERATE SIZED CIRCUIT SWITCH - FUNCTIONAL CONFIGURATION

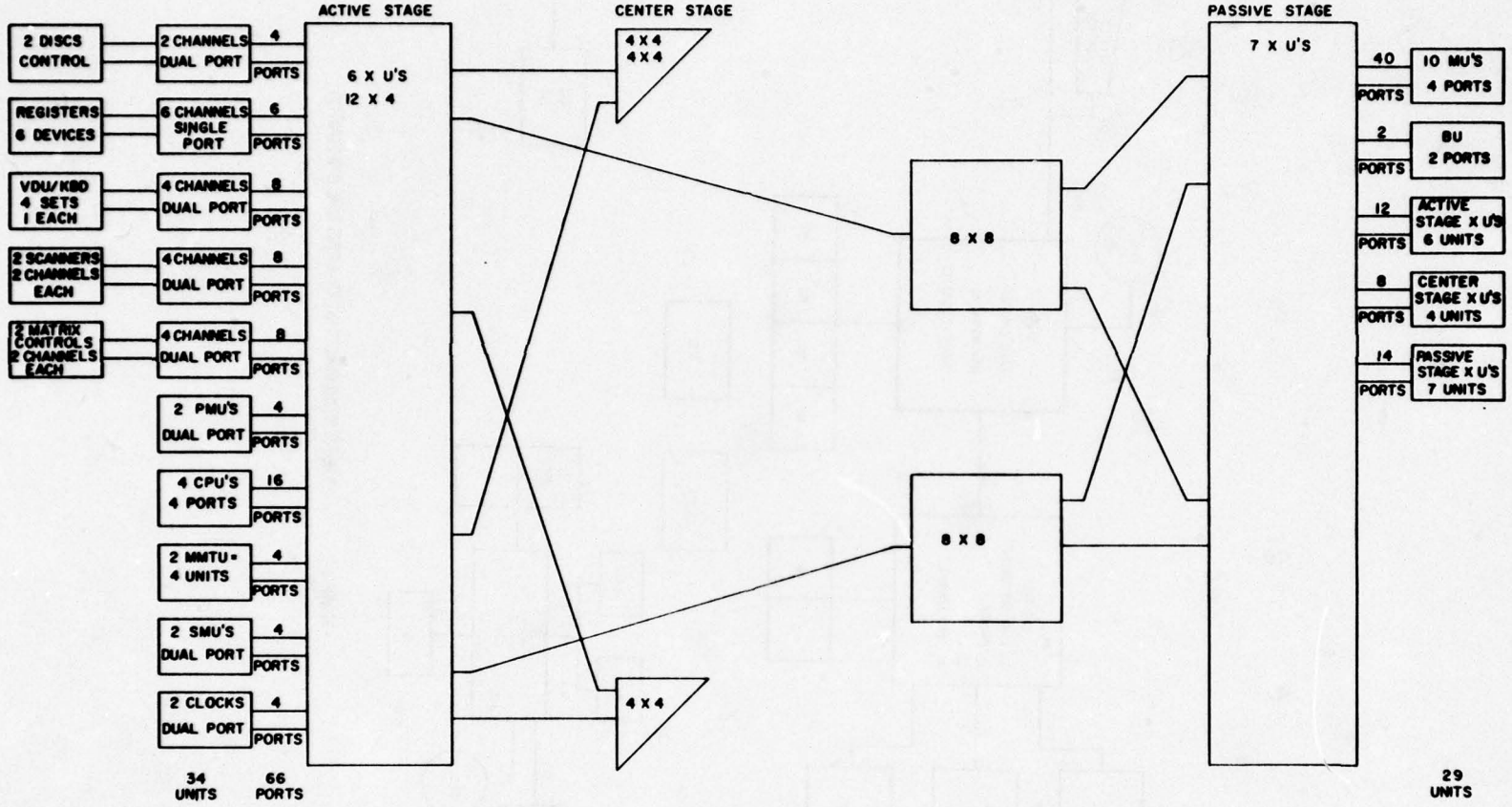


FIGURE I-10 PHYSICAL CONFIGURATION OF MEDIUM CIRCUIT SWITCH

system has only a moderate amount of this activity and consequently a minimal sized 4 x 4 triangular submatrix will suffice.

3.11.4.4 Very Small Circuit Switch

Figure I-11 shows the functional configuration of an unattended (except for traffic service) circuit switching system, such as might be used for a large PBX or equivalent. It is a dual thread system in keeping with the typical military mission, but does not have the usual built-in test hardware. If the CPU or any component should fail, a service technician, equipped with the proper tools will be sent. Repair may require temporary shut-down. The corresponding physical configuration is shown in Figure I-12. Units have been simplified in keeping with reducing the complexity of the recovery programs and the fact that there are no trained technicians on the premises. An active-active triangular stage is not provided. Such traffic would have to be accommodated by transferring data to memory first.

Figure I-13 shows a single thread version of the same system. In this case, the multiplicity of ports have been provided only for traffic reasons and not for viability reasons. This allows the use of a single stage matrix. The SMU is only used to provide failure alarms and automatic bootstrapping of the system from what are (hopefully) transient malfunctions. In such a case, the SMU would effectively force a re-bootstrapping of the system from the copy of the program stored in the BU.

3.11.5 Tech Control System

The tech control system functional configuration is similar to that which is shown in Figure I-7 for circuit switching systems. The configuration is shown in Figure I-14. There is still a matrix control function. In this case, it is used to control automated patch-bays. Traffic service positions are replaced by stations at which tests can be initiated and directed. In addition to the scanner/distributor, a number of specialized (analog) test equipment is provided. Disc units are used to store channelization details. As in all of these systems, minor circuit switching and message switching functions must be performed and appropriate line termination and sensing devices are provided.

3.11.6 Digital Concentrator

The physical configuration of a small concentrator would be similar to that of the small circuit switch. The scanner would be replaced by low speed line multiplexers or by individual line termination devices if medium and high speed lines are used. Concentrators are typically single thread systems and the same savings as were obtained in the small circuit switch can be obtained here.

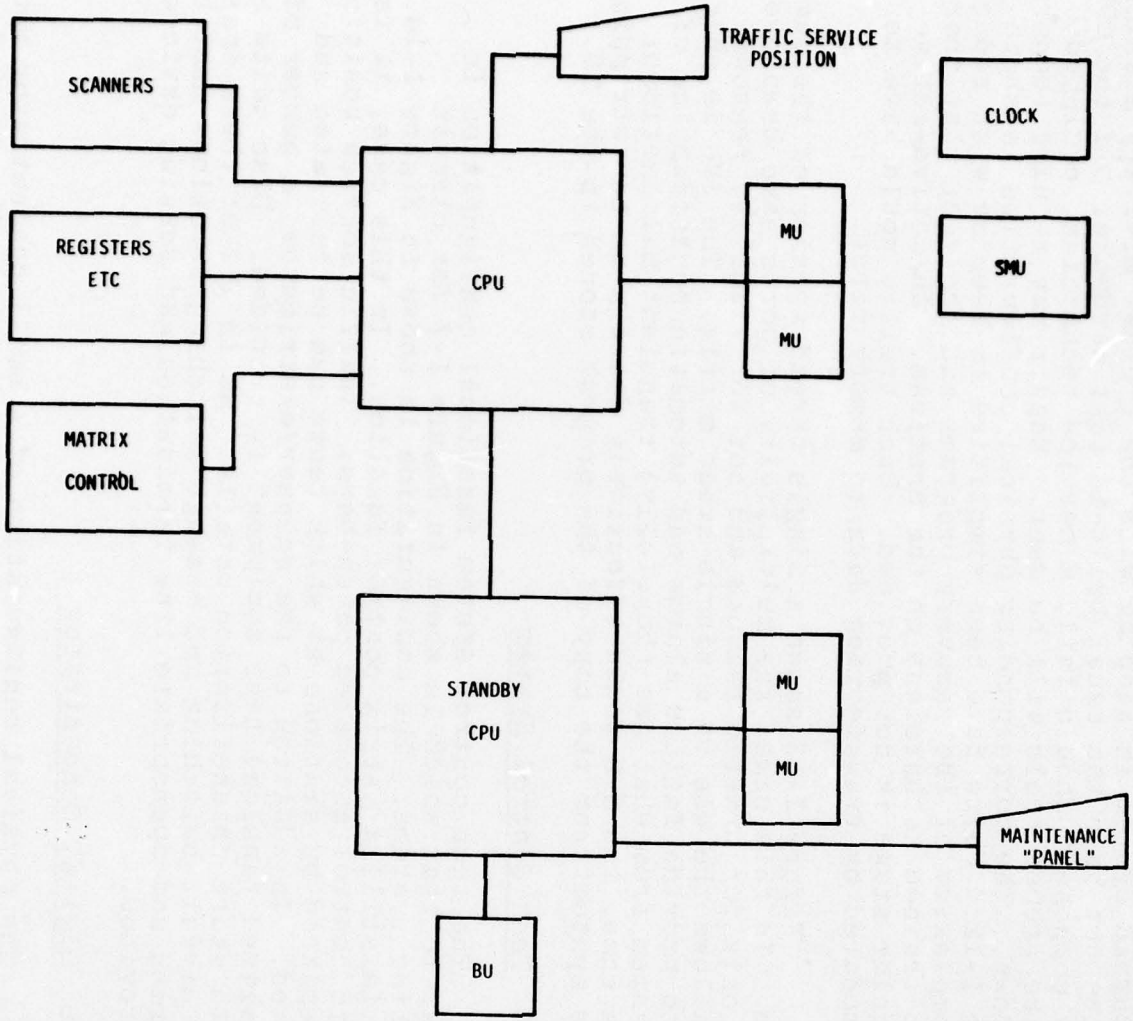


FIGURE 1-11 UNATTENDED CIRCUIT SWITCH

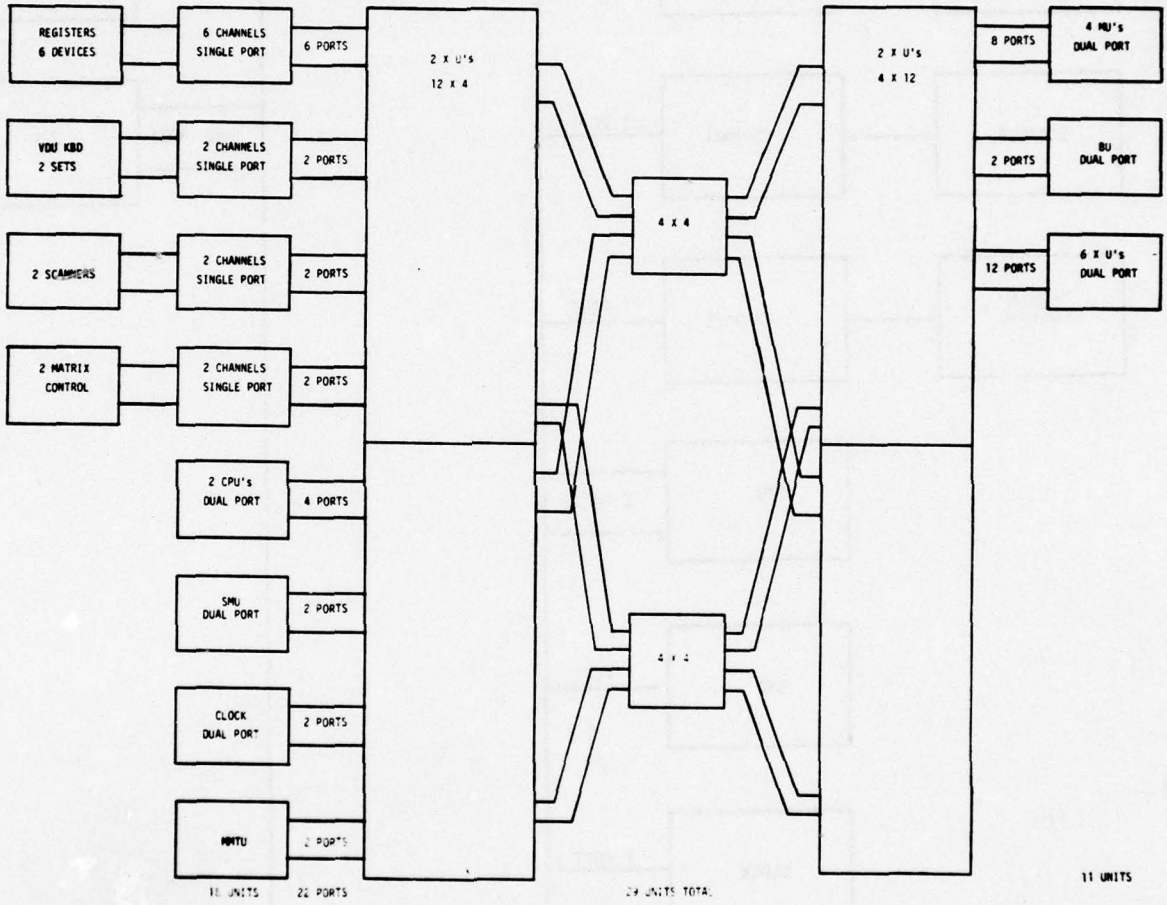


FIGURE I-12 PHYSICAL CONFIGURATION - SMALL CIRCUIT SWITCH

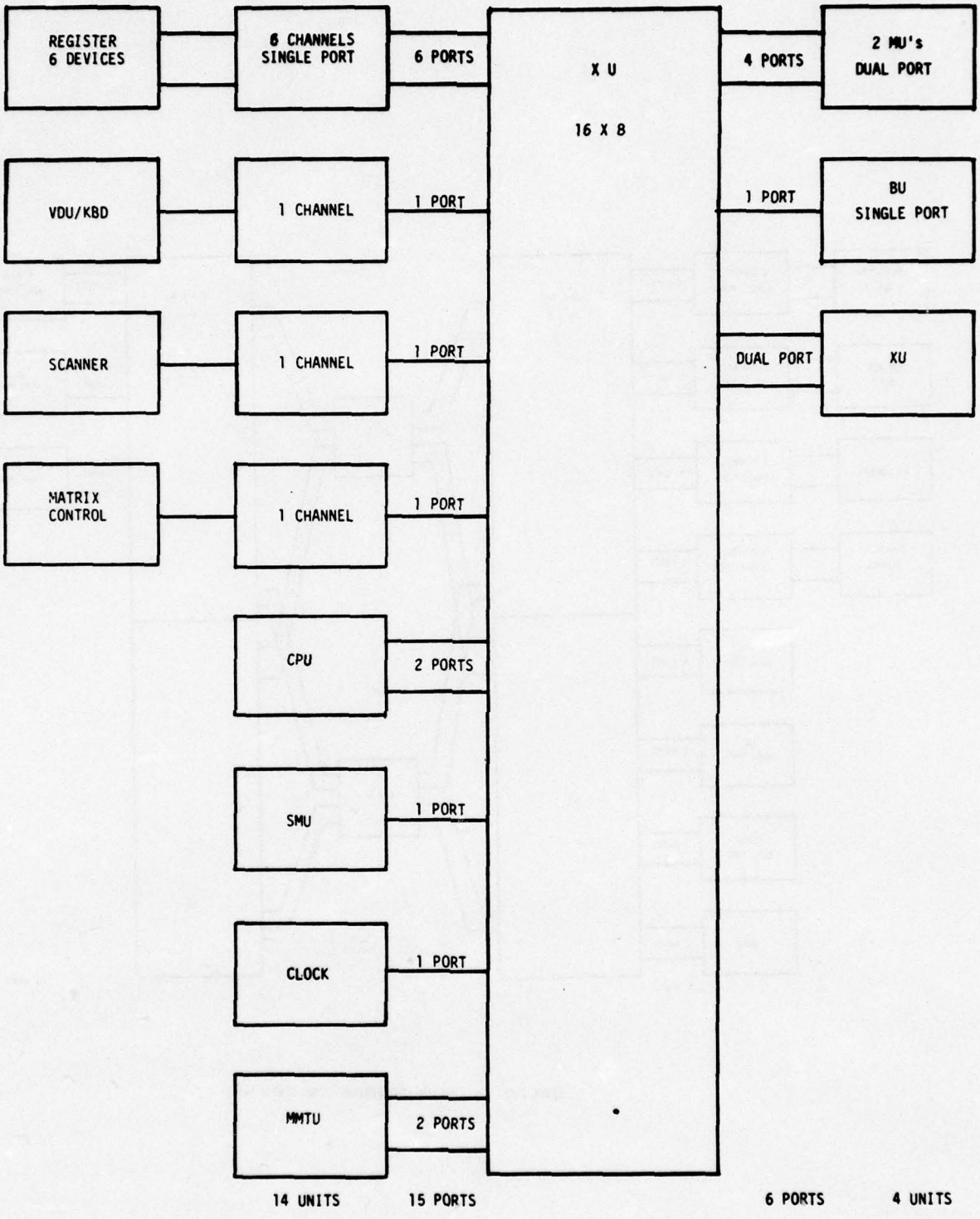


FIGURE I-13 SINGLE THREAD SMALL CIRCIUT SWITCH

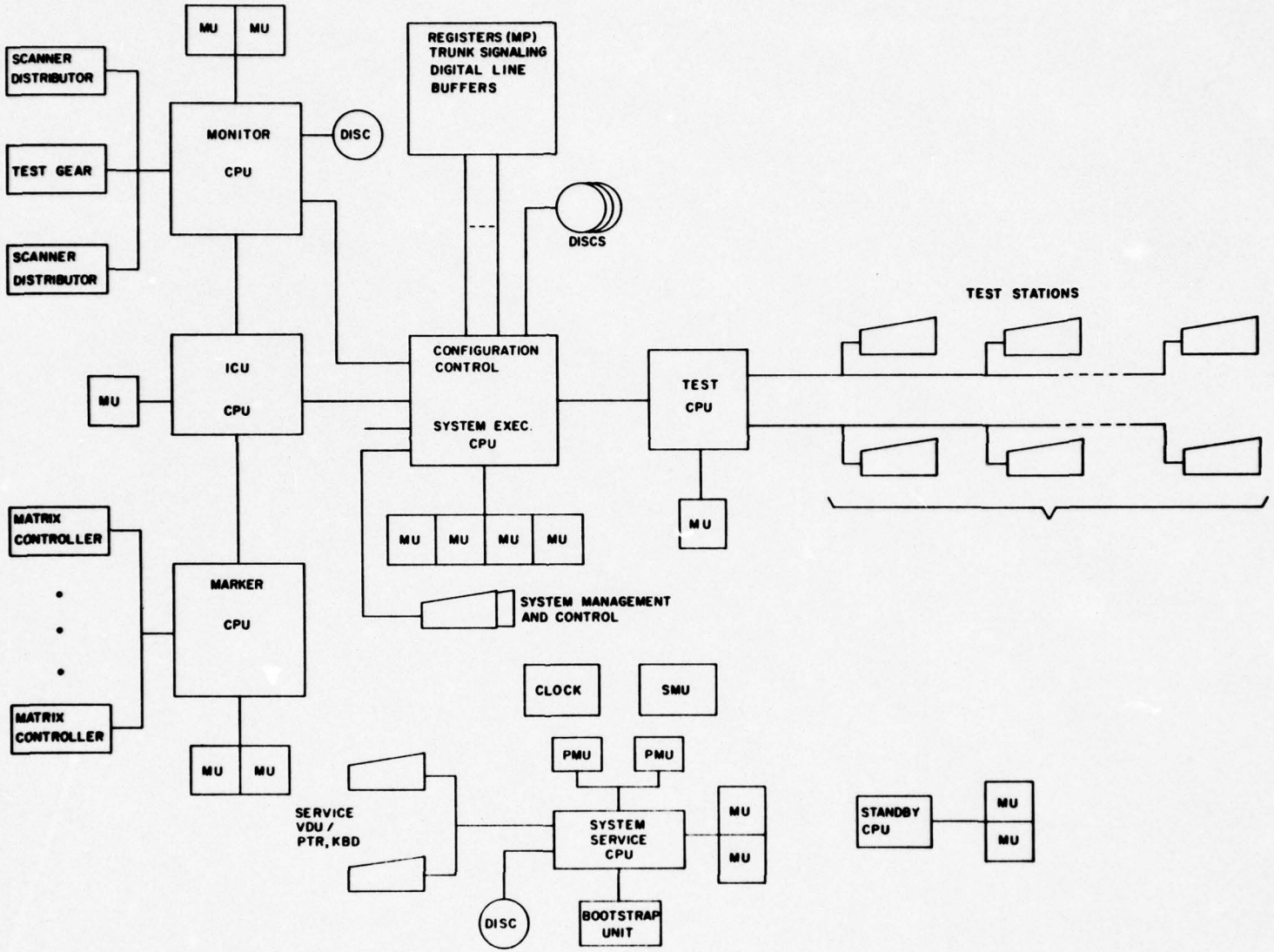


FIGURE I - 14 FUNCTIONAL CONTROL OF TECH CONTROL SYSTEM

SECTION II

SPECIFICATION FOR THE COMMUNICATIONS PROCESSING UNIT (CPU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE
COMMUNICATIONS PROCESSING UNIT
OF THE
CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	II-1
2.0	Applicable Documents	II-1
3.0	Communications Processing Unit (CPU) Specification	II-1
3.1	General Characteristics	II-1
3.1.1	Overview	II-1
3.1.2	Addressing	II-1
3.1.3	Registers	II-1
3.1.4	Register Sets	II-2
3.1.5	Instruction Stack	II-2
3.1.6	Unit Characteristics	II-3
3.1.7	Arithmetic	II-3
3.1.8	Machine Versus Assembler Addressing	II-4
3.1.9	Extended Range Addressing	II-4
3.2	Memory Reference and Indexing	II-4
3.2.1	Unit Identification	II-4
3.2.2	Index Arithmetic	II-5
3.2.3	Index Modes	II-5
3.2.4	Indirect Modes	II-6
3.2.5	Application	II-6
3.3	Instruction Repertoire	II-6
3.3.1	General	II-7
3.3.2	Register Set Instructions	II-7
3.3.3	Subroutine Call and Return Instructions	II-9
3.3.3.1	General	II-9

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.3.3.2	Operation of the Stack	II-9
3.3.3.3	Stack Protection	II-10
3.3.3.4	Call and Return Instructions	II-11
3.3.4	Skips, Jumps and Executes	II-12
3.3.4.1	General	II-12
3.3.4.2	Unconditional Jump Instructions	II-12
3.3.4.3	Unconditional Execute Instructions	II-13
3.3.4.4	Conditional Skip Instructions	II-13
3.3.4.4.1	General	II-14
3.3.4.4.2	Bit Conditioned Register Skips	II-14
3.3.4.4.3	Skip on Word, Character, Half-Word	II-14
3.3.4.4.4	Double Element Skips	II-15
3.3.4.4.5	Memory Reference, Single Operand Skips	II-16
3.3.4.4.6	Dual Operand, Memory Reference Skips	II-16
3.3.4.5	Conditional Executes	II-17
3.3.5	Load/Store Memory Instructions	II-17
3.3.5.1	Load/Store Registers	II-17
3.3.5.2	Load/Store, Half-Word, Character, NN Mode	II-17
3.3.5.3	Load/Store via Registers	II-18
3.3.6	Single Operand in Memory via Register	II-19
3.3.7	Bit Manipulation in Memory via Register	II-20
3.3.8	Operations between Registers and Memory	II-20
3.3.9	Shifts and Rotates	II-23
3.3.9.1	Logical Shifts	II-23
3.3.9.2	Arithmetic Shifts	II-24
3.3.9.3	Rotates	II-24
3.3.10	Count Instructions	II-24

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.3.10.1	Count in Registers	II-24
3.3.10.2	Count in Memory	II-25
3.3.11	Tally Instructions	II-26
3.3.12	Double Register Operations	II-27
3.3.13	Triple Register Operations	II-28
3.3.14	Single Register Immediate Operand	II-29
3.3.15	Dual Register, Immediate Operands	II-29
3.3.16	Stack and Queue Management Instructions	II-30
3.3.16.1	Push and Pop Instructions	II-30
3.3.16.2	Pull and Shove Instructions	II-31
3.3.16.2.1	FIFO Chain Structure	II-31
3.3.16.2.2	FIFO Chain Operation	II-33
3.3.16.2.3	Instructions	II-33
3.3.17	Miscellaneous Instructions	II-34
3.3.17.1	Analyze	II-34
3.3.17.2	Halt	II-35
3.3.17.3	Select Next Level	II-35
3.4	Inter-Unit (IU) Instructions	II-35
3.4.1	General	II-35
3.4.2	Single Character IU Micro-Instructions	II-35
3.4.3	Two and Three Character IU Micro-Instructions	II-36
3.4.4	Issue of IU Command via Register	II-37
3.4.5	Issue of IU Command via Memory	II-37
3.5	The CPU as a Unit	II-38
3.5.1	Control Memory	II-38
3.5.2	Command Protection	II-38

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.5.3	Port Priority Control	II-40
3.5.4	Port Command Stack	II-40
3.5.5	IU Commands	II-41
3.6	CPU Alarm Processing and Traps	II-42
3.6.1	General	II-42
3.6.2	Self-Detected Alarms	II-42
3.6.3	Trap Operation	II-42
3.6.4	Trap Related Instructions	II-43
3.7	Flags and Breakpoints	II-45
3.8	Operating Modes	II-46

1.0 SCOPE

This document is the specification for the Communications Processing Unit (CPU) of the CPS Central Processor (CP). It defines the architecture, features and instruction repertoire of the CPU. The CPU is one of several units which together comprise the CP.

2.0 APPLICABLE DOCUMENTS

Central Processor Specification (Section I of the CPS Central Processor Specification).

3.0 COMMUNICATIONS PROCESSING UNIT (CPU) SPECIFICATION

3.1 General Characteristics

3.1.1 Overview

The processing unit of the CP is a general purpose stored program digital computer. It has a basic operational word length of 32 bits. While the CPU has a general purpose instruction repertoire, many aspects of that repertoire as well as specialized instructions, make it optimum for the communications task.

3.1.2 Addressing

Three types of elements can be addressed in memory: words, half-words and characters. An address can be either directly incorporated into an instruction, or can be in a register which is referred to by the instruction. In addition, normal indirect addressing and a variety of indexing modes are provided. Every memory reference must either directly or indirectly incorporate a unit ID which identifies the memory unit which is to be accessed. Five character length fields, stored in registers, are used to provide a unit ID implicitly. One field is part of the program counter. The other four fields are specified by the value of a two bit field in the memory reference instructions. This two bit field, called the AUGMENT field, specifies which of four characters in the registers is to be used to augment the basic 16 bit address, thereby identifying the memory unit.

3.1.3 Registers

The CPU is a generalized register machine. It has 16 registers of 32 bits each. These can be addressed and manipulated as: 16 registers of 32 bits, 32 registers of 16 bits, 64 registers of 8 bits, or any sensible combination thereof.

This provides the programmer with the ability to split a 32 bit register into four tally counters, or an address field and two counters, etc.

The contents of registers 0, 1, and 2 are used for special functions:

Register 0 - 24 bit program counter, 8 condition bits.

Register 1 - 4 Unit ID's - for memory unit address augmentation (i.e., page registers).

Register 2 - subroutine stack pointer, 8 flag bits.

While the interpretation of the contents of these registers is predetermined, the programmer can (with caution) manipulate them at will.

3.1.4 Register Sets

Every CPU has four complete sets of 16 registers. At any instance of time, the CPU is in one of four program states, corresponding to which of the four register sets is in use. Externally generated interrupts may cause the CPU to shift from one register set to another. Similarly, the programmer can select which register set is in operation at any instant of time. The shift from one register set to another takes only the time required to execute the register set or program state shift instruction. Instructions are provided to allow the entire contents of a register set to be loaded from memory or to be stored in memory, effectively allowing the implementation of an arbitrary number of program states.

3.1.5 Instruction Stack

An eight word instruction stack for instruction look-ahead is provided in every CPU. This stack can be used to store from 7 to 32 instructions. It should not be construed from this statement that instructions have variable lengths. CPU instructions are 32 bits long but the stack mechanism takes advantage of the op-code structure to compact the representation of some instructions. Instructions on the stack may vary in length from 8 to 36 bits. This is an internal matter and transparent to the programmer. The most often used instructions will average 16 bits in length, consequently allowing the stack to hold 8 instructions.

The instruction stack facilities are used to set up tight loops and other repetitive processes that have a minimum number of memory references. The CPU uses whatever opportunities arise to make additional memory fetches of instructions.

For example, if operands are not being fetched as a result of a particular instruction execution, an additional instruction will be fetched instead and placed into the stack. Eventually, the stack is filled with instructions. New instructions enter the bottom of the stack and displace previously used instructions out of the top of the stack. New instructions fetched are always pushed to the topmost available location in the stack. If the stack is full, new instructions entering at the bottom will force old (used) instructions to be displaced from the top. This process continues unless a branch instruction is fetched. This includes conditional and unconditional JUMPS, SKIPS, and EXECUTE instructions. The occurrence of such an instruction in the stack stops the migration of instructions, but allows the stack to be filled up to capacity. Instruction execution progresses until the branch type instruction is reached. If the branch is to an instruction within the stack, that instruction is pushed to the top and additional instructions are fetched in to fill out the bottom. If the branch is to an instruction not presently on the stack, the stack is cleared and a new fetch and stack build cycle is started.

All stack operations are transparent to the programmer. However, a knowledge of the operational details of the stack mechanism is essential if advantage is to be taken of it in the construction of tight loops, executed wholly out of the stack. The execution time for stacked instructions is typically 250 nanoseconds, providing an effective four fold increase in CPU speed when compared to executions which involve memory references.

3.1.6 Unit Characteristics

Some of the functions traditionally associated with CPU's are not implemented in the CPU of the CP. These functions are accomplished by other units. Predominant among them are memory protection and interrupt/priority control. However, the CPU is still a unit of the CP. Consequently, it has all the typical unit characteristics - modifiable LID, priority/protection cache memory, response to I/U commands, etc. The CPU can be provided with 1, 2, or 4 ports. In most cases, the 4 port configuration will be used. As with any other CP unit, the CPU has a unit specific repertoire which defines its behavior in inter-unit operations. Its unit specific repertoire has features common to memory units and channel units. (Refer to Sections VI and VII).

3.1.7 Arithmetic

The number representation system is 2's complement. In all arithmetic operations between elements of different sizes

(e.g. word and character) justification is with the smaller element in the least significant position. Results are always in terms of the larger element. Overflow and underflow flags are set in accordance to that of the larger element involved. Overflow and underflow flags remain set until explicitly reset by the programmer.

3.1.8 Machine Versus Assembler Addressing

All addressing specifications in this specification are described as absolute addresses. The reader should not construe from the description of the various instructions, particularly the variable length skip instructions, that absolute or relative to absolute addressing is required in the assembly source language. As in all assemblers, symbolic addressing is used. The target of a skip instruction is treated like the symbolic address normally used for a jump. The assembler will provide the proper skip value. If this is out of the permissible range of the skip instruction, appropriate assembler diagnostics will be issued.

3.1.9 Extended Range Addressing

Where the address portion of an instruction which references memory is found in a half-word register, or in memory, as in an indirect reference, the contents of that registers location is interpreted in terms of the unit being referenced. Since a half-word register is 16 bits long, it can be used to reference 65,536 words, half-words, or characters. Thus, word reference via a half-word register has a span of 4 units; half-word reference a span of 2 units and character reference a span of one unit. The inclusion of indexing can further extend the range. If a half-word index mode is applicable, the word range can be extended to 8 units, the half-word range to 4 units, and the character range to 2 units. This characteristic should be taken into account when setting up to load unit ID's in the Augment (AUG) fields. Often, because of extended range addressing, it will not be necessary to reload the AUG field.

3.2 Memory Reference and Indexing

3.2.1 Unit Identification

Every memory reference must ultimately, directly, indirectly, implicitly, or explicitly, result in the specification of at least one unit ID. There are two ways in which such unit ID's can appear in an instruction: immediately as a part of the instruction, or via a pointer to one of the four characters in register 1. If the UID appears immediately in the instruction, it is an eight bit field. If the UID is in register 1, it is

pointed to by means of a two bit AUGMENT field which specifies which of four characters are to be used.

3.2.2 Index Arithmetic

Index arithmetic operations are carried out with all participating elements interpreted as positive integers. Justification is to the smaller element of the pair. All index arithmetic operations are carried through to the UID segment of the address, permitting indexing across unit boundaries. However, a corollary to this is the fact that overflow will occur only for attempts to get beyond unit 477, and underflow only for attempts to get below unit 000.

3.2.3 Index Modes

There are three index options: none, addition, and B + C*I. The normal indexing (addition) is available for some instruction classes in both a pre-and post index mode. This results in four sets of indexing operations: N(one), B(ase addition, pre indirect), P(ost indirect addition), and C(onstant multiplier). If indexing has been specified, then the first UID is always via an AUGment field.

B Mode - This corresponds to normal pre-indirect indexing operations in which the index value is added to the base address to produce the resulting address for that part of the effective address calculation. The index value can be contained in a half-word register or in a register character.

P Mode - This mode applies only if an indirect operation has been specified. The indirect operation is first carried out, fetching a new address, to which the index value is added. The index value can be contained in a half-word or a register character.

C Mode - In this mode of indexing the product of a multiplier constant and an index value is formed. This is added to the base address in the instruction. The index value can be contained in a half-word register or in a register character. The multiplier can be contained in a register character or be an immediate field in the instruction. The C mode is not available with all instruction classes. Indexing always precedes indirect operations in this mode.

3.2.4 Indirect Modes

The first memory access of an indirect operation is always a fetch of a half-word which can contain a word, half-word or character address. The UID (first UID) can be via an AUGMENT field, or be an immediate operand in the instruction itself. A second UID must be specified to direct which unit the indirect address refers to. This can also be an immediate eight bit field in the instruction or be specified by means of the AUG field. However, only one of the UID's can be immediate. More specific rules apply in combination with the indexing modes as follows:

N Modes - If no indexing has been specified, one or the other but not both UID's can be immediate. Alternatively, both UID's can be obtained by specifying the AUG field.

B,P,C Modes - If indexing in any mode has been specified, then both UID's are obtained via the AUGment field.

3.2.5 Application

Not all combinations of indexing and indirect apply to all classes of instructions. The particular index/indirect modes which do apply are specified with each instruction class, by means of the following mnemonics: NN, NI, BN, CN, BI, PI, CI.

NN - no index, no indirect

NI - no index, indirect

BN - Base plus constant index, no indirect

BI - Base plus constant index, with indirect

CN - Base plus constant times multiplier, without indirect

CI - Base plus constant times multiplier, with indirect

PI - index addition post indirect

For each of these, there is a further specification of the location of the UID(s), the index value, and the index constant, as appropriate to the instruction class. Where a specification does not exist, the option is not allowed for that instruction class.

3.3 The Instruction Repertoire

3.3.1 General

The instructions are presented from a functional point of view, as a sequence of fields. The proper range of values for each field is included in the instruction specification. For example:

$$\text{SHFC::OP(L(eft)), R(ight)): CHAR}_{0}^{255} \text{ : VALUE}_{1}^{8}$$

is the specification for a shift within a specified register character. The interpretation of the specification is:

OP - Indicates a variation of the basic operation. In this case, L (left) or R (right).

SHFC - Mnemonic for a character shift instruction.

CHAR_{0}^{255} - A register character address, whose value is anything from 0 to 255.

VALUE_{1}^{8} - Indicates a shift from one to 8 bits.

In source language, the instruction has the following appearance:

```
SHFC L  60  3
```

Indicating a shift left of character 60 by 3 bits.

Instruction structures are described only from the point of view of their functional appearance. These do not necessarily correspond to the bit patterns that they are translated into by the assembler. There is no simple, one-for-one representation of instructions that corresponds to specific bit patterns in the instructions.

3.3.2 Register Set Instructions

These commands are used to modify the working register set. The sets are numbered 0 to 3, with set 0 having the highest priority. Upon shifting register sets, control is transferred to the new set. The contents of the old set is unaffected.

$$\text{GOTON :: VALUE}_{0}^{3}$$

Unconditional transfer to register set VALUE.

BLOCK :: $VALUE_0^3$

Prevents interrupts whose priority is VALUE from causing a shift to register set VALUE.

BLOND :: $VALUE_0^3$

Prevents interrupts whose priority is VALUE or lower from causing a shift to another register set.

UBLOK :: $VALUE_0^3$

Unblocks interrupts at priority VALUE. Has no effect if already unblocked.

UBLND :: $VALUE_0^3$

Unblocks interrupts at VALUE or lower.

STALL :: $VALUE_1^4$

Stalls all interrupts until after the execution of the next N instructions, where N is specified by VALUE.

Interrupt stalls, such as those obtained through the use of the STALL instructions, or other instructions which have a STALL option, are cumulative and apply to instruction executions. For example, the instruction sequence:

```
STALL 4
STALL 4
STALL 4
```

causes a cumulative stall of the next 12 instructions, two of which were the second and third STALL's - consequently there would be a cumulation of 10 instruction stalls beyond the third STALL. An 8 bit stall counter is maintained for each CPU. Since only the active level can be stalled, only one stall counter is kept. If the cumulative stall plus the additional stalls exceeds 255, a stall of 255 will be executed. As each instruction is executed, the stall count is reduced by one. The following commands in this set automatically create a stall of one instruction: GOTON, WAITS, SLECT. Note that this means, GOTON, WAITS, and SLECT, when issued at their own level (i.e., the program is in state 3 and GOTON 3 is issued) resets the stall counter to 1.

WAITS :: $VALUE_0^3$

Transfers control to register set VALUE and halts. The next instruction will be executed only upon the receipt of an

interrupt at level VALUE. Higher level interrupts will override this function.

$$\text{SLECT :: VALUE}_0^3$$

Transfers control to set VALUE and causes a four instruction stall. All registers are wiped clean except 0, 1, and 2.

3.3.3 Subroutine Call and Return Instructions

3.3.3.1 General

Two instructions are provided to effect subroutine calls and returns. The issuance of a CALLS instruction causes the proper jump to the called subroutine as well as the storage of the return address in a stack set up for that purpose. In addition to the basic return address data, the system will store an 8 bit flag character. Similarly, the RETURN instruction causes a jump back to the proper return point, with a return 8 bit flag character, as well as adjusting the stack pointer. The programmer can specify as many or as few separate stacks as he wishes. All stacks are limited to 256 locations (e.g. nesting of 256 subroutine calls per stack). Should either end of a stack be violated, the CPU will jump to programmer defined standard locations for that stack.

3.3.3.2 Operation of the Stack

The format of the contents of register 2 is:

FLAG BITS	UNIT ID	14 BIT WORD ADDRESS	X X

The first 8 bits contain flags which can be manipulated and acted upon by the programmer. The next character contains the unit ID in which the subroutine stack control word will be found. The next 14 bits specifies the location of the subroutine stack control word. Bits XX have functions discussed elsewheres. Issuing a CALLS instruction causes the contents of the subrcutine stack control word to be fetched from memory. The subroutine stack control word has the following format:

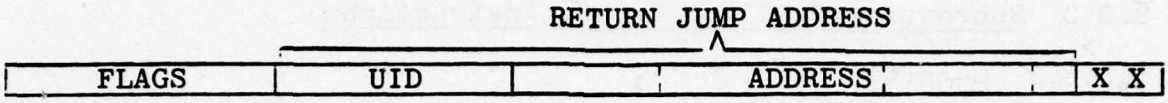
COUNT	UID	STACK ENTRY ADDRESS BASE	X X

The first character contains the present stack count, which can be anything from 0 to 255 entries. The next character specifies the (memory) unit in which the stack is to be found, and next 14 bits specify the top of the stack. Having fetched the stack control word, the following events take place:

(1) The count is decremented by one.

(2) The new stack entry address is calculated relative to the stack base address.

(3) The following information is placed in the stack entry:



The flag field will be set to the values of the flag character in register 2. The UID-ADDRESS fields will be set to the return address (the value of the PC + 1). Bits XX will be set to an optional value specified in the call instruction.

The issuance of a RETRN instruction reverses the procedure as follows:

(1) The return address is fetched from the current stack position, via the contents of register 2 and the appropriate stack control word.

(2) The flag bits contained in the stack entry are used to replace the flag bits in register 2.

(3) The count field is incremented by one.

(4) The stack entry is decremented and checked.

(5) The program counter is replaced by the return address.

(6) A stall of XX is added to the present value of the stall counter.

3.3.3.3 Stack Protection

Locations one word past the last valid stack address (i.e., that address at which the count field is equal to -1) and locations one before the first valid stack address (current stack address < stack base address) should contain the address of jump instructions, stall value and flag bits.

If the end of the stack should be overrun, the CPU will execute a jump to the specified location, setting flag bits as stored in the return word, with the indicated number of instruction stalls. Similarly, if the top of stack should be "underrun", a jump to the location specified by the programmer

for that condition will be executed. In this way, the programmer can specify the maximum subroutine call depth, separately for each stack, and can further specify what corrective action will take place in the event that either the top or the bottom of the stack should be overrun.

3.3.3.4 Call and Return Instructions

The call instruction has the following formats:

$$\text{CALLS :: AUG}_0^3 \text{ : ADDRESS}_0^{16,383} \text{ : STALLIN}_0^3 \text{ : STALLBACK}_0^3$$

$$\text{CALLA :: UID}_1^{254} \text{ : ADDRESS}_0^{16,383} \text{ : STALLIN}_0^3 \text{ : STALLBACK}_0^3$$

The two different forms of this instruction allow a subroutine call via the AUG field specified (register 1 UID's) or in the absolute mode (CALLA) the explicit specification of the unit being called. In either case, a word address is provided. The STALLIN field specifies an addition from 0 to 3 to stall counter. The STALLBACK field specifies the number of instructions to be stalled upon return from the subroutine call.

There are two return instructions: RETRN and RTJMP. RETRN performs the return as described above and there are no optional fields. The specification for RTJMP is given below:

$$\text{RTJMP :: REGISTER SET}_0^3 \text{ : UNIT ID}_0^{255} \text{ : WORD ADDRESS}_0^{16,383}$$

The normal return from a subroutine is done, placing flags in register 2, updating the stack control word, etc. However, instead of jumping to the normal return address, a register set change is performed as well as a jump to the specified address. Should return stack underflow occur as a result of executing this instruction, the underflow jump will override the specified jump, and the register set change will not occur. Interrupts are stalled for one instruction past the RTJMP.

These instructions are also available with a B mode indexing, in which the index value is found in a register character. The return instruction RETRN can in the indexed mode, perform an indexed jump based on the contents of a specified register character. Typically, it would be on the contents of the return flag character in register 2. This is used in conjunction with multiple exits from subroutines. The formats are:

$$\text{CALLS :: AUG}_0^3 \text{ : ADDRESS}_0^{16,383} \text{ : STALLIN}_0^3 \text{ : STALLBACK}_0^3 \text{ :}$$

$$\text{B-INDEX CHARA}_0^{63} \text{ : BN}$$

$$\text{RETRN :: B-INDEX CHARACTER ADDRESS}_0^{63} \text{ : BN}$$

3.3.4 Skips, Jumps, and Executes

3.3.4.1 General

Three types of branching instructions are provided: jumps, skips, and executes. Jumps have a wide range and can go from unit to unit with a variety of indexing and indirect modes. Jumps are all unconditional. The skip instructions provided here are all conditional and variable length in either direction. The indexing for the skip instructions are operand based. The counterpart to the jump and skip instructions are the unconditional and conditional execute instructions. The former execute a (short) list of specified instructions unconditionally. Like the jump instructions, they have a broad range, with indexing and indirect options.

3.3.4.2 Unconditional Jump Instructions

JUMPA :: UID_1^{253} : $ADDRESS_0^{16,383}$: NN

JUMPS :: AUG_0^3 : $ADDRESS_0^{16,383}$: NN

JUMPS :: AUG_0^3 : $ADDRESS_0^{16,383}$: $CHARA_0^{63}$: BN

JUMPS :: AUG_0^3 : $ADDRESS_0^{32,767}$: $AUG2_0^3$: NI

JUMPS :: AUG_0^3 : $ADDRESS_0^{32,767}$: $CHARA_0^{63}$: $AUG2_0^3$: BI

JUMPS :: AUG_0^3 : $ADDRESS_0^{32,767}$: $CHARA_0^{63}$: $AUG2_0^3$: PI

JUMPH :: AUG_0^3 : $ADDRESS_0^{16,383}$: $HWRA_0^{31}$: BN

JUMPH :: AUG_0^3 : $ADDRESS_0^{32,767}$: $HWRA_0^{31}$: $AUG2_0^3$: BI

The fields are interpreted as follows:

UID - absolute Unit ID of 8 bits.

AUG - specification of augment field in register 1.

ADDRESS - address of instruction (0-16,383 in direct modes) or address of address (0-32,767 for indirect mode)

CHARA - name of register character.

AUG2 - second augment field to provide second memory unit number for indirect operations.

HWRA - name of half-word register.

The instruction specifications are interpreted as follows:

JUMPA - direct jump to absolute instruction address specified by the unit ID and address in the instruction.

JUMPS :: NN - direct jump using the AUGMENT field for the UID.

JUMPS :: BN - indexed jump via AUGMENT field, with indexing by the contents of the specified character field. (CHARA) or specified half-word register (HWRA).

JUMPS :: NI - indirect jump with AUG and AUG2 providing the first and second UID respectively.

JUMPS :: BI - indexed indirect jump with index value in specified character (CHARA) or in specified half-word register (HWRA).

JUMPS :: PI - indexed indirect jump with indexing occurring post indirect.

3.3.4.3 Unconditional EXECUTE Instructions

Field interpretations are as in the jump instructions. The new field "VALUE" specifies the number of instructions to be executed. EXECUTE's of JUMP's and other EXECUTE's is allowed, however, the return data is lost for double EXECUTE's; control returns to the most recently issued EXECUTE. EXECUTE of a JUMP is equivalent to jumping and coming back without doing anything. Caution is advised.

$$\text{EXECT :: AUG}_0^3 : \text{ADDRESS}_0^{16,383} : \text{VALUE}_0^{15} : \text{NN}$$

$$\text{EXECT :: AUG}_0^3 : \text{ADDRESS}_0^{16,383} : \text{VALUE}_0^{15} : \text{CHARA}_0^{63} : \text{CN}$$

$$\text{EXECT :: AUG}_0^3 : \text{ADDRESS}_0^{32,767} : \text{VALUE}_0^{15} : \text{AUG2}_0^3 : \text{PI}$$

The NN mode performs a normal direct execute. There is no BN mode. This is provided by issuing a CN mode EXECT with VALUE = 1. The CN mode performs an indexing of the form ADDRESS + (CHARA) * VALUE and executes VALUE instructions. As usual, the notation " (XXX) " means "the contents of XXX".

3.3.4.4 Conditional Skip Instructions

3.3.4.4.1 General

All conditional skip instructions provide a forward or backward skip of 0 to 255 instructions except SKPMB:CN which has a range of -128 to +127. Skip values are best visualized in terms of what happens to the program counter.

SKIP 1	PC → PC+2	
SKIP+N	PC → PC +N+1	
SKIP 0	PC → PC +1	(SKIP 0 instructions behave like NOOP's in this respect)
SKIP-1	PC → PC	(Loop in place)
SKIP-N	PC → PC -N+1	

3.3.4.4.2 Bit Conditioned Register Skips

SKIPB :: $CHARA_0^{63}$: BIT_0^7 : $SKIP_0^{255}$: TF_F^T : FB_F^B

SKIPS :: $CHARA_0^{63}$: BIT_0^7 : $SKIP_0^{255}$: TF_F^T : FB_F^B

SKIPB skips on the value of BIT in CHARA, an amount equal to SKIP. The skip can be executed on the bit value being TRUE = 1 = set, or the bit value being FALSE = 0 = reset. SKIPS operates the same way with regard to the skipping portion of the instruction, with the following differences. If the skip is on TRUE, and the skip is taken, the bit will be reset. Similarly, if the skip is on FALSE and the skip is taken, the bit will be set. These instructions could be called: SKIP ON SET AND RESET and SKIP ON RESET AND SET. If a skip value of 0 is used, the SKIPS instruction is used to reset bits (TF = T) or to set bits (TF = F). The FB field indicates forward or backward. A SETBIT pseudo instruction is provided in assembly language.

3.3.4.4.3 Skip on Word, Character, Half-Word

SKIPW :: WRA_0^{15} : $SKIP_0^{255}$: COND : FB_F^B

SKIPH :: $HWRA_0^{31}$: $SKIP_0^{255}$: COND : FB_F^B

SKIPC :: $CHARA_0^{63}$: $SKIP_0^{255}$: COND : FB_F^B

These instructions differ only as to whether a word, half-word, or character register is referenced, as indicated by

the WRA, HWRA, and CHARA fields respectively. The following condition options may be used for all of these instructions:

POSI - indicated element is numerically positive

NEGA - indicated element is numerically negative

ZERO - indicated element has value zero

NERO - indicated element is not zero

Odd and even valued skips can be obtained by using the appropriate SKIPB or SKIPS instruction of the least significant bit of the element. These are also provided as assembler pseudo-ops.

3.3.4.4.4 Double Element Skips

These skips depend upon the relative values of two elements being considered. The following combinations are mechanized: WORD-WORD, HALF-WORD-HALF-WORD, HALF-WORD-CHARACTER, and CHARACTER-CHARACTER.

The following conditions apply for the skips:

$A > B$:	A is greater than B
$A \geq B$	A is greater than or equal to B
$A = B$	A is equal to B
$A \leq B$	A is less than or equal to B
$A < B$	A is less than B
$A \neq B$	A is not equal to B
$A = \lvert B \rvert$	A is equal to the absolute value of B
$A = -B$	A is equal to -B

The instruction formats are:

SKPWW :: WRA_0^{15} : WRA_0^{15} : $SKIP_0^{255}$: COND : FB_F^B

SKPHH :: $HWRA_0^{31}$: $HWRA_0^{31}$: $SKIP_0^{255}$: COND : FB_F^B

SKPHC :: $HWRA_0^{31}$: $CHARA_0^{63}$: $SKIP_0^{255}$: COND : FB_F^B

SKPCC :: $CHARA_0^{63}$: $CHARA_0^{63}$: $SKIP_0^{255}$: COND : FB_F^B

3.3.4.4.5 Memory Reference, Single Operand Skips

$$\text{SKPMB} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{BIT}_0^7 : \text{SKIP}_0^{255} : \text{TF}_T^F : \text{FB}_F^B : \text{NN}$$

$$\text{SKPMB} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{BIT}_0^7 : \text{SKIP}_{-128}^{+127} : \text{TF}_T^F : \text{CHARA}_0^{63} : \text{VALUE}_0^{15} : \text{CN}$$

In both of these instructions the skip is conditional on the value of a bit in memory. The character address is contained in the half-word register. BIT, SKIP, TF are defined as before. The CN mode provides indexing by the contents of character register CHARA multiplied by VALUE. With VALUE = 1, this provides the equivalent of the NI mode.

$$\text{SKPMW} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

$$\text{SKPMH} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

$$\text{SKPMC} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

The address is contained in a half-word register for these instructions. The conditions are POSI, NEGA, ZERO, NERO.

3.3.4.4.6 Dual Operand, Memory Reference Skips

$$\text{SKRMH} :: \text{AUG}_0^3 : \text{HWRA}_0^{31} : \text{HWRA}_0^{31} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

$$\text{SKRMC} :: \text{AUG}_0^3 : \text{HWRA}_0^3 : \text{CHARA}_0^{63} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

$$\text{SKRCC} :: \text{AUG}_0^3 : \text{HWRA}_0^3 : \text{CHARA}_0^{63} : \text{SKIP}_0^{255} : \text{COND} : \text{FB}_F^B$$

In all cases: The unit ID is in the indicated AUG field: The address of the memory referenced element is in the first half-word register. The address of the second element being compared is in the second half-word or character address. The conditions are as in the double element skips, $A > B$, $A \geq B$, $A = B$, $A \leq B$, $A < B$, $A \neq B$, $A = |B|$, $A = -B$. SKRMH compares a register half-word to a memory half-word. SKRMC compares a memory character to a register half-word, and SKRCC compares a memory character to a register character.

3.3.4.5 Conditional Executes

EXECB :: AUG_0^3 : $ADDRESS_0^{16,383}$: $CHARA_0^{63}$: BIT_0^7 : $COUNT_0^{15}$: TF_F^T

EXECC :: AUG_0^3 : $ADDRESS_0^{16,383}$: $CHARA_0^{63}$: COND : $COUNT_0^{15}$

EXECH :: AUG_0^3 : $ADDRESS_0^{16,383}$: $HWRA_0^{31}$: COND : $COUNT_0^{15}$

EXECW :: AUG_0^3 : $ADDRESS_0^{16,383}$: WRA_0^{15} : COND : $COUNT_0^{15}$

EXECB executes the instruction at the specified address depending upon whether the bit specified in the register character CHARA and BIT is true or false as specified by the TF field. Zero to fifteen instructions can be executed. EXECC, EXECH, and EXECW perform conditional executes on the condition of characters, half-words and words respectively, where the condition is either POSI, NEGA, ZERO, or NERO. This provides executes on: Positive, Positive or zero, negative, negative or zero, or zero.

3.3.5 Load/Store Memory Instructions

3.3.5.1 Load/Store Registers

LOADR :: AUG_0^3 : $ADDRESS_0^{16,383}$: $REGA_0^{15}$: $NUMB_1^{16}$: SET_0^3

STORR :: AUG_0^3 : $ADDRESS_0^{16,383}$: $REGA_0^{15}$: $NUMB_1^{16}$: SET_0^3

These instructions load/store NUMB registers (32 bits) starting with register REGA in register set SET. LOADR and STORR will carry over into the next register set. If the specified total is greater than register 15 in set 3, and the operation is LOADW, the excess words will be placed in register set 0. Similarly, if the operation is a STORW, the data will be taken out of set 0. Caution is advised in the use of this instruction.

3.3.5.2 Load/Store, Half-Word, Character, NN Mode

LOADH :: AUG_0^3 : $ADDRESS_0^{32,767}$: $HWRA_0^{31}$

STORH :: AUG_0^3 : $ADDRESS_0^{32,767}$: $HWRA_0^{31}$

LOADC :: AUG_0^3 : $ADDRESS_0^{65,535}$: $CHARA_0^{63}$

STORC :: AUG_0^3 : $ADDRESS_0^{65,535}$: $CHARA_0^{63}$

These instructions perform the indicated load or store for the character or half-word register specified from the memory address specified.

3.3.5.3 Load/Store via Registers

LORDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CHARAINDEX_0^{63}$: BN

STRDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CHARAINDEX_0^{63}$: BN

LORDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CHARAINDEX_0^{63}$: BN

STRDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CHARAINDEX_0^{63}$: BN

LORDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CHARAINDEX_0^{63}$: BN

STRDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CHARAINDEX_0^{63}$: BN

LORDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

STRDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

LORDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

STRDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

LORDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

STRDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: $AUG2_0^3$: BI

LORDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CINDEX_0^{63}$: AUG_0^3 : PI

STRDW :: AUG_0^3 : $HWRA_0^{31}$: $REGA_0^{15}$: $CINDEX_0^{63}$: AUG_0^3 : PI

LORDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: AUG_0^3 : PI

STRDH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: AUG_0^3 : PI

LORDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: AUG_0^3 : PI

STRDC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: AUG_0^3 : PI

These instructions perform load/store of words, half-words, and characters. The address portion of the instruction is contained in the specified half-word register. Note that the range of the contents of the specified register depends upon whether the item is a word, half-word, or character. For words, the range is 65,536 words, or four memory units, or 3 memory units beyond the one specified by the AUG field. For half-words, the range is 65,536 half-words, or 2 units, while for characters, the range is one unit - that which is specified by the AUG field. The second specification (REGA, HWRA, or CHARA) specifies the source/destination element as a register, half-word register, or character register. All of these instructions are indexed by the contents of a specified character register CINDEX. The indirect modes, as usual, have a second AUG specification given by AUG2, as usual.

3.3.6 Single Operand in Memory via Register

OPERW : AUG_0^3 : $HWRA_0^{31}$: OPS : NN

OPERH : AUG_0^3 : $HWRA_0^{31}$: OPS : NN

OPERC : AUG_0^3 : $HWRA_0^{31}$: OPS : NN

OPERW : AUG_0^3 : $HWRA_0^{31}$: $CINDEX_0^{63}$: $VALUE_1^{16}$: OPS : CN

OPERH : AUG_0^3 : $HWRA_0^{31}$: $CINDEX_0^{63}$: $VALUE_1^{16}$: OPS : CN

OPERC : AUG_0^3 : $HWRA_0^{31}$: $CINDEX_0^{63}$: $VALUE_1^{16}$: OPS : CN

AUG and HWRA as usual specify the AUG field the half-word register that contains the address. Extended range addressing applies. CINDEX is the register character containing the index value. VALUE contains the multiplier for the CN mode. VALUE=1

converts these instructions to the BN mode, however, the assembler provides pseudo ops OPERX:BN. The operations specified by OPS are: CLEAR (set all 0's), LOGICAL COMPLEMENT, ARITHMETIC NEGATE, and ARITHMETIC ABSOLUTE VALUE.

3.3.7 Bit Manipulation in Memory via Register

SETBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : NN

CLRBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : NN

COMBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : NN

SETBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: AUG2 : BI

CLRBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: AUG2 : BI

COMBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: AUG2 : BI

SETBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: $VALUE_1^{16}$: CN

CLRBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: $VALUE_1^{16}$: CN

COMBT : AUG_0^3 : $HWRA_0^{31}$: BIT_0^7 : $CINDEX_0^{63}$: $VALUE_1^{16}$: CN

These instructions provide the setting (SETBT), resetting (CLRBT), and logical complementing (COMBT) of the specified BIT, to be found at the character address in half-word register HWRA. The index is given by character CINDEX. The CN mode is also used to provide the BN mode by using a VALUE of 1. Assembler pseudo-ops are provided by SETBT:BN, CLRBT:BN and COMBT:BN.

3.3.8 Operations Between Registers and Memory

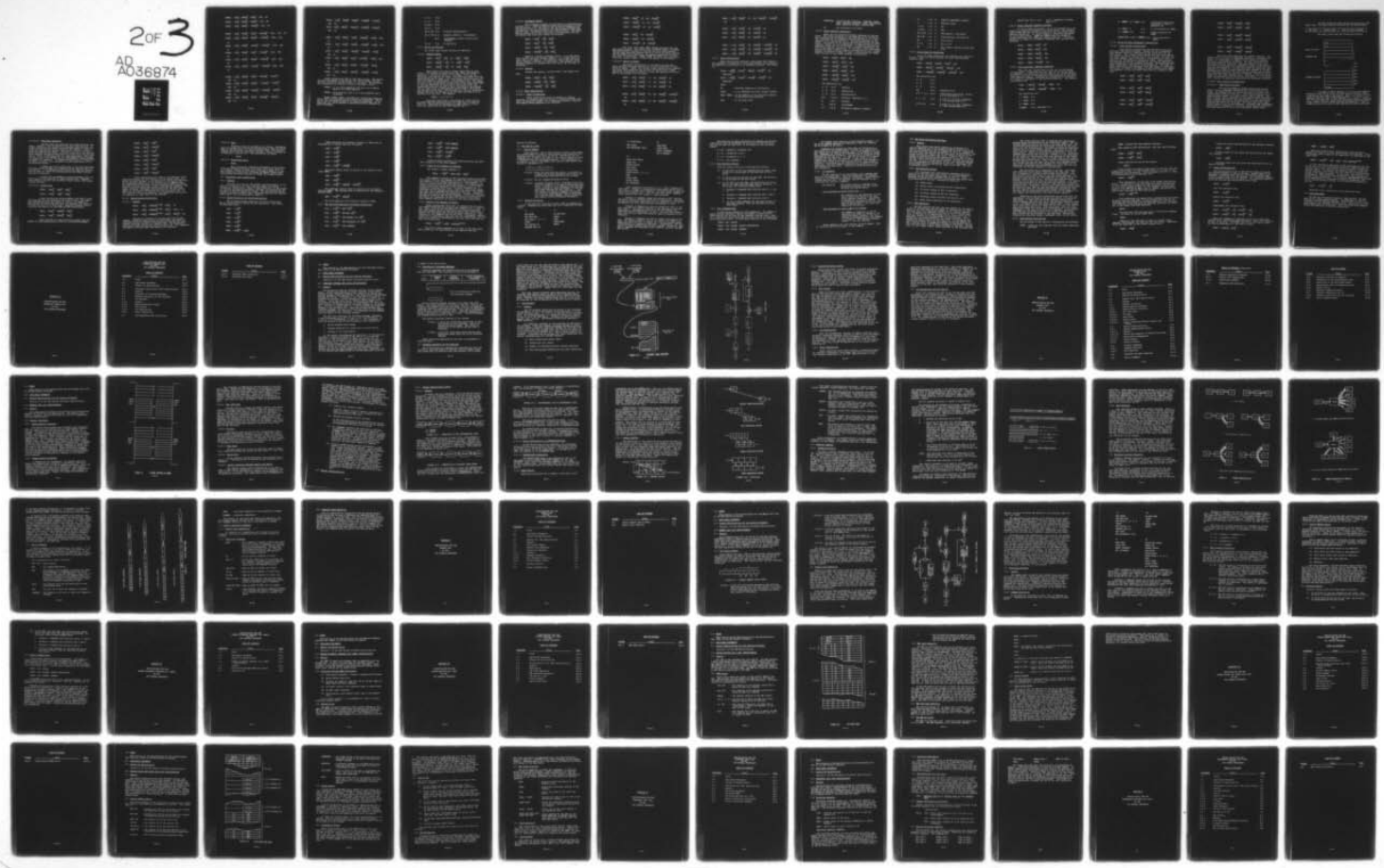
DOTMW :: AUG_0^3 : $HWRA_0^{31}$: WRA_0^{15} : OPS : NN

DOTMH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: OPS : NN

DOTMC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: OPS : NN

AD-A036 874 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394-VOL-8 NL

2 OF 3
AD
AD36874



```
DOTWM :: AUG_0^3 : HWRA_0^31 : WRA_0^15 : OPS : NN

DOTHM :: AUG_0^3 : HWRA_0^31 : HWRA_0^31 : OPS : NN

DOTCM :: AUG_0^3 : HWRA_0^31 : CHARA_0^63 : OPS : NN


DOTMW :: AUG_0^3 : HWRA_0^31 : WRA_0^15 : CINDEX_0^63 : AUG2 : OPS : BI

DOTMH :: AUG_0^3 : HWRA_0^31 : HWRA_0^15 : CINDEX_0^63 : AUG2 : OPS :
  BI

DOTMC :: AUG_0^3 : HWRA_0^31 : CHARA_0^15 : CINDEX_0^63 : AUG2 : OPS :
  BI

DOTWM :: AUG_0^3 : HWRA_0^31 : WRA_0^15 : CINDEX_0^63 : AUG2 : OPS :
  BI

DOTHM :: AUG_0^3 : HWRA_0^31 : HWRA_0^15 : CINDEX_0^63 : AUG2 : OPS :
  BI

DOTCM :: AUG_0^3 : HWRA_0^31 : CHARA_0^15 : CINDEX_0^63 : AUG2 : OPS :
  BI


DOTMW :: AUG_0^3 : HWRA_0^31 : WRA_0^15 : CINDEX_0^63 : VALUE_1^16 :
  OPS : CN

DOTMH :: AUG_0^3 : HWRA_0^31 : HWRA_0^31 : CINDEX_0^63 : VALUE_1^16 :
  OPS : CN

DOTMC :: AUG_0^3 : HWRA_0^31 : CHARA_0^15 : CINDEX_0^63 : VALUE_1^16 :
  OPS : CN

DOTWM :: AUG_0^3 : HWRA_0^31 : WRA_0^15 : CINDEX_0^63 : VALUE_1^16 :
  OPS : CN
```

DOTHM :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{15}$: $CINDEX_0^{63}$: $VALUE_1^{16}$:
OPS : CN

DOTCM : AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{15}$: $CINDEX_0^{63}$: $VALUE_1^{16}$:
OPS : CN

DOTMW :: AUG_0^3 : $HWRA_0^{31}$: WRA_0^{15} : $CINDEX_0^{63}$: $AUG2_0^3$: OPS :
PI

DOTMH :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: $AUG2_0^3$: OPS :
PI

DOTMC :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: $AUG2_0^3$:
OPS : PI

DOTWM :: AUG_0^3 : $HWRA_0^{31}$: WRA_0^{15} : $CINDEX_0^{63}$: $AUG2_0^3$: OPS :
PI

DOTHM :: AUG_0^3 : $HWRA_0^{31}$: $HWRA_0^{31}$: $CINDEX_0^{63}$: $AUG2_0^3$:
OPS : PI

DOTCM :: AUG_0^3 : $HWRA_0^{31}$: $CHARA_0^{63}$: $CINDEX_0^{63}$: $AUG2_0^3$:
OPS : PI

These instructions come in four sets of six. The different sets correspond to the NN, BI, CN, and PI modes, with BN obtained from CN by setting VALUE to 1 as usual. The mnemonics are interpreted as follows in terms of operands A and B:

DOTMW - it is a word operation and the A is in memory while B is a word register.

DOTWM - word operation with A in a word register and B in memory.

DOTMH, DOTHM, DOTMC, and DOTCM are interpreted similarly, except that the operands are half-words and characters. OPS is one of the following arithmetic or logical operations. In all cases but one, the B operand is modified and the A operand remains unchanged.

A + B → B

A - B → B

A and B → B

A or B → B

$A\bar{B}$ or $A\bar{B}$ → B (Logical inequivalence)

AB or $\bar{A}\ \bar{B}$ → B (Logical identity - coincidence)

A ⇔ B (Interchange A and B; both A and B changed)

$\bar{A}$ or B → B (A implies B)

3.3.9 Shifts and Rotates

All shifts and rotates are done in registers.

3.3.9.1 Logical Shifts

SHIFW :: WRA_0^{15} : LR_L^R : ZO : NUM_0^{31} : NUM_0^{31}

SHIFH :: $HWRA_0^{31}$: LR_L^R : ZO : NUM_0^{15} : NUM_0^{15}

SHIFC :: $CHAR_0^{63}$: LR_L^R : ZO : NUM_0^7 : NUM_0^7

These shifts are done on words, half-words or characters. The element is specified by WRA, HWRA, and CHARA respectively. All shifts occur in two steps - first in one direction, and then in the other. The direction taken first is specified by the LR field. If LR is L(eft) there will be a left shift followed by a right shift. The number of bits shifted on the first shift is specified by the first NUM field. The number of bits on the second shift is specified by the second NUM field. If only one NUM field is specified in the source code, the second NUM field will be set to zero, resulting in an ordinary left or right shift as specified by the LR field. Bits which are shifted out are lost. The ZO field is a two bit field which specify what will be shifted in on the right and what will be shifted in on the left. For example:

SHIFH :: 14 L 01 7 3

Specifies that first a left shift of 7 bits will be performed and the vacated bits on the right will be replaced with 1's. This will be followed by a right shift of 3 bits, with the vacated bits on the left replaced by 0's.

3.3.9.2 Arithmetic Shifts

Left arithmetic shifts are equivalent to multiplication by powers of 2. Right arithmetic shifts are equivalent to division by powers of 2. The sign of the operand is preserved and the shift-in bits are consistent with the arithmetic shift mode and the sign. Arithmetic shifts are unidirectional. The appropriate overflow bits (word, half-word, and character) will be set should that occur as a result of the shift.

$$\text{SHATD} :: \text{WRA}_0^{15} : \text{LR}_L^R : \text{NUM}_0^{31}$$

$$\text{SHATW} :: \text{WRA}_0^{15} : \text{LR}_L^R : \text{NUM}_0^{31}$$

$$\text{SHATH} :: \text{HWRA}_0^{31} : \text{LR}_L^R : \text{NUM}_0^{15}$$

$$\text{SHATC} :: \text{CHARA}_0^{63} : \text{LR}_L^R : \text{NUM}_0^{7}$$

SHATD is a double arithmetic shift. In a left shift, this carries over into register (WRA)-1. In a right shift the carry over is to (WRA)+1. The Overflow flag is set only upon double register overflow on a left shift. The contents of the adjacent register is cleared before the shift and the sign is corrected.

3.3.9.3 Rotates

Rotates are logical, unidirectional, and single element.

$$\text{ROTAW} :: \text{WRA}_0^{15} : \text{LR}_L^R : \text{NUM}_0^{31}$$

$$\text{ROTAH} :: \text{HWRA}_0^{31} : \text{LR}_L^R : \text{NUM}_0^{15}$$

$$\text{ROTAC} :: \text{CHARA}_0^{63} : \text{LR}_L^R : \text{NUM}_0^{7}$$

3.3.10 Count Instructions

3.3.10.1 Count in Registers

These instructions count the number of leading, trailing, or internal zero's or one's of a specified word, half-word, or character element into a specified character register. The element being examined is not affected.

KONTW :: WRA_0^{15} : LT : ZØ : $CHARA_0^{63}$

KONTH :: $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$

KONTC :: $CHARA_0^{63}$: LT : ZØ : $CHARA_0^{63}$

KONEW :: WRA_0^{15} : ZØ : $CHARA_0^{63}$

KONEH :: $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$

KONEC :: $CHARA_0^{63}$: ZØ : $CHARA_0^{63}$

The first field (WRA, HWRA, CHARA) specifies the element being counted. The second field in KONTW, KONTH, and KONTC specifies whether leading or trailing bits are being counted. The ZØ field specifies whether Zero's or one's are being counted, and the CHARA field identifies the register character in which the count is being made. KONEW, KONEH and KONEC count the specified bits without regard to where they appear in the element.

3.3.10.2 Counts in Memory

Equivalent instructions to the above are available which performs the same function for a word, half-word, or character in memory. Again, the element being counted is not affected. The first field specifies the AUG and the second field (HWRA) specifies the half-word register in which the address of the element to be counted is to be found.

KOMTW :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: NN

KOMTH :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: NN

KOMTC :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: NN

KOMTW :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$:
BN

KOMTH :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$:
BN

KOMTC :: AUG_0^3 : $HWRA_0^{31}$: LT : ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$:
BN

KOMEW :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: NN

KOMEH :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: NN

KOMEC :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: NN

KOMEW :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$: BN

KOMEH :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$: BN

KOMEC :: AUG_0^3 : $HWRA_0^{31}$: ZØ : $CHARA_0^{63}$: $CINDEX_0^{63}$: BN

3.3.11 Tally Instructions

These instructions perform a conditional skip based on what happens to a character or half-word register as a result of an addition or subtraction of an immediate operand. The format is:

TALYH :: OP_{SUB}^{ADD} : $VALUE_1^{16}$: $HWRA_0^{31}$: $SKIP_0^{255}$: FB :
CONDITION

TALYC :: OP_{SUB}^{ADD} : $VALUE_1^{16}$: $CHARA_0^{63}$: $SKIP_0^{255}$: FB :
CONDITION

Where:

OP - specifies addition or subtraction.

VALUE - is an immediate positive integer operand.

HWRA, CHARA - is the address of the half-word register or character being modified.

SKIP - is the skip value.

CONDITION - is one of the following: OVERFLOW, UNDERFLOW, POSITIVE, NEGATIVE, ZERO, POSITIVE OR ZERO, NEGATIVE OR ZERO, POSITIVE ONE.

FB - the direction of the Skip.

3.3.12 Double Register Operations

These are all operations between registers of various sizes. The fourth character in the mnemonic is by convention the A operand and the mnemonic indicates whether it is a word (W), half-word (H) or character. The fifth character in the mnemonic is by convention the "B" operand and follows the same rules. The results of the operation is always placed in the B register, except for the interchange. B register overflow and underflow flags will be set for the arithmetic operations, but not for the logical operations. In the case of the interchange operations where a smaller element is being moved into a larger element, the smaller element is right justified and the higher order portions of the larger element is unaffected.

OPSWW :: WRA_0^{15} : WRA_0^{15} : OPS

OPSHW :: $HWRA_0^{31}$: WRA_0^{15} : OPS

OPSCW :: $CHARA_0^{63}$: WRA_0^{15} : OPS

OPSHH :: $HWRA_0^{31}$: $HWRA_0^{31}$: OPS

OPSCH :: $CHARA_0^{63}$: $HWRA_0^{31}$: OPS

OPSCC :: $CHARA_0^{63}$: $CHARA_0^{63}$: OPS

The operations are:

A + B	$\longrightarrow$	B	Addition
A - B	$\longrightarrow$	B	Subtraction
A * B	$\longrightarrow$	B	Multiplication
B/A	$\longrightarrow$	B,A	Division, remainder in A.
A	$\longrightarrow$	B	Transfer
A	$\longleftrightarrow$	B	Interchange
-A	$\longrightarrow$	B	Arithmetic negative transfer

$\overline{A}$	⟶	B	Logical complement transfer
A	⟶	B	Absolute value
A or B	⟶	B	OR
A and B	⟶	B	AND
AB V $\overline{A}$ $\overline{B}$	⟶	B	Equivalence, coincidence
A$\overline{B}$ V $\overline{A}$B	⟶	B	Inequivalence, exclusive OR
A$\overline{B}$	⟶	B	Not B and A
$\overline{A}$ V B	⟶	B	Not A or B
$\overline{AB}$	⟶	B	Not A and B, Sheffer Stroke function

3.3.13 Triple Register Operations

In all of these operations, the elements are identical in size. There are three versions for word, half-word and character mode:

TOPSW :: $WRAA_0^{15}$: $WRAB_0^{15}$: $WRAC_0^{15}$: OPS

TOPSH :: $HWRAA_0^{31}$: $HWRAB_0^{31}$: $HWRAC_0^{31}$: OPS

TOPSC :: $CHARAA_0^{63}$: $CHARAB_0^{63}$: $CHARAC_0^{63}$: OPS

The operations are:

A + B	⟶	C	
A - B	⟶	C	
A . B	⟶	C	
B/A	⟶	C	remainder in B
A $\overline{C}$ V B C	⟶	B	conditioned disjunction, select A or B according to C.
A V C B	⟶	B	A or B if C is True; otherwise A - OR through mask in C.
A ($\overline{C}$ V B)	⟶	B	A and B if C is True; otherwise A - AND through mask in C.

$C(\overline{A}\ \overline{B} \vee AB) \vee \overline{C}\ A \rightarrow B$ $A \equiv B$, otherwise A through mask in C.

3.3.14 Single Register Immediate Operand

In all of these operations, one of the operands is to be found as an immediate operand of the instruction. The register always takes the role of the B register in the equivalent double register operations. All double register operations discussed in 3.3.12 above apply. IMMHW and IMMCH stand for immediate operand half-word, and immediate operand character. The interpretation is logical or arithmetic depending upon the operation selected.

MPSHW :: IMMHW : WRA_0^{15} : OPS

MPSCW :: IMMCH : WRA_0^{15} : OPS

MPSHH :: IMMHW : $HWRA_0^{31}$: OPS

MPSCH :: IMMCH : $HWLA_0^{31}$: OPS

MPSCC :: IMMCH : $CHARA_0^{63}$: OPS

3.3.15 Dual Registers, Immediate Operands

There are two instructions in this set for half-words and characters. Other than the fact that one of the operands is immediate, the operations are the same as for triple register functions, with one of the registers being replaced by the specified immediate operand. Operands are interpreted as arithmetic or logical in accordance to the selected operation. The instruction formats are:

TIMOH :: $HWRAA_0^{31}$: $HWRAB_0^{31}$: IMMHW : OPS

TIMOC :: $CHARA_0^{63}$: $CHARAB_0^{63}$: IMMCH : OPS

The operations are:

A + IMMOP $\rightarrow$ B

A - IMMOP $\rightarrow$ B

A * IMMOP $\rightarrow$ B

A/IMMOP $\rightarrow$ B, remainder in A.

$A * \overline{IMMOP} \vee B * IMMOP \rightarrow B$		conditioned disjunction according to value of immediate op.
$A \vee IMMOP * B \rightarrow B$		A or B selected by IMMOP.
$A * (\overline{IMMOP} \vee B) \rightarrow B$		A and B selected by IMMOP.
$IMMOP*(\overline{A}*\overline{B} \vee A*B) \vee \overline{IMMOP}*A \rightarrow B$		$A \equiv B$ selected by IMMOP.

3.3.16 Stack and Queue Management Instructions

3.3.16.1 Push and Pop Instructions

Push and Pop instructions are provided in the word, half-word, and character modes. Control is via a designated word register which is identified by the first field of the instruction. This control word contains the UID of the memory unit where the stack control word is to be found, the stack control word address, and 8 flag bits. The operation is like the operation of the subroutine entry-exit stack. Stack overrun and underrun protection is provided by placing the appropriate jump instruction in the boundary locations. Half-word and character stacks must terminate on word boundaries. Two forms of these instructions are available; with the number of characters, half-words, or words to be PUSH'ed or POP'ed given as an immediate operand; or with the number specified in a register character.

$$PUSHW :: WRA_0^{15} : IMM_1^{16} : WRA_0^{15}$$

$$POPPW :: WRA_0^{15} : IMM_1^{16} : WRA_0^{15}$$

$$PUSHH :: WRA_0^{15} : IMM_1^{32} : HWRA_0^{31}$$

$$POPPH :: WRA_0^{15} : IMM_1^{32} : HWRA_0^{31}$$

$$PUSHC :: WRA_0^{15} : IMM_1^{64} : CHARA_0^{63}$$

$$POPPC :: WRA_0^{15} : IMM_1^{64} : CHARA_0^{63}$$

PUSCW :: WRA_0^{15} : $CHARA_0^{63}$: WRA_0^{15}

POPCW :: WRA_0^{15} : $CHARA_0^{63}$: WRA_0^{15}

PUSCH :: WRA_0^{15} : $CHARA_0^{63}$: $HWRA_0^{31}$

POPCH :: WRA_0^{15} : $CHARA_0^{63}$: $HWRA_0^{31}$

PUSCC :: WRA_0^{15} : $CHARA_0^{63}$: $CHARA_0^{63}$

POPCC :: WRA_0^{15} : $CHARA_0^{63}$: $CHARA_0^{63}$

The first field specifies the control register. The second field is either an immediate operand or a character address. In either case, it specifies the number of elements which are to be PUSH'ed or POP'ed. The last field specifies the first register word, half-word, or character which will participate in the operation. Assembler checks are provided for those instructions with immediate operands to see that overruns are not specified. If more characters are specified than possible (e.g. PUSH 14 words starting with register 6, or POP 26 characters starting with character 44) loading and unloading will take place only for the real registers. There will be no carry into the other register sets. Similarly, only the proper number of characters will be written into memory. An interrupt will, however, be generated indicating the attempted violation.

3.3.16.2 Pull and Shove Instructions

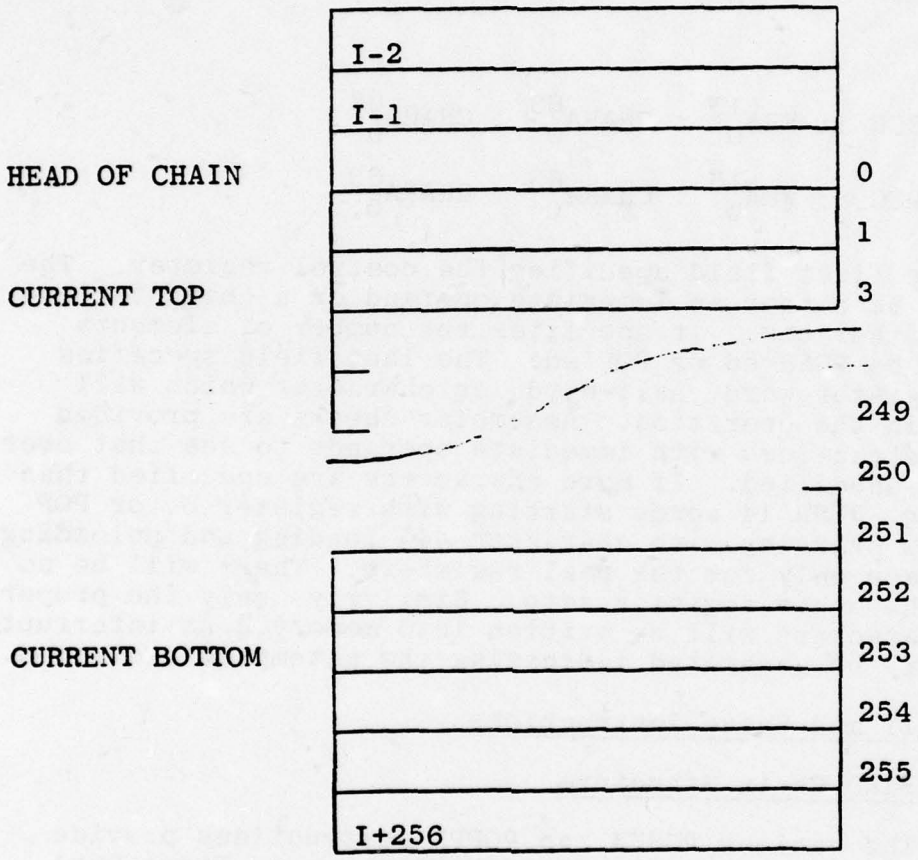
3.3.16.2.1 FIFO Chain Structure

The various PUSHX and POPPX instructions provide control over a programmer specified LIFO (Last-in-First-Out) stack. An analogous set of instructions provide control over a FIFO chain (First-in-First-out). The chain controlled via a register which points to a chian control word in memory. The first (highest order) character of the register does not participate in these instructions and is unaffected. The second character contains the UID of the memory unit in which the control word and the chain will be found. The third and fourth characters contain a 16 bit word address. Extended range addressing applies here; consequently, the chain itself and the chain control word could be in any of four contiguous memory units.

All FIFO chains are word chains and are exactly 256 words long. The chain control word has the following format:

TOP COUNT	BOTTOM COUNT	HEAD OF CHAIN ADDRESS

The chain itself has the following structure:



The Head of Chain Address is a 16 bit word address relative to the UID in which the chain control word is found. It points to the "head of the chain", which is a marker for the first of 256 words that comprises the chain. It is not to be confused with the "TOP" or the "BOTTOM" of the chain. Locations I-1, I-2 and I+256 are loaded with programmer specified instructions, used to control chain overruns and underruns. The I-2 location will not be used in most applications and is optional.

3.3.16.2.2 FIFO Chain Operation

The TOP indicator specifies the oldest entry in the chain. Normally, data is taken off the chain from the TOP. If a PULL instruction is issued, the specified number of words are "PULL'ed" from the TOP location and the TOP counter is decremented and appropriate amount. The chain is rotary, so that if words are PULL'ed from before location 0 relative to the head of chain address, subsequent words will be pulled from location 255 and up. Prior to the execution of a PULL instruction, the relative position of the BOTTOM and the (TOP-NUMBER OF WORDS PULLED) is checked for underrun. If underrun would occur as a result of the PULL, the PULL is not executed, but the instruction in I-1 is executed instead.

A SHOVE instruction adds words to the chain starting with the current BOTTOM, also wrapping around location 255 to 0 as required. If the BOTTOM should overrun the TOP, the instruction at location I+256 will be executed and the SHOVE instruction will be aborted.

Current TOP and BOTTOM are always assumed to be correct prior to the execution of these instructions. Hardware logic properly controls the switchover which occurs, when going "around the bend".

3.3.16.2.3 Instructions

$$\text{SHOVE} :: \text{WRA}_{0}^{15} : \text{IMM}_{1}^{16} : \text{WRA}_{0}^{15}$$

$$\text{PULLS} :: \text{WRA}_{0}^{15} : \text{IMM}_{1}^{16} : \text{WRA}_{0}^{15}$$

These instructions perform SHOVE's and PULL's using the first WRA for the address of the control word. The immediate operand specifies the number of words to be SHOVE'd or PULL'ed and the second WRA specifies the register from which or to which the data is to come or go. Register boundary conventions are the same as for the PUSH and POP instructions.

$$\text{SHOVE} :: \text{WRA}_{0}^{15} : \text{CHARA}_{0}^{63} : \text{WRA}_{0}^{15}$$

$$\text{SHOVE} :: \text{WRA}_{0}^{15} : \text{CHARA}_{0}^{63} : \text{WRA}_{0}^{15}$$

These perform the same operations except that the number of words manipulated is found in the specified CHARA.

APEND :: WRA_0^{15} : IMM_1^{16}

APEND :: WRA_0^{15} : $CHARA_0^{63}$

BHEAD :: WRA_0^{15} : IMM_1^{16}

BHEAD :: WRA_0^{15} : $CHARA_0^{63}$

CURTL :: WRA_0^{15} : IMM_1^{16}

CURTL :: WRA_0^{15} : $CHARA_0^{15}$

UPEND :: WRA_0^{15} : IMM_1^{16}

UPEND :: WRA_0^{15} : $CHARA_0^{15}$

These instructions all refer to the contents of the specified register for the location of the control word. The number of words operated on is specified by either the IMM field as an immediate operand, or the contents of the CHARA field. The APEND instruction moves the bottom pointer DOWN the number of word locations specified, without clearing the contents. The CURTL instruction moves the bottom pointer UP the specified number of words. The BHEAD instructions move the top pointer DOWN, and the UPEND instructions move the top pointer UP. In all cases of overruns or underruns, the appropriate instruction in locations I-1 and I+256 will be executed.

3.3.17 Miscellaneous Instructions

3.3.17.1 Analyze

ANALZ :: AUG_0^3 : $ADDRESS_0^{16,383}$: WRA_0^{15} : NN

ANALZ :: AUG_0^3 : $ADDRESS_0^{16,383}$: WRA_0^{15} : AUG2 : NI

ANALZ :: AUG_0^3 : $ADDRESS_0^{16,383}$: WRA_0^{15} : $HWRA_0^{31}$: BN

Fetches the contents of the word specified by AUG - ADDRESS, interprets that word as an instruction, performs all indicated effective address calculations and returns the final address including its AUG into register WRA. The operations specified by that instruction are not carried out. All registers and memory locations other than WRA are not affected.

3.3.17.2 Halt

Performs a halt of the CPU at all levels. Processing does not resume until initiated externally by a OPEN unit micro. This is the equivalent of a self-inflicted CLOS. The external command could have been generated through a channel by means of an operator's console, etc. The instruction is simply:

HALTS

3.3.17.3 Select Next Level

SNAPS

This instruction terminates processing at the present level and resumes processing at the next lower level which is not blocked. Return to this level following a SNAPS is to the next sequential instruction following the SNAPS, unless other instructions have been executed to change the program counter.

3.4 Inter-Unit (I/U) Instructions

3.4.1 General

The CPU, as a unit of the system, and particularly in its role as a controller of units, must be capable of issuing various I/U instructions. Many of these come about in the course of ordinary instruction execution; but most I/U instructions are not part of the CPU repertoire itself. I/U instructions may be issued in several different ways, depending upon the I/U instruction itself, and the mode used to issue it.

3.4.2 Single Character I/U Micro-Instructions

The following single character I/U micro-instructions may be issued directly within the CPU repertoire. The format is specified below:

GOOF :: UID_0^{255}

REST :: UID_0^{255}

CLOS :: UID_0^{255}

OPEN :: UID_0^{255}

POWR :: UID_0^{255}

RTST :: UID_0^{255} : CHARA

CHARA specifies the character location in which the responding unit's status code will be placed.

XORO :: UID_0^{255}

LOCK :: UID_0^{255}

UNLK :: UID_0^{255}

FINT :: UID_0^{255}

FUST :: UID_0^{255} : $CHARA_0^{63}$

The units control state is placed in the character register CHARA.

LART :: UID_0^{255}

GART :: UID_0^{255}

INXT :: UID_0^{255} : $CHARA_0^{63}$: $STATE_0^{255}$

The resultant control state is placed in the character register CHARA. The forcing state is specified by the immediate operand STATE.

QUEP :: UID_0^{255} : $CHARA_0^{63}$

The units physical ID is placed in register CHARA.

3.4.3 Two and Three Character I/U Micro-Instructions

SLID :: UID_0^{255} : NEW LID_0^{255}

SMID :: UID_0^{255} : NEW MAX ID_0^{255}

SICU :: UID_0^{255} : NEW PRIMARY ICU ID_0^{255}

SISU :: UID_0^{255} : NEW SECONDARY ICU ID_0^{255}

CLPN :: UID_0^{255} : PORT NUMBER$_1^4$

OLPN :: UID_0^{255} : PORT $NUMBER_1^4$

SPNP :: UID_0^{255} : PORT $NUMBER_1^4$

STAT :: UID_0^{255} : $STATE_0^{255}$

CNID :: PID_0^{255} : LID_0^{255}

A unit ID does not have to be specified since the UID = 000 is implicit in the CNID command.

3.4.4 Issue of I/U Command via Register

The format of these commands is:

ISSUE :: UID_0^{255} : DESC:COMM : WRA_0^{15}

The command is placed in the registers starting with WRA. The second field of the command is either the first character of the command, or the command descriptor if a third person, indirect or chained mode has been specified. The command is extracted starting with the specified register. If the command should overrun the registers in the active register set, a diagnostic interrupt will occur. Caution is advised in using this mode for the transfer micros since the result will depend upon what other commands were pending at the time. Generally, the data will come to and from registers. Caution is similarly advised in using the SST and FST commands in this mode since it may result in overlaying the CPU's own control cache memory. Such commands should all be issued in the third person mode.

3.4.5 Issue of I/U Command via Memory

These instructions fetch the I/U command from the specified memory locations and re-issue the command as if it were its own. This is not to be confused with the third person mode. The same command issued in the third person mode, would make it appear that the memory had issued the micro-instruction. The issuance of this command ties up two ports. Subsequent instructions will be executed if they are not I/U instructions. The I/U instruction length is determined by the port logic and does not have to be specified.

ISSUM :: UID_0^{255} : AUG_0^3 : $ADDRESS_0^{16,383}$

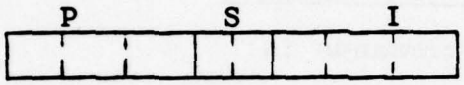
The UID of these commands can be that of the CPU itself. This allows all I/U instructions to be issued by the CPU.

Caution is advised.

3.5 The CPU as a Unit

3.5.1 Control Memory

Every CPU has a 128 or 256 character control cache memory. The control memory is used to store priority and protection information for all other units in the CP. The SID of the requesting unit is used as an index to the control memory to fetch a character with the following structure: the interpretation is similar to that of the memory unit control cache memory (see Section V, paragraph 3.2).



P field - Three bit port priority field. A priority of 0 means that direct communications between the specified unit and this CPU is not allowed.

S field - Two bit command protection field.

I field - Interrupt priority field. Provides a built-in priority category to be used in servicing interrupts. A value of zero indicates that the unit is not allowed to issue interrupts to this CPU. The normal setting for this field is 1. With this value, the CPU will respond only to the I/U interrupt commands discussed in paragraph 3.5.4 and the FINT command. Other values produce special actions discussed elsewheres.

3.5.2 Command Protection

If the priority (P-field) is zero, then no commands are allowed. Commands are subdivided into four categories as shown below:

11	10
SET STATE	GO OFF-LINE
SET UNIT MAX	CLOSE
SET STATE 1, 2, 3, 4	OPEN
INXMT	POWER DOWN
SET LOGICAL ID	RESET
CONVERT NULL ID	

11 (Continued)	01
SET ICUID	OPEN PORT
SET SECONDARY ICUID	CLOSE PORT
	FORCE INTERRUPT
	SET N PRIMARY

00

FETCH UNIT STATE
QUERY PID
RETURN STATUS
LOCK PATH
UNLOCK PATH
FETCH STATE 1, 2, 3, 4
XORO
LOCAL ABORT
GLOBAL ABORT
ALL TRANSFERS

These categories are called 3, 2, 1, and 0 commands respectively. Category 0 commands are those which can be used in the course of normal data transfers, or which cannot change the state of the memory unit. That is, they are "safe" commands - their execution are not likely to cause problems.

Category 1 commands change the routing of data through the matrix and can thereby affect matrix efficiency. Similarly, forcing an interrupt improperly will not generally cause system malfunction but only cause spurious processing to occur.

Category 2 commands are used to change the state of the unit, but not its logical identity or the contents of the control memory. Inappropriate execution of this command will definitely affect performance, in fact, can cause the system to fail, but will not cause the system's logical integrity to be corrupted.

Category 3 commands are used to change the state of the unit, its logical identity, and in the case of the INXMT command, can cause the unit to behave in a totally bizarre manner (deliberately for diagnostic purposes). Consequently, the execution of these commands must be well guarded. Typically, category 3 commands are executable only by CPU's, the ICU, the SMU, and the BU.

The selection of which categories of commands are allowed a particular unit is determined by the value of the S-field in the control word for that unit, in accordance to the following scheme:

S = 00 - category 0 commands only

S = 01 - categories 0 and 1

S = 10 - categories 0, 1, 2

S = 11 - all commands

3.5.3 Port Priority Control

Priority control rules are established as follows:

(1) If the SID's of the two commands are not equal, then priority is established by the priority in the P-field.

(2) If the P-field priorities are the same, the priority is established by the port order.

(3) If the SID's are the same, the priorities are necessarily the same since the same unit is involved. In this case, the following scheme holds:

a. Category 3 commands have priority over 2, 1, and 0.

b. Category 2 commands have priority over 1 and 0.

c. Category 1 commands have priority over 0.

d. If two or more commands for the same SID are in the same category, they are serviced in strict FIFO order.

3.5.4 Port Command Stack

A stack entry consists of four characters. The number of stack entries provided is double the number of memory ports. Stack entries can be used for any commands from any port - that is, a particular stack entry does not have a built-in association with a particular port. A stack entry consists of the following:

(PORT), SID, OPCODE

(PORT), SID, OPCODE, OPCODE CONTINUATION

(PORT), SID, OPCODE, ADDRESS

The OPCODE could take one to four character spaces. If the total command requires two characters, then two commands can be stacked in one entry.

Commands are stacked only when they are received on a port which is in the locked state. Control line signaling by the source unit is used to specify that a new command is on the way. Commands are stacked in accordance to the priority scheme discussed in section 3.5 above. The new SID is compared to the previous SID for that port - if they do not match, all stacked commands for that port are aborted. A port could be in the locked state either as a result of an explicit lock command or because it is waiting for the completion of an ongoing memory operation. The port remains locked as long as there is a command stacked for that port. Instruction stacking operations are transparent.

3.5.5 I/U Commands

The CPU responds to all the generic unit commands when given in the proper mode. There are additional I/U commands related to the handling of external interrupts. These are discussed under the ICU. In addition, the CPU responds to the following unit specific commands:

SET STATE:XX — The program state is changed to XX. Execution continues with the current PC value for the new state.

LOAD REGISTERS:XX:CHARA:LENGTH:DATA .....

The registers in the specified set are force loaded starting with the character register specified. The number of characters is given by LENGTH. All processing is suspended while this command is taking place. All other port operations are held up.

DUMP REGISTERS:XX:CHARA:NUMBER:UID:ADDRESS

The NUMBER of registers in set XX starting with CHARA are dumped to the specified memory location. All processing is suspended while this command is in effect. The register contents themselves are not affected.

These commands, may with caution, be self-issued - that is, with the CPU as the object of the command.

3.6 CPU Alarm Processing and Traps

3.6.1 General

Most events that would classify as interrupts in a typical computer are either queued in memory directly as a result of I/O operations, or are handled by the ICU. Should the ICU determine that the CPU in question is to be interrupted, the ICU will force the subject CPU to the proper state and to the proper instruction within that state. However, there are a number of conditions which are detectable by the CPU, in which the CPU itself can and should take action, without waiting for the ICU. In all of these cases, however, an interrupt message will nevertheless be transmitted to the ICU. Subsequent ICU action might override the action taken by the CPU. This is a programming matter.

3.6.2 Self-Detected Alarms

All interrupts and interruptable conditions detected directly by the CPU itself will result in a self-interrupt in addition to the interrupt message sent to the ICU. Self-detected interrupts are limited to malfunction and alarm conditions detected by the CPU. Included in these are:

(1) Power alarm.

(2) Parity error in received matrix transmission.

(3) I/U micro issued by CPU is rejected.

(4) I/U micro issued by CPU not completed.

(5) Interrupt chain violation (see Section III).

(6) Other alarm conditions (to be specified).

3.6.3 Trap Operation

The CPU contains the UID, the word address of the head of a list of instructions and the number of instructions for each entry on the list (trap entry length) in an internal register. The trap entry list consists of a list of set of instructions, (e.g. 17 entries of 12 instructions each). Every entry on the list must consist of the same number of instructions. Upon detection of an alarm condition, all normal processing is suspended and control shifts to the trap list. If the memory unit required to execute the trap list cannot be reached because of say, a parity error or permanent blockage in the matrix, the CPU will halt and issue a major alarm interrupt to the ICU. Failing that, it will issue a major alarm interrupt to the alternate ICU.

The alarm condition code is combined with the present program state and multiplied by the trap entry length. The resulting number is used as an index to the trap entry list. The instruction starting with the calculated location in the trap entry list are then executed. This mode of operation takes precedence over all other program states. Each entry in the trap entry list can consist of an instruction sequence of up to 16 instructions. Jumps and skips may be taken at will. Similarly, EXECUTE instructions may be issued. However, at the conclusion of the number of instructions that were declared as the length of the trap entry list, control will revert back to the current program state and PC value. This could be a different state and PC than the one that had been left, since the programmer could have issued instructions that changed the state and/or the PC. Additional interrupts (self-generated) can occur, but will not interrupt the current sequence of instructions.

The trap mode has no registers of its own. All normal mode registers can be used as desired in the trap mode. The set is selected by issuing a state change instruction. Note that the CPU will revert back to the last state selected. Instructions executed in the trap mode will not modify the PC of whatever state the CPU is in. Overflow and underflow flags, however, will be set and reset in the normal manner. All instruction executions while in the trap mode are relative to the UID stored in the trap control register. This UID will override all other PC UID's. Similarly, in all other instructions which cause an effective jump, such as jump instructions, subroutine entry and exit instructions, program state change instructions, etc., the base UID for instruction execution will not be changed - it will remain the UID stored in the trap control register.

Execution of the first instruction in the trap list corresponding to the interrupt code has the effect of turning off that alarm. If more than one alarm condition should occur simultaneously, the trap will be to the smallest numerical value - e.g., corresponding to the lowest number alarm. Other alarms which occur while in the trap mode are not lost; but are ignored until acted upon explicitly. When acted upon, they are reset. Additional alarms or a given type occurring prior to the acknowledgement of that type of alarm will be ignored. Operation in the trap mode overrides stalls in the normal program states. Stall counts are not affected by the trap mode. Effectively, entering the trap mode causes a permanent stall for all levels.

3.6.4 Trap Related Instructions

The following IU unit specific instructions are provided:

LTRAP - Loads the trap register with the three characters provided.

STRAP - Stores the trap register contents.

When issued as CPU instructions, they take the following form:

LTRAP :: UID_0^{255} : WRA_0^{15}

STRAP :: UID_0^{255} : WRA_0^{15}

These instructions can be self-issued.

SKPST::

This instruction causes a skip equal to one more than the program state value - if the program state is 0, a skip of one instruction will be executed. If the program state is 1, a skip of 2 instructions will be executed, etc.

RMAIN :: IMM_0^{255}

This instruction, if executed in the normal mode has no effect. If executed in the trap mode will cause the trap mode to be in effect for an additional IMM instruction executions, up to a total maximum of 255. If, for example, the trap mode counter is at 200 and an RMAIN::200 is issued, the CPU will remain in the trap mode for 255 instructions.

ALLAY::: $IMM_0^{16,383}$

This instruction is issued from a program state and forces an entry into the trap mode, at the location specified by the immediate operand relative to the base address in the trap control register. Once the shift is executed, operation is identical to the normal trap mode operation.

DOWNN::

Forcible exit from the trap mode to the current program state. No effect if in the normal mode.

DSKPS::

Remains in the trap mode if there is another alarm pending, and executes the trap list for that alarm. Otherwise, exits the trap mode to the current program state.

CLEAR :: IMM_0^{15}

Clears the alarm code specified by the immediate operand.

BLOCK :: IMM_0^{15}

Blocks alarms for the alarm code specified by the immediate operand.

CLARM :: IMM_0^{15}

Unblocks alarms for the alarm code specified by the immediate operand.

3.7 Flags and Breakpoints

The CPU provides for up to 256 control flags to be used for various purposes in sets of 32 each. These flags are independent of program state and are not affected except under explicit instructions from the program. All operations applied to non-existent flags will be ignored. The following instructions are provided for manipulating these flags:

SETFG :: $FLAG_0^{255}$

Sets the indicated flag.

RSTFG :: $FLAG_0^{255}$

Resets the indicated flag.

COMFG :: $FLAG_0^{255}$

Complements the indicated flag.

SKIPF :: $FLAG_0^{255}$: TF_T^F : $SKIP_0^{255}$: FB_F^B

SKUPF :: $FLAG_0^{255}$: TF_T^F : $SKIP_0^{255}$: FB_F^B

These two commands are analogous to the bit conditioned register skip instructions SKIPB and SKIPS, except the condition is that of the flag. The SKUPF instruction will set or reset the flag as appropriate if the skip is taken.

LOADF :: $RANGE_0^7$: WRA_0^{15}

DUMPF :: RANGE_0^7 : WRA_0^{15}

These two instructions, respectively, load and store a range of 32 flags to or from the indicated word register. Flags are grouped into 8 ranges of 32 flags each, as indicated by the range field.

RETRF :: FLAG_0^{255} : TF_T^F : AUG_0^3 : WORD $\text{ADDRESS}_0^{16,383}$

If the instruction specifies a TRUE condition and the flag is not set, or if the instruction specifies a FALSE condition and the flag is set, the instruction will execute a normal subroutine return. If however, the indicated flag is in the required condition, the instruction will execute an unconditional branch to the named location. In this case, the stack will not be affected. This instruction can always be used instead of a normal return instruction. It is used for performance monitoring for establishing the equivalent of console breakpoints, and halts, without paying a penalty if the condition indicated is not met.

HALTF :: FLAG_0^{255} : TF_T^F

Executes a halt conditional on the specified flag.

3.8 Operating Modes

The CPU has two operating modes - STEP and RUN. The RUN mode is the normal operating mode. The STEP mode is used for diagnostic and de-bugging purposes. These modes are set by use of a pair of unit specific IU instruction. Receipt of the first STEP command sets the STEP mode. Each additional STEP command allows the execution of the next instruction.

SECTION III

SPECIFICATION FOR THE INTERRUPT CONTROL UNIT OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE INTERRUPT CONTROL UNIT OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	III-1
2.0	Applicable Documents	III-1
2.1	General CP Specification	III-1
3.0	Interrupt Control Unit (ICU) Specification	III-1
3.1	General	III-1
3.2	Structure of Interrupt Messages	III-2
3.3	Hardware Operation of the Interrupt	III-2
3.4	ICU Software	III-3
3.4.1	General	III-3
3.4.2	System Wide Major Alarms	III-6
3.4.3	Unit Alarms	III-6
3.4.4	I/O Terminations	III-6
3.4.5	Other Terminations	III-6
3.5	ICU Designation and ICU Failure	III-7

LIST OF FIGURES

FIGURE	TITLE	PAGE
III-1	Interrupt Table Structure	III-4
III-2	Interrupt Flow Chart	III-5

1.0 SCOPE

This section is the specification for the Interrupt Control Unit (ICU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Specification for the Central Processor

(Section I of the CPS Central Processor Specification)

3.0 INTERRUPT CONTROL UNIT (ICU) SPECIFICATION

3.1 General

The CP does not have an interrupt facility in the ordinary sense. Unlike other systems in which interrupt lines connect devices to the (typically single) CPU, there is no direct connection between a unit and the ICU. The connection is established through the matrix. Furthermore, the identity of the ICU is dynamically variable. At any instance of time, two ICU's may be defined. Units that would in an ordinary system respond to a situation by issuing an interrupt to a particular CPU, will respond instead by transmitting an interrupt message to the primary ICU. Failing to connect with it, it will re-issue the interrupt message to the secondary ICU. Once the ICU has accepted the interrupt message, the unit is effectively freed of its responsibility regarding that interrupt. Its subsequent action depends upon the nature of the interrupt message.

The ICU, upon receiving the interrupt message, processes it and determines which CPU (if any) is to be interrupted. If a CPU is to be interrupted, it will do so. That interrupt action may consist of any of the following in any combination:

a. Forced program state change.

b. Loading registers of a given set to stored values.

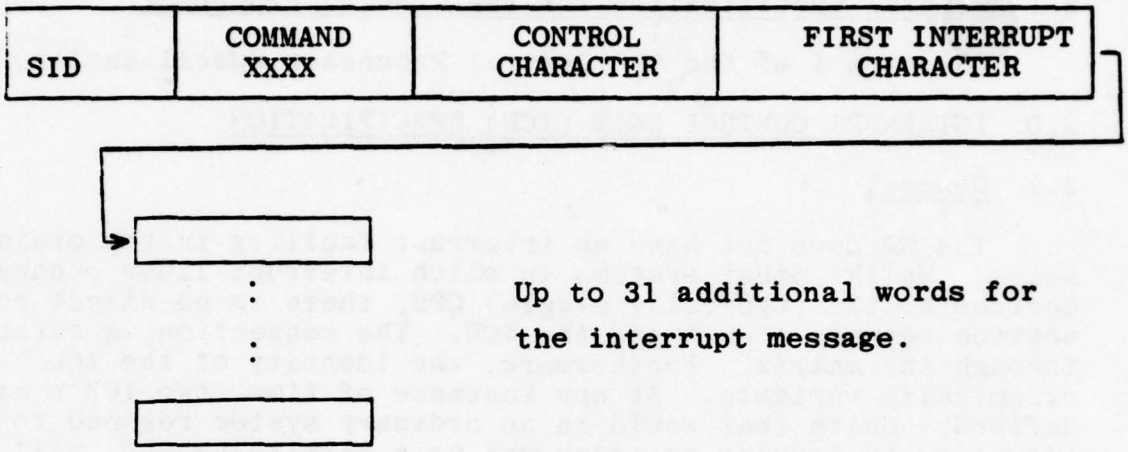
c. Loading of the cache memory.

In the course of processing the interrupt, the ICU may have recourse to examine the state of the CPU it is to interrupt, therefore, it has the ability to query its logical ID, its physical ID, the contents of its cache memory, the contents of its registers, etc. It is clear, that the ICU can be called upon to perform somewhat complicated actions. Actions that are normally wired to the logic of the interrupt control hardware. However, in the CP, the ICU functions are performed by just another CPU that has been assigned to that task. Therefore, all of the features and capabilities of the CPU described in Section

II apply to the ICU as well.

3.2 Structure of Interrupt Messages

Interrupt messages are transmitted as IU micro-commands with the code XXXX. The structure of the entire message is:



Every interrupt message consists of at least four characters. The first character, as usual is the SID. This is followed by the command, with the code XXXX. This is followed by a control character (described below) and at least one additional character with interrupt specific data. This can be followed by up to 31 more words of interrupt data. Interrupt messages are always a multiple of 4 characters in length. Interrupt messages are transmitted one word at a time.

The control character consists of two fields:

C-field - a three bit field which identifies the general nature of the interrupt. 000 is reserved for major alarms. The coding is particular to the device and the type of interrupt.

L-field - a five bit field which specifies how many additional words there are in the interrupt message.

These fields are generated by the unit in accordance to built-in logic.

3.3 Hardware Operation of the Interrupt

The C-field value is combined (by concatenation) with the I-field in the cache memory to provide a six bit index value. This index value is used with the trap register UID to create

a word index for the low order 64 words of that memory unit. If the index value is 0, the interrupt will be handled in the trap mode with index value 0, which should have been set up to correspond to the treatment of external interrupts which are to be handled by the trap mode. Otherwise, the contents of the referenced word is fetched. It is interpreted as a FIFO chain control word using the same ICU as a base. The interrupt message length is used to determine if there will be an overrun. If an overrun will occur, then instruction at I-2 of the chain will be executed (effectively in the trap mode), otherwise, successive words of the interrupt message will be stored on the chain. CPU instruction execution is locked out until the check has been performed. The structure of the tables and registers involved is shown in Figure III-1. Figure III-2 is a flow chart for the operation.

Note that because extended range addressing applies, the interrupt FIFO chains can be in any of the 4 adjacent memory units starting with the UID specified in the trap control register. However, the chain control words must be in the first 64 locations of the UID specified by the trap control register.

3.4 ICU Software

3.4.1 General

The ICU hardware identifies the nature of the interrupt and decides if it is to be treated in the trap mode or in the normal interrupt mode. If it is to be treated in the interrupt mode, the interrupt message, as originally received is queued on one of up to 63 FIFO chains. Hardware provides protection against chain overrun. The hardware effectively performs a SHOVE operation.

All remaining treatment of the interrupt is performed by ICU software. Since the original interrupt message is queued without change, arbitrary relations between the interrupt type, its priority, the unit it came from, the CPU (if any) which is to be interrupted, the program state and location to which the interrupt is to force the CPU, etc., may be established by the software. Interrupts can be divided into three convenient categories, depending upon the intended response:

(1) Major alarms with system impact.

(2) Alarms with unit impact.

(3) Normal I/O terminations which require interrupt.

(4) Non-interrupting terminations and other conditions.

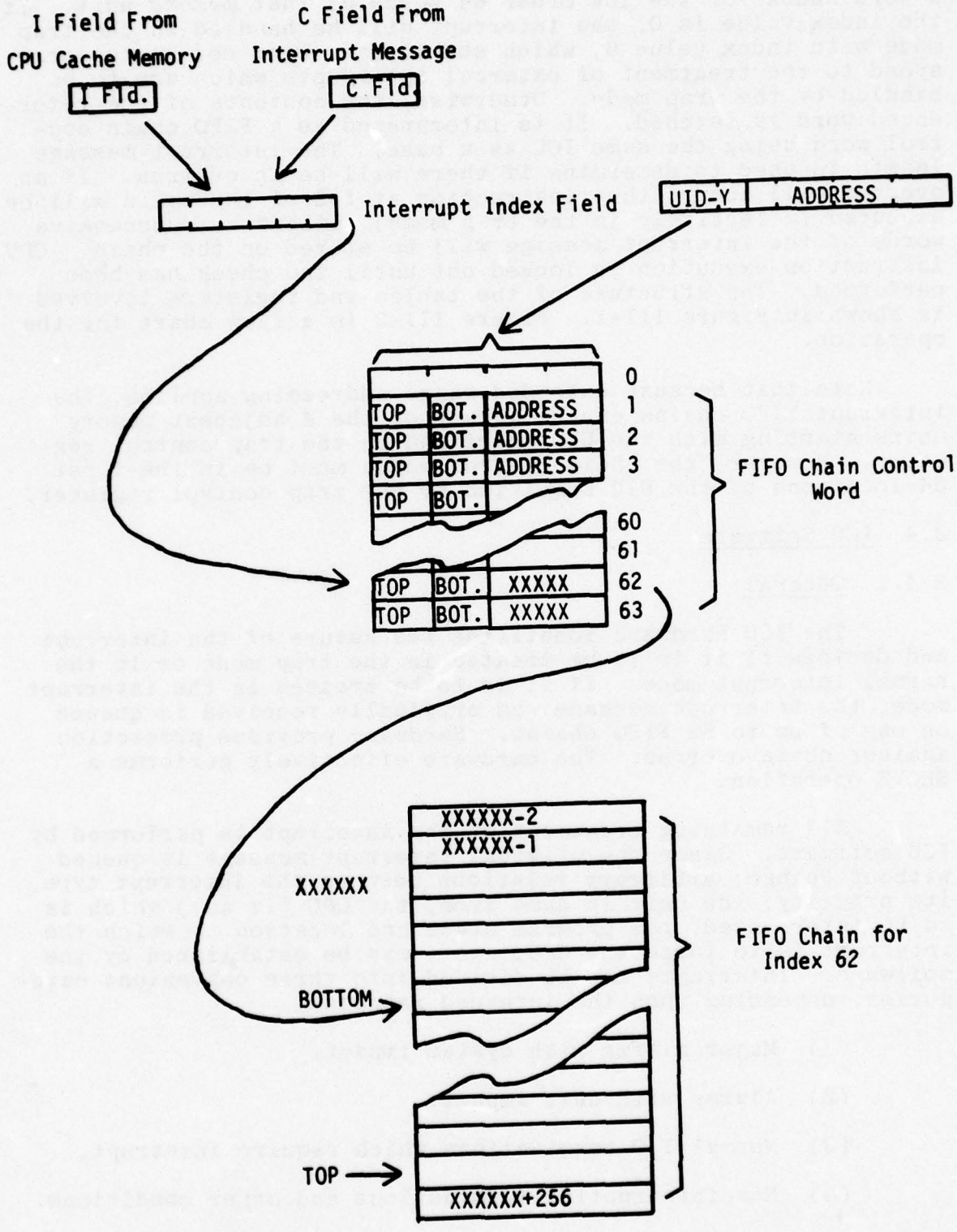


FIGURE III-1 INTERRUPT TABLE STRUCTURE

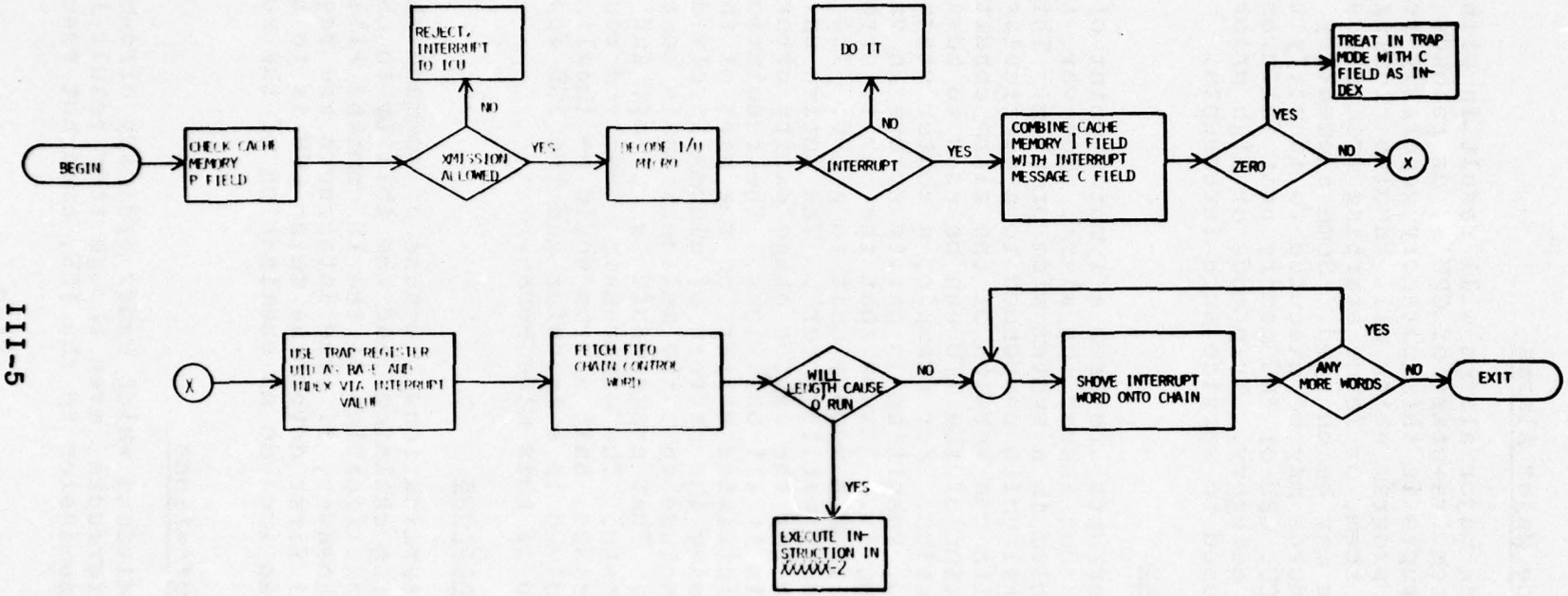


FIGURE III-2 INTERRUPT FLOW CHART

3.4.2 System Wide Major Alarms

System wide major alarms will result in either reconfiguration of the system, re-start of CPU's, or reloads of program memories. Interrupts in this category generally result in executing a complex program which will involve closing units down, re-bootstrapping them, or force starting from predetermined locations. LID's may be changed. Some elementary test and diagnostic procedures may be executed to identify the malfunctioning unit. ICU, CPU or ICU memory unit failures should be included in this category. Trap mode or high priority program state should be used to service such interrupts.

3.4.3 Unit Alarms

These interrupts are from a symptom point of view localizable to the unit that issued the alarm. However, it can turn out that the problem is a system wide problem. This would be the case if several units responded to a particular situation, each one giving its own version of the alarm condition. The centralized position of the ICU can be used to advantage to make this distinction. For example, a center stage matrix unit has malfunctioned, resulting in parity errors in various transmission traversing it. Assume that the parity error occurred only on the path from a memory unit to a CPU. The CPU provides an alarm for "memory parity error". The active unit stage also provides an alarm for the center stage parity error. The center stage, however, is itself oblivious. The resolution of this bug would be a self-initiated restart on the part of the CPU. The ICU, however, seeing the pattern of alarms involved, could be programmed to conclude that the malfunction is most likely in the center stage. That stage would be cut off and handed over for diagnostic tests. The other units involved would merely retry. In most cases, unit alarms would be localized to the one of two units involved in a transfer and the ICU would simply take retry action if this made sense.

3.4.4 I/O Terminations

Most I/O terminations, because of command chaining, data chaining, condition chaining, and the ability to chain interrupts as part of the specification of the IU command will not result in interrupts. However, if true interrupts are required, the ICU software will first determine which CPU is to be interrupted and how. This can include an examination of the subject CPU's state and PC.

3.4.5 Other Terminations

Other conditions which under ordinary circumstances might be treated by interrupts, even though they result in an interrupt message transmission to the ICU, need not result in the

bona fide interrupt of any other CPU. This could come about because of peculiarities of the device design, or because of the functional role played by the device in question. Thus, all channels will give interrupts on missed transfer conditions. However, if the channel in question is being manipulated by a diagnostic program in a diagnostic CPU, it may be desirable not to actually interrupt that CPU, but rather to queue the condition as detected. Similarly, other "off-line" diagnostic tests could, because of unit malfunctions, cause all kinds of events that would normally result in interrupts, but because of the interposition of the ICU, they could be treated in a less frenzied manner.

3.5 ICU Designation and ICU Failure

Every unit contains the UID of the ICU and some other CPU designated as the alternate ICU. Failure to transfer an interrupt message and receive confirmation thereto from the primary ICU will result in the transferring unit's unilateral transfer of the interrupt message to the designated alternate ICU. The ICU CPU itself must contain the same two UID's. The primary ICU for the primary ICU should be the alternate ICU. The alternate ICU for the primary ICU should be some third CPU, typically concerned with system level executive functions, or otherwise one allowed to execute instructions out of the memory unit(s) that contain the recovery procedures for ICU failure. The primary ICU for the alternate ICU could be the primary ICU, while the secondary ICU for the secondary ICU could be the same third CPU. Note that the designated alternate need not "contain" (i.e., be allowed access) to the entire ICU tables and memory, a small subset of this is all that is necessary.

SECTION IV

SPECIFICATION FOR THE CHANNEL UNIT OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE
CHANNEL UNIT
OF THE
CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	IV-1
2.0	Applicable Documents	IV-1
2.1	General CP Specification	IV-1
3.0	Channel Unit (CU) Specification	IV-1
3.1	General	IV-1
3.2	Channel Interfaces	IV-1
3.2.1	Channel-Matrix Interface	IV-1
3.2.2	Channel-Device Interface	IV-1
3.2.2.1	DCA, CDA Lines	IV-3
3.2.2.2	YT Lines	IV-3
3.2.2.3	Data Lines	IV-3
3.2.2.4	Parity Lines	IV-3
3.2.2.5	Control Transfers Between Channel and Device	IV-3
3.3	Channel Characteristics	IV-4
3.3.1	Channel Master/Slave States	IV-5
3.3.1.1	General	IV-5
3.3.1.2	Device Side Behavior of Master/Slave Mode	IV-6
3.3.1.3	Master/Slave Designation	IV-6
3.3.2	Speed Control	IV-6
3.3.3	Channel Buffers	IV-7
3.4	Transfer Commands	IV-8
3.4.1	Command Structure	IV-8
3.4.2	Data Chaining	IV-11
3.5	Bilateral and Dual Transfers	IV-11
3.6	Device Commands	IV-14

TABLE OF CONTENTS (Continued)

PARAGRAPH	TITLE	PAGE
3.7	Channel Imperative Commands	IV-16
3.7.1	Generic Unit Imperatives	IV-16
3.7.2	Additional Commands	IV-16
3.8	Immediate Mode Operation	IV-17

LIST OF FIGURES

FIGURE	TITLE	PAGE
IV-1	External Interface of Channel	IV-2
IV-2	Connection of Two Asynchronous Lines	IV-5
IV-3	Connection to Variable Speed Boram	IV-5
IV-4	Asynchronous Line to Synchronous Line	IV-6
IV-5	Buffer Action	IV-7
IV-6	Transfer Command Structure	IV-10
IV-7	Transfer Combination Set Up	IV-12
IV-7a	Transfer Combination Set Up Continued	IV-13
IV-8	Device Command Format	IV-15

1.0 SCOPE

This section is the specification for the Channel Unit (CU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Specification for the Central Processor

(Section I of the CPS Central Processor Specification).

3.0 CHANNEL UNIT (CU) SPECIFICATION

3.1 General

All communications between units within the CP and devices external to the CP is via channel units. The Channel Units (CU's), therefore, provide the means for the CP to communicate with the outside world.

3.2 Channel Interfaces

3.2.1 Channel-Matrix Interface

The channel has a normal unit interface with the matrix. That is, there is a validation/priority cache memory (256-512 bits for three priority levels) and LID, MID, ICU ID, etc., storage. In addition, the channel stores in a number of registers, additional information required to execute I/O transfers or other I/U commands. These registers contain a unit ID, and depending upon the unit type, an additional two characters of address data. The following information can be stored: data address, data count, command chain address, data chain address, condition chain address, and termination instruction. A channel can be involved in three simultaneous transfers (2 inputs and one output) and consequently the above information can be held for each of the three transfers. These instruction storages are over and above the registers used to store micro-commands. The micro-command storage registers are still required because a given I/U macro is executed as a sequence of unit generic micro-commands.

3.2.2 Channel-Device Interface

A channel has two interfaces: an internal interface which is a standard unit interface to the matrix, and an external interface, which is also a standard interface. While the internal interface may have one or more ports, the external interface is always a single port interface. That interface is a full duplex, character parallel interface, with a number of control and timing lines. Figure IV-1 shows the structure of the external interface.

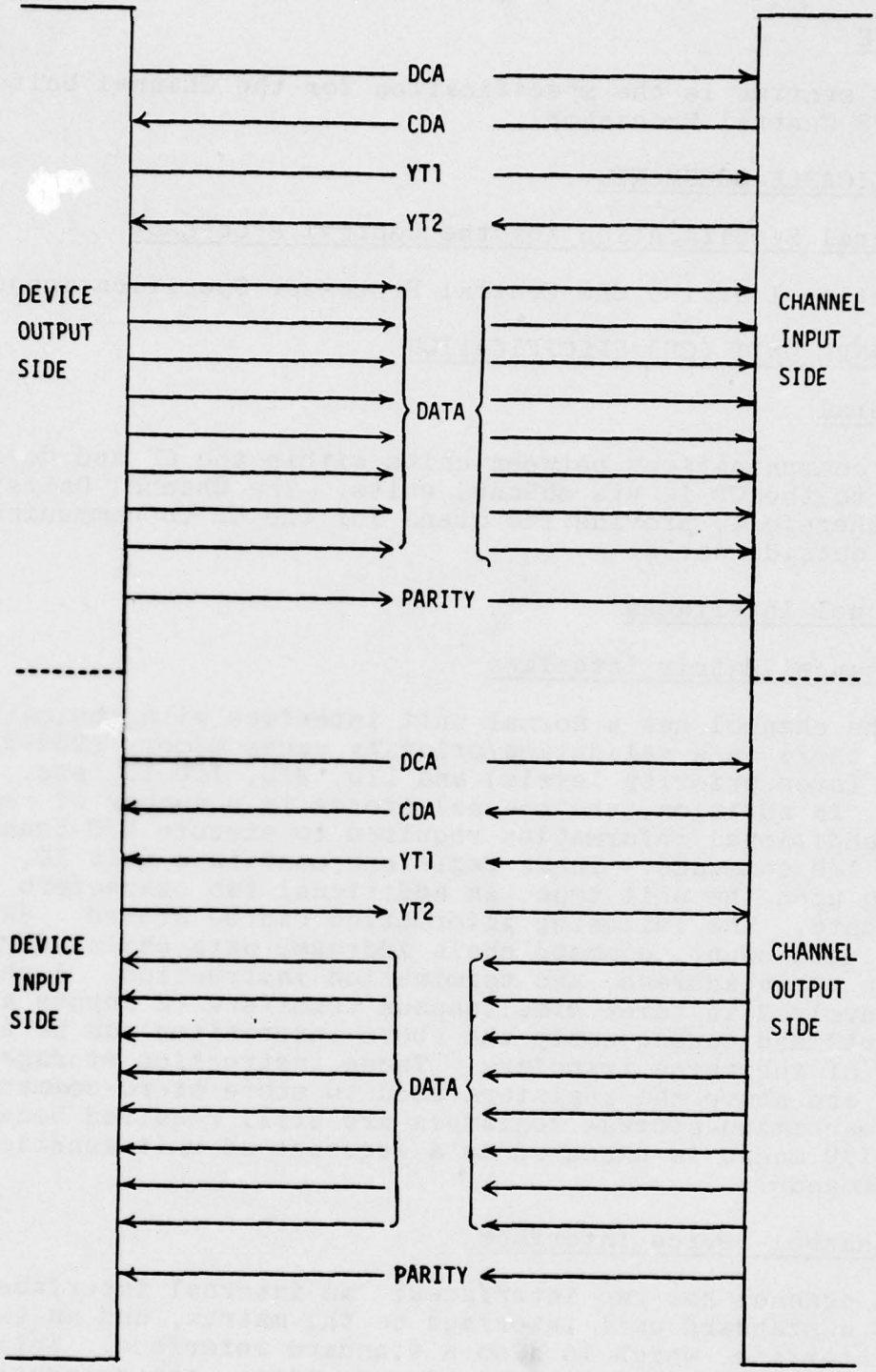


FIGURE IV-1 EXTERNAL INTERFACE OF CHANNEL

The interface is symmetrical and can operate in the full duplex mode. Except for the direction of the signals involved, the interface from the device input side to the channel output side, and the interface from the device output side to the channel input side are identical. There are a number of other control lines which have not been shown in the interest of clarity. Each half of the interface consists of the following lines: DCA, CDA, YT, DATA, and PARITY. Each is discussed below.

3.2.2.1 DCA, CDA Lines

The DCA line is a signal line whose logical value specifies that a single character transfer of data or control is to occur. If the DCA line value is "true" then the character is on its way, and has been initiated on the behest of the device. That is, the device is providing the clock. In this case, the channel input side is slaved to the device output side. Alternatively, if the CDA line is activated, it indicates that the character is transferred on the behest of the channel - in which case, the device output side is slaved to the channel input side. The DCA and CDA line perform identical functions on the other half of the interface.

3.2.2.2 YT Line

The logical value of the YT line ("true-false") indicates whether the current character is to be treated as data or if it is to be interpreted by the recipient. Thus, in a transfer from the device to the channel, the YT line specifies if the current character is data or control. Similarly, in the opposite direction, the state of the YT line specifies if the device is to interpret the character as data or control.

3.2.2.3 Data Lines

The data lines are 8 lines in parallel, used to transfer a character between the device and the channel or vice versa.

3.2.2.4 Parity Line

Every character transferred across the interface has a parity bit. That parity bit is checked by the recipient of the transfer.

3.2.2.5 Control Transfers Between Channel and Device

The channel passes control information to the device. Some of that control information is given to the channel as data, while some of it is derived by the channel and re-packaged, so to speak for the device. The former data is not interpreted by

the channel, the latter must be. Similarly, there is a need to pass control type information from the device to the channel. Again, there is information which is to be treated by the channel as data, and information which is to be interpreted by the channel, converted and re-packaged for some other unit to handle. There is, therefore, in operation over the control interface between the channel and device a primitive instruction repertoire. The operation of this repertoire is completely transparent to the programmer. It is of interest primarily to the designer of a device, particularly, the designer of a device-channel interface.

(1) Control over transfer length:

Transfer length, be it of control information or data, is done by means of the condition of the appropriate DCA or CDA line.

(2) Distinction between control and data:

If the characters are to be interpreted by the device or (in the opposite direction) by the channel, the appropriate YT line will be raised.

(3) All commands, whatever the direction or the mode of transfer consist of at least one character. A command given to the device by the channel is either an INPUT (to the channel), OUTPUT (from the channel) or device imperative. In all three cases, this may be followed by device specific data which is interpreted by the unit. In particular, the device imperative must be followed by a character that specifies the particular imperative command.

(4) A device communicating control information to the channel has the following choices: error termination, normal termination, special condition information. Each type identifies the direction (if applicable) of the transfer involved. The completion of the execution of any command is signalled by a termination character sequence. Like the channel-device commands, the device-channel information can be followed by a number of auxiliary characters which are not to be interpreted by the channel, but rather, are to be passed on by the channel to the appropriate unit. Special condition information can be sent at any time and does not necessarily correspond to the termination of a command.

3.3 Channel Characteristics

3.3.1 Channel Master/Slave States

3.3.1.1 General

A channel has two interfaces - with the device and with the matrix. In any transfer of data, be it into or out of the system, in addition to specifying the direction of the transfer, we must specify which device is to provide the clocking for the transfer. For example, if the device is an asynchronous input line, it is clear that the channel cannot tell when the next character is to be expected. Somehow or other, the device must initiate the transfer. In this case, the device is the master and the channel is the slave. Carry this further, and say that the characters are being transferred to memory. The memory cannot initiate the transfer since it does not know when the channel will have a character ready for it. For this interface, it is the channel which is the master and the memory which is the slave. Consider now the interconnection of two such full duplex asynchronous lines, as shown in Figure IV-2.

M = MASTER
S = SLAVE

FIGURE IV-2 - CONNECTION OF TWO ASYNCHRONOUS LINES

Both channels are in the same state, being MASTER on output to the device and SLAVE'd to it on input. The matrix interface, however, has the situation reversed. Consider now, the connection of a block addressed random access memory which can operate at whatever speed it is capable of, being connected to a disc unit, which has a preset clock speed. The situation is now depicted in Figure IV-3.

FIGURE IV-3 - CONNECTION TO VARIABLE SPEED BORAM

This connection should make it clear that MASTER/SLAVE states have nothing to do with transfer direction and that the two paths could be in different states independently. There will be, in this case, an alternation of MASTER/SLAVE across the

channel. If an asynchronous line is now hooked to a synchronous line, the situation of Figure IV-4 might develop.

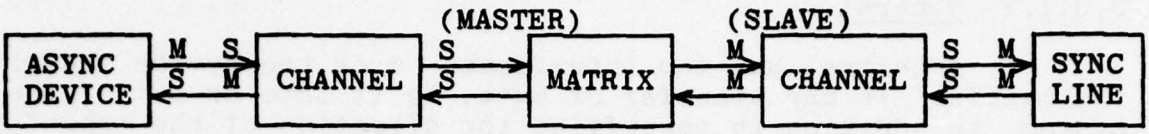


FIGURE IV-4 - ASYNCHRONOUS LINE TO SYNCHRONOUS LINE

Now one or the other channels must have a SLAVE/SLAVE set up. This kind of hook-up would not be made unless there were some assurance that things would not go wrong. But, there is a limited amount of buffering in the channel. At very slow character rates, in cases where delays were not important, this configuration could conceivably come up.

The final configuration need not be drawn. It is clearly the MASTER/MASTER mode, in which the channel is acting as a master in both directions. Under these circumstances, the channel initiates all character transfers at a programmed speed.

The MASTER/SLAVE designation is applied separately as to the direction of the transfer, however, all ports will have the same designation in the same direction. MASTER/SLAVE designation applies to the controlling clock for data transfers. It does not apply to the unit's ability to receive commands, to issue commands, etc.

3.3.1.2 Device Side Behavior of MASTER/SLAVE Mode

On the device side of the interface, the equivalent action is determined by whether the DCA or CDA line is raised to indicate if a character is on the way. If the DCA line is used, the channel is in the SLAVE mode. If the CDA line is used, the channel is in the MASTER mode.

3.3.1.3 MASTER/SLAVE Designation

The MASTER/SLAVE mode is set through the use of the unit generic micro-command STAT (SET STATE) with the code 0000XXXX. Where the X bits designate MASTER (0), or SLAVE (1), in the following order: device side input leg, device side output leg, matrix side input (all ports), matrix side output (all ports).

3.3.2 Speed Control

Speed can be selected for a channel if and only if all

interfaces are in the MASTER mode. That is, the channel must be controlling the data transfers in both directions and providing the timing pulses to both the internal unit (matrix side) and the device. This is done by using a channel specific non-transfer command "SET SPEED". The speed code is a four character binary field (32 bits). A speed of 0 is interpreted to mean that the channel will free-run and will transfer characters on request characters at the maximum rate it is capable, without, however, overrunning buffers or otherwise messing up through lost characters and the like.

Speed control is achieved by using the speed control bits as a countdown value. At each matrix character transfer clock period (approximately 50 nanoseconds) a counter in the channel is decremented. When that count reaches zero, the transfer requests are issued and the counter is reset to the value programmed. In effect, the programmed count value provides a division of the matrix clock rate. For example, to provide 5 level baudot output at 45.5 baud, a count value of 3252747_{10} is programmed. The channel may not succeed in transferring the character the first time it tries because of matrix blockage, busy units, etc. If the character is transferred at any time preceding the conclusion of the current countdown, it will be considered as having successfully transferred the character. If the character is not transmitted (or received) prior to the conclusion of the countdown period, then a missed transfer interrupt with appropriate information is generated.

3.3.3 Channel Buffers

The channel has a four character data buffer for each inlet and outlet. That is, there is a four character input buffer on the device side, a four character output buffer on the device side, as well as four character input and output buffers on the matrix side. All ports share the matrix side buffers. The two buffers for a given direction act as a pair. The way in which characters are transferred from one buffer of a pair to the other can be controlled. The pairs in both directions always operate in the same buffering mode. Figure IV-5 depicts the action of a buffer pair in one direction:

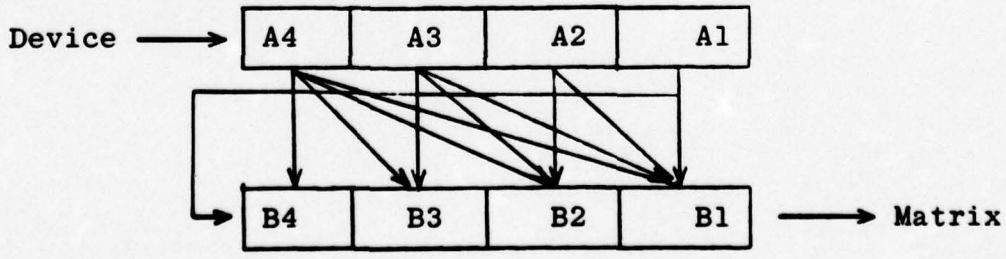
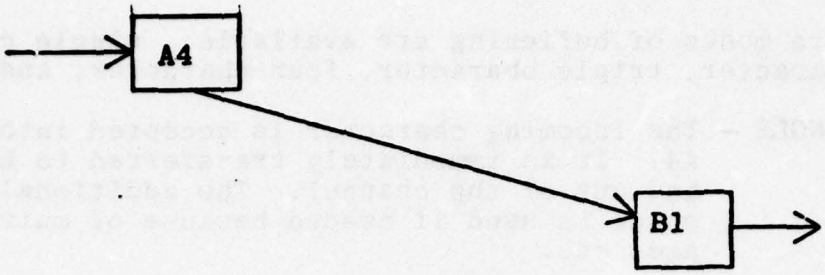
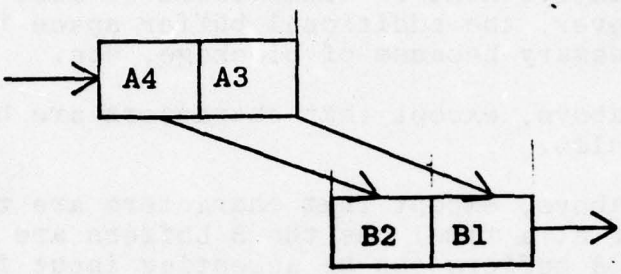


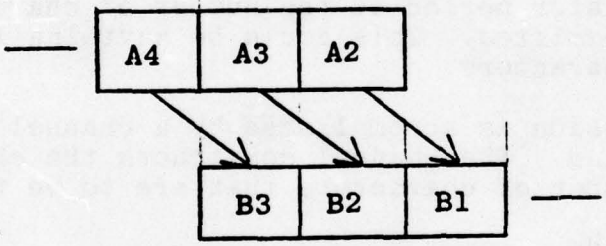
FIGURE IV-5 - BUFFER ACTION



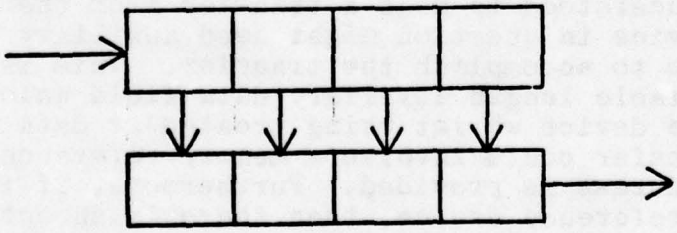
SINGLE CHARACTER BUFFER



TWO CHARACTER BUFFER



THREE CHARACTER BUFFER



FOUR CHARACTER BUFFER

FIGURE IV-5 - Continued

Five modes of buffering are available: single character, double character, triple character, four character, and free.

SINGLE - The incoming character is accepted into buffer A4. It is immediately transferred to buffer B1 and out of the channel. The additional buffer space is used if needed because of matrix blockage, etc.

DOUBLE - Characters are transmitted a pair at a time. Normally, A3A4 is transferred to B2B1 and out. However, the additional buffer space is used if necessary because of blockage, etc.

TRIPLE - As above, except that characters are handled as triples.

FOUR - As above, except that characters are transferred four at a time. As the B buffers are outputting, the A buffers can be accepting input from the device.

FREE - The buffers are treated as a serial shift register in which A1 is tied to B4. As many characters as possible are accumulated. The number of characters accumulated at the next available transfer period is the number of characters transmitted. This could be anything from 1 to 8 characters.

Each transmission is accomplished by a channel generated transfer micro-command. The channel constructs the command appropriate to the number of characters that are to be transferred.

3.4 Transfer Commands

3.4.1 Command Structure

Two basic transfer commands are available: IN and OUT. "IN" is understood to mean a transfer from a device into the CP. "OUT" is understood to mean a transfer from the CP to a device. The device in question might need auxiliary data in order to be able to accomplish the transfer. This is provided by adding a variable length auxiliary data field which will be passed on to the device whilst being treated as data by the channel. A transfer could involve a memory reference, in which case a memory address is provided. Furthermore, if the transfer is to a memory reference device, then there is an option for data chaining. Input transfers are similar to output transfers, however, dual transfers may be specified, so that input signals

can simultaneously be routed to two different memories. The channel must in this case have at least two ports. Transfer lengths of 1 to 32,768 characters can be set up. In addition, a free-running transfer, with no specified length can be established. Each direction of the transfer is established by a separate command.

The I/O command structure is shown in Figure IV-6.

The skeleton of the command is as usual; consisting of the DID, SID, DESCRIPTOR, the I/O command proper, and the termination command, if any. Additional fields due to person, chaining, and indirect operations are not shown. The I/O command itself consists of a control character followed by a number of additional, optional fields. The control character has five subfields, interpreted as follows:

I - defines the direction as IN or OUT.

M - specifies if the data will involve memory reference. Note that this does not necessarily imply the use of a memory but any unit for which an additional 16 bits is meaningful. If memory reference has been specified, the channel will issue X2XN or X2RN commands for each data group transferred. If memory reference has not been specified, the micro-command will be of the form XOXN or XORN, as appropriate to input or output. If memory reference is employed, there will be an additional two address characters in the optional fields.

A - this field specifies if auxiliary data is to be transferred to the device. The auxiliary data in question could be, for example, the sector and track address for a disc unit.

LENT - this specifies the number of characters in the auxiliary data. Up to 16 auxiliary characters can be sent, with 000 interpreted as 16, as usual.

C - specifies data chaining to be used.

The control character is followed by transfer length characters. The first bit of these characters indicates if the transfer is fixed length or of indefinite length. If the transfer is fixed, then the remaining 15 bits specify the length in characters, with the usual convention of 0 as 32768.

Following the length field indicator, are the auxiliary characters which are transferred to the device. This is followed by two address characters, if memory reference has been

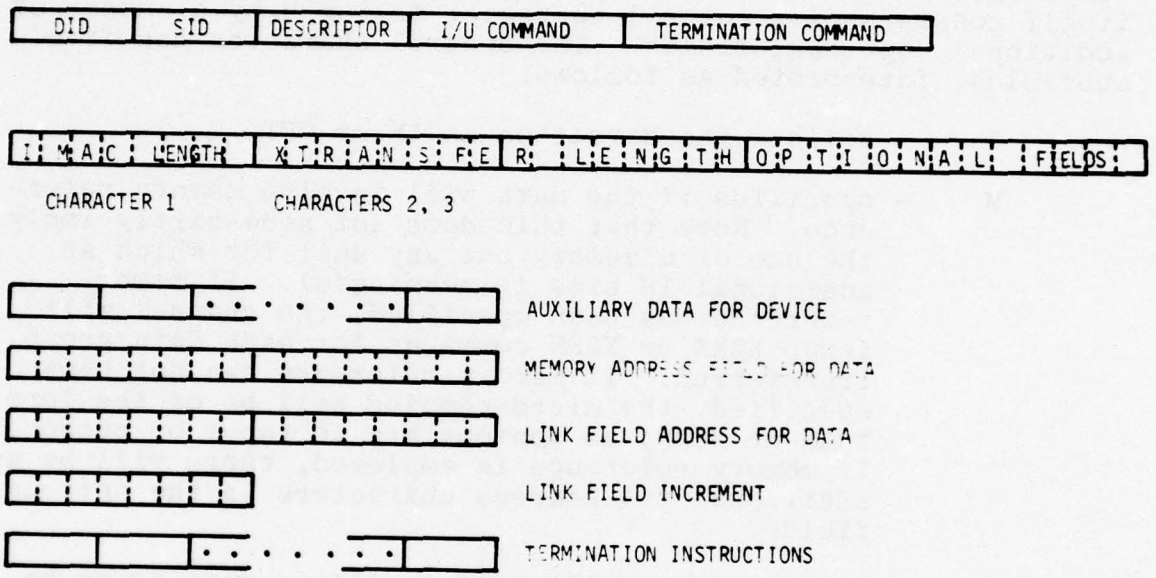


FIGURE IV-6 TRANSFER COMMAND STRUCTURE

specified. These characters are the address of the first character in to which data will be stored or from which data will be fetched, as appropriate. If data chaining has been specified, additional characters follow before the normal command. These characters specify the termination sequence for each component of the transfer. These are not to be confused with the termination action of the I/O command, which is independent of the data block termination action.

3.4.2 Data Chaining

Data chaining operates under command chaining. That is, a chain of I/O commands can be set up, each of which can specify a chain of data transfers. Data chaining can only be used with memory reference modes. At the conclusion of the transfer, the channel will read two additional characters from the memory at a link address location in a manner identical to what was specified for condition chaining. If the operation had been an input operation, then it would read the two characters from the link field address and the link field increment. These characters will be interpreted as a link to the next block of data - i.e., the next block to be written to, or to be read from. Clearly, data chaining must be contained within the same memory module. The new address is used to replace the old address characters and the command is essentially re-issued. A chain address of 177777_8 is interpreted as the end of the chain, and therefore, the conclusion of the command.

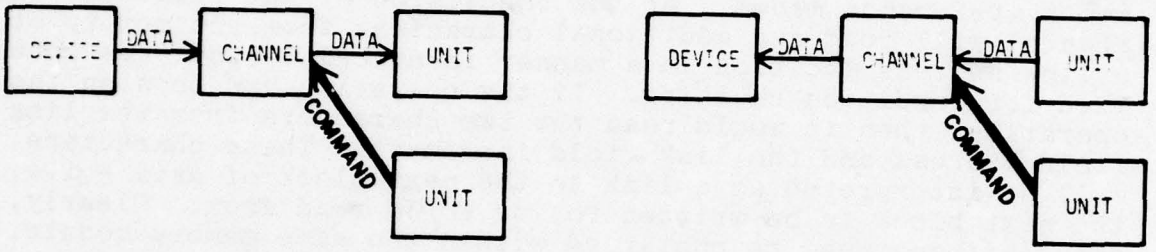
The termination character specifies one of three termination modes: do nothing, interrupt, linked. The do-nothing mode needs no elaboration; the data transfer continues until the chain is depleted. If the interrupt mode has been selected an interrupt containing the usual SID, DID information as well as an I/O transfer code and the final address to which or from which data was transferred. If a linked mode has been specified, the termination data will be added to the condition chain.

3.5 Bilateral and Dual Transfers

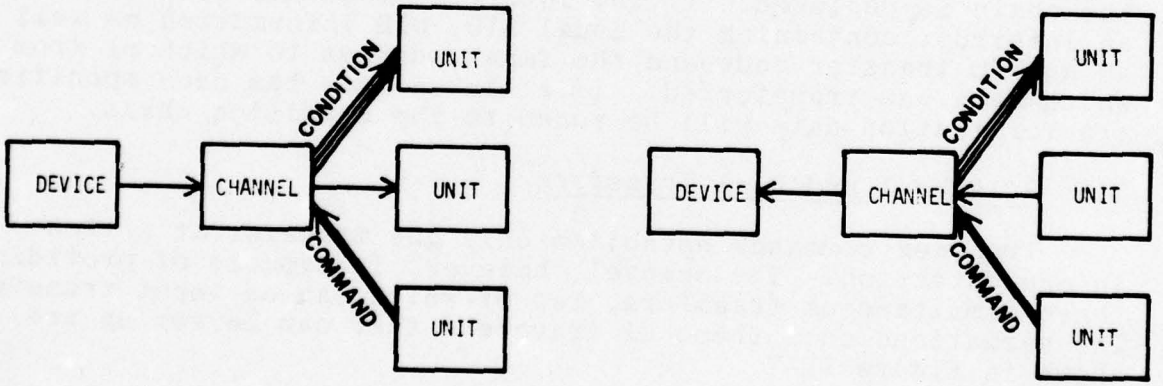
Transfer commands establish only one transfer at a time - in one direction. The channel, however, is capable of providing three simultaneous transfers, two of which can be input transfers. The variations on a theme of transfers that can be set up are shown in Figure IV-7.

In Figure IV-7a, a simple transfer has been set up, say, with data chaining, separately in one or the other direction. The channel stores the address of the link fields for data chaining and keeps track where each successive character is to go. In Figure IV-7b, a chaining has been added, but a unidirectional transfer mode has been maintained. Now, in addition

A - SIMPLE TRANSFERS

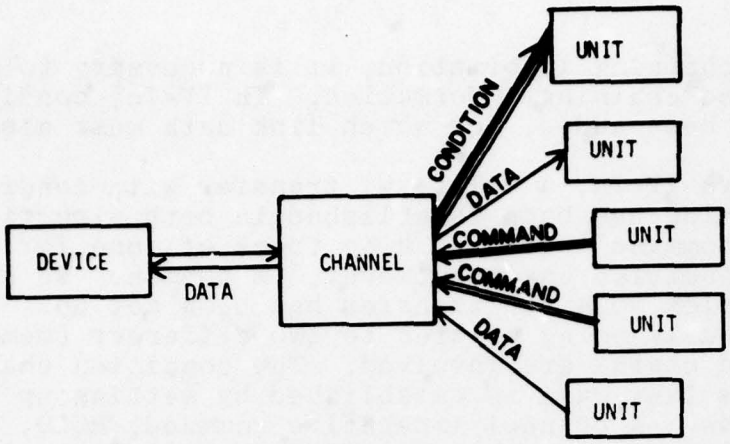


B - SIMPLE TRANSFER WITH COMMAND CHAINING

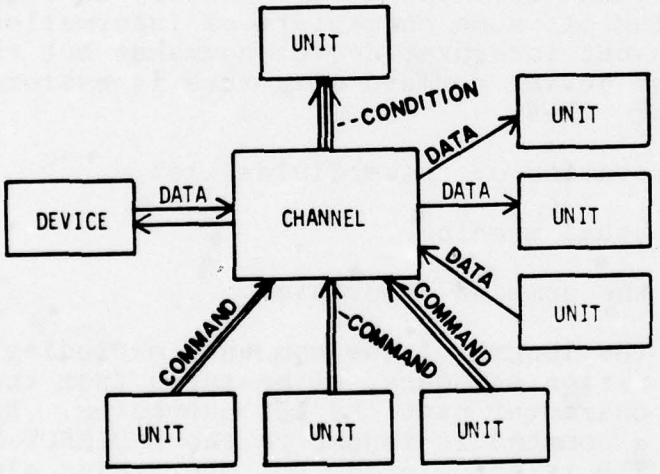


C - SIMPLE TRANSFER WITH COMMAND AND CONDITION CHAINING

FIGURE IV-7 TRANSFER COMBINATION SET-UP



D - BILATERAL TRANSFER WITH COMMAND AND CONDITION CHAINING



E - DUAL WRITE, BILATERAL TRANSFER WITH COMMAND AND CONDITION CHAINING

FIGURE IV-7A TRANSFER COMBINATION SET-UP CONTINUED

to the data chaining information, it is necessary to keep track of the command chaining information. In IV-7c, condition chaining has been added, for which link data must also be kept.

In Figure IV-7d, a bilateral transfer with condition and command chaining has been established in both directions. There are now two command chains to keep track of, one for each direction. The condition chain, however, is common. In Figure IV-7e, the most complex possible transfer has been set up. The input is simultaneously being written to two different (memory) units. Three command chains are involved. The condition chain is again common. This last case is established by setting up three separate commands. A channel imperative command, HOLD, is used to delay the initiation of the transfers until another imperative command, GO, is given. This can be used to enhance synchrony between input and output, or between two elements of a dual write operation. Note that synchrony is not guaranteed. The dual write operation cannot be performed unless the channel has at least two ports. Furthermore, the dual write operation cannot be performed to the same unit.

3.6 Device Commands

A device command is not interpreted by the channel. It is passed from the channel to the device exactly as received, except for stripping off some characters of information. Since the channel does not interpret device commands but rather treats them as data, the device command structure is extremely simple as shown in Figure IV-8.

The interpretation of these fields are:

DID, SID - usual meaning.

DES - the command descriptor.

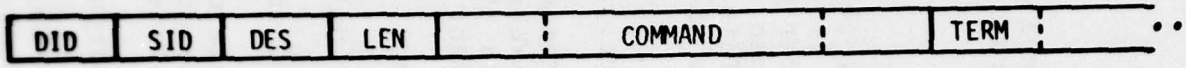
LEN - the length of the command, excluding the termination sequence, as measured from the first character past the LEN character. Note that if a command is issued in the INDIRECT MODE, the SID is not stored, but everything else must be. Lengths can be from 1 to 256 characters, with the value 00000000 interpreted as 256.

LID3 - the logical ID of the DID specified for the third person mode.

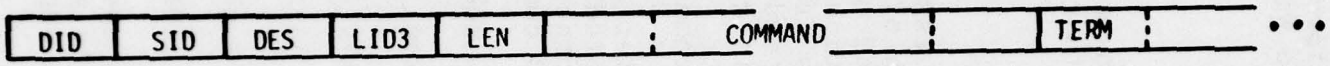
UID - the unit ID to which the command is chained.

ADDRESS - the address in the unit to which the command is chained.

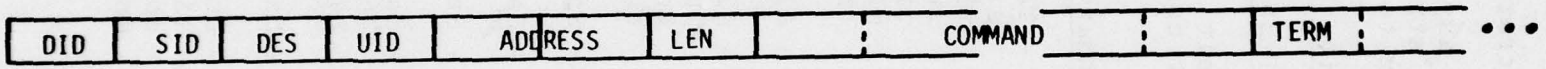
FIRST PERSON, UNCHAINED:



THIRD PERSON, UNCHAINED:



FIRST PERSON, CHAINED:



THIRD PERSON, CHAINED:

FIGURE IV-8 DEVICE COMMAND FORMAT

TERM - the first character of the termination command.

COMMAND - arbitrary characters.

The channel, or any other unit which can respond to a device format command, strips off all information and passes on the "COMMAND" portion to the unit to do with as it wishes.

3.7 Channel Imperative Commands

3.7.1 Generic Unit Imperatives

All generic unit imperatives can be issued with a descriptor, effectively converting them to channel imperative commands.

3.7.2 Additional Commands

HOLD	- This command is issued to delay the initiation of a chain of commands. It is used when an input and output transfer must be initiated simultaneously, or when dual write operations are set up. This command holds all pending commands until the GO command is given.
GO	- Allows pending commands to start.
HALT IN	- Allows the completion of the present input command, including all data chaining, but curtails further commands in the chain until a GO command is given.
HALT OUT	- Same as HALT IN except for output.
GO IN	- Same as GO but applies to input only.
GO OUT	- Same as GO but applies to output only.
HALT IN DATA	- Same as HALT IN but also curtails data chains in progress, allowing the present block in the data chain to complete.
HALT OUT DATA	- Same as HALT IN DATA, but for output.
NO-OP	- Does nothing, is used to change the condition chaining data when it is desired to do so without an explicit transfer command.

3.8 Immediate Mode Operation

The I/U repertoire in conjunction with the design details of the device can be used to implement an immediate mode I/O command operation. In this mode, true data is contained as part of the command. Similarly, input data is placed in a buffer area which is part of the command itself. This can be implemented by designing the device or device controller so that it interprets the device specific data which is part of the I/U command, as data rather than as control information. Condition chaining can be used to accomplish the same thing on output, where the address of the command would be used as the chaining data.

SECTION V

SPECIFICATION FOR THE MEMORY UNIT OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE MEMORY UNIT OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	V-1
2.0	Applicable Documents	V-1
2.1	General CP Specification	V-1
3.0	Memory Unit (MU) Specification	V-1
3.1	General	V-1
3.2	The Control Memory	V-1
3.3	Memory Cycle Operation	V-2
3.4	Protection Features	V-4
3.4.1	General	V-4
3.4.2	Command Protection	V-4
3.4.3	Data Transfer Protection	V-6
3.4.4	Control Memory Layout	V-7
3.5	Priority Control	V-7
3.6	Memory Command Stack	V-8

LIST OF FIGURES

FIGURE	TITLE	PAGE
V-1	Control Memory Entry Format	V-1
V-2	Memory Cycle Overview	V-3

1.0 SCOPE

This section is the specification for the Memory Unit (MU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Specification for the Central Processor

(Section I of the CPS Central Processor Specification)

3.0 MEMORY UNIT (MU) SPECIFICATION

3.1 General

The standard memory unit of the CP is a 65,536 character addressable module. Memory units are self-contained and have individual power supplies and power transient protection. Character parity is generated and/or checked on every memory operation. Memory units can be equipped with 1, 2, or 4 ports into the matrix. A full memory address is a 24 bit field, of which the most significant 8 bits is the LID of the memory unit.

3.2 The Control Memory

Every memory unit has a 256 or 512 character control memory. The control memory is used to store priority and protection information for all other units in the CP. The SID of the requesting unit is used as an index to the control memory, to fetch a pair of characters. The character pair has the following structure:

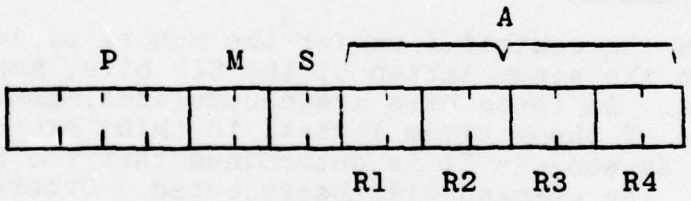


FIGURE V-1 - CONTROL MEMORY ENTRY FORMAT

P-field - A four bit field which designates the priority that the source unit will have relative to other source units that can use the memory. A priority of 0 is interpreted as not allowing that source unit access to the memory.

M-field - A two bit field that determines the interpretation of the R/A field. If it is zero, the R/A field is interpreted as a set of four read/write restrictions. If it is non-zero, the R/A field is interpreted as an address (in the control memory) where a set of restrictions for that source unit will be found.

S-field - A two bit field that specifies what kinds of non-transfer commands can be executed by that source unit in this memory unit.

R-field - Two bit fields. The first bit specifies if reading is allowed, the second bit specifies if writing is allowed.

A-field - An eight bit address field which points to one of 256 character pairs in the control memory.

The layout of the control memory is established by assembly language declarations. If a physical memory unit has been assigned to a new logical role (i.e., its LID has been changed), its control memory must be loaded to the values appropriate to that role. This is done through the use of the SET STATE-2 or SET STATE-3 commands (see Section I, paragraph 3.6.4.1). A 256 character control memory is loaded by using the SET STATE-2 command. A 512 character control memory (256 character pair entries) is loaded by using the SET STATE-2 followed by the SET STATE-3 command.

3.3 Memory Cycle Operation

Figure V-2 shows the control flow for the memory cycle. The operation begins with the accumulation of the SID bits, most significant bit first. As these bits are accumulated, they are compared to the value of the maximum logical ID (MID) stored in that memory module. As soon as it is determined that the SID is greater than the MID, the command will be rejected. Otherwise, the control logic waits until the entire SID has been received. This is shown as a parallel operation since the rejection could occur prior to the accumulation of the SID, but the SID comparison takes slightly longer (for a full 8 bits) than the accumulation of the SID bits.

Once the SID has been accumulated, it is used as an address to the control memory, to fetch the character pair corresponding to the SID. The priority is checked. A zero priority indicates that the source unit has no privileges in this memory unit, whereupon the command is rejected. If there is a non-zero priority, the priority logic is entered. This could cause some

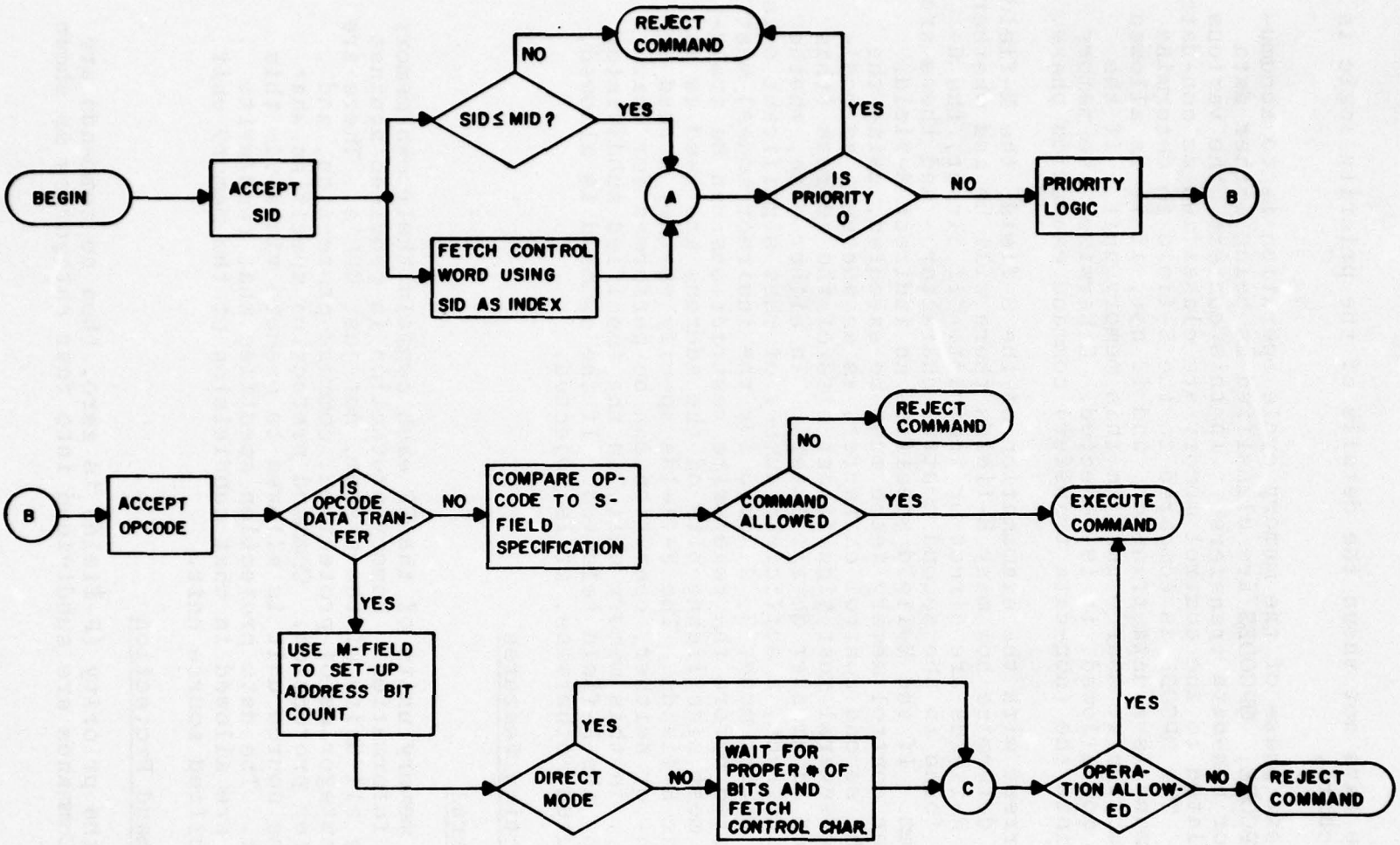


FIGURE V-2 MEMORY CYCLE OVERVIEW

delays. We have not shown the details of the priority logic in this flow chart.

The next phase of the memory cycle operation is to accumulate the OPCODE. OPCODES are classified as being either data transfers or non-data transfers. In this context, the various OPCODES related to the control memory are classified as non-data transfers. The OPCODE is compared to the S-field to determine if the command is a data transfer, and if not, if it is allowed for that particular source unit in this memory unit. If the command is not allowed, it is rejected. Otherwise, the memory unit goes into the (non-data transfer) command execution phase.

Concurrent with the examination of the S-field, the M-field is used to determine how many R-fields there will be and whether or not the R-fields are direct or indirect. If direct, the R-fields are found in the second control character - and there are four of them. If the M-field specifies an indirect R-field, then another control memory fetch must be executed, using the S-field (the second control character) as an address, as modified by the several most significant bits of the address (this is explained in greater detail later). In either case, whether direct or indirect, a sufficient number of most significant bits (2 for the direct mode, 3, 4, or 5 for the indirect modes) must be accumulated before the read/write restrictions can be evaluated. The most significant bits of the address are used as an index to the R-fields. The R-fields specify whether a read or write, both, or neither, operation can be performed for that source unit, in this memory unit, in the specified subdivision indicated by the R-field selected. If the command is allowed, it is executed, otherwise, it is rejected.

3.4 Protection Features

3.4.1 General

The memory units of the CP each contain their own memory protection information. Memory protection is provided against failures by all units of the system, not just CPU's. There are two major categories of protection: command protection, and data transfer protection. Command protection specifies what commands the source unit is allowed to execute vis-a-vis this memory unit. The data protection specifies what read/write operations are allowed in what subdivision of the memory unit by the specified source unit.

3.4.2 Command Protection

If the priority (P-field) is zero, then no commands are allowed. Commands are subdivided into four categories as shown below:

11	10
SET STATE	GO OFF-LINE
SET UNIT MAX	CLOSE
SET STATE 1, 2, 3, 4	OPEN
INXMT	POWER DOWN
SET LOGICAL ID	RESET
CONVERT NULL ID	
SET ICUID	
SET SECONDARY ICUID	

01	00
OPEN PORT	FETCH UNIT STATE
CLOSE PORT	QUERY PAD
FORCE INTERRUPT	RETURN STATUS
SET N PRIMARY	LOCK PATH
	UNLOCK PATH
	FETCH STATE 1, 2, 3, 4
	XORO
	LOCAL ABORT
	GLOBAL ABORT
	ALL TRANSFERS

These categories are called 3, 2, 1, and 0 commands respectively. Category 0 commands are those which can be used in the course of normal data transfers, or which cannot change the state of the memory unit. That is, they are "safe" commands - their execution are not likely to cause problems.

Category 1 commands change the routing of data through the matrix and can thereby affect matrix efficiency. Similarly, forcing an interrupt improperly will not generally cause system malfunction but only cause spurious processing to occur.

Category 2 commands are used to change the state of the unit, but not its logical identity or the contents of the control memory. Inappropriate execution of this command will definitely affect performance, in fact, can cause the system to fail, but will not cause the system's logical integrity to be corrupted.

Category 3 commands are used to change the state of the unit, its logical identity, and in the case of the INXMT command, can cause the unit to behave in a totally bizarre manner (deliberately for diagnostic purposes). Consequently, the execution of these commands must be well guarded. Typically, category 3 commands are executable only by CPU's, the ICU, the SMU, and the BU.

The selection of which categories of commands are allowed a particular unit is determined by the value of the S-field in the control word for that unit, in accordance to the following scheme:

S = 00 - category 0 commands only

S = 01 - categories 0 and 1

S = 10 - categories 0, 1, 2

S = 11 - all commands

3.4.3 Data Transfer Protection

Data transfer protection is provided by subdividing the entire unit into sequences which lie on binary boundaries. The smallest set of subdivisions is 4. That is, memory is divided into four regions corresponding to character addresses: 0-16,383; 16,384-32,767; 32,768-49,151; and 49,152-65,535. The M-field specifies how many partitions there will be in accordance to the following values:

M = 00 - The R/A field is interpreted as four protection fields. The memory is divided into four quadrants, each with their own 2 bit protection specification, which separately specifies read and write protection. This is called the "direct protection mode". All other values of M are indirect protection modes.

M = 01 - The R/A field is interpreted as a cache memory address of a pair of control characters which contain 8 R subfields. The memory is subdivided into 8 octants.

M = 10 - The R/A field is interpreted as an address of a set of 4 control characters. These specify 16 subdivisions for protection purposes.

M = 11 - The R/A field is interpreted as an address of a set of 8 control characters which provide 32 subdivisions of the memory.

From the above, it can be seen that read/write operations can be protected in segment sizes ranging from a minimum of 2048 characters to a maximum of the entire memory unit. These specifications are made by declarations in the source code.

3.4.4 Control Memory Layout

The assembler performs the assignment of the indirect addresses (where specified) in an optimum manner to achieve a maximum packing of the control memory. In most cases, the direct mode is sufficient. Usage of the indirect mode should be restricted since the control memory cannot be expanded beyond 512 characters.

Control memory space can be conserved through a judicious assignment of logical identities. All passive units (including memories) should be given high order LID's. Ordering of LID's should follow the scheme depicted below:

(1) Units which can have access to all memories.

(2) Units which can have access to some memories.

(3) Units which cannot have access to memories.

(4) Passive units other than memories.

(5) Memories.

With this assignment, the MID (maximum logical identity) is set not to the MID of the system, but to the highest value ID that can have access to that logical memory unit. Thus, if there were 175 units in the system, but units with LID's above 120 were not allowed memory access, then the MID would be set to 120 rather than to 175. This allows the use of 270 characters for indirect specifications as compared to 160 had 175 been specified as the maximum. If there were very few indirect specifications, setting the MID to 120 could allow the use of a 256 character cache memory rather than a 512 character cache memory to store the same specification.

3.5 Priority Control

Priority control rules are established as follows:

(1) If the SID's of the two commands are not equal, then priority is established by the priority in the P-field.

(2) If the P-field priorities are the same, the priority is established by the port order.

(3) If the SID's are the same, the priorities are necessarily the same since the same unit is involved. In this case, the following scheme holds:

a. Category 3 commands have priority over 2, 1, and 0.

b. Category 2 commands have priority over 1 and 0.

c. Category 1 commands have priority over 0.

d. If two or more commands for the same SID are in the same category, they are serviced in strict FIFO order.

3.6 Memory Command Stack

A stack entry consists of four characters. The number of stack entries provided is double the number of memory ports. Stack entries can be used for any commands from any port - that is, a particular stack entry does not have a built-in association with a particular port. A stack entry consists of the following:

(PORT), SID, OPCODE

(PORT), SID, OPCODE, OPCODE CONTINUATION

(PORT), SID, OPCODE, ADDRESS

The OPCODE could take one to four character spaces. If the total command requires two characters, then two commands can be stacked in one entry.

Commands are stacked only when they are received on a port which is in the locked state. Control line signaling by the source unit is used to specify that a new command is on the way. Commands are stacked in accordance to the priority scheme discussed in paragraph 3.5 above. The new SID is compared to the previous SID for that port - if they do not match, all stacked commands for that port are aborted. A port could be in the locked state either as a result of an explicit lock command or because it is waiting for the completion of an ongoing memory operation. The port remains locked as long as there is a command stacked for that port. Instruction stacking operations are transparent.

SECTION VI

SPECIFICATION FOR THE MEMORY-TO-MEMORY TRANSFER UNIT (MMTU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE MEMORY-TO-MEMORY TRANSFER UNIT (MMTU) OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	VI-1
2.0	Applicable Documents	VI-1
2.1	General CP Specification	VI-1
3.0	Memory-to-Memory Transfer Unit (MMTU) Specification	VI-1
3.1	General	VI-1
3.2	Differences Between MMTU and Channel	VI-1
3.3	Method of Use	VI-1

1.0 SCOPE

This section is the specification for the Memory-to-Memory Transfer Unit (MMTU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General CP Specification

(Section I of the CPS Central Processor Specification)

3.0 MEMORY-TO-MEMORY TRANSFER UNIT (MMTU) SPECIFICATION

3.1 General

The MMTU is named such because this is expected to be its primary use. In fact, its functions are somewhat broader. It is used for all passive unit to passive unit transfers. Functionally, it can be viewed as two interconnected channel units, with a more limited set of capabilities.

3.2 Difference Between MMTU and Channel

(1) Fixed speed transfers - always at maximum matrix speed.

(2) Master Master mode only.

(3) No HOLD, GO, HALT IN, HALT OUT, GO IN, GO OUT, HALT IN DATA, HALT OUT DATA, commands.

(4) No buffer control; four character mode is always used.

(5) No dual input transfers.

(6) Functionally a half-duplex rather than a full-duplex unit.

In all other respects, it corresponds to a pair of interconnected channel units.

3.3 Method of Use

The MMTU is used by issuing a third person command to the MMTU to establish the transfer from the passive unit which is to be the source unit. A second third person command establishes the connection to the destination unit. The transfer is not initiated until the second command of the pair has been received.

SECTION VII

SPECIFICATION FOR THE SYSTEM MONITOR UNIT (SMU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE SYSTEM MONITOR UNIT (SMU) OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	VII-1
2.0	Applicable Documents	VII-1
2.1	General CP Specification	VII-1
3.0	System Monitor Unit (SMU) Specification	VII-1
3.1	General	VII-1
3.2	PROM Layout	VII-1
3.3	SMU Input Operation	VII-3
3.4	SMU Self-Test Operation	VII-3
3.5	The SMU as a Unit	VII-3
3.6	Control Memory	VII-4
3.7	Application Notes	VII-4

LIST OF FIGURES

FIGURE	TITLE	PAGE
VII-1	SMU PROM Layout	VII-2

1.0 SCOPE

This section is the specification for the System Monitor Unit (SMU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Specification for the Central Processor

(Section I of the CPS Specification)

3.0 SYSTEM MONITOR UNIT (SMU) SPECIFICATION

3.1 General

The SMU is the watchdog unit over the CP. The SMU contains a PROM (Programmable Read Only Memory) whose contents specify the response of the SMU to various conditions. Basically, the SMU expects various units to transmit a test signal periodically. Should the unit in question fail to transmit the proper signal, or transmit it in the work order, the SMU will react by alerting a unit which is specified in the PROM.

3.2 PROM Layout

Figure VII-1 shows the layout of the control PROM of the SMU. The contents of this memory are specific to the system. The PROM may have anything from 16 to 512 characters. The characters are interpreted as follows:

"MIN UID" - the identity of the minimum valued LID to which the SMU will respond.

"MAX UID" - the identity of the maximum valued LID to which the SMU will respond.

"SMUID" - the logical identity of the SMU itself.

"UID X, Y, Z" - various LID to which the SMU will report should a malfunction be detected.

"T1, T2" - time period indicators, as taken from a clock signal. These are specific to the unit in the table.

"UID" - the logical ID of the unit to which the SMU will respond should a particular unit fail to report on time.

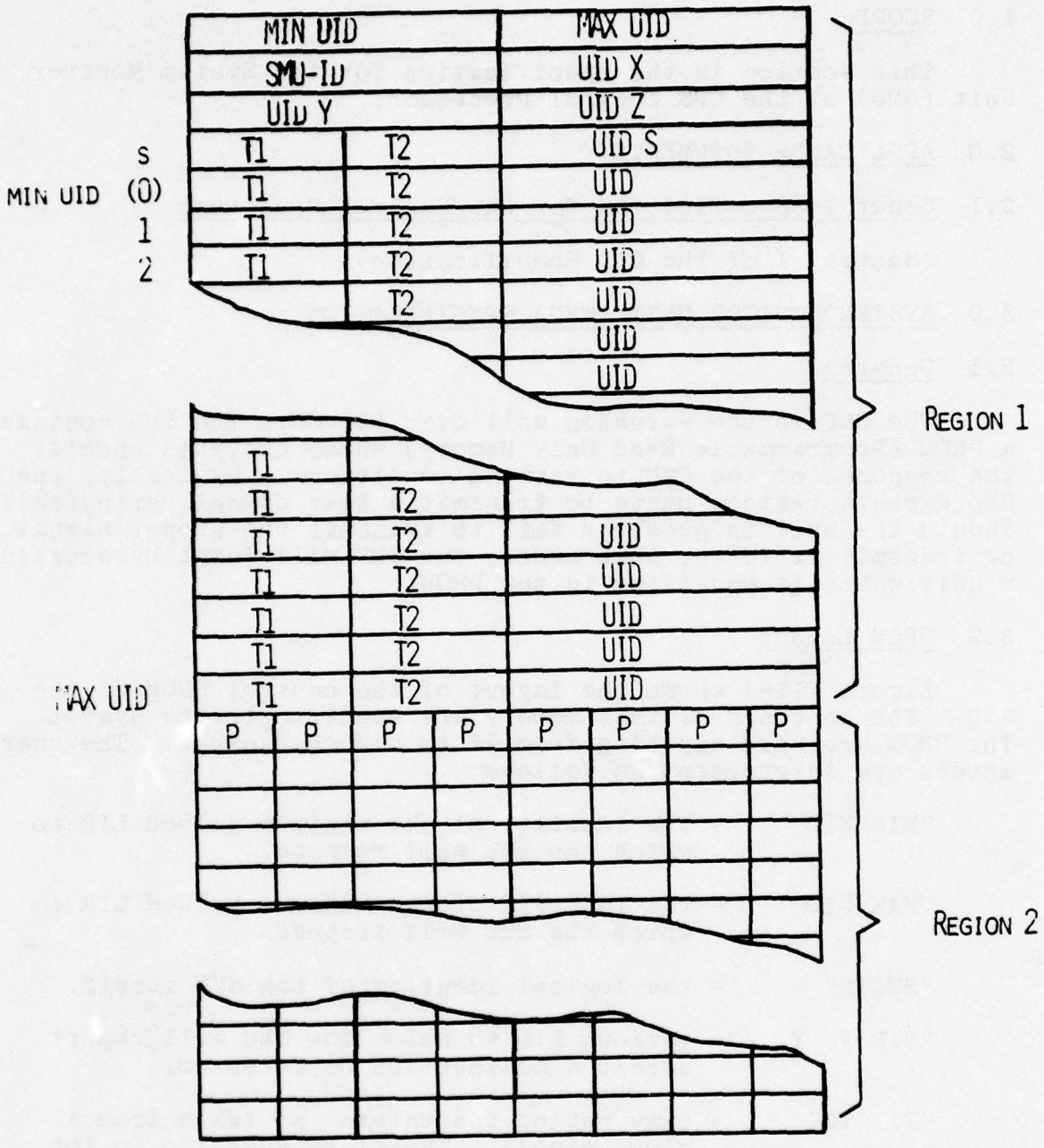


FIGURE VII-1 SMU PROM LAYOUT

P - the P-fields following the MAX UID entry are two bit port priority fields, with 00 interpreted as not allowing the specified unit to access the SMU.

3.3 SMU Input Operation

The SMU is tied directly (not via the matrix) to a clock source. Typically, a 100 millisecond period, although this is arbitrary. A period of not less than 5 milliseconds and not greater than 1 second is recommended. The same clock source is used for all monitor functions. The T1, T2 entries specify the number of clock intervals that should take place between each signal transmitted by the unit to which that set of T1, T2 entries correspond. The LID of the unit is established relative to the MIN UID stored. For example, if the MIN UID were 44, then the entry labeled MIN UID in Figure VII-1 would be for unit 44, the next for unit 45, etc. Assume that the second entry (labeled 2) in the PROM corresponded to unit 46. Furthermore, that T1 = 7 and T2 = 13. The SMU would then expect a signal from unit 46 every 7 units of time, and every 13 units of time (as established by the clock). If either or both of these signals should fail to arrive, within the specified time period, or if signals should arrive at a higher rate than specified by T1 and T2, the SMU will respond by transmitting an alarm to the unit whose LID is stored in the second character of the pair for unit 46. A value of T1 or T2 of 0 means that that period is ignored. If both T1 and T2 are zero, and a signal from that unit does arrive, then the alarm will be sent to the specified UID.

If the SMU should not be able to reach the specified UID for alarming, then it will attempt to alarm the unit specified as UIDX. If the clock signal should fail to arrive within the specified time boundaries for the clock, the UNIT identified as UIDY will be alerted. In addition to transmitting the alarm to the specified unit, upon repetition of the alarm condition, the SMU will transmit an alarm message to the unit specified as UIDZ. The identities of UIDX, UIDY, UIDZ, or for that matter any of the UID's is arbitrary and could be one and the same.

3.4 SMU Self-Test Operation

The location marked "S" in Figure VII-1 establishes two time intervals and a UID for the SMU's self-test. At periods T1 and T2, the SMU will send itself a test message. Should either or both of these test messages fail to get through, the SMU will alarm UID's.

3.5 The SMU as a Unit

The SMU is a two port unit. Both ports must be operational at all times. The SMU responds to the following commands:

GOOF - it goes off-line.

CLOS -

OPEN -

XORO - the normal test signal transmitted and received by the SMU for monitoring purposes.

FINT -

INXT -

In addition, it has the following unit specific commands:

RESET T1 NNNN - causes the T1 counter for unit NNNN to be reset. Does not affect the PROM value T1.

RESET T2 NNNN - causes the T2 counter for unit NNNN to be reset. Does not affect the PROM value T2.

RESET ALL - resets all counter values for all units.

3.6 Control Memory

The SMU contains a control memory of one character per UNIT which is represented in the PROM. This memory contains a counter for the accumulated T1 and T2 counts.

3.7 Application Notes

The size of the UID PROM and the extent of the checks to be employed depends upon the details of the system design and the extent to which on-line tests are to be implemented. Typically, only a small fraction of the units in the system would be tied to the SMU. As a minimum, we would have the following: clock units, SMU, ICU, alternate ICU, system executive CPU, all other CPU's. One might wish to also tie the various MU's and XU's to the SMU. However, since these units cannot initiate I/U commands in the normal mode, one would have to use the third person mode to get them to contact the SMU. T1 and T2 would be chosen as two relatively prime numbers - e.g., (2,3), (2,5), (2,7), (2,11), (2,13), (2,15), (3,4), (3,5), (3,7), etc. This minimizes the possibility of getting into a looping situation which could cause the SMU signal to be transmitted in the right period. Two relatively prime signals are used since a single signal could be properly generated in a loop. The UID to which the SMU responds is typically the standby CPU for the particular function. For example, the UID used for the ICUX would be the alternate ICU. The UID used for the alternate ICU would be the primary ICU.

UIDX could be chosen as the standby CPU, while UIDY might be the system executive processor. UIDZ should be a channel connected to the systems operations console, designed to give a visible and audible alarm. The UID to which the SMU responds should it fail to react to itself should be the system executive processor or a manual control position.

SECTION VIII

SPECIFICATION FOR THE SYSTEM TIMING AND CLOCK UNIT (SCU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE SYSTEM TIMING AND CLOCK UNIT (SCU) OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	VIII-1
2.0	Applicable Documents	VIII-1
2.1	General CP Specification	VIII-1
3.0	System Timing and Clock Unit (SCU) Specification	VIII-1
3.1	General	VIII-1
3.2	Control Memory Layout	VIII-1
3.3	Wired Timings	VIII-3
3.4	Programmed Timings	VIII-3
3.5	Time of Day	VIII-4
3.6	Synchronization	VIII-4
3.7	The Clock as a Unit	VIII-5
3.8	Alarm Detection	VIII-5
3.9	Clock Display	VIII-5

LIST OF FIGURES

FIGURE	TITLE	PAGE
VIII-1	Clock Control PROM Layout	VIII-2

1.0 SCOPE

This section is the specification for the System Timing and Clock Unit (SCU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General CP Specification

(Section I of the CPS Central Processor Specification)

3.0 SYSTEM TIMING AND CLOCK UNIT (SCU) SPECIFICATION

3.1 General

The system timing and clock unit provides precise time intervals for use by the rest of the complex. It maintains a time-of-day clock which is accessible to all units in the complex. Time of day information is available by the normal inter-unit communication method. The time intervals are wired to all units that require them. The system generally will have two clock units which are logically locked to one another. Timings are distributed from both clock units to all units that require them. In addition, the clock unit can be programmed to provide the distribution of "clock interrupts" to such units as might require them.

3.2 Control Memory Layout

The control memory of the clock unit is contained in a PROM. The fields of this PROM are interpreted as follows: (see Figure VIII-1)

MAX UID - maximum unit LID for units which are allowed to access the clock via the matrix.

MIN UID - minimum unit LID for units which are allowed to access the clock via the matrix.

SELF UID - the logical ID of the clock unit.

ICU-ID - the logical ID of the primary ICU.

ICU-ID(s) - the logical ID of the back-up ICU.

ALARM ID - the logical ID of the unit which is to be notified for self-detected alarm conditions.

P-field - two bit port priority/validation field.

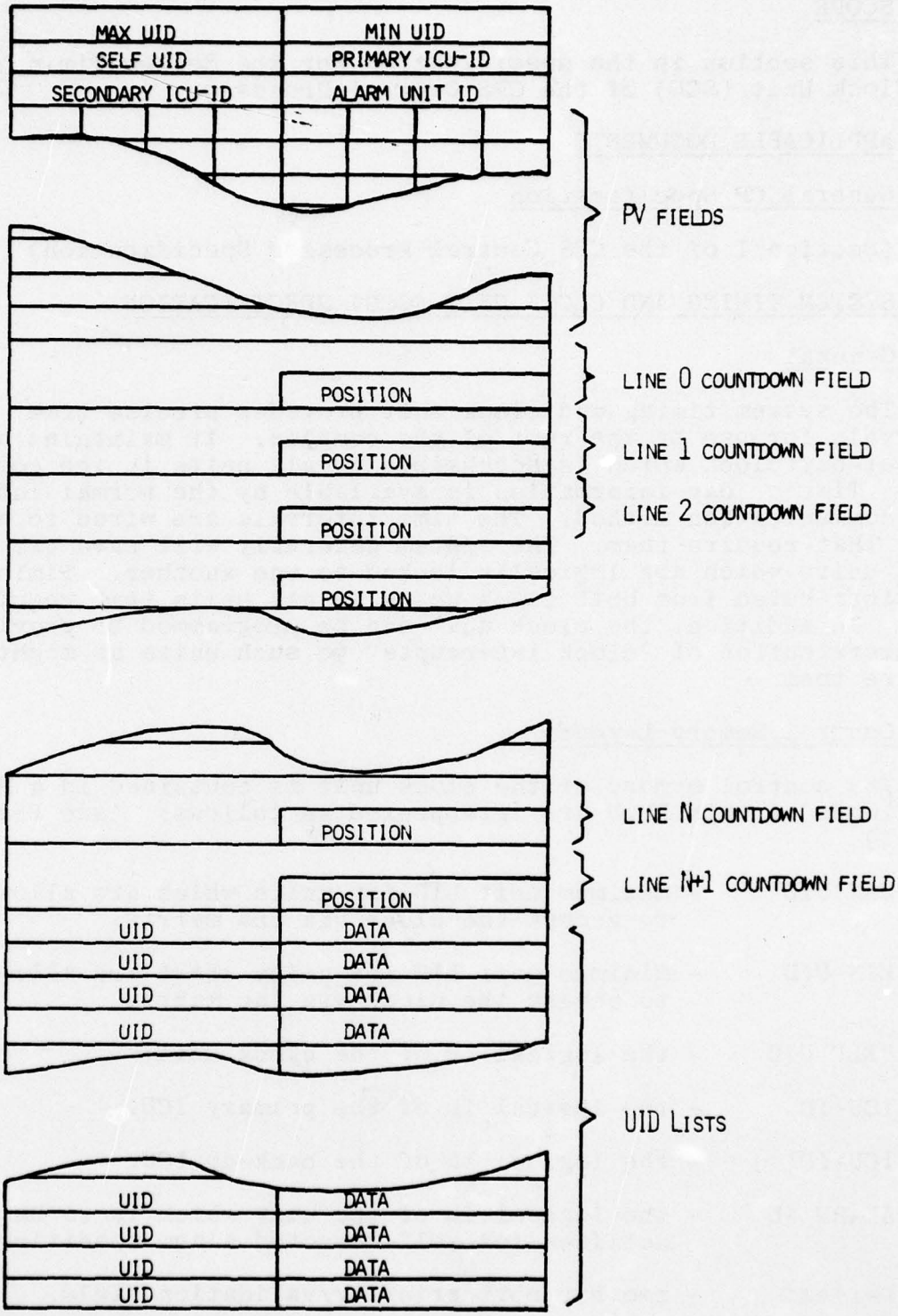


FIGURE VIII-1 CLOCK CONTROL PROM LAYOUT

COUNTDOWN - up to 254 fields of 24 bits each containing countdown values to be used to develop the "wired" timings.

POSITION - a character address in the PROM which specifies the first UID to be used in a list of "Programmed Timings".

UID LISTS - lists of UID's to be used in programmed timings. The total size of the PROM may not exceed 256 characters.

DATA - specifies three bits to be issued in the C-field and 5 bits to be inserted in the interrupt character of the generated interrupt message.

3.3 Wired Timings

A clock can be equipped with a number of lines which provide "wired-in" timings. The number of such lines is limited by the total size of the PROM. If the P fields consumed 10 characters (for a total of 40 units which are allowed to query the clock), and there were no UID lists, then up to 60 "wired timings" could be provided. A "wired timing" entry does not necessarily correspond to a physical clock line, but may be used only in conjunction with the programmed timings discussed below.

A wired timing entry in the PROM is a specification for a countdown of the basic 1 megacycle clock. The 24 bit entry in the PROM is counted down in a countdown register and a timing pulse is produced on the specified line each time the countdown is completed. At that point, the countdown register is restored to the value stored in the PROM and a new cycle is started.

At least one "wired timing" for clock synchronization is required. This is typically set up to a one second increment. The twenty-four bit countdown field allows this timing to be set to as high as 16.67 seconds.

3.4 Programmed Timings

Every wired timing entry in the PROM has an 8 bit field which is a pointer to the head of a list of UID's. If the pointer field has a value of 0, it is interpreted as not pointing to any UID's. Whenever a wired timing counter runout occurs, the clock unit uses the pointer field to point to the UID list, and then generates clock interrupts for the units specified on the list. A unit number of 0 is interpreted as the end of the list.

The pointer can point to any position on a list. For example, a list can be set up specifying units A, B, C, D, E, F, G. Timer 1 can point to A, thereby producing interrupts for A through G. A different timer, say 3, can point to F, thereby producing interrupts for units F and G. Unit names can appear on more than one list, or more than once on a given list. Interrupt messages will be transmitted in the order they appear on the list. Failure to communicate with the interrupted device will result in an interrupt to the ICU or alternate ICU but will not suppress the transmission of the other interrupts on the list.

3.5 Time of Day

The clock unit can be queried to provide the time of day in a variety of formats:

(1) 32 bit binary time, to the microsecond, with a maximum range of 1 hour. This is reset every hour.

(2) 32 bit binary time to the millisecond, with a maximum range of 28 to 31 days, reset at the end of every month (with proper corrections for leap years including the four year, 100 year, and corrections for the year 2000).

(3) 32 bit binary time to the second, not reset, providing turnover every 136 years.

(4) 32 bit time in four characters; providing time of day in 100 microsecond increments, 60 seconds binary, 60 minutes binary, and 24 hours binary.

(5) 32 bit BCD time, providing tenths of hours, hours, year day, and year from 0 to 99.

(6) 32 bit BCD time providing month day, month and absolute year.

(7) Various standard IRIG formats.

The absolute year is used as a base for all leap year corrections.

3.6 Synchronization

Synchronization is achieved by connecting the clock to a timing source, typically a one second increment, provided either by another clock (a timing clock) unit, or from a WWV receiver. The clock is loaded with the proper time data using the SET STATE IU micro-command. This is followed by a SYNC command.

The clock oscillator is suppressed until the synchronization pulse is received. Thereafter the clock continues in the normal fashion. Synchronization is to the nearest microsecond following the receipt of the sync pulse.

3.7 The Clock as a Unit

The clock responds to normal unit commands. It does not accept XNXN or XNRN commands. Single character non-transfer IU micro-commands have their normal interpretation. The two character non-transfer commands have thier normal interpretation. In addition, the clock interprets the following unit specific commands:

HOLD	- stops the clock oscillator at the present value.
START	- starts the clock upon receipt of the command.
SYNC	- starts the clock at the next sync pulse.
TIME1 - TIME7	- fetches the time of day in one of the specified 7 formats.
RESET NNNN	- resets the specified countdown timer to the 24 bit value given as part of the command.
RTIM1 - RTIM7	- resets one of the seven formats to the specified time.
RESET AND SYNC NNNN - RTIMX AND SYNC	- these commands do the same as the RESET NNNN and the RTIMX commands, but resetting is held up until the next sync pulse.

3.8 Alarm Detection

The clock expects a sync pulse every second, either from the other clock, or from a synchronization source. Should this signal fail to occur, the clock will generate an interrupt to the unit whose logical identity is stored in the PROM. In addition, a "sync failure" light will blink on the clock display.

3.9 Clock Display

The clock is fitted with a visible display unit which can be mounted on the system control console. The display provides the following information: Year, Month, Day, Hour, Minute, and Second.

SECTION IX

SPECIFICATION FOR THE BOOTSTRAP UNIT (BU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE BOOTSTRAP UNIT (BU) OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	IX-1
2.0	Applicable Documents	IX-1
2.1	General CP Specification	IX-1
3.0	Bootstrap Unit (BU) Specification	IX-1
3.1	General	IX-1
3.2	Bootstrap Sequences	IX-1
3.3	Bootstrap PROM	IX-2
3.4	The Bootstrap Unit as a Unit	IX-2
3.5	Manual Initiation of Bootstrap	IX-2
3.6	Special Bootstrap Commands	IX-2

1.0 SCOPE

This section is the specification for the Bootstrap Unit (BU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General CP Specification

(Section I of the CPS Central Processor Specification).

3.0 BOOTSTRAP UNIT (BU) SPECIFICATION

3.1 General

The Bootstrap Unit provides automatic bootstrapping of selected unit types. Bootstrapping can be initiated either through the matrix by another unit, or directly and external to the matrix via operator commands which are keyed into the unit. Bootstrapping consists of a set sequence of operations followed by a program stored in the Bootstrap Unit PROM.

3.2 Bootstrap Sequence

Bootstrap commands consist of: the physical identity of the unit which is to be bootstrapped, the logical identity that it is to assume, and the specification of a bootstrap program/ data set which is to be used. The unit is assumed reset to the 0000 LID. The bootstrap sequence consists of the following commands:

CNID - converts the logical ID of the unit to the new logical ID.

REST - safety reset of the unit.

GART - global abort of all pending commands as a safety measure.

QUEP - safety query of unit's physical ID.

BOOTSTRAP SEQUENCE COMMANDS.

The bootstrap sequence is a set of stored unit micro-commands and data which are issued to the unit to be bootstrapped. Should any part of the bootstrap sequence fail, the sequence is aborted. If the sequence was initiated internally, an interrupt will be transmitted to the ICU. If initiated externally, an alarm will be triggered. The conclusion of a bootstrap sequence is signalled by an interrupt to the ICU and/or a visual signal on the BU indicator panel.

3.3 Bootstrap PROM Memory

The bootstrap PROM is up to 65,536 characters in length. The first 512 characters contain up to 256 entry points for bootstrap sequences. A bootstrap sequence is a set of unit micro-commands and data in the proper order. The bootstrap unit interprets these commands and sequences properly (character by character) to the next command which is then issued to the unit being bootstrapped.

3.4 The Bootstrap Unit as a Unit

The Bootstrap Unit contains a single bit (per unit) cache memory which identifies the units which are allowed to issue bootstrap commands. Once a bootstrap command is issued, ports are closed and no other commands except GART (global abort) can be accepted. The logical ID of the Bootstrap Unit is fixed. It responds to the normal unit micro-commands except for XNRN, SST2, SST3, SST4, FST2, FST3, FST4, and the three and four character versions of these commands. SST1 is used to set the control memory. FST1 is used to dump the content of the control memory. In addition, it responds to the following unit specific commands:

BOOT PHYSICAL UNIT ID AS LOGICAL UNIT ID WITH PROGRAM XXXX.

3.5 Manual Initiation of Bootstrap

Manual initiation of bootstrapping is done by keying in the following sequence from the Bootstrap Unit panel:

"PID.LID.PRG."

Where PID - three digit physical ID of the unit to be bootstrapped.

LID - three digit logical ID to be taken by unit.

PRG - three digit program ID to be used for bootstrapping.

3.6 Special Bootstrap Commands

The Bootstrap Unit can execute unconditional and conditional branches within the bootstrap program. These are not issued as IU micro-commands. The commands are:

SET FLAG 1	RESET FLAG 1	JUMP IF FLAG 1
SET FLAG 2	RESET FLAG 2	JUMP IF FLAG 2
SET FLAG 3	RESET FLAG 3	JUMP IF FLAG 3

SET FLAG 4	RESET FLAG 4	JUMP IF FLAG 4
HALT	PAUSE	

All flags are reset upon the initiation of a bootstrap sequence. A bootstrap sequence is halted by the HALT command. A PAUSE command halts the program at that step in the sequence and waits until another bootstrap command is issued (either internally or externally). The pause light will be lit on the console when this occurs and an interrupt is generated to the ICU. These commands can be used to advantage to share bootstrap sequences. The LID (DID for issued I/U commands) is always supplied as the specified LID of the unit being bootstrapped. Each PAUSE provides an opportunity to change the LID of the unit being bootstrapped.

SECTION X

SPECIFICATION FOR THE PERFORMANCE MONITOR UNIT (PMU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE PERFORMANCE MONITOR UNIT (PMU) OF THE CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	X-1
2.0	Applicable Documents	X-1
2.1	General CP Specification	X-1
3.0	Performance Monitor Unit (PMU) Specification	X-1
3.1	General	X-1
3.2	The PMU Hardware	X-1
3.2.1	General	X-1
3.2.2	Control Memory Layout	X-3
3.2.3	Port Mode	X-4
3.2.4	IU Mode	X-5
3.2.5	Combined Modes	X-6
3.2.6	Other Messages	X-6
3.2.7	Priority and Stacking	X-6
3.3	The PMU as a Unit	X-7
3.4	PMU Software	X-7
3.4.1	General	X-7
3.4.2	Port Mode Probe Message Processing	X-7
3.4.3	IU Mode Software	X-8
3.4.4	CPU Monitoring	X-8
3.4.5	Use of Flag Instructions	X-9

LIST OF FIGURES

FIGURE	TITLE	PAGE
X-1	PMU Probe Installation	X-2

AD-A036 874 NORTH ELECTRIC CO GALION OHIO F/G 17/2
COMMUNICATIONS PROCESSOR SYSTEM.(U)
JAN 77 K HAGSTROM, B BEIZER F30602-73-C-0314
UNCLASSIFIED RADC-TR-76-394-VOL-8 NL

3 OF 3
AD
A036874

END

DATE
FILMED
4-77

1.0 SCOPE

This section is the specification for the Performance Monitor Unit (PMU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Central Processor Specification

(Section I of the CPS Central Processor Specification)

3.0 PERFORMANCE MONITOR UNIT (PMU) SPECIFICATION

3.1 General

The CP can be equipped with one or more performance monitor units. These are used to gather and process performance data with a minimum of perturbation to the on-line programs. The PMU itself is only a part of a broader performance monitor and analysis subsystem which includes the PMU, CPU's and memories as well. This section deals not only with the specialized PMU hardware, but also with the ways in which CPU's and other units are used to accomplish the performance monitoring task.

The hardware that comprises the PMU is basically a set of probes which can be interjected on various ports of the system, and which can be programmed to trap inter-unit messages. This aspect of PMU activity does not perturb the on-line configuration at all, except for the increased matrix load. The rest of the PMU function is essentially software - both in the programs being monitored and in specialized performance monitor and analysis packages.

3.2 The PMU Hardware

3.2.1 General

The specialized PMU hardware, as embodied in the performance monitor unit, is a set of high impedance probes that can be placed on any port of the CP. A probe connector is introduced in the backplane of the port control logic (see Figure X-1).

The PMU probe connector can be inserted into any port of the CP. It can sample all signals at the port interface without perturbing what is happening at the port or in the unit to which the port is attached. It is entirely passive and will not, except possibly under a failure, inject any signals into a port that is being monitored. The number of ports which a PMU can monitor cannot exceed 254 simultaneously. However, the ability to logically terminate that number of ports does not guarantee

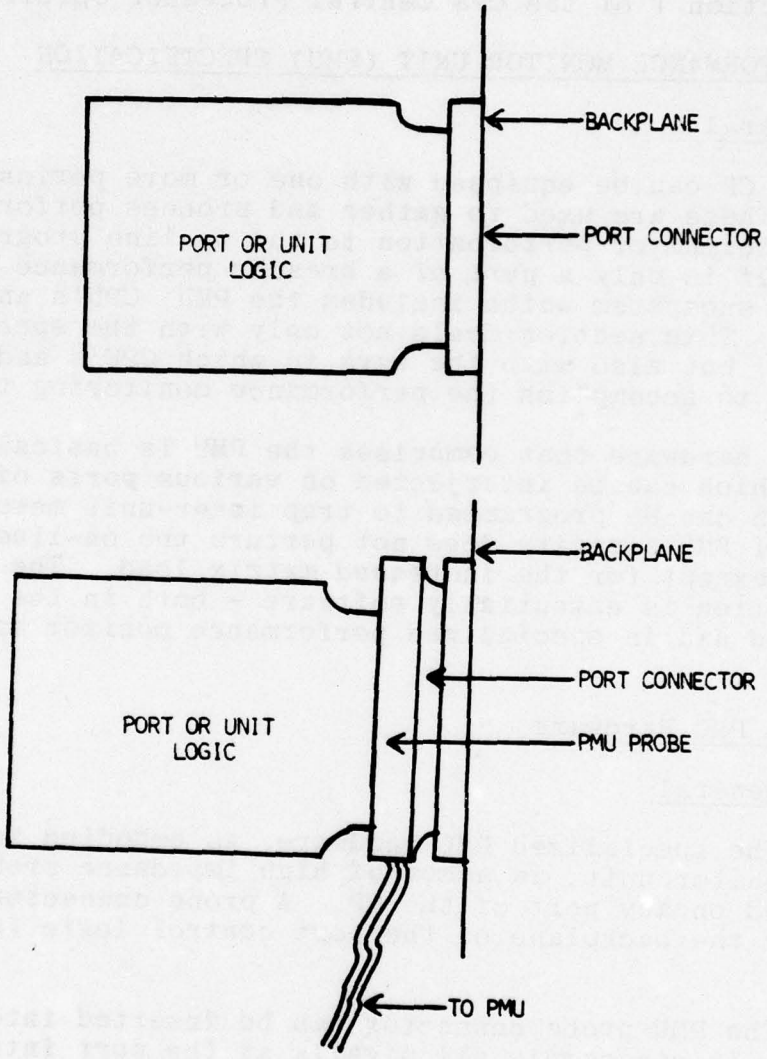


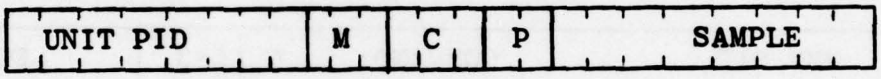
FIGURE X-1 PMU PROBE INSTALLATION

the ability to monitor that many. The ability to monitor them simultaneously depends upon the internal matrix traffic.

A probe can be set to monitor in one of two modes: IU mode and port mode. In the IU mode, the PMU traps and logs IU commands that are actually executed. In the port mode, the specific IU operation executed is of no importance; instead the attention is focused on path completions, path rejections, blockages, delays, etc. The IU mode is primarily used to monitor the utilization of non-matrix units. The port mode is used primarily to monitor the utilization of the matrix units.

3.2.2 Control Memory Layout

The control memory is split into two sections. The first section is indexed by the probe's physical number. The second section is indexed by either the SID or the DID of the matrix transmission. The probe indexed entry is used for all monitoring functions. It has the following format:



UNIT PID - the physical ID of the unit to which the probe is to be attached.

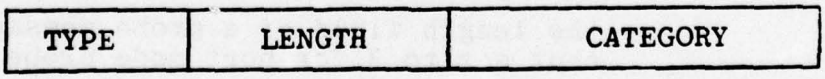
M - the probe mode - Port, SID, or DID, or any combination thereof. A 000 mode turns the probe off.

C - the C-field entry to be placed in the probe message.

P - the probe priority.

SAMPLE - the sample rate to be used. If SAMPLE = 1, all transmissions are examined. If SAMPLE = 14 every 14th transmission is sampled.

The DID or SID indexed entry has the following format:



TYPE - unit type; either XU, MU, CPU, MMTU, SMU, STCU, BU. All other unit types.

LENGTH - maximum length of generated probe message.

CATEGORY - eight categories of instruction types, whose interpretation depends upon the unit type.

3.2.3 Port Mode

The beginning of a port control line state change signals the onset of an IU operation of some sort. The port number is used as an index in the control memory and the control word is fetched. A sample counter for that port is incremented, and if it becomes zero, the sample will be made, and the counter reset to the value specified in the control memory. The IU message being transmitted is captured and a probe message is transmitted to the CPU which is doing the performance monitoring. The probe message has the same format as an interrupt message and it is handled by the PMU-CPU in exactly the same way. The message structure is:

CONTROL CHAR.

PMU SID	COMMAND	C	L=3	SEQUENCE #

PROBE #	DID	SID	LENGTH

STATE I	STATE F	CODE	TYPE	TIME	CI

The fields have the following interpretation:

PMU SID - the normal SID in an IU message - this time the ID of the PMU.

COMMAND - the interrupt message command - XXXX.

C - the C-field value used with probe messages.

L - the length field of a probe message, except set to 3 for port mode probe messages.

SEQUENCE # - a running sequence number of the message transmitted for that probe.

PROBE # - the physical ID of the probe.

DID	- the DID of the IU message that was examined.
SID	- the SID of the IU message that was examined.
LENGTH	- the length of the IU message in characters.
STATE I	- the port state prior to the receipt of the IU message (e.g., busy, free, locked, etc.)
STATE F	- the port state after the transmission of the IU message.
CODE	- the status of the transmission attempt: COMPLETE, NO PATH, REJECTED SID, REJECTED DID, REJECTED OPX, etc.
TYPE	- generic unit micro, unit specific micro, etc.
TIME	- the least significant 8 bits of the millisecond clock.
CI	- the first character of the IU message.

This message, which we shall refer to as a "probe message", while having the general structure of an interrupt is not sent to the ICU or the alternate ICU, but to the CPU that is handling performance monitoring. The subsequent treatment of the probe message by the monitor CPU will closely parallel the treatment of interrupts by the ICU CPU. The C-field combined with the values stored in that CPU will be used to place the probe message on one of 64 (actually, one of 8 since there is typically only one PMU) FIFO stacks, etc. The actual transmission of a probe message does not occur until the probe has sensed that the command has terminated.

3.2.4 IU Mode

The IU mode begins the same way as the port mode. If a DID mode has been selected, the DID of the IU message will be used as an index to the control memory, and the contents thereof together with the interpreted command will determine if this message or event is to be sampled. This occurs after the sample counter. If the SID mode has been selected, the SID is used as an index instead. The unit type in the control memory specifies the way in which the command should be interpreted. The command is categorized and if the control memory specifies that it should be reported, a probe message is generated. If the category is not interesting, the probe ignores further activity on that port

until a new IU command is initiated or received. Then the sample counter is checked and the cycle begins anew. The structure of the probe messages for the SID or DID IU modes is:

CONTROL CHAR.

PMU SID	COMMAND XXXX	C	L	SEQUENCE #

PROBE #	TIME	DID	SID
			. . .
			DATA

The fields are interpreted the same way as the fields of the port mode messages, except that the length can be variable. The DATA is the first 4-8 characters of the IU message that was intercepted. The transmission of the probe message is held up until it has been confirmed that the monitored IU message had been accepted by the destination.

3.2.5 Combined Modes

If the port mode and one of the two IU modes has been selected, then two different probe messages may be sent. If both the SID and DID mode has been selected, then the control memory will first be accessed using the DID as the index. If this results in a probe message, the SID entry will be ignored. If the DID index did not result in a probe message, the SID index will be tried.

3.2.6 Other Messages

Whenever a probe sequence number "turns over" the PMU sends a message in interrupt format, to the monitor CPU containing the binary time of day in millisecond format, along with the probe number. Additional messages of a diagnostic nature are also transmitted. These are discussed elsewheres.

3.2.7 Priority and Stacking

Completed probe messages are stored on a stack prearatory to transmission to the PMU CPU. Generally, this port and the associated path should be kept locked at all times in order to minimize perturbations to the rest of the complex. The stack

contains 256 characters. Priority plays no role unless the stack is more than half full. In such cases, messages are placed on the stack in decreasing priority order. If a message should fail to get stacked, and if a new IU transmission should be monitored by that probe, it will overwrite the old message, that had not yet been transmitted. In this case, a "missed message counter" will be incremented for that probe. A diagnostic message bearing the probe number, the time, and the number of missed messages will be transmitted to the PMU CPU at the first opportunity. If the missed message counter for that probe should run out, the diagnostic message will be transmitted with the highest priority. A message is "missed" in this context only if the sample interval criteria was not met, and if the type criteria was not met for the IU mode.

3.3 The PMU as a Unit

The PMU responds to the generic unit micro-commands with SST instructions being used to load and the FST instructions used to fetch the control memory contents. The PMU does not have a validation memory and will accept commands from only the designated PMU CPU, which is determined by the SMID (SET UNIT MAX) command, and another CPU defined as the MIN UID. ICU ID's are also specified and are used for diagnostic and alarm purposes. To minimize the perturbations caused by the PMU on the rest of the system, the primary ICU ID should be set as that of the PMU.

3.4 PMU Software

3.4.1 General

PMU software can be divided into two categories: (1) the software required to process probe messages, and (2) the software required to process software probes. It is not the intention of this section to discuss either software set in detail as this may differ from application to application. The intention here is to indicate the ways in which the various facilities of the CP can be used to advantage in creating such software packages.

3.4.2 Port Mode Probe Message Processing

The port mode probe messages should be reserved almost exclusively to the determination of internal matrix traffic and general statistics on the kinds of IU operations that are performed, without tying things down to specific application dependent characteristics. The port mode messages can be used to determine (statistically) the percentage of all IU messages that were rejected and the reason for their rejection. Path conflicts and blocking probabilities can be determined in this manner. Similarly, we can determine if a particular memory unit is being

overloaded with requests, without however identifying the specifics of the overload.

Because the port mode has relatively little discrimination powers, it can be readily overloaded. That is, if the sample interval is set too low (e.g., every I/U message) any one port could be reporting at approximately memory speed. Ultimately, every probe message ends up in a memory unit. If a probe was set to each of four ports on a given memory unit which was to be monitored, and all messages were sampled, we would have a three word probe message generated for every IU message monitored. This would effectively mean that we would have to do at least 4 memory fetches/loads for every memory fetch/load monitored. This is clearly a non-converging approach. Sampling is the key to the use of the port mode, especially when tied to a CPU, memory or matrix.

The number of ports of a given unit to which a probe is attached is another way of sampling. If it is known that (statistically) ports are being used with equal frequency, then by attaching a probe to only one port rather than four, we can cut the sampling rate by four again. Since it is relatively easy to overload the PMU when using the port mode, the number of ports that are simultaneously sampled should be minimized. Alternatively, we can sample on one port for a few seconds, shift to another for the next few seconds, etc. If such samples are taken periodically over a period of time, good (stable) statistics can be gathered without overloading the PMU or its control CPU. Where the application warrants intensive high frequency samplings in the port mode, more than one PMU and CPU can be used for monitoring.

The application level software typically will consist of examining the contents of the probe message, categorizing it in accordance to the features of interest, and tallying a counter corresponding to that condition. More elaborate statistical messaging should be done off-line, or on-line at low priority.

3.4.3 IU Mode Software

The same general philosophy that applied to the use of the port mode should be applied to the IU mode probe message processing. However, because the IU modes have a finer discrimination capability, it is possible to set the sample rate relatively high. IU modes should be used to detect the time at which fairly specific events took place, such as the initiation and termination of I/O operations.

3.4.4 CPU Monitoring

The PMU CPU can act just like an ICU and consequently can sample the program state of any CPU, can examine the PC, etc.

If this is done (with caution) and at relatively infrequent intervals, time spent in each program state, time spent accessing tables, time spent accessing data, net number of instruction executions per second, etc., can be determined. However, since most of these operations block processing, they should not be overdone lest the monitoring function overwhelms the processing.

3.4.5 Use of Flag Instructions

The various flag instructions are the key to software monitoring. A number of flags should be reserved for exclusive use in performance monitoring. If the specified flag is not set, the instruction effectively behaves as a NOOP and the only penalty paid is the execution of that instruction. If the condition for the specified flag is set, the CPU will transfer to the software probe region, where appropriate instructions will be executed to record that which is being monitored. The flag conditioned subroutine return instruction is the key software monitoring instruction. It allows the programmer to jump to a software monitor area if a flag is set, but imposes no penalty if the flag is not set. Therefore, it imposes no penalty on a program or subroutine which is not being monitored.

SECTION XI

SPECIFICATION FOR THE MATRIX AND MATRIX UNIT (XU) OF THE CPS CENTRAL PROCESSOR

SPECIFICATION FOR THE
MATRIX AND MATRIX UNIT (XU)
OF THE
CPS CENTRAL PROCESSOR

TABLE OF CONTENTS

PARAGRAPH	TITLE	PAGE
1.0	Scope	XI-1
2.0	Applicable Documents	XI-1
2.1	General CP Specification	XI-1
3.0	The Matrix and Matrix Unit (XU) Specification	XI-1
3.1	General	XI-1
3.2	Matrix Structure	XI-2
3.3	Connection Conventions	XI-2
3.4	Links, Link States, and Crosspoint State	XI-4
3.4.1	General	XI-4
3.4.2	Free Links	XI-6
3.4.3	Reserved Links	XI-6
3.4.4	Busy State	XI-6
3.4.5	Locked State	XI-6
3.4.6	Error State	XI-6
3.4.7	Dead State	XI-7
3.4.8	Additional Link States	XI-7
3.4.9	Crosspoint States	XI-7
3.5	Saturation Signaling and Unit Control Sequences	XI-7
3.5.1	General	XI-7
3.5.2	Signaling Procedure	XI-7
3.6	Unit Generic Commands for the Matrix Unit (XU)	XI-17
3.6.1	General	XI-17
3.6.2	Interpretation of Generic Unit Commands for XU's	XI-17
3.6.3	Priority/Validation Cache Memory	XI-18

LIST OF FIGURES

FIGURE	TITLE	PAGE
XI-1	Matrix Schematic	XI-3
XI-2	Submatrix Structure	XI-5
XI-3	First Stage of Link Set Up	XI-9
XI-4	Link Set Up 1	XI-10
XI-5	Link Set Up 2	XI-11
XI-6	Link Set Up 3	XI-12
XI-7	Link Set Up 4	XI-14
XI-8	Link Set Up 5	XI-15
XI-9	Link Set Up 6	XI-16

1.0 SCOPE

This section is the specification for the Matrix and the Matrix Unit (XU) of the CPS Central Processor.

2.0 APPLICABLE DOCUMENTS

2.1 General Central Processor Specification

(Section I of the CPS Central Processor Specification).

3.0 THE MATRIX AND MATRIX UNIT (XU) SPECIFICATION

3.1 General

The operation of the matrix is transparent to the programmer. Functionally, it plays the same role as the more common bus system. However, for any particular application, the system architect has a wide choice of matrix configurations which he can construct out of the same basic submatrix units. The structure of the matrix for any specific application depends upon the following factors:

(1) The total number of active and passive units implemented.

(2) The number of matrix ports required for each unit.

(3) The total number of memory units implemented.

(4) The amount of active unit to passive unit traffic.

(5) The required path availability - degree of graceful degradation required.

(6) The processing load on the system.

Units can be implemented with one to four ports into the matrix. Typically, a CPU will have 2 or 4 ports, as will memory units. Channels which serve critical functions or high traffic functions such as discs and trunk lines, would be implemented with two ports. Low speed lines, service lines, VDU's, etc., would be implemented with a single port channel. If the unit's operation is to be retained despite a single matrix unit failure, then two ports are required for that unit. Otherwise, the choice of more than one port is based on the speed and traffic handled by that unit. Note that in this context "traffic" refers to system internal data transfers through the matrix and not the external traffic handled by the system.

Matrix sizing affects speed and reliability, but not functional capabilities. If the matrix is too small, the system

reliability and throughput will suffer. If too large, the system cost will suffer.

3.2 Matrix Structure

Figure XI-1 shows the overall structure of the matrix. The matrix consists of three stages, called respectively, the "active unit state", and the "center stage", and the "passive unit stage".

The center stage is further divided into two types - the passive unit interconnect center stages, and the active unit interconnect center stages. Each stage consists of a number of component switching submatrices, typically with 8 inlets and 4 outlets. A submatrix allows any inlet to be connected to any free outlet without interferring with pre-existing connections in that submatrix. A submatrix is said to be "non-blocking" for this reason. Each inlet or outlet provides a full duplex character serial interface. In addition, there are a number of control lines which follow the same path.

The matrix employs a distributed control which has inherent graceful degradation. The control scheme is based on saturation signaling.

A path through the matrix consists of an active or passive stage path, followed by a center stage path. If the connection is to a passive unit, a rectangular center stage is selected. The path then continues through the passive stage to the appropriate memory unit. If the connection is to be made to another active unit, a triangular stage is selected and the path is continued to the appropriate receiving unit by means of another active unit stage. Memory-to-memory transfers are accomplished by selecting a path from the memory unit, to the passive stage submatrix, through a rectangular center stage to an active stage submatrix to a memory-to-memory transfer unit, back through (possibly, but not necessarily) a different active unit stage submatrix, a rectangular stage submatrix, a passive unit stage submatrix, and finally, to the other memory unit.

3.3 Connection Conventions

From the point of view of the matrix, there are two types of units. Units which can and do normally initiate transfers (active units) and units which do not normally initiate transfers (passive units). Memory units and matrix control units do not normally initiate transfers. A connection between units is always a full duplex connection. Every unit has both a physical and a logical identity. The identity is established by the unit number (ID). The physical identity (DID) of the unit is a strapped number (wired-in). The logical identity of the unit (LID) is an eight bit unit number stored by the unit itself.

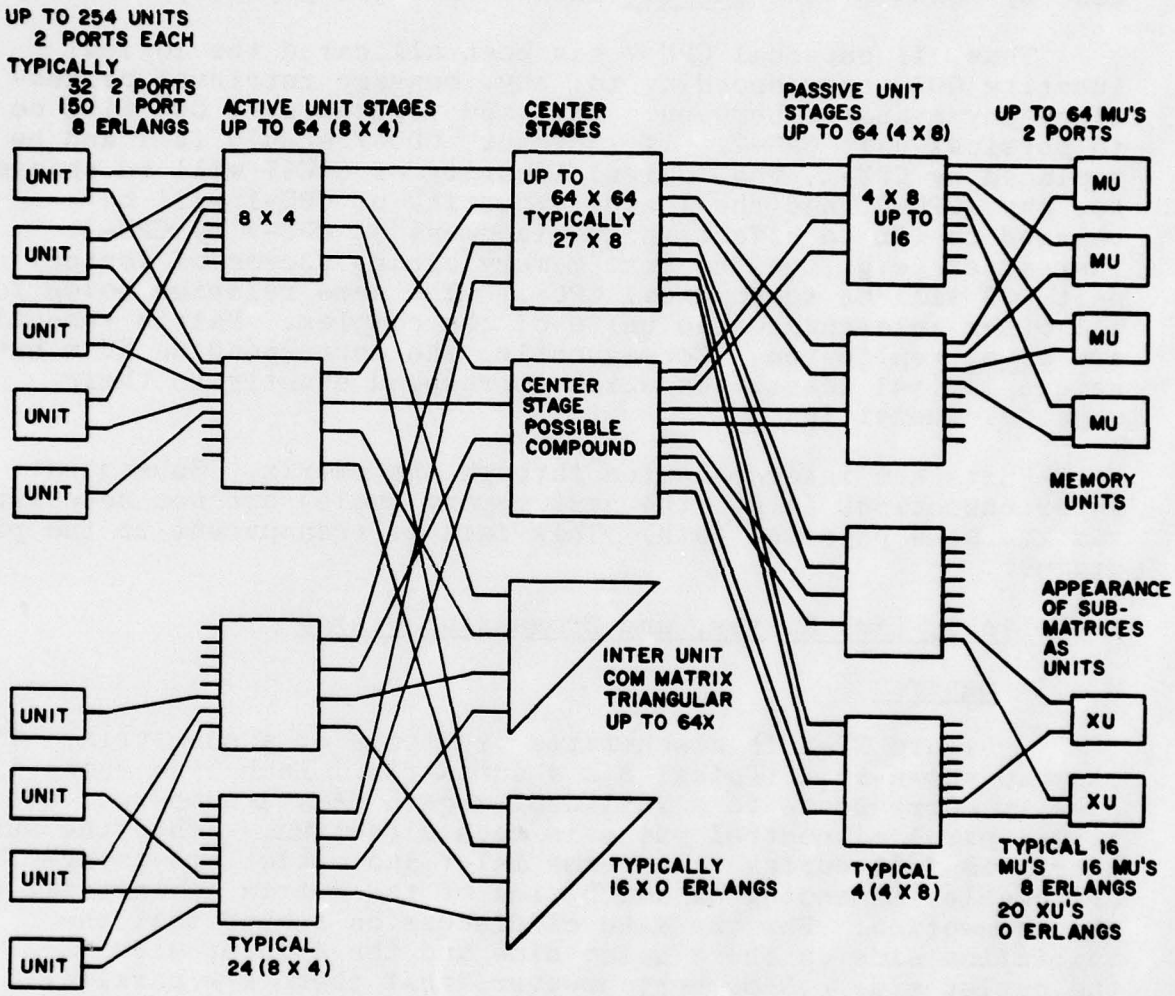


FIGURE XI–1 MATRIX SCHEMATIC

A unit can change its logical identity, or have its logical identity changed for it. It cannot change its physical identity. Logical identities are established by the programmer and are generally used to identify functional elements of the complex. Physical identities are wired-in. The failure of a unit will result in the reassignment of its logical identity to a non-failed unit of the same type. This is done under program control. Addressing of units and the interconnection of units is done by means of the logical rather than the physical identity.

Thus, if physical CPU-7 has been allocated the logical identity 005, corresponding to, say, message retrieval processing, any transfer between a unit and logical unit 005 will be to physical unit CPU-7. If, however, CPU-7 should fail and be replaced by CPU-1, the logical identity of CPU-7 will be changed to, say, CPU-0, and the logical identity of CPU-1 will be changed to 005 to effect the replacement of CPU-7 by CPU-1. Thereafter (e.g., on the next memory cycle) access of logical unit 005 will be to physical CPU-1. The same relation holds for all other interchangeable units of the complex. Matrix sub-units are never replicated. Consequently, the corresponding XU's have unique logical identities which correspond exactly to their physical identities.

Units are interconnected through the matrix. Subsequent interconnections (e.g., the next memory cycle) are not necessarily via the same physical path. This fact is transparent to the programmer.

3.4 Links, Link States, and Crosspoint States

3.4.1 General

Figure XI-2 is a schematic rendition of a submatrix. The example shown is a typical 8 x 4 submatrix. Each line shown in reality corresponds to a full duplex path plus a number of additional parallel control paths in each direction. Since the submatrix is full duplex, the terms inlet and outlet are interchangeable, depending on which side of the matrix is initiating the connection. For the sake of discussion assume that the initiation side is the 8 inlet side and the 4 inlet side is the outlet side. Remember, however, that there are parallel paths in the opposite direction. Any one inlet link is capable of being connected to one or more outlet links simultaneously. However, while a transfer is actually in progress, inlets are connected to exactly one outlet (and vice versa). Telephony jargon is used to distinguish between links by calling them horizontal and vertical. A horizontal link can be in one of the following states:

FREE: not latched to any vertical.

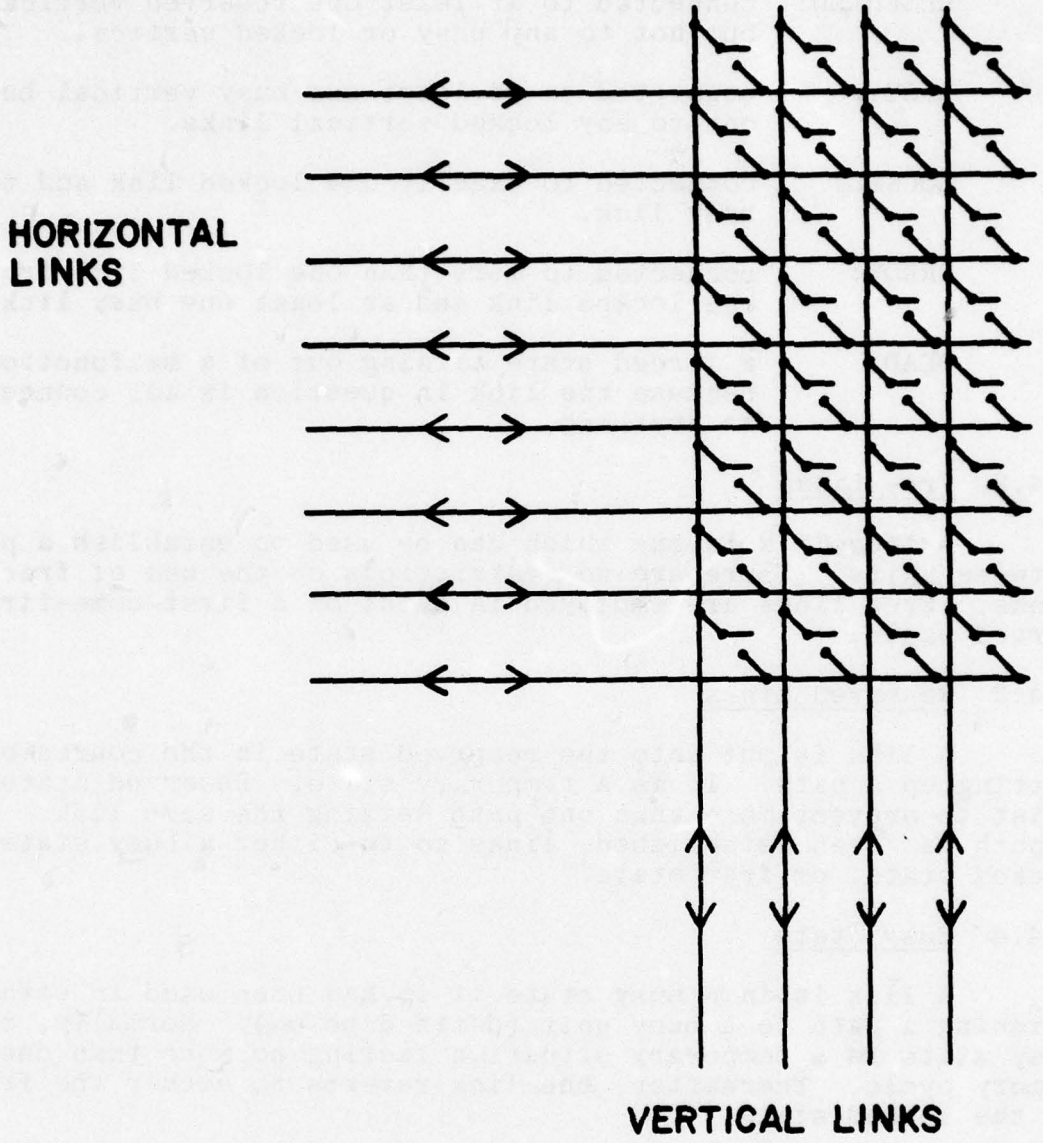


FIGURE XI-2 SUBMATRIX STRUCTURE

RESERVED: connected to at least one reserved vertical but not to any busy or locked vertical.

BUSY: connected to at least one busy vertical but not to any locked vertical links.

LOCKED: connected to exactly one locked link and to no busy link.

ERROR: connected to more than one locked link or to one locked link and at least one busy link.

DEAD: a forced state arising out of a malfunction or because the link in question is not connected to anything.

3.4.2 Free Links

A free link is one which can be used to establish a path between units. There are no restrictions on the use of free links. Free links are employed in paths on a first-come-first-served basis.

3.4.3 Reserved Links

A link is put into the reserved state in the course of setting up a path. It is a temporary state. Reserved states exist to prevent more than one path seizing the same link. Once a path has been established, links go to either a busy state, locked state, or free state.

3.4.4 Busy State

A link is in a busy state if it has been used in establishing a path to a busy unit (defined below). Normally, the busy state is a temporary situation lasting no more than one memory cycle. Thereafter, the link reverts to either the free or the locked state.

3.4.5 Locked State

A link is locked if it is used in a path which is presently involved in a transfer. It remains locked for the duration of the transfer and reverts to the free or reserved status thereafter. A link can be locked by means of an explicit link lock command. It will then remain locked until unlocked.

3.4.6 Error State

An error state is a temporary state occurring upon the detection of an error condition. The detection of an error state results in the generation of an interrupt.

3.4.7 Dead State

A link in the dead state may not be used for any purpose whatsoever. A link remains in the dead state until it is forced to the free state by means of an explicit command.

3.4.8 Additional Link States

There are additional link states intermediate to these which occur during the course of setting up a path. These are totally internal to the operation of the matrix.

3.4.9 Crosspoint States

Connections between horizontal and vertical links are made by activating the appropriate crosspoint (an automatic action, not a programmed action). A crosspoint is basically in one of two states - latched or unlatched.

3.5 Saturation Signaling and Unit Control Sequence

3.5.1 General

Unit interconnection is accomplished by a technique called "Saturation Signaling". In a telephone network, this would be analogous to the following:

You wish to call a specific party, say "JOE", whose name is unique. You pick up the telephone - your telephone is immediately connected to all non-busy telephones in the network. The recipients pick up their handsets, whereupon you shout "JOE?". Everybody but JOE hangs up; he confirms the connection. Some telephones could have been busy. If JOE has answered, you signal the busy telephones to stop trying. If JOE was busy, you keep all paths open to all busy telephones until JOE does answer.

3.5.2 Signaling Procedure

The following steps are employed in establishing a connection between a source and destination unit:

(1) The unit selects a port (assuming that it has more than one) on which the transfer is to occur. The specific port chosen is a matter of concern to the unit and not to the matrix or other units. The selection procedure for ports are particular to unit types and will be discussed in that context.

(2) The source unit transmits the DID to the matrix along the link between the chosen port and the active stage submatrix. The DID is propagated in parallel

throughout the matrix along all free links encountered. This propagation proceeds through all matrix stages creating a tree which fans out as it goes. A branch of the tree ends when either the branch reaches a free port, or all available vertical links are busy, reserved, or locked. Ports use the same conventions with regards to busy, locked, free, or reserved that links do; however, the reason for selecting a particular state may differ from unit to unit.

This stage of the set up is shown schematically in Figure XI-3. The source unit is unit A. Since no other units are busy, all paths are free, and the request (DID) goes to all destination units.

(3) Assume that the destination port was free, and that the destination unit was willing to accept a transfer command. All free units that receive the DID examine the DID and compare it with their own logical identities. If the identities do not match, the free unit transmits a path kill signal, which will release all links used between the source unit itself that were not used in any other path. If the DID matches the logical ID of the destination unit, the destination will respond by echoing the DID back to the source and transmitting a BUSY or LOCK signal back along the path. The choice depends upon the destination unit.

This sequence of events is shown pictorially in Figures XI-4 through XI-6. In XI-4, various destination units have responded with a NO. This signal frees the links that were in use. In XI-4, the freeing has progressed through the destination stage. At the same time, A's destination has properly matched the DID and is simultaneously sending a reserve link signal and the DID echo.

In Figure XI-5, the signals have continued to the center stage, and in Figure XI-6, the path reservation is complete.

(4) The echoed DID traverses the matrix back to the source unit. At the same time, the BUSY or LOCK signal traverses back to the source unit. As it does so, it switches all links in the tree which are not directly involved in the source to destination path back to the FREE state. The end result will be that a single path has been established between the source and the destination.

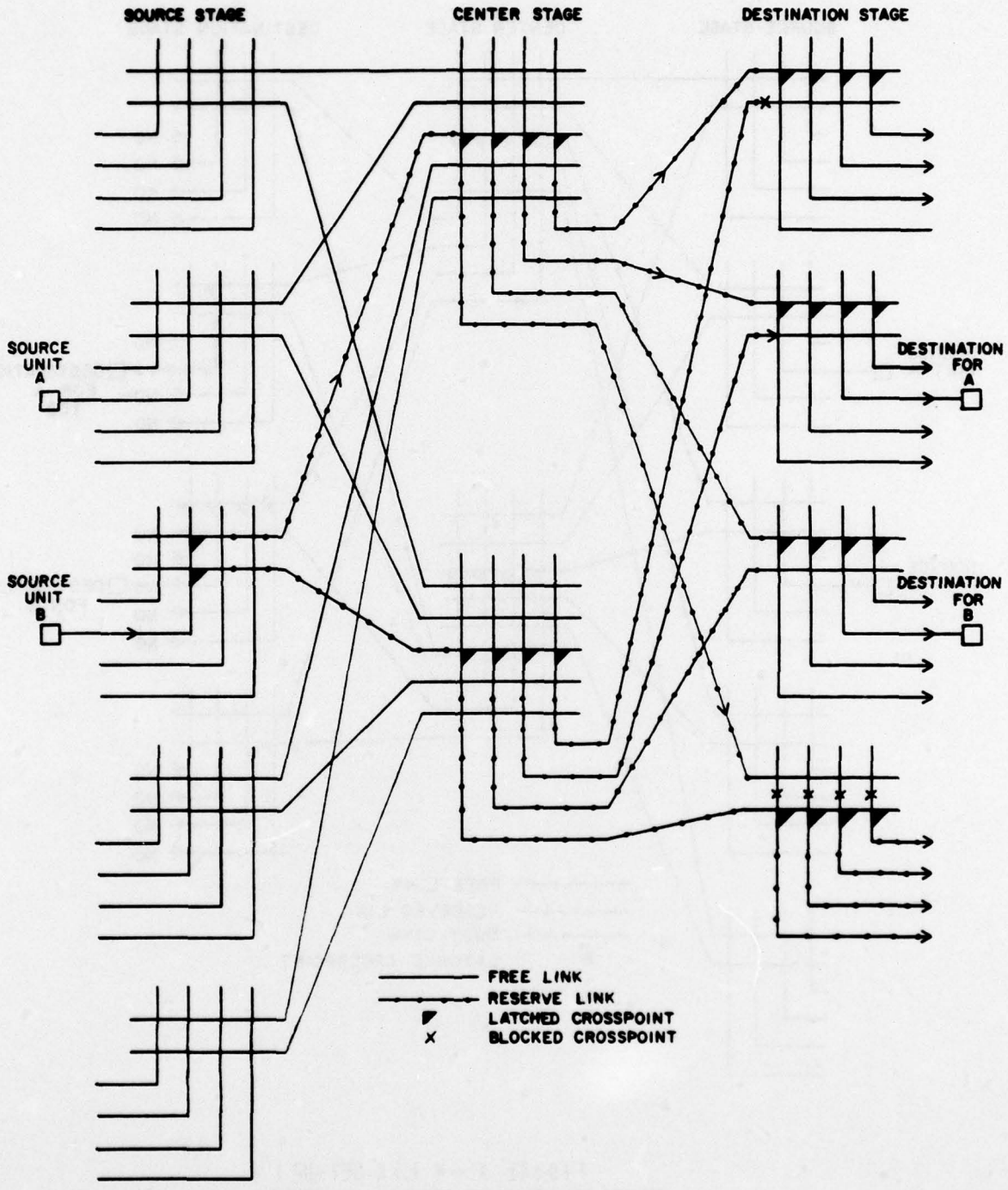


FIGURE XI—3 FIRST STAGE OF LINK SET-UP

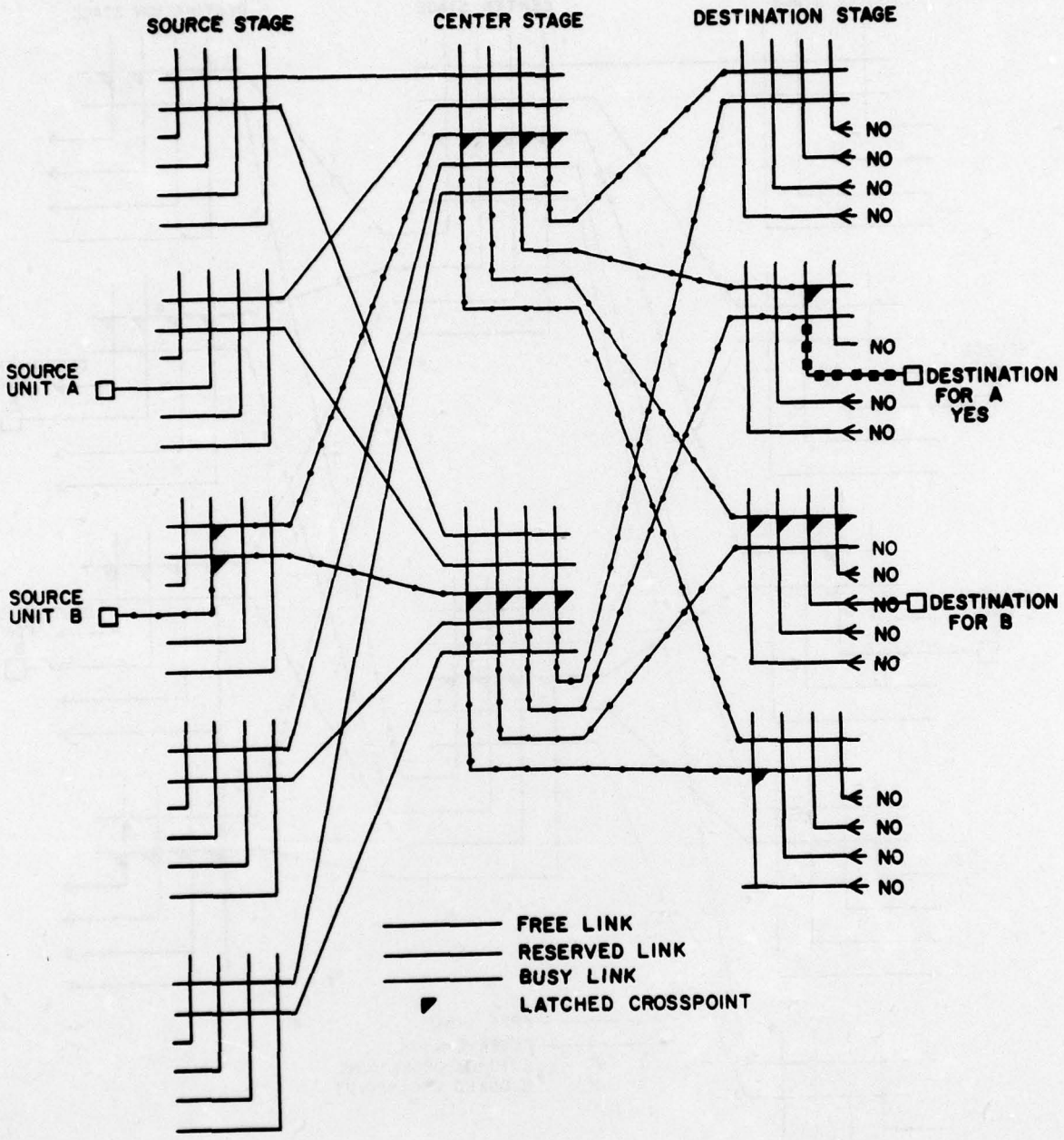


FIGURE XI–4 LINK SET-UP I

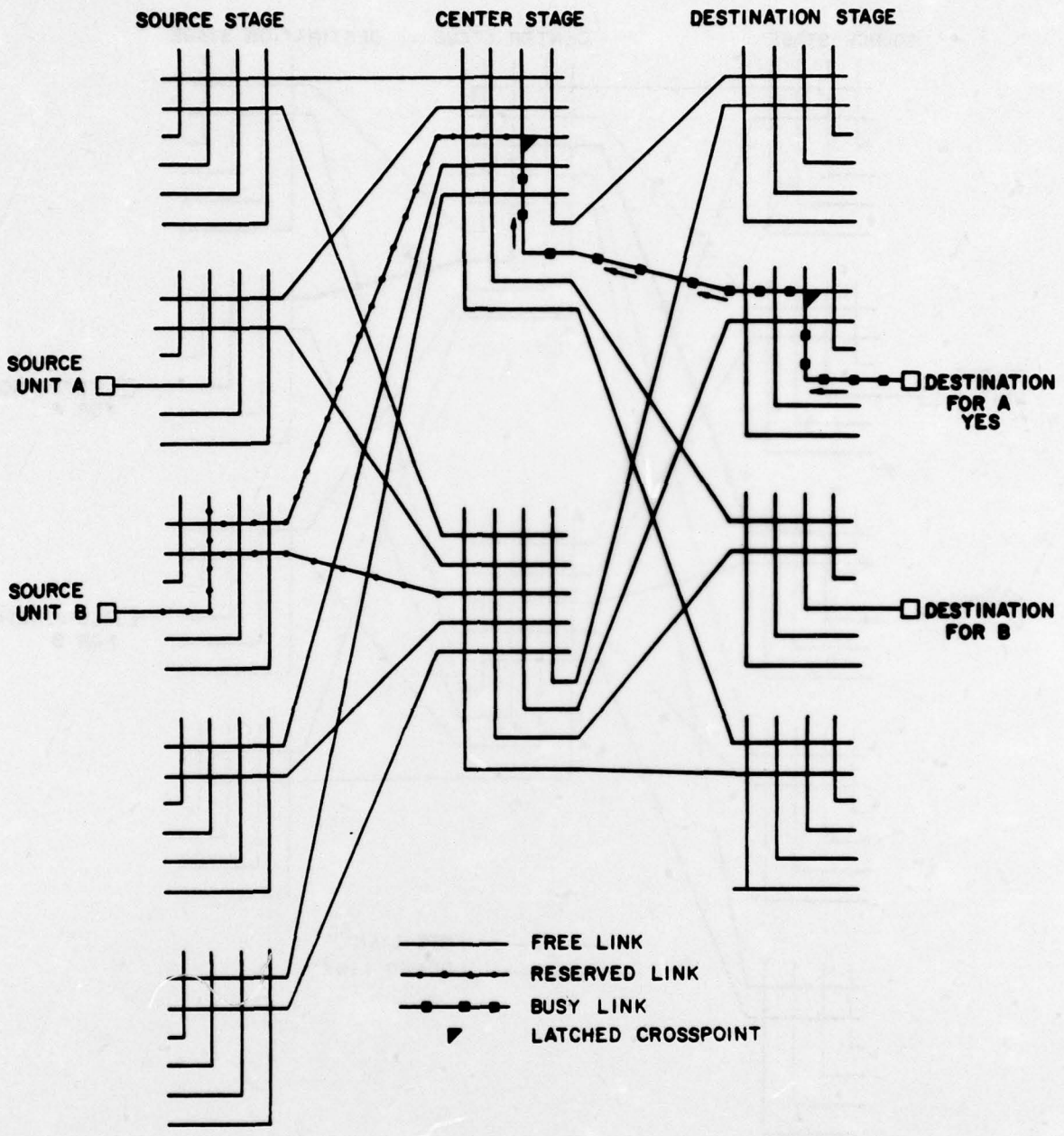


FIGURE XI-5 LINK SET-UP 2

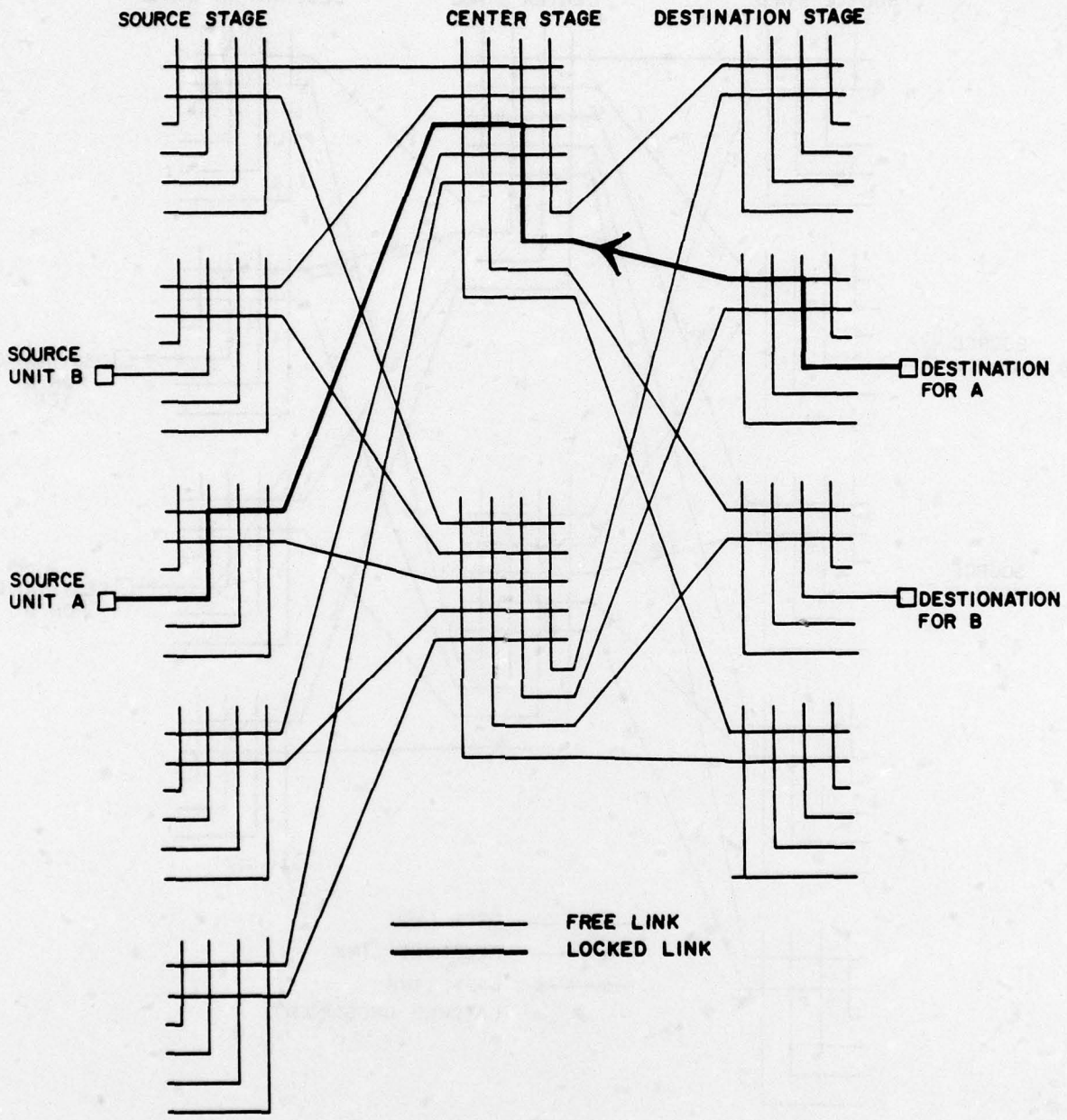


FIGURE XI-6 LINK SET-UP 3

Figures XI-7 through XI-9 show the same thing repeated under the assumption that a path had already been set up.

(5) If either all paths from the source to the destination were blocked (because they were in use in some other source-destination pair) or if the destination unit of interest was busy or locked, there would have been no LOCK or BUSY command propagated back along the matrix and hence, the part of the tree connected to blocked or busy links would not have reverted back to the FREE state. The parts of the tree leading to improperly selected units would have been killed by the action described in 3 above. The links thus involved (in the BUSY or BLOCKED paths) remain in the reserved state until something happens. Eventually, paths are unblocked and/or BUSY or LOCKED units become free, in which case they respond with a kill signal, or the destination unit becomes free in which case the action proceeds as before, with the path being established.

(6) If the source unit received a no-path signal, depending upon the nature of the unit and the desired transfer, one of the following alternatives may be taken:

a. Retry until a connection is made.

b. Try the connection via an alternate port (if there is one).

c. Try the connection with a usurpation mode.

(7) The usurpation modes are used to force connections for important signals such as interrupts. Connections are made in the normal manner, except that reserved links will be usurped.

(8) Once the path has been established, the source unit examines the echoed DID and compares it to the one that was sent. If they do not match, an error condition has occurred and an appropriate interrupt is manufactured and transmitted.

(9) If the reflected DID matches, the source unit transmits its own ID (SID) to the destination unit. This is necessary because the mere establishment of a path does not guarantee that a transfer will be allowed. The priorities with which a transfer will be allowed is up to the destination unit and depends

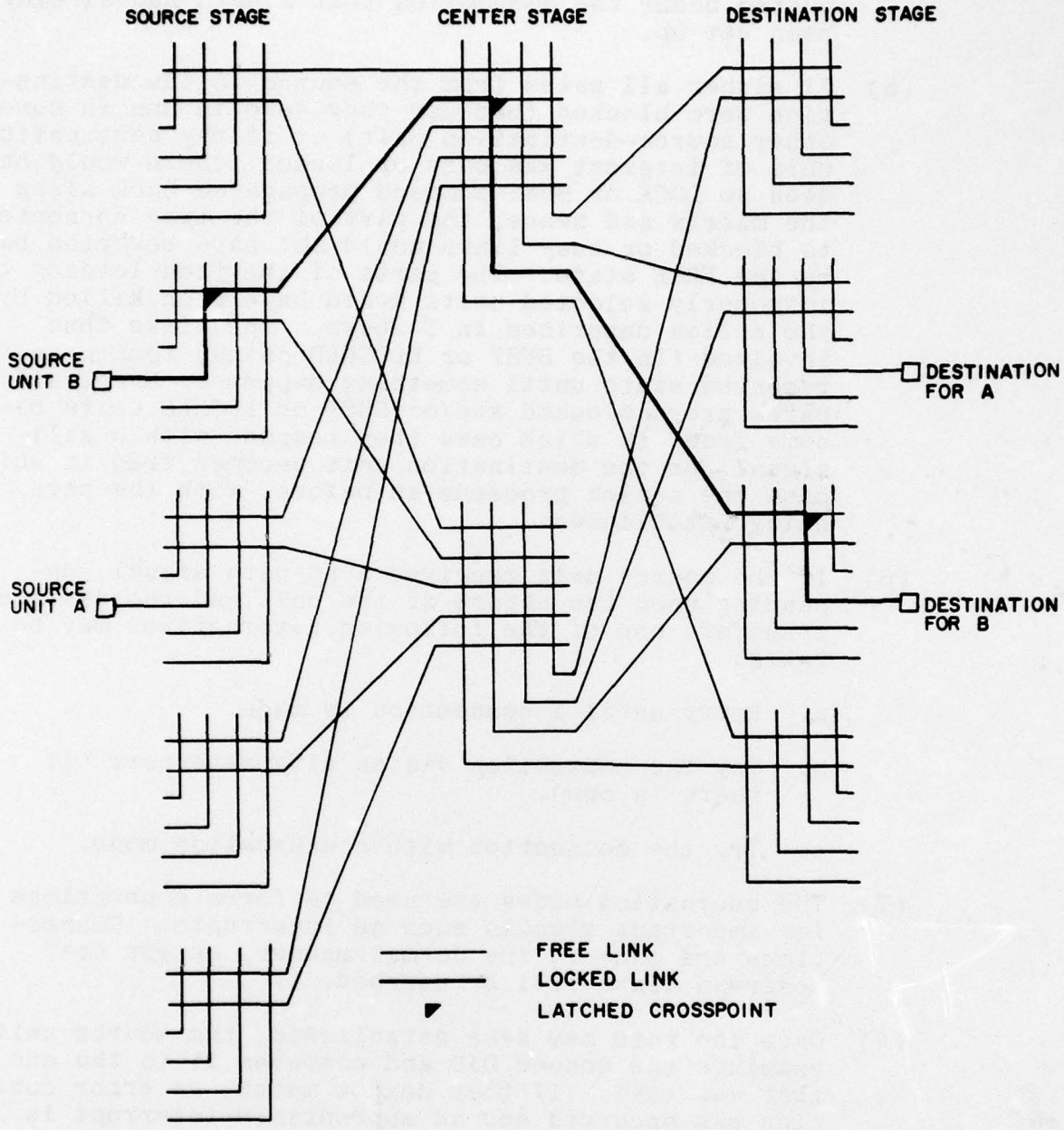


FIGURE XI-7 LINK SET-UP 4

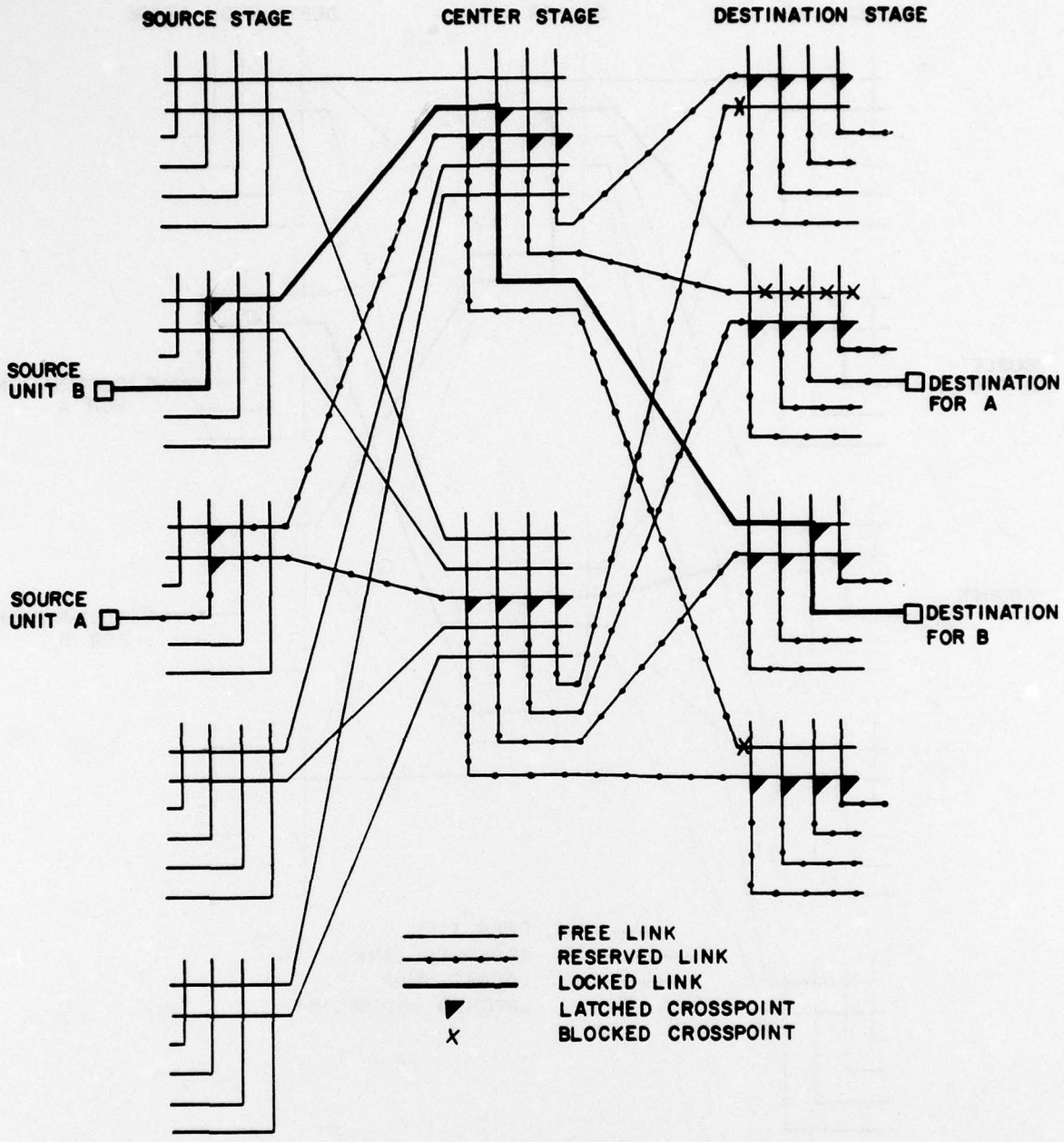


FIGURE XI-8 LINK SET-UP 5

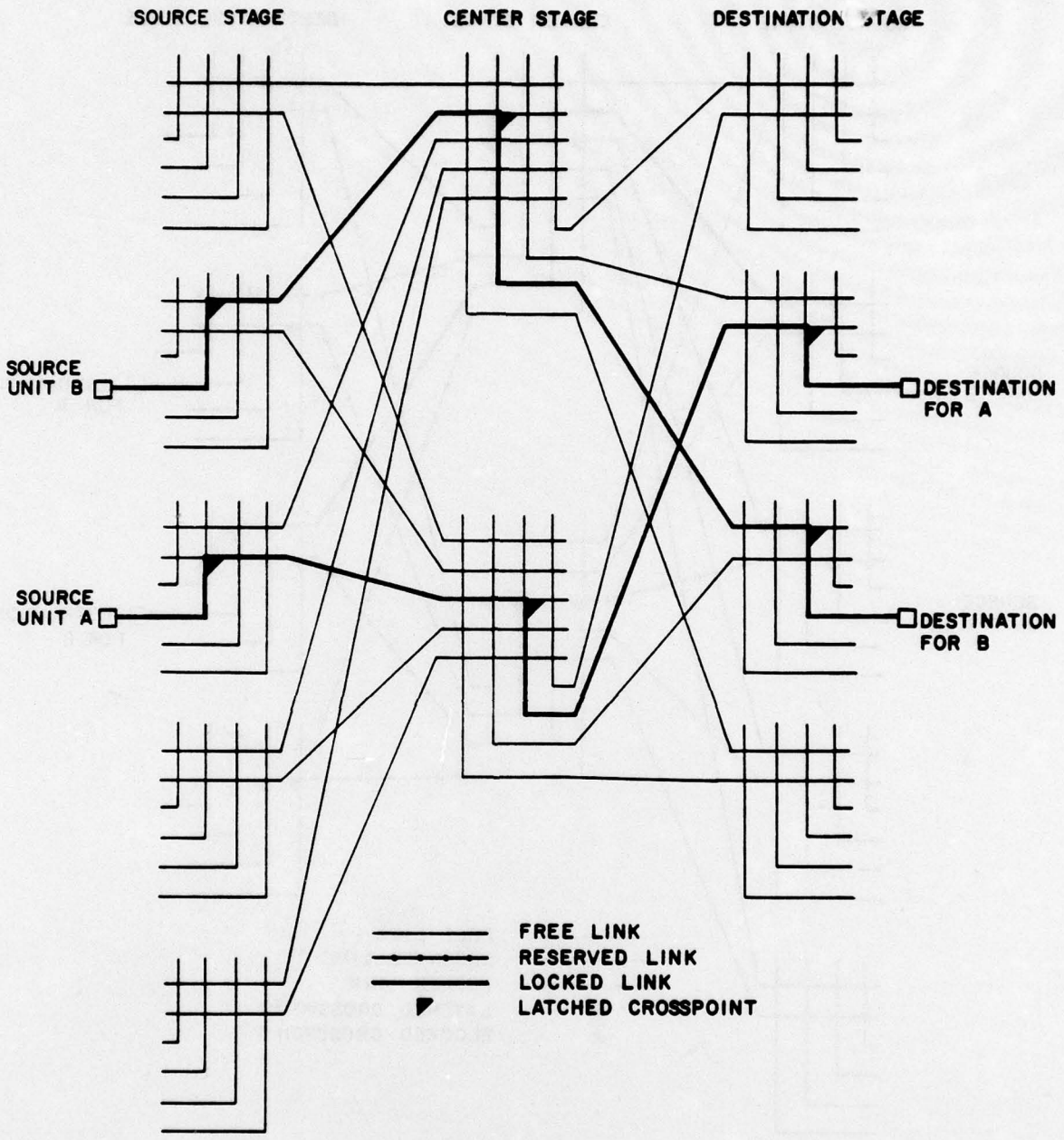


FIGURE XI-9 LINK SET-UP 6

upon (in general) the SID, the internal state of the destination unit, and the priority structure employed by that unit. This is particularly true for memories and CPU's. Channel units will typically accept any request.

(10) The destination unit then changes the state of the path to LOCKED, BUSY, RESERVED, or FREE, depending upon what action it intends to take. Typically, the state is changed to LOCKED. A FREE state might occur as a result of receiving an improper SID relative to that destination - for example, an attempt to connect two clock units to each other, say. An improper SID will result in an interrupt being generated.

(11) If the SID is correct, the source unit then transmits a command to the destination unit, describing the kind of transfer or action which is to take place. Command validity is checked by the destination unit and if correct, is acted upon.

(12) If the command is valid, data (if any) is transmitted possibly in either or both directions.

(13) At the conclusion of the transfer, the state of the path is changed to RESERVED, holding the path open for future transfers, unless usurped. Other termination states are possible, if explicitly commanded. In all cases, the status of the path following the transfer is the lower priority (FREE is lower than LOCKED) of that established by the source and destination units.

3.6 Unit Generic Commands for the Matrix Unit (XU)

3.6.1 General

Each submatrix of the matrix is itself a unit, with a normal unit interface to the matrix. The ports of an XU should not be terminated on the self-same XU. XU's are not normally treated as units. There is no reason to issue a unit command to an XU save in the interest of testing, diagnosing, or bootstrapping the system. The XU commands described here are from the point of view of the appearance of the XU as a unit and not from the point of view of its function as part of the matrix. The LID of the XU is identical to its PID.

3.6.2 Interpretation of Generic Unit Commands for XU's

Most commands operate in the normal manner and need not

be discussed again. Since the XU's have no data memory (as distinct from a cache memory) memory referencing commands and the indirect mode are forbidden. The specific action for various commands are given below:

GO OFF-LINE	- all links are marked DEAD. All crosspoints are unlatched. Reset requires operator action.
RESET	- all links are marked FREE. All crosspoints are unlatched.
CLOSE	- normal CLOSE commands. Link states and crosspoint states remain unchanged. The unit will not respond to control line signals on matrix links.
RETURN STATUS	- responds with a unit wide control state (OPEN, CLOSED, POWER DOWN, etc.).
LOCK PATH	- normal LOCK PATH command for the XU control links. This does not affect the XU matrix links.
UNLOCK	- normal UNLOCK command for the XU control links.
SET STATE	- the major control state of the XU is set as specified.
FETCH STATE	- normal operation.
SET STATE 1	- sets the bits in the validation memory.
SET STATE 2	- sets the crosspoints in accordance to the prescribed pattern.
SET STATE 3	- sets the link states in accordance to the prescribed pattern.
FETCH STATE 1	- dumps the bits of the validation memory.
FETCH STATE 2	- dumps the crosspoint states.
FETCH STATE 3	- dumps the link states.

3.6.3 Priority/Validation Cache Memory

Each XU has a 256 or 512 bit priority/validation memory. The two bit code (P-field) is interpreted as follows:

P = 00 - named unit cannot function as source for this XU.

P = 01, 10, 11 - named unit has priority P with respect to execution of XU unit commands.

The XU does not have command stacking. Priority operation and rules are as discussed in Section I, the CP Specification.

METRIC SYSTEM

BASE UNITS:

Quantity	Unit	SI Symbol	Formula
length	metre	m	...
mass	kilogram	kg	...
time	second	s	...
electric current	ampere	A	...
thermodynamic temperature	kelvin	K	...
amount of substance	mole	mol	...
luminous intensity	candela	cd	...

SUPPLEMENTARY UNITS:

plane angle	radian	rad	...
solid angle	steradian	sr	...

DERIVED UNITS:

Acceleration	metre per second squared	...	m/s
activity (of a radioactive source)	disintegration per second	...	(disintegration)/s
angular acceleration	radian per second squared	...	rad/s
angular velocity	radian per second	...	rad/s
area	square metre	...	m
density	kilogram per cubic metre	...	kg/m
electric capacitance	farad	F	A·s/V
electrical conductance	siemens	S	A/V
electric field strength	volt per metre	...	V/m
electric inductance	henry	H	V·s/A
electric potential difference	volt	V	W/A
electric resistance	ohm		V/A
electromotive force	volt	V	W/A
energy	joule	J	N·m
entropy	joule per kelvin	...	J/K
force	newton	N	kg·m/s
frequency	hertz	Hz	(cycle)/s
illuminance	lux	lx	lm/m
luminance	candela per square metre	...	cd/m
luminous flux	lumen	lm	cd·sr
magnetic field strength	ampere per metre	...	A/m
magnetic flux	weber	Wb	V·s
magnetic flux density	tesla	T	Wb/m
magnetomotive force	ampere	A	...
power	watt	W	J/s
pressure	pascal	Pa	N/m
quantity of electricity	coulomb	C	A·s
quantity of heat	joule	J	N·m
radiant intensity	watt per steradian	...	W/sr
specific heat	joule per kilogram-kelvin	...	J/kg·K
stress	pascal	Pa	N/m
thermal conductivity	watt per metre-kelvin	...	W/m·K
velocity	metre per second	...	m/s
viscosity, dynamic	pascal-second	...	Pa·s
viscosity, kinematic	square metre per second	...	m/s
voltage	volt	V	W/A
volume	cubic metre	...	m
wavenumber	reciprocal metre	...	(wave)/m
work	joule	J	N·m

SI PREFIXES:

Multiplication Factors	Prefix	SI Symbol
1 000 000 000 000 = 10^{12}	tera	T
1 000 000 000 = 10^{9}	giga	G
1 000 000 = 10^{6}	mega	M
1 000 = 10^{3}	kilo	k
100 = 10^{2}	hecto*	h
10 = 10^{1}	deka*	da
0.1 = 10^{-1}	deci*	d
0.01 = 10^{-2}	centi*	c
0.001 = 10^{-3}	milli	m
0.000 001 = 10^{-6}	micro	μ
0.000 000 001 = 10^{-9}	nano	n
0.000 000 000 001 = 10^{-12}	pico	p
0.000 000 000 000 001 = 10^{-15}	femto	f
0.000 000 000 000 000 001 = 10^{-18}	atto	a

* To be avoided where possible.

MISSION
of
Rome Air Development Center

RADC plans and conducts research, exploratory and advanced development programs in command, control, and communications (C^3) activities, and in the C^3 areas of information sciences and intelligence. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.

*U.S. GOVERNMENT PRINTING OFFICE: 1977-714-025/108