

AD-A066 241 DATA SYS DIV, LITTON SYS, INC., VAN NUYS, CAL
MULTIMODE CPU DESIGN STUDY (U)

F/G 9/2

UNCLASSIFIED JUNE 1978 GARY L. MALLALEY
AFAL-TR-78-134

F33615-77-C-1158

N/L

1 OF 2
ADA
066241



AD A066241

LEVEL *IV*

2

F/G 9/2

AFAL-TR-78-134

MULTIMODE CPU DESIGN STUDY

Data Systems Division
Litton Systems, Inc.
8000 Woodley Avenue
Van Nuys, California 91409

JUNE 1978

TECHNICAL REPORT AFAL-TR-78-134

Final Report for Period MAY 1977 - MAY 1978

Approved for public release; distribution unlimited.

Prepared for

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AFB, OHIO 45433

DDDC
RECEIVED
MAR 21 1978
C-1

DDC FILE COPY

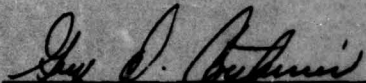
79 08 21 001

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever, and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

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

This technical report has been reviewed and is approved for publication.

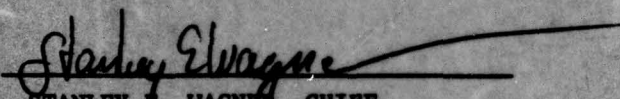


GUY D. COUTURIER, PROJECT ENGINEER



DR. RONALD A. BELT, ACTING CHIEF
PROCESSOR TECHNOLOGY GROUP

FOR THE COMMANDER



STANLEY E. WAGNER, CHIEF
MICROELECTRONICS BRANCH
ELECTRONIC TECHNOLOGY DIVISION

"If your address has changed, if you wish to be removed from our mailing list, or if the addressee is no longer employed by your organization please notify AFAL/DHE-1, W-PAFB, OH 45433 to help us maintain a current mailing list."

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER 18 AFAL TR-78-134	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 MULTIMODE CPU DESIGN STUDY.	5. TYPE OF REPORT & PERIOD COVERED 9 Final Report, May 1977 to May 1978	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) 10 Gary L. Mallaley, Dr. Alfred Ess Alexander Mihich	8. CONTRACT OR GRANT NUMBER(s) 15 F33615-77-C-1158	
9. PERFORMING ORGANIZATION NAME AND ADDRESS Data Systems Division Litton Systems, Inc. 8000 Woodley Ave., Van Nuys, CA 91409	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 16 Exploratory Development Program 6096 Task 31 Work Unit 02	
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory Air Force Systems Command Wright-Patterson AFB, Ohio 45433	12. REPORT DATE 11 June 1978	13. NUMBER OF PAGES 171
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) 12 172p.	15. SECURITY CLASS. (of this report) Unclassified	15a. DECLASSIFICATION DOWNGRADING SCHEDULE
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)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Microprocessor Cosine Transform Signal Processor Electronic Warfare FFT Array Processing Coordinate Conversion Multiplier Structures CFAR		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A study was conducted to define a Multimode CPU architecture in which the CPU could handle instruction addressing, data addressing, and data processing. A problem set of signal processing tasks was defined from which the architectural design evolved. The multiplier/FFT interaction was identified as a major architectural constraint. A RALU structure was defined to perform the data addressing and data processing. A microsequence structure was designed to perform the instruction addressing. Two complex-data signal processors were defined and designed at the register level. These processors, along with the Raytheon Micro Signal Processor and the Tracor/RCA General Processing Unit were addressed on their ability to perform benchmarks from the initial problem set.		

2204E

209390

Handwritten signature and arrow pointing to the top right.

FOREWORD

This report was prepared by the Data Systems Division of Litton Systems, Inc., Van Nuys, California under USAF contract F33615-77-C-1158. This work was administered under the direction of the Air Force Avionics Laboratory, Mr. Guy D. Couturier, Project Engineer and Capt. Thomas Margraff.

This final report covers work conducted during the period May 1977 to May 1978 and was submitted by Data Systems Division in June 1978.

Contributions to this report were made by Mr. Gary L. Mallaley, Dr. Alfred Ess and Mr. Alexander Mihich.

Publication of this report does not constitute Air Force approval of the reports' findings or conclusions. It is published only for the exchange and stimulation of ideas.

ACCESSION for	to Section <input checked="" type="checkbox"/>
NTIS	to Section <input type="checkbox"/>
ADP	to Section <input type="checkbox"/>
DISTRIBUTION AND SECURITY CODES	
BY	SPECIAL
A	

TABLE OF CONTENTS

Section		Page
I	INTRODUCTION AND SUMMARY	1
II	GENERAL SIGNAL PROCESSING PROBLEM	7
	2.0 INTRODUCTION	7
	2.1 MODELLING SIGNAL PROCESSING PROBLEMS	9
	2.2 BENCHMARKS	12
	2.2.1 Fast Fourier and Weighted Fast Fourier Transform Benchmarks	12
	2.2.2 Coordinate Conversion Benchmark Definition	15
	2.2.3 Constant False Alarm Rate (CFAR) Benchmark Definition	18
	2.2.4 Cosine Transform Benchmark	20
	2.3 ELECTRONIC WARFARE PROBLEM	23
	2.3.1 Basics on Radar Characteristics	23
	2.3.1.1 Single Pulse Characteristics	23
	2.3.1.2 Multiple Pulse Characteristics	24
	2.3.2 Necessary EW Functions	24
	2.3.3 Channelized Front End	25
	2.3.4 System Architecture	25
	2.3.4.1 System Flow	25
	2.3.4.2 Processing Flow	26
	2.3.5 Architectural Necessities	28
	2.3.6 Benchmarks	29
	2.3.6.1 Representation of the Received Signal	29
	2.3.6.2 Mean and Variance Determination	30
	2.3.6.3 Pulse Classification Algorithm	31
	2.3.6.4 PRF Sorting	34
	2.3.6.5 References	37
III	MULTIMODE CPU ARCHITECTURE	38
	3.0 INTRODUCTION	38
	3.1 PRIMITIVE COMPUTING STRUCTURE	38
	3.2 BUS SPEED, WIDTH, EFFICIENCY	39
	3.2.1 Bus Speed	39
	3.2.2 Bus Width	41
	3.2.3 Bus Efficiency	42
	3.3 MULTIPLIER STRUCTURES	42
	3.3.1 FFT Butterfly	42
	3.3.2 Multiplier/FFT Structures	42
	3.3.3 System Impact of Multiplier/FFT Structures	44
	3.3.3.1 Case 1: Multiplier	44
	3.3.3.2 Case 2: Multiply/Accumulator	47
	3.3.3.3 Case 3: Multiplier/Accumulator with Holding Registers	48
	3.3.3.4 Case 4: Multiplier/FFT	48
	3.4 COMPLEX PROCESSOR	49
	3.4.1 Processor 1	49
	3.4.1.1 Data Processors	49
	3.4.1.2 Data Addresser	49

TABLE OF CONTENTS – Continued

Section		Page
	3.4.1.3 Data Memories	49
	3.4.1.4 Multiplier	49
	3.4.1.5 Control	51
3.4.2	Processor II	52
	3.4.2.1 Data Processor/Addresser (DP/DA)	52
	3.4.2.2 Data Memory	52
	3.4.2.3 Multiplier/FFT	54
	3.4.2.4 Control	54
3.4.3	Complex Processor Performance	54
3.5	DATA PROCESSOR/DATA ADDRESSER	55
3.5.1	Design Rationale	55
	3.5.1.1 Registers	55
	3.5.1.2 Arithmetic/Logic Unit	55
	3.5.1.3 RALU Structure	56
	3.5.1.4 Additional Comments	56
3.5.2	Two DP/DA Structures	56
3.6	INSTRUCTION ADDRESSING	59
3.6.1	Program Control Unit	59
3.6.2	Interrupt Control Unit	59
3.6.3	Flag Logic	61
3.6.4	Loop Counter	61
3.7	ARRAY PROCESSING	61
3.7.1	Array Processor Element	62
3.7.2	Parallel Array Multiprocessor	63
IV	LSI TECHNOLOGY SUMMARY	66
4.0	INTRODUCTION	66
4.1	TECHNOLOGY SURVEY	66
4.1.1	Custom LSI	66
	4.1.1.1 SiGate MNOS	66
	4.1.1.2 N-Channel Depletion-Enhancement Mode SOS-MOSFET	67
	4.1.1.3 VMOS	67
	4.1.1.4 DMOS	67
	4.1.1.5 C ² L/MOS	67
	4.1.1.6 CMOS/SOS	67
	4.1.1.7 First and Second Generation I ² L/MTL	67
	4.1.1.8 S ² L	68
	4.1.1.9 SFL	68
	4.1.1.10 SCHOTTKY I ² L	68
	4.1.1.11 Up-Diffused I ² L	68
	4.1.1.12 I ³ L	68
	4.1.1.13 Table 6 Description	69
4.1.2	LSI/VLSI Memories	71
4.1.3	Figure of Merit	72

TABLE OF CONTENTS – Continued

Section		Page
4.2	MACROCONSTRAINTS	74
4.3	THE TECHNOLOGY DECISION	77
4.3.1	System Requirement Categories	77
	4.3.1.1 Architecture	77
	4.3.1.2 Environment	77
	4.3.1.3 Physical Characteristics	78
	4.3.1.4 Viability – Reliability, Availability, Maintainability and Survivability	78
	4.3.1.5 Cost	78
	4.3.2 Forced-Pair Comparison	79
	4.4 REFERENCES	82
V	LSI DESIGN AND DEVELOPMENT	83
5.0	INTRODUCTION	83
5.1	REGISTER-LEVEL DESIGN DISCUSSION	83
	5.1.1 The Multiport RAM (MPR)	83
	5.1.2 The Arithmetic Logic Unit	86
	5.1.3 The Bidirectional I/O Data Port	86
	5.1.4 Miscellaneous Functions	87
5.2	GATE ESTIMATES FOR THE RALU's	88
5.3	LSI DEVELOPMENT APPROACHES	88
	5.3.1 Chip Size	88
	5.3.2 Power	91
	5.3.3 Fundamental Speed	92
	5.3.4 Availability	92
5.4	IA CHIP	93
VI	SIGNAL PROCESSOR COMPARISON	95
6.0	INTRODUCTION	95
6.1	DEFINITIONS	95
6.2	MACRO COMPUTER	98
	6.2.1 Tracor/RCA Macro Computer	98
	6.2.2 Raytheon Macro Computer	98
	6.2.3 Litton Macro Computer	101
6.3	BUILDING BLOCKS	101
6.4	CENTRAL PROCESSING UNIT (CPU)	102
	6.4.1 Tracor GPU	102
	6.4.2 Raytheon Arithmetic	102
	6.4.3 Litton Multimode Central Processing Unit (MMCPU)	102
6.5	CONTROLLER OR SEQUENCER	107
	6.5.1 Tracor/MC2909 Controller	107
	6.5.2 Raytheon Controller	107
	6.5.3 Litton Controller	107
6.6	MICROCOMPUTER	107
	6.6.1 Tracor/MC2901 Microcomputer	110
	6.6.2 Raytheon Microcomputer	112
	6.6.3 Litton Microcomputer	112

TABLE OF CONTENTS -- Continued

Section		Page
6.7	BENCHMARK COMPARISON	116
	6.7.1 Algorithms	116
	6.7.2 Instructions and Timing	117
	6.7.3 Tracor and Litton Microinstructions	118
	6.7.4 Raytheon Timing	119
	6.7.5 Fast Fourier Transformation (FFT)	123
	6.7.6 Coordinate Conversion	124
	6.7.7 Constant False Alarm Rate (CFAR)	125
	6.7.8 Microprocessor Cycles for Benchmarks	125
6.8	COMPARISON OF RESULTS	126
	6.8.1 Architecture Comparison	126
	6.8.2 Timing Comparison	127
	6.8.3 Conclusions	128
6.9	REFERENCES	129
VII	CONCLUSIONS AND RECOMMENDATIONS	130
	7.0 INTRODUCTION	130
	7.1 CONCLUSIONS	130
	7.1.1 Benchmarks	130
	7.1.2 The Architecture	131
	7.1.3 The Architectural Comparison	131
	7.1.4 The Technology and the MMCPU	132
	7.2 RECOMMENDATIONS	133
	7.2.1 Electronic Warfare	133
	7.2.2 Array Processing	133
	7.2.3 Demonstration of MMCPU	133
	7.2.4 VMOS Technology	133
Appendix		
A	TIMING BENCHMARKS FOR COMPLEX PROCESSORS	134
B	BENCHMARKS -- CODING	140
C	BENCHMARKS -- TIMING	153

LIST OF ILLUSTRATIONS

Figure		Page
1	Processor 1	2
2	Processor 2	3
3	DP/DA for Processor 1	4
4	DP/DA for Processor 2	5
5	IA Chip	6
6	A 32 Point Complex Decimation-in-Time Fast Fourier Transform	14
7	CFAR Block Diagram	19
8	32 Point FFT Modified for Computing Cosine Transform	22
9	Comparison Properties	36
10	Primitive Structure	40
11	Minimum Time Path for Address to Register	41
12	Complex Butterfly	43
13	Hardware Implementation of FFT Butterfly	43
14	Multiplier with Accumulator and Holding Registers	45
15	Complex Processor	46
16	Processor 1	50
17	Complex Processor Control Unit	52
18	Processor II	53
19	Processor I RALU	57
20	Processor II RALU	58
21	Instruction Addressing Function	60
22	Parallel Array Multiprocessor	63
23	Control Detail	64
24	LSI Technology, vs Figure of Merit	73
25	Forced-Pair Comparison Chart	80
26	Forced-Pair Comparison Example	81
27	RALU (DP/DA) for Processor 1	84
28	RALU (DP/DA) for Processor 2	85
29	Bidirectional I/O Data Port	86
30	16-Bit RALU Gate Estimates	89
31	8-Bit RALU Gate Estimates	90
32	Controller Without Microsequencer	94
33	Definition Block Diagram (Simplified)	97
34	Advanced Micro Device Microcomputer	99
35	Raytheon Macro Computer	100
36	Litton Macro Computer	101
37	Tracor GPU Block Diagram	103
38	Advanced Micro Device Microprocessor Slice	104
39	Raytheon Arithmetic	105
40	Litton Multimode CPU (Used for DP/DA)	106
41	Advanced Microdevice Controller	108
42	Raytheon Controller	109
43	Raytheon Controller Detail	110
44	Litton Controller	111
45	Microprogrammed Architecture Around MC2901's	112
46	Microcomputer with Tracor GPU	113
47	Raytheon Microcomputer	114

LIST OF ILLUSTRATIONS – Continued

Figure		Page
48	Litton Microcomputer	115
49	Microinstruction Symbols (Tracor/Litton)	118
50	Microinstruction Types (Tracor/Litton)	120
51	Microinstruction Timing (Tracor/Litton)	121
52	Raytheon Timing	122
53	Microprocessor Cycles for Benchmarks	123
54	Microprocessor Cycles for Benchmarks	127
55	32 Point FFT Module for Computing Core Transform	8
56	Companion Processor	9
57	Primitive Structure	10
58	Minimum Time Path for Address to Register	11
59	Complex Butterfly	12
60	Hardware Implementation of FFT Butterfly	13
61	Multiplex with Accumulator and Holding Register	14
62	Complex Processor	15
63	Processor I	16
64	Complex Processor Control Unit	17
65	Processor II	18
66	Processor I RALU	19
67	Processor II RALU	20
68	Instruction Addressing Function	21
69	Parallel Array Multiprocessor	22
70	Control Detail	23
71	LSI Technology vs. Family of Modules	24
72	Forward Path Comparison Circuit	25
73	Forward Path Comparison Example	26
74	RALU (RDA) for Processor I	27
75	RALU (RDA) for Processor II	28
76	Bidirectional Data Port	29
77	16 Bit RALU Gate Structures	30
78	8 Bit RALU Gate Structures	31
79	Control Without Microprocessor	32
80	Partitioned Block Diagram (Implementation)	33
81	Advanced Micro Device Microprocessor Time	34
82	Raytheon Micro Computer	35
83	Litton Micro Computer	36
84	Tracor CPU Block Diagram	37
85	Advanced Micro Device Microprocessor Time	38
86	Raytheon Architecture	39
87	Litton Multichip CPU (Used for DADA)	40
88	Advanced Micro Device Controller	41
89	Raytheon Controller	42
90	Raytheon Controller Detail	43
91	Litton Controller	44
92	Microprocessor Architecture (Advanced Micro Device)	45
93	Microcomputer with Tracor CPU	46
94	Raytheon Microcomputer	47

LIST OF TABLES

Table		Page
1	Important Facets of Signal Processing	10
2	Mapping of Sine and Cosine to One Quadrant of the Sine Function	16
3	Classification of Radar Waveforms	27
4	Benchmarks and Indicated Architectural Characteristics	39
5	Bus Activity as a Function of the Multiplier/FFT Structure	47
6	Technology Survey	70
7	Memory Device and Performance	71
8	Comparison of LSI Candidates	76
9	ALU Operations and Status Flags	87
10	Technology Analysis	91

SECTION I

INTRODUCTION AND SUMMARY

The Multimode CPU Design Study was undertaken by the Air Force and Litton Data Systems Division to define a multimode CPU architecture, to assess the microprocessor design (IC process interdependencies intrinsic to the processor approaches), to establish a detailed chip design (at the register level) for an advanced 8-bit, bit-sliced processor element, and to assess the multimode chip design set and existing CPUs for their ability to perform processing tasks.

The problem set, defined in Section II, is a representative set of signal processing tasks required through a major portion of the 1980's. The set was used as the benchmarks by which the Multimode CPU (MMCPU) design could be bounded.

The architectural design was attempted initially without the constraints of LSI technology. The results are presented in Section III. The design process started from the bus system (data and data addressing) and work out to a CPU architecture. Because the FFT represents the most difficult of the problems in the set, the impact of the multiplier/FFT special function structure was investigated and two processor structures were presented (see Figures 1 and 2). The various blocks of the processor were analyzed and two register arithmetic logic unit (RALU) structures were defined (Figures 3 and 4). Each RALU is designed to perform both the data processing (DP) and data addressing (DA) functions. An instruction addressing and microcontrol structure was defined. Figure 5 is the instruction addresser without its microinstruction memory.

Before the feasibility of the architecture as an LSI candidate could be tested, the state-of-the-art was assessed. The results are presented in Section IV. The gate level design of the RALU structures are presented in Section V, concluding that an 8-bit, bit-sliced RALU for the DP/DA functions is feasible.

Three microcomputer architectures, one based on the Tracor/RCA GPU, one based on the Litton DP/DA RALU, and one embodied in the Raytheon Micro Signal Processor, were assessed on their ability to perform the benchmarks from the problem set discussed in Section II. The methodology, initial assumptions, self-imposed constraints, and results and conclusions are presented in Section VI.

In summary, the attempt to design a single large scale integrated circuit, the MMCPU, has revealed some interesting insights into the signal processing environment, LSI technology, and processor design. Analysis showed that the main functions in Section III are all within the reach of current LSI technology, but two chip types will be necessary to accomplish the total function of the Multimode CPU.

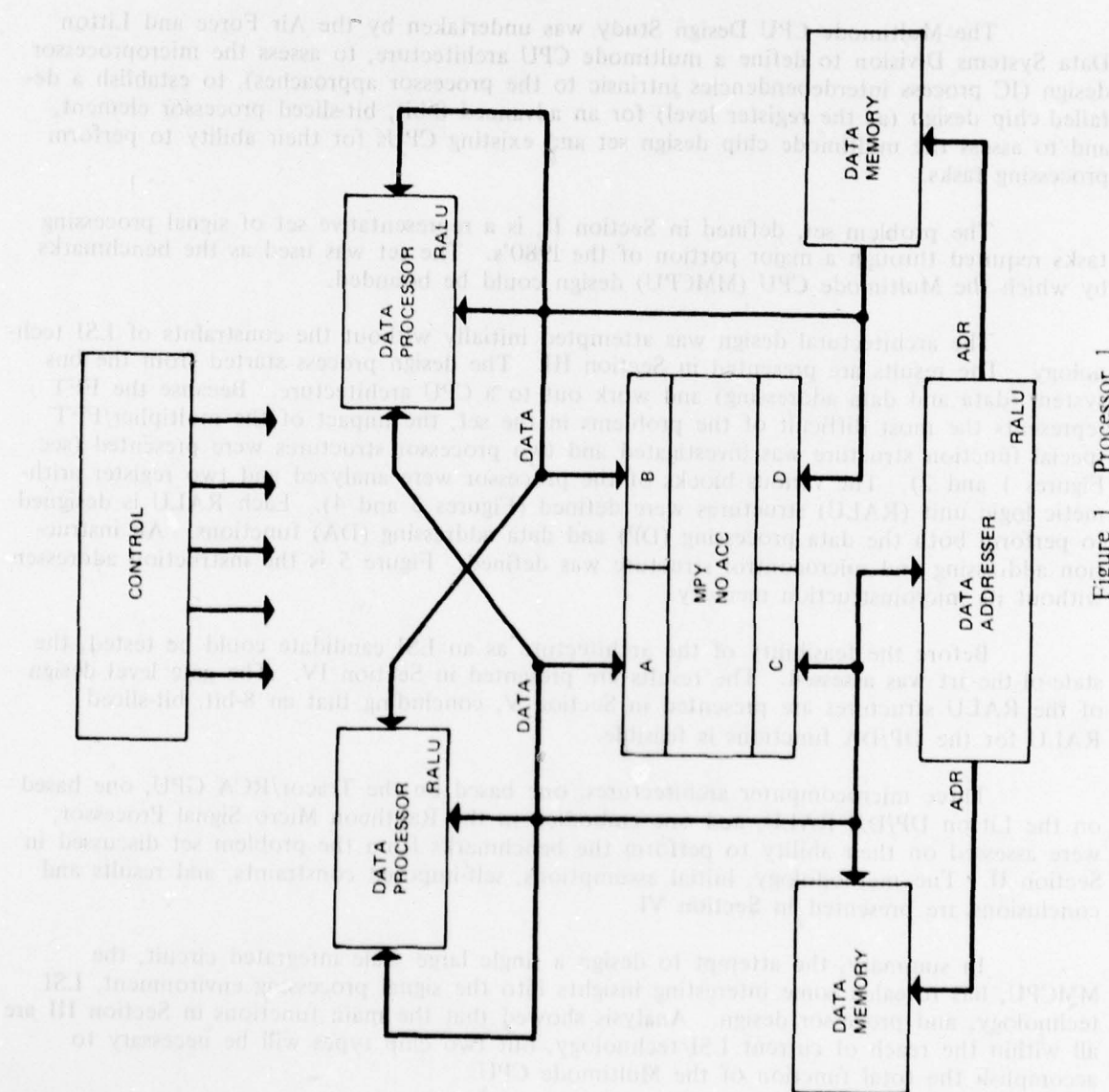
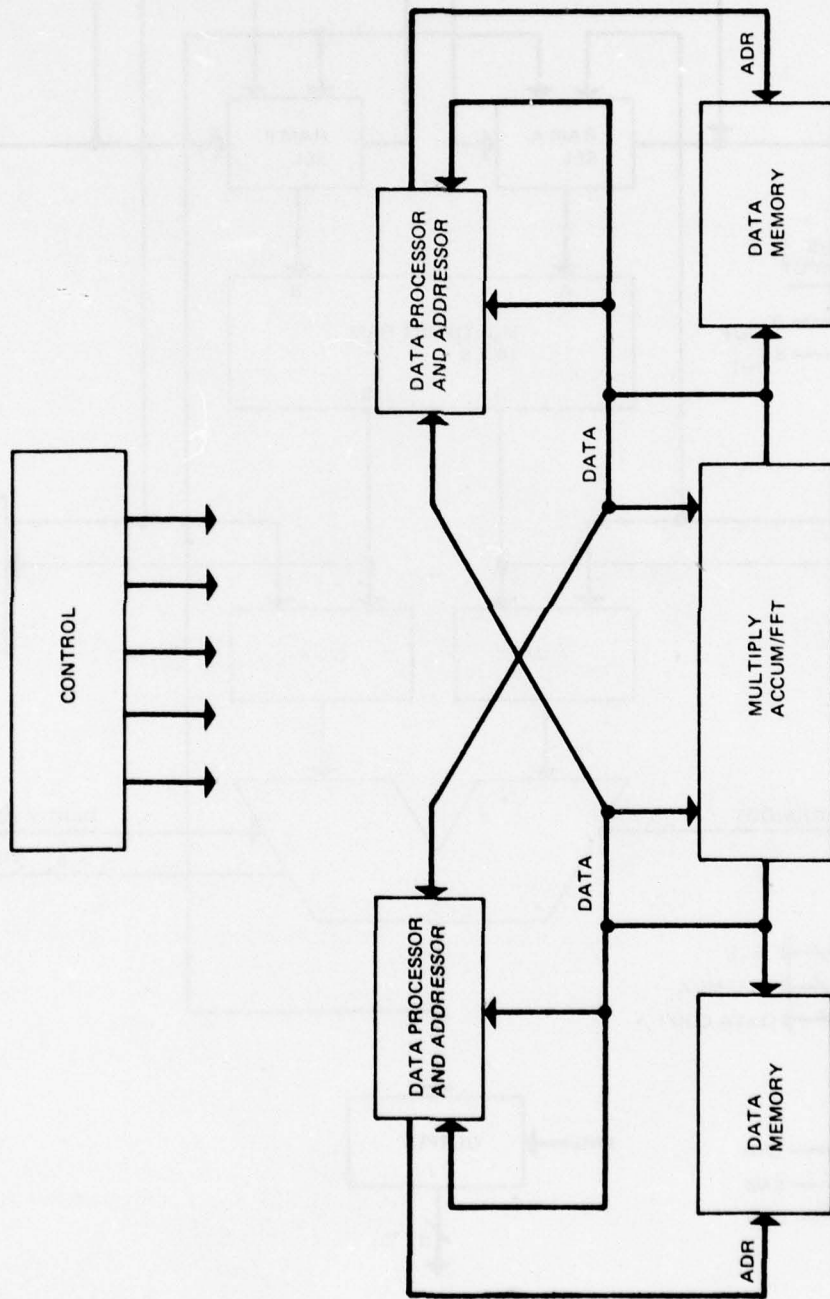


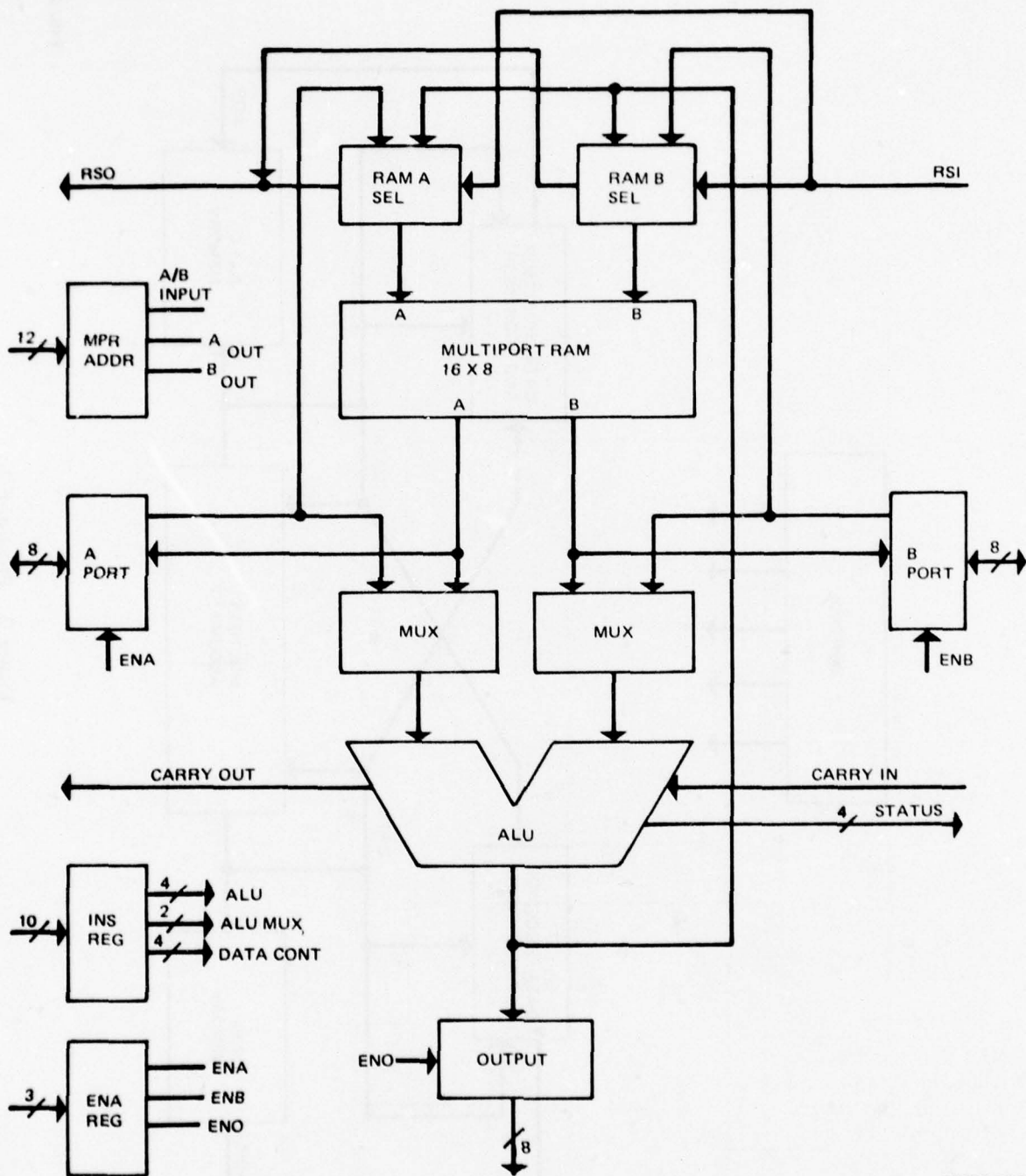
Figure 1. Processor 1

78453-21



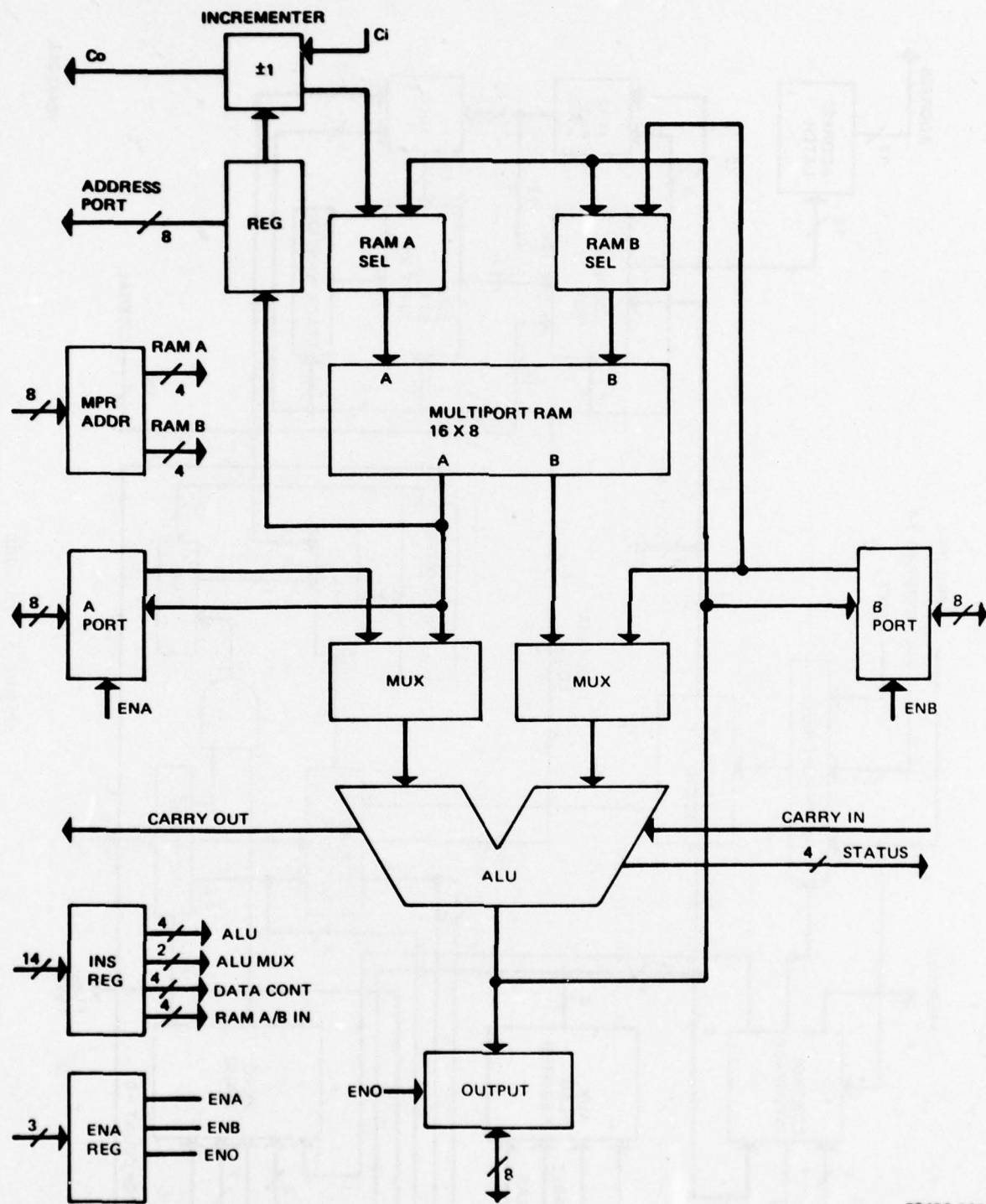
78453-22

Figure 2. Processor 2



78453-23A

Figure 3. DP/DA for Processor 1



78453-24A

Figure 4. DP/DA for Processor 2

SECTION II

GENERAL SIGNAL PROCESSING PROBLEM

2.0 INTRODUCTION

The objective of this study is to define and do a top level design of LSI circuitry that will have significant impact upon the capabilities, costs, environmental factors and performance of future military systems that must deal with the information content of analog waveforms (signals) to perform their assigned tasks. The systems or portions of systems directly addressed in this program consist of those techniques, in both mathematics and implementation, used to transform the signal information content into a form suitable for a known user (whether the user is a human, an automated tracking system, etc., depends upon the actual system being implemented).

The problem will be further bounded by assuming that the required signal processing will be performed using digital circuitry and that recommended circuits should show the promise of spanning a wide range of applications. These circuits are intended for use in the construction of systems from 1980 through approximately 1990, most of which have not as yet been conceived. An initial effort has, therefore, been expended in trying to predict the direction of applications of digital signal processing to military problems for the next 10 to 15 years. It was recognized that this program could do more than aid these applications but, if properly executed, would speed and alter the course of new applications. Care was exercised during the study of present and "drawing board" systems for use as a basis of predicting future system requirements. The actual system goals were studied rather than specific implementations that exist as approximations to desired systems. The desired systems often cannot be produced cost effectively using today's circuits.

The field of applications to be addressed can be made clearer by first considering the nature of signal processing itself. Signal processing problems are typified by:

- a. Analog signal or signals to be processed (in this case digitally, thus requiring A/D converters).
- b. An uncooperative (noisy) environment which corrupts the desirable signals.
- c. Low information-rate-to-data-rate ratio permitting averaging for signal-to-noise ratio improvement.

Due to the uncooperative environment and low information-rate-to-data-rate ratio, the incoming signal can be, and most often is, converted to what mathematicians call "sufficient statistics". The idea is to transform the large amount (and often highly redundant) incoming data into a relatively small amount of data which contains all or, in practice, almost all of the information content of the initial signal. Once this transformation is performed subsequent processing and memory requirements simplify because less data must be handled at each processing step.

Signal processing tasks can be separated into high speed and low speed processing requirements because of the sufficient statistic concept where high speed and low speed are relative to input sampling rate for a specific problem and are not absolute. For example, the high speed processing of a sonar problem may be slower than the low speed processing of a radar problem in terms of actual hardware requirements. The dependence of processing speed on sampling rate

and the high and low speed processing requirements points out the fact that for many signal processing jobs the total job could be done using a programmable CPU approach but, as sampling rates increase, a point will be reached where high speed processing will have to be out-boarded and, for high rates, pipelined for maximum throughput. The low speed requirements could almost always be performed in a programmable CPU, however.

Characterization of the signal processing problems in the 1980-1990 time frame can be accomplished by considering the basic nature of such problems and the analytical techniques being used to address the problems. The objective is always one of determining and suitably formatting the information content of a signal which has been corrupted by an uncooperative environment. Due to the large number of variables in this type of problem (number of samples, system states, etc.) problems are cast in the form of matrix equations. This method of analysis permits a certain ease of manipulation of large numbers of variables and/or equations. As a direct result, the first statement of the problem solution is in the form of a matrix equation or, more typically, the equations for computing sufficient statistics are matrix equations.

The implementation problem can then be viewed as a reduction of these matrix equations to the point where they can be implemented by existing hardware. In the past this has required reducing all equations to Boolean operations because design was done at the gate level. As levels of integration increased, the ALU became a readily available part so that algorithms needed only to be reduced to adds, subtracts and logical operations.

The very common operation of real multiplication has recently been attacked in an attempt to reduce it to cost-effective hardware and it seems reasonable to expect that the divide problem will also become available as a hardware component. Thus, we see a common approach by commercial semiconductors to ease the manipulation of real scalar quantities in numerical calculations.

In order to increase the hardware/algorithm boundary further so that less effort will be required to implement signal processing algorithms, hardware needs to address the operations involved in complex vector and matrix mathematics. It is unlikely that in the near future single chips will perform such functions as a vector multiply, but rather a CPU that has a multiplier under its control could be organized so that vector operations become easy to program and are efficiently implemented.

This philosophy indicates that the direction of thought toward defining an ideal micro-signal processing chip set be such that the chip set should:

- a. Provide a hardware complex multiply.
- b. Control the multiply and memory so that matrix manipulations are extremely efficient.
- c. Simplify the programmers' task for performing matrix calculations.
- d. Not compromise the ease of performing scalar arithmetic and logical operations.
- e. Be capable of handling large I/O data rates.

2.1 MODELLING SIGNAL PROCESSING PROBLEMS

All possible signal processing problems cannot be considered during the design of a versatile signal processor and a chip set. Instead, the problem must be modelled by a small, manageable set of problems, representative of the baseline scenario from which general computational requirements of all signal processing problems can be derived. With the concurrence of the technical staff of the Processor Technology group (AFAL/DHE-1) of the Air Force Avionics Laboratory, Litton Data Systems used the benchmarks, discussed in this chapter, as a good representative set of problems for use in the design of a Multimode CPU chip set. The following paragraphs briefly discuss each benchmark and the characteristics of the signal processing indicated by the benchmark.

The first benchmark is a complex 1024 point FFT. In addition to being the most common signal processing benchmark in the entire signal processing industry for comparing signal processing equipment, this problem illustrates the first four facets of signal processing listed in Table 1.

The popularity of the Fourier Transform over other transforms involving orthogonal basis functions is by no means accidental. It is a direct result of the fact that the Fourier basis functions (sine and cosine) are the eigenfunctions of all linear systems and, therefore, are the only functions which will preserve their functional form (except for parameter changes) from input to output of any linear system. For example, if $A \cos(\omega t)$ is used as input to any linear system, the response will be of the form $B \cos(\omega t + \phi)$ and no additional frequencies (basis functions) will be produced (this cannot be claimed for Walsh or other orthogonal function representations). Since all systems are modelled as linear whenever possible due to the enormous gain in mathematical simplicity, it is assumed that Fourier Transforms will continue as the most popular transform technique and, as implementation becomes less costly, their use will grow considerably.

The basic operation performed during a Discrete Fourier Transform is the multiplication of a vector times a matrix. If this operation could be solved quickly and efficiently by, say, a matrix multiply chip there would no longer be any interest in the collection of algorithms known generally as the Fast Fourier Transform. However, semiconductor technology will not solve the matrix problem in the time frame under consideration so that taking advantage of the cyclic properties of the Transform matrix (FFT) will continue as one of the most important signal processing computational problems.

The second benchmark is a modification to the FFT by the application of a windowing function to the data to reduce the side lobe effects inherent in the FFT algorithm. This operation, if performed in the time domain, is an example of a high speed function product common in modulation/demodulation processes and digital filtering via the FFT. If performed in the frequency domain this algorithm is an example of a high speed convolution common in finite impulse response digital filtering. In either case, this benchmark also illustrates the facets (1) through (4) listed in Table 1.

Table 1. Important Facets of Signal Processing

Benchmark	Facets of Signal Processing
1 and 2	<ol style="list-style-type: none"> 1. High speed calculation using complex arithmetic on data arrays. 2. With the exception of numerical scaling, the high speed algorithm is independent of the input data. 3. Array indexing is orderly although not always simple. 4. Multiply/add is a common arithmetic pair of operations.
3	<ol style="list-style-type: none"> 5. Tight data dependent loops. 6. Double precision during less time-critical processing. 7. Numerical scaling. 8. Data dependent jumps.
4	<ol style="list-style-type: none"> 9. Sliding window data manipulation. 10. Averaging (integration). 11. Data dependent decisions. 12. Bit manipulation.
5	<ol style="list-style-type: none"> 13. High I/O rates between memory and the outside world. 14. High speed calculation on small input data blocks. 15. Repetitive use of a very short program.
6	<ol style="list-style-type: none"> 16. Fast/tandem address generation. 17. Data dependent address generation. 18. Efficient memory organization. 19. Data comparisons.

The third benchmark, coordinate conversion, involves the computation of several common functions such as sine, cosine, divide, square root, etc. These functions are common to a wide range of signal processing problems particularly modulation, demodulation, detection, averaging and standard deviation estimation. Demonstrating the ability to handle these functions will indicate the ability to handle a variety of other functions such as logarithm, anti-logarithm, error function, Marcum's Q function, etc., in a similar manner. The double precision requirement is indicative of the fact that these functions are often required to great precision and that, under the same circumstance, calculation speed is generally less critical so it would make sense to use double precision programming rather than a machine with a larger word size. In addition, calculation of these functions often involves iterative loops of a form that requires results of a calculation before the next calculation can be performed. This problem illustrates the signal processing characteristics numbers 5, 6, 7 and 8 as listed in Table 1.

The fourth benchmark is an example of Constant False Alarm Rate (CFAR) detection commonly used to improve performance of radars particularly when operating in a high clutter environment. This benchmark illustrates facets 9, 10, 11 and 12 of Table 1.

The sliding window is an important aspect of signal processing wherein an algorithm is applied to a set of data points (in this case averaging and a threshold based decision) and then a new data point is introduced to the set while the oldest point is deleted and the algorithm is repeated. Conceptually the same process is involved in finite impulse response (transversal) filters, convolution, generation of algebraic codes, etc. In this benchmark the output is a series of 0/1 decisions which, for memory and communication efficiency should be packed into computer words (e.g., 16 decisions in a 16 bit word).

The fifth benchmark is the use of a Cosine Transform as found in the front end processing of an image bandwidth compression problem. The basic algorithm is an FFT and in that sense is similar to benchmark 1. This problem differs, however, in that a high input data rate is required while the transform itself is only 32 points. The problem is, therefore, one of performing a short, fast operation including fast I/O thus straining processor I/O, memory address and store capabilities and calculating power. The facets of signal processing illustrated by this benchmark are 13, 14 and 15.

The sixth benchmark is a pulse classification algorithm characteristic of general pattern recognition problems but specifically oriented toward signal sorting for electronic warfare. To accomplish this benchmark, the signal processor must have a highly flexible memory organization with efficient data memory control, sophisticated data address generation for subsequent data processing, data dependent, data address generation and conditional jump/branch capability.

Because the specification of signal sorting algorithms is fairly incomplete in the literature, Litton Data Systems was asked to help extend the specification for the Processor Technology group (AFAL/DHE-1). Included in the succeeding sections is a detailed discussion of the signal sorting problem and the general EW problem.

The use of a specific benchmark set has provided greater insight into the general properties that a micro-signal processor chip set should possess. The benchmarks have been shown to generally represent the totality of signal processing problems and it appears certain that a machine architecture that can provide features 1 through 9 of Table 1 will be applicable to real world signal processing problems.

2.2 BENCHMARKS

2.2.1 Fast Fourier and Weighted Fast Fourier Transform Benchmarks

The Discrete Fourier Transform Equation for an N point complex sequence $f(n)$ is defined as follows:

$$F(k) = \sum_{n=0}^{N-1} f(n) W_N^{kn} \quad k = 0, 1, \dots, N-1 \quad (1)$$

where

$$W_N = e^{-j2\pi/N}$$

A decimation-in-time Fast Fourier Transform can be readily derived from equation (1) by defining two $(N/2)$ point sequences as the even and odd members of $f(n)$. Because of the highly cyclic nature of W_N , it can be shown that Equation (1) can be computed by first computing

$$F_1(k) = \sum_{n=0}^{\frac{N}{2}-1} f(2n) W_{N/2}^{nk} \quad (2)$$

and

$$F_2(k) = \sum_{n=0}^{\frac{N}{2}-1} f(2n+1) W_{N/2}^{nk} \quad (3)$$

and then combining Equations (2) and (3) to achieve the result of Equation (1) as follows

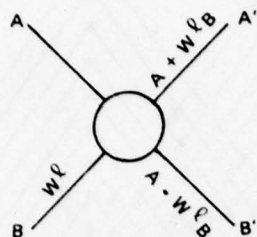
$$F(k) = F_1(k) + W_N^k F_2(k) \quad (4)$$

This process can be continued with a significant computational advantage gained at each step.

The basic operation resulting from Equation (4) is called the Decimation-in-Time butterfly defined as

$$\begin{aligned} A' &= A + W^{\ell} B \\ B' &= A - W^{\ell} B \end{aligned} \quad (5)$$

This operation can be visualized in flow graph form as follows:



The quantity W^p is commonly called a rotation vector or 'twiddle factor'. A combination of butterflies arranged to produce a 32 point complex FFT is shown in Figure 6. This structure results from the repetitive application of Equations (2), (3) and (4) and the butterfly operation Equation (5). The same process can be used to define a 1024 point FFT and it is this algorithm that has been chosen as the FFT benchmark. It should be noted that the output values are unordered and that reordering is a necessary step to be included in the benchmark problem.

The algorithm will assume 14-bit input data which will be sufficient for almost all signal processing problems. The nature of the algorithm, however, is such that numerical values tend to increase through a butterfly operation. The largest data value out of a butterfly will be at least as large but no greater than twice as large as the largest data value into the butterfly. Overflows can be prevented by dividing by two at the output of every butterfly, but this results in many unnecessary underflows. The scaling scheme employed involves keeping all data values at 14 bits and, at the end of each group or course, checking to see if any data value has overflowed into the 15th bit. Whenever this occurs all input data to the next course will be divided by two. This scheme insures no actual overflows while maintaining as much precision as possible in the final results.

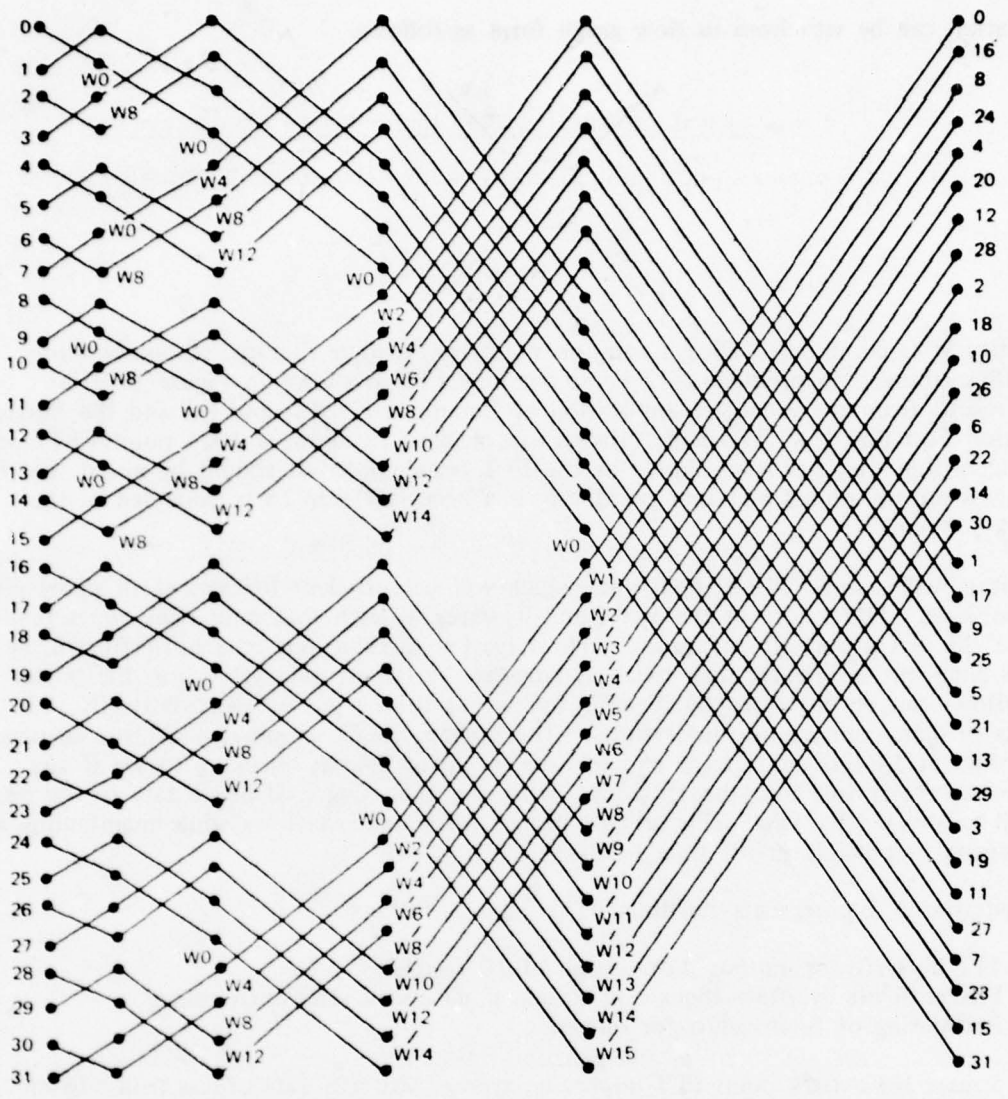
The computational requirements can then be defined as follows:

- a. 512 butterfly operations during each of 10 courses
- b. Fifteenth bit overflow check and scaling if necessary after each course
- c. Reordering of final results for output.

A rate of 5 msec for a 1024 point FFT implies an average butterfly rate of less than 976 nsec, but time must be allotted for processor dependent data loading, loop set-up and control and output reordering. This time, thus, represents an absolute upper bound to the actual butterfly time which will depend upon processor architecture.

In a similar manner, an absolute upper bound on butterfly time for a rate of 0.5 msec for a 1024 point FFT is 98 nsec with this number decreasing as a function of processor architecture.

The additional benchmark of windowing the FFT data will be performed in the time domain and consists of a premultiplication of all input data values by a window function. The technique adopted is to store the window function in memory thus requiring the processor to perform 1024 complex multiplies as well as associated fetches and stores. This technique permits the arbitrary selection of any window function at no additional computational cost since the function will be prestored in memory.



77357-221A

Figure 6. A 32 Point Complex Decimation-in-Time Fast Fourier Transform

2.2.2 Coordinate Conversion Benchmark Definition

The coordinate conversion benchmark of polar-to-rectangular and rectangular-to-polar in single and double precision demonstrates the fact that functions generated from input variables are often required in signal processing. Normally they occur in post-processing applications where relatively low speed calculations are required. Therefore, the use of double precision programming is preferred when increased accuracy is required rather than implementing a larger word-size machine.

The polar-to-rectangular conversion problem is defined as follows:

Given a point specified by magnitude R and angle θ , determine the rectangular coordinates X and Y where

$$X = R \cos \theta$$

and

$$Y = R \sin \theta \quad (6)$$

The problem is basically one of computing sine and cosine functions and can be solved using a nested polynomial approach to the Taylor series for either function over $\pi/2$ and, by symmetry, determining the functional value.

First assume that $\sin \theta$ can be computed for $0 \leq \theta \leq \pi/2$. Then θ can be mapped to θ as a function of the quadrant of θ and the function (sine or cosine) desired as shown in Table 2. Therefore, any sine or cosine value can be computed using a Taylor series expansion for $\sin \theta$ in the range $0 \leq \theta \leq \pi/2$. The expansion is

$$\sin \theta = \sum_{j=0}^{\infty} (-1)^j \frac{\theta^{(2j+1)}}{(2j+1)!} \quad (7)$$

The expansion must be limited to a finite number of terms, the number used reflecting the desired numerical accuracy. Assume, for example, the first five terms are sufficient for a particular application. Then

$$\sin \theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \frac{\theta^9}{9!} \quad (8)$$

Table 2. Mapping of Sine and Cosine to One Quadrant of the Sine Function

θ	$\cos \theta$	$\sin \theta$
$0 \leq \theta \leq \pi/2$	$\sin [\pi/2 - \theta]$	$\sin [\theta]$
$\pi/2 \leq \theta \leq \pi$	$-\sin (\theta - \pi/2)$	$\sin (\pi - \theta)$
$\pi \leq \theta \leq 3\pi/2$	$-\sin \left(\frac{3\pi}{2} - \theta \right)$	$-\sin (\theta - \pi)$
$3\pi/2 \leq \theta \leq 2\pi$	$\sin \left(\theta - \frac{3\pi}{2} \right)$	$-\sin (2\pi - \theta)$

which can be expressed as a set of nested polynomials of the form

$$\sin \theta = \theta \left[1 + K_1 \theta^2 \left(1 + K_2 \theta^2 \left(1 + K_3 \theta^2 \left(1 + K_4 \theta^2 \right) \right) \right) \right] \quad (9)$$

With a change of definition of constants and defining $\psi = \theta^2$ the expression can be written as

$$\sin \theta = \theta \left[k_1 + \psi (k_2 + \psi (k_3 + \psi (k_4 + \psi))) \right] \quad (10)$$

which can be programmed as an iterative loop as follows

$$\begin{aligned} Z_0 &= k_4 + \psi \\ Z_{n+1} &= k_{3-n} + \psi Z_n \quad n = 0, 1, 2 \end{aligned} \quad (11)$$

Now consider the rectangular-to-polar conversion problem. Given a point specified by X and Y determine the polar coordinates R and θ where

$$\begin{aligned} R &= \sqrt{X^2 + Y^2} \\ \theta &= \begin{cases} \sin^{-1} \left(\frac{|X|}{R} \right) & \text{if } 0 \leq \frac{|X|}{R} \leq \pi/4 \\ \sin^{-1} \left(\frac{\pi/2 - |X|}{R} \right) & \text{if } \pi/4 \leq \frac{|X|}{R} \leq \pi/2 \end{cases} \end{aligned} \quad (12)$$

with appropriate quadrant modifications on θ based upon the signs of X and Y.

There are three function problems here; square root, divide, and arc sine. A Taylor series can be used for the arc sine and implemented exactly as Equation (11) above with a different set of constants, k. The series is

$$\sin^{-1}\theta = \theta + \frac{\theta^3}{6} + \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{\theta^5}{5} + \frac{1}{2} \cdot \frac{3}{4} \cdot \frac{5}{6} \cdot \frac{\theta^7}{7} + \dots \quad (13)$$

It is possible to show, using the Contraction Mapping Theorem, that a divide can also be implemented in the form of Equation (11). The resulting algorithm is as follows

if $Z = \frac{X}{Y}$ and $-1 \leq Y \leq .5$ then the iteration

$$Z_{n+1} = k_X + k_Y Z_n \quad (14)$$

will converge at a rate exceeding two bits per iteration to Z if

$$\left. \begin{array}{l} k_X = -X(Y + 2) \\ k_Y = (Y + 1)^2 \end{array} \right\} \quad (15)$$

and

$$Z_0 = k_X$$

Note that k_X and k_Y remain constant for all iterations. The last required function to complete the problem is the square root. The Contraction Mapping Theorem can again be used to show that, if

$$0.0625 \leq N \leq 0.5625$$

then the recursion

$$X_{n+1} = \epsilon_n - X_n \quad (16)$$

will converge to \sqrt{N} where

$$\epsilon_n = N - X_n^2 \quad (17)$$

The algorithms presented above represent one possible solution to the coordinate conversion problem. In addition, they will demonstrate the goals of the benchmark in that they require a flexible processor capable of handling real and double precision data.

2.2.3 Constant False Alarm Rate (CFAR) Benchmark Definition

The CFAR benchmark is defined by assuming that 6-bit positive values are input from the detection stage of a radar with the following characteristics:

- a. A 70-mile range
- b. A 200-nanosecond compressed pulse width
- c. A 1-millisecond pulse repetition interval

This implies that 4352 data points are to be processed resulting in 4096 binary decisions by a CFAR algorithm using a 256-point sliding window.

The algorithm computes, for each decision, an average of the previous 128 points and the next 128 points for use as a decision threshold. This threshold is modified by a constant threshold parameter and then compared to the window center point resulting in a binary decision that the point is above the threshold or is not. The algorithm is illustrated in block diagram form in Figure 7.

The output of the algorithm is, then, a single bit decision for each input value which would be forwarded to another processor that would, probably, perform some scan-to-scan operation. For the purpose of efficiently transferring the decisions, the signal processor is required to pack 16 consecutive decisions into a 16-bit data word. This process will be included in the CFAR benchmark.

The sliding window function of the benchmark can be efficiently implemented by computing the average of the first 256 points and then updating the average, A_i , as follows

$$A_{i+1} = A_i + X_{i+128} - X_{i-128} \quad (18)$$

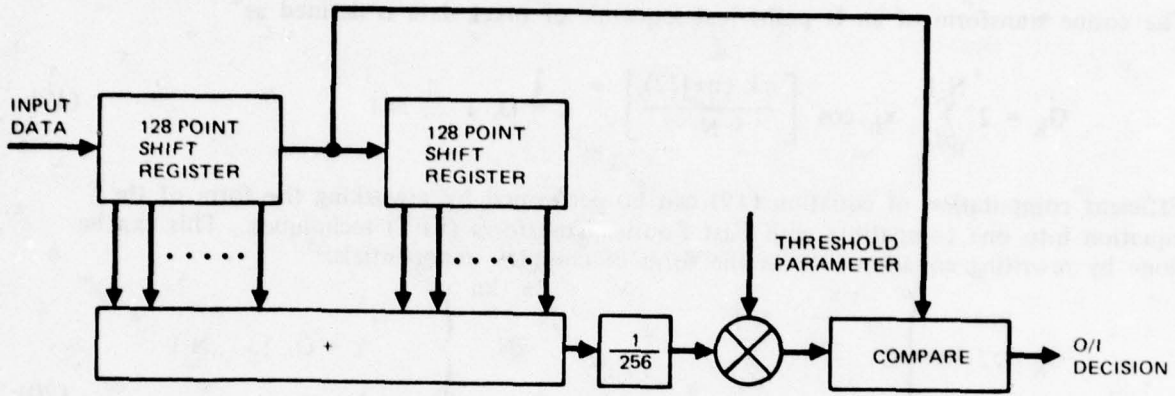
As a result, most of the points will require only three adds (one is part of the compare) and one multiplication (the division by the number of points in the window can be combined with the constant threshold parameter). Therefore, the CFAR algorithm will require

$$3 * 4096 + 256 = 12544 \text{ adds/millisecond}$$

and

$$3 * 4096 = 12288 \text{ multiplies/millisecond}$$

in addition to the output bit packing.



77357-223

Figure 7. CFAR Block Diagram

2.2.4 Cosine Transform Benchmark

The cosine transform of an N point real sequence of pixel data is defined as

$$G_k = 2 \sum_{n=0}^{N-1} x_n \cos \left[\frac{\pi k (n+1/2)}{N} \right] \quad k = 0, 1, \dots, N-1 \quad (19)$$

Efficient computation of equation (19) can be performed by reworking the form of the equation into one compatible with Fast Fourier Transform (FFT) techniques. This can be done by rewriting equation (19) in the form of complex exponentials.

$$G_k = 2 \operatorname{Re} \left\{ e^{-j \frac{\pi k}{2N}} \sum_{n=0}^{N-1} x_n e^{-j \frac{2\pi kn}{2N}} \right\} \quad k = 0, 1, \dots, N-1 \quad (20)$$

If X_n is artificially extended to a double length even sequence of length $2N$, equation (20) can be written as

$$G_k = \operatorname{Re} \left\{ e^{-j \frac{\pi k}{2N}} \sum_{n=0}^{2N-1} X_n e^{-j \frac{2\pi kn}{2N}} \right\} \quad k = 0, 1, \dots, N-1 \quad (21)$$

Defining $w_{2N} = e^{-j \frac{2\pi}{2N}}$

$$G_k = \operatorname{Re} \left\{ e^{-j \frac{\pi k}{2N}} \sum_{n=0}^{2N-1} X_n w_{2N}^{kn} \right\} \quad k = 0, 1, \dots, N-1 \quad (22)$$

The summation term in equation (22) is recognized as the Fourier Transform of the even sequence X_n and is, therefore, a real quantity for all values of k . Thus, equation (22) can be written as

$$G_k = \cos \frac{\pi k}{2N} \left\{ \sum_{n=0}^{2N-1} X_n w_{2N}^{kn} \right\} \quad k = 0, 1, \dots, N-1 \quad (23)$$

For the purposes of image bandwidth compression, the multiplication term $\cos \left(\frac{\pi k}{2N} \right)$ is of no value since it is not data dependent and, therefore, contains no information of interest.

The term in brackets can be computed using a $2N$ length FFT. The cosine transform of X_n (less the multiplicative cosine factor) will be the first N output points of the FFT algorithm.

A more efficient implementation can be determined by recalling that X_n has been made into an even function and, as a result, the FFT output will be real. Consider forming a complex sequence using X_n and an additional set of new data Y_n also even extended as the imaginary part. Thus

$$X'_n = X_n + j Y_n \quad (24)$$

The Fourier Transform of X'_n is

$$F(X'_n) = F(X_n) + j F(Y_n) \quad (25)$$

because F is a linear operator. Now, since X_n and Y_n are even sequences, $F(X_n)$ and $F(Y_n)$ will both be real functions. The implication of equation (25) is, therefore, that two cosine transforms can be simultaneously computed by a single double length FFT and that the results will be found in the first N real FFT outputs for X_n and the first N imaginary FFT outputs for Y_n .

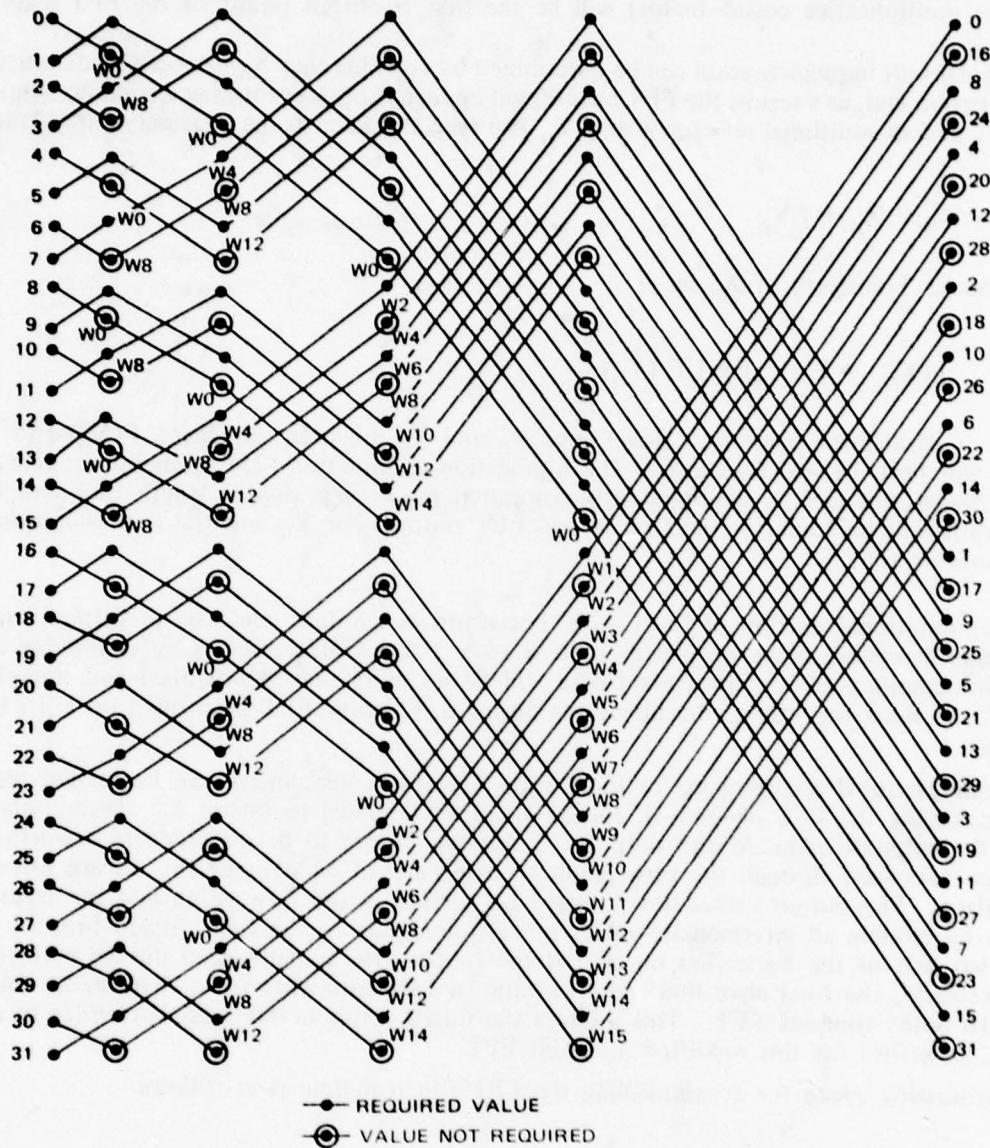
The computational portion of the Cosine Transform benchmark could be performed using the technique described above on groups of two 16×1 input pixel vectors by employing a 32 point complex FFT. Input pixel data will be no more than 8 bits/pixel and, therefore, there is no need for scaling considerations during a 32 point FFT performed on a 16 bit machine.

The computational requirements of the Cosine Transform Benchmark can be further reduced by considering the flow diagram of the 32 point FFT shown in Figure 8. Since only the first 16 results are required and outputs 16 through 31 are to be discarded it is worthwhile to trace backward through the flow graph to determine at what point the discard can actually take place. The output values not required are circled in the flow graph and the trace is shown by circling all intermediate values not required due to the final discard process. It is seen that half of the butterflies on all but the first course actually need not be performed at all. Actually, the final algorithm requires only 16 half butterflies for a first course followed by a 16 point complex FFT. This reduces the initial count of 80 total butterflies to a count of 48 butterflies for this modified 32 point FFT.

The processing speed for accomplishing the FFT's in real time is as follows:

Number of pixels transformed		= 8064000 pixels/second
Number of 32 point complex FFTs	$\frac{8064000}{32}$	= 252000 FFTs/second
48 Butterflies/32 point FFT	48(252000)	= 12096000 butterfly/second

Therefore, the average butterfly rate is one each 82.6 nanoseconds.



77357-225A

Figure 8. 32 Point FFT Modified for Computing Cosine Transform

2.3 ELECTRONIC WARFARE PROBLEM

2.3.1 Basics on Radar Characteristics

In general, at the receiver of any Electronic Warfare equipment, the received signal can be represented as:

$$y(t) = S(t, a_1, a_2, \dots, a_m) + n(t) \quad (26)$$

where S is the transmitted signal which is a function of time and a number of parameters, a_1, a_2, \dots, a_m ; and $n(t)$ is noise. By using the theory of estimation of parameters, the m parameters of the transmitted signal give considerable information about the specific emitter responsible for the received signal. The frequency spectrum, pulse characteristics, pulse repetition frequency, beam pattern, scan pattern and rate, angle of arrival and antenna polarization are some important characteristics that may be utilized in the classification of an unknown emitter.

The frequency spectrum includes the center frequency and modulation of a CW type radar, the center frequency and pulse modulation of single or multiple frequency pulsed radars and agilities. Modulations range from the simple rectangular pulse with no FM to complicated FM and coded waveform. In many cases, the frequency domain is the only way to classify the more complex waveforms.

Pulse characteristics include pulse rise time, fall time, amplitude width and jitter. These parameters are in the time-domain and may be useful in quick classification of simpler, pulsed radars along with center frequency.

The pulse repetition frequency or interval is primarily a derived parameter, gotten from the sorting of a number of individual pulse. PRFs fall into four categories, monofrequency, staggered, jittered, and random, of which the first three can be definitively classified and tracked.

The beam pattern, scan pattern and scan rate are functions of the radar type such as tracking, surveillance, height-finding, etc. A given type of radar will generally exhibit given beam and scan characteristics. These characteristics represent a transformation of frequency and time domain information into a spatial picture of the radar beam and the beams sweep pattern; therefore, these characteristics are also derived.

Finally, the angle of arrival is the angle of the maximum energy for the transmitted signal relative to the direction of flight of the aircraft. The angle of arrival is a single pulse parameter in the spatial domain; however, to be most accurate, the angle of arrival should be transformed into direction of arrival to compensate for the in-flight motion of the aircraft.

2.3.1.1 Single Pulse Characteristics

Using the information available in a single pulse, it is possible, assuming perfect conditions, to identify the emitter type and location. To classify the general type, the frequency spectrum and pulse characteristics are the only useful information available in a single pulse. A blanket statement may be made that the measurements of the parameters in equation (26) will be spread in some random fashion and will represent a stochastic process. If the

various frequency and pulse characteristics are used to model emitter classes, then from the theory of estimation, probabilities can be established using the estimated mean and variance to identify a given incoming pulse with a modelled emitter class.

For a simple radar, center frequency, pulse width, and rise and fall times are probably sufficient to make an estimate. More complex types may need details of the frequency spectrum of the pulse or the agilities to identify them.

The location of an emitter can be gotten from the conversion of the angle of arrival into earth coordinates, often called the direction of arrival. If properly done, this parameter is the least sensitive to pulse-to-pulse variations because the emitter can not move significantly from pulse-to-pulse. The frequency band and the angle of arrival are often used as a first coarse sorting of an incoming pulse.

By classifying the general type of emitter and its location, the specific emitter may be identified for further processing, such as pulse train classification, range, lethality, etc. Techniques for this classification will be discussed in Section 2.3.6.

2.3.1.2 Multiple Pulse Characteristics

When the individual pulses are successfully classified at least by general type, additional information can be extracted about the emitters responsible for these pulses. For many classes of radar, this information is uninteresting and single pulse classification is enough to display, counter, or disregard. However, the more lethal new threats have new agilities, interesting scan patterns, and various processing techniques that require sophisticated processing on the part of the Electronic Warfare receiver to characterize the emitter. Pulse train characteristics, such as PRF, require a number of pulses to detect with any degree of accuracy and may be modelled as a Markov process. Furthermore, scan pattern and beam pattern can be determined using the pulse train information with pulse amplitude; however, extensive processing is necessary. A discussion on pulse train classification is given in Section 2.3.6

2.3.2 Necessary EW Functions

There are three major types of electronic warfare functions: electronic reconnaissance, electronic support measure (ESM), and electronic countermeasures (ECM). Electronic reconnaissance is the specific reconnaissance directed toward the collection of electromagnetic radiations, e.g. ELINT, COMINT, SIGINT, etc. Two functions are served by the reconnaissance and analysis of the radiations: 1) Intelligence gathering to obtain information for the electronic order of battle, and 2) Basis of ECM designs or redesigns.

Electronic support measures are for monitoring the direction and type of potentially hostile systems, generally using a priori reference data.

Electronic countermeasures is to deny or degrade the enemy's use of his electromagnetic systems in order to obtain a tactical advantage. Both active and passive measures exist.

The above major EW functions are supported by similar subsystems to the extent that the major function needs them. All the major EW functions need to be able to receive signals, determine the significant parameters of the signals, sort the signals, associate the signals with

emitter classes and/or specific emitters. The ESM and ECM functions must be able to prioritize emitters according to their lethality, and display the hostile emitters. Finally, the ECM system must be able to process the emitters, analyze the available resources of the electronic defensive systems, decide on the most effective countermeasures, and activate those countermeasures. The bulk of this section will be devoted to the sub-functions necessary to sort signals, associate signals with classes, and determine lethality primarily for the ESM/ECM mission.

2.3.3 Channelized Front End

No specific system is defined for the ensuing benchmarks, however, the availability of signal parameters is assumed. The most promising system in development today contains a channelized front-end receiver, and the processing is done on a channel or subchannel basis.

Typical first problems are dense signal environments and radar characteristics that cover multibeam, multifrequency transmission, PRF agility, RF agility, CW, and intrapulse frequency agility. As a minimum, a receiver must possess:

1. An ability to handle multiple frequencies simultaneously
2. A near-unity probability of detection
3. Good frequency measurement, resolution, and accuracy
4. Single-pulse acquisition and parameter measurement.

To handle high pulse densities spread over a wide frequency range requires a wide instantaneous bandwidth. Furthermore, a wide bandwidth allows instant, single pulse acquisition. The complex PRF agile radars require sorting by single pulse parameters, forcing the need of good frequency measurement, resolution, and accuracy and a high probability of intercept. The channelized receiver concept have a wide instantaneous bandwidth and high signal sensitivity, allowing high probability of detection over several octaves. An excellent discussion of channelized receivers and the impact of surface acoustic wave devices may be found in *Electronic Warfare*, September/October 1977.

2.3.4 System Architecture

From a generic point of view, a fantastic system can be postulated that will process any incoming pulse, fully characterize it, classify it, correlate it with its train of pulse, and direct countermeasures, and whatever else is necessary. From a practical point of view, this system must handle a multitude of pulses with exceedingly different characteristics. This section will attempt to define a system flow and point out the strengths and weaknesses of the various steps in the flow. To attempt this definition, firstly, the top-level system flow of a "pulse" processor will be discussed; secondly, the processing flow will be analyzed; and lastly, the architectural necessities will be presented.

2.3.4.1 System Flow

Incoming Signals: The incoming pulse is received and generally converted to base band. The various pulse parameters, such as center frequency, pulse-width, rise time, fall time, AOA,

etc. are extracted from the pulse, and digital words are passed to the signal "sorter". The digital words, more or less, characterize the received pulse.

Sorting of Signals: As the digital words enter the signal processor ("sorter"), the processor must attempt to classify the generic type of radar so that the proper countermeasures may be chosen; the potential of danger may be ascertained, and further processing may be simplified. Furthermore, the processor should identify the specific emitter so that any directivity of countermeasures may be specified, as well as, the probability of a radar mode change to a dangerous mode may be estimated. If the generic type and/or the specific emitter can be classified, then multipulse statistics can be gathered to refine further the processor's knowledge of how to jam and when this threat will become dangerous. Finally, based on available data, the processor must prioritize the threats for display to an operator or for automatic deployment of countermeasures. In the airborne ECM case, this prioritization is critical because the ECM gear has definitely limited quantities of jammer power or deployable passive countermeasure to use.

2.3.4.2 Processing Flow

Signal Parameters: As the incoming pulse is being received, a number of operations begin that extract information about the pulse. The minimal set usually includes center frequency (f_c), pulse width (PW), angle of arrival (AOA), and time of arrival (TOA). These require very little processing to extract the information and may be handled primarily in analog form while being processed, and then converted to digital signals.

However, the more exotic the emitter class, the more sophisticated the processing must become. A considerable amount of preprocessing may be necessary if a good characterization of complex radars is desired. Radars that have modulated waveforms, train and/or coded pulses, spread spectrum characteristics, or CW may require greatly enhanced preprocessing on a single-pulse basis to be successfully characterized. Table 3 indicates some of the variety that can be seen in radar waveforms. Each type has favorable properties that are useful in relation to the range-velocity ambiguity function.

To characterize these pulses successfully, additional frequency domain information in the form of spectral analysis, or additional time domain information such as rise and fall time, and pulse amplitude may be necessary for emitter-type classification, as well as, a spatial domain information transformation from AOA to direction of arrival (DOA) may be necessary for specific-emitter classification. A significant problem or conflict arises because the volume of incoming pulses is high; therefore, the pre-processing rates for spectral information will be exceedingly high, approaching 200-500 million operations per second for a 60-100 MHz channel at baseband. Unfortunately, the spectral analysis must be done before any emitter-type classification can be performed. If the easily classified pulses could be stripped away either from the raw analog or in the digital data, the spectral analysis could be done on the spread spectrum pulses at a much lower processing rate. Current equipment can only perform limited amounts of this preprocessing because of hardware limitations.

Probabilistic Type Classification: The signal parameters are passed to the signal ("sorter") processor for classification and action (display, countermeasures, etc.). The parameters

Table 3. Classification of Radar Waveforms

SINGLE PULSE	
	Rectangular, No FM
	Spread Spectrum
	Rectangular - Linear FM
	Linear V FM
	Stepped FM
	Quadratic FM
	Gaussian, Linear FM
TRAIN PULSE	
	Equally Spaced, Identical
	Equally Spaced, Identical With Constant Complex Multiplier
	Non-Identical
	Multiple Frequency
	Staggered PRF
	Multiple Carrier
	Pseudo-Random Coded
	Barker
	Maximum - Length Sequence
	Polyphase Sequence (Ternary, Quaternary)
	Huffman
CW	
	Simple
	Frequency - Modulated
	Multi-Frequency

will be a best approximation of the parameter set actually needed to characterize the received signal. Recalling equation (26), the signal processor will receive $\hat{y}(t, b_1, \dots, b_n)$ which will approximate y , that is

$$\hat{y}(t, b_1, \dots, b_n) \approx y(t, a_1, \dots, a_m) \quad (27)$$

where \hat{y} is the signal processor's representation of y and b_1, b_2, \dots, b_n are the signal representation parameters. A given emitter-type will have a "characteristic" set of representation parameters, which will be spread in a random fashion about some mean.

Using probabilistic techniques for multivariate data analysis, a Chi-squared method may be employed. The proximity of a given pulse to the "characteristic" set of parameters for a

given type will be a measure of the probability of identification. If the measure exceeds a given threshold, classification will be declared, and the probability of this type will be updated. A further discussion will be given in the Section 2.3.6.

The threshold criterion is included to insure that "over-classifying" does not occur. A pulse can arrive at the receiver that does not fit into any a priori emitter-type class. The processor must first be able to create a new class if a sufficient statistic can be formed. Furthermore, the threshold may be warped to allow an easier classification of emitter-types that have exceeding wide diversity or a high lethality. By establishing a low threshold on the more lethal threats, the receiver/processor will declare more false classifications of the lethal emitter-types; however, this higher-error rate may be justified as safety feature (the ounce of prevention rather than the pound of cure).

Specific Emitter-Classification: A specific emitter can be identified primarily from type classification and AOA/DOA. In the state-of-the-art case, today's receiver/processors use frequency/frequency band and AOA. This classification can also be done as a Chi-squared distance measure; however, the statistic for the AOA/DOA will have to be built up as the emitter-type probability is created. To minimize the processing, DOA is preferred because it relates the emitter to a fixed geometric reference which is insensitive to the motion of the platform. The establishment of a statistical parameter will be discussed in the Section 2.3.6.

Furthermore, the probability of a specific emitter can be updated, as well as, a current "active emitter" list may be established and updated. A given specific emitter may be modelled as a Markov process, that is, if an emitter has been detected, the likelihood is high that it will be observed again. A scan-to-scan correlation may be observed. The theory of Markov chains may be applied to describe the scan-to-scan correlation. In the theory of Markov processes, the routine of any particular event is not assumed to be independent of other events. This theory has been applied to radar returns at the radar antenna and can be likewise applied to the EW receiver.

Using the above rationale, multipulse statistics and properties can be ascertained. Pulse train characteristics, scan pattern and scan rate, and beam width can be determined. These characteristics require sophisticated processing whose difficulty increases as a power (N^2 or N^3) of the number of items being processed.

Further statistics may be gathered if interchannel communication is permitted, that is, if a number of parallel processors are assumed to be handling portions of the spectrum; multi-frequency threats, or spread spectrum threats may appear in a number of the channels. The correlation of interchannel information for specific classes of emitters will provide positive identification.

2.3.5 Architectural Necessities

The processing flow for an EW receiver requires a diverse set of processing capabilities which will be outlined herein and expanded in the coding and discussion in the Section 2.3.6. The most severe are primarily in the address generation and data comparison areas.

The memory organization must be highly flexible, such that if a given frequency band has a high degree of activity, the memory space can be reallocated to accommodate the higher

activity. Two functions or architectural necessities are indicated: 1) efficient data memory control and organization and 2) sophisticated data address generation for subsequent data processing. These necessities, when combined with the high pulse densities of the current and future EW environment, dictate a dedicated data address function to control the reading and writing of data into memory as well as to control the allocation of memory based on need.

Data comparisons has implications in all areas of the processor – data addressing, data processing and control. Even though a very sophisticated algorithm can be devised to improve the classification processes, a final decision must be made by the processor. This decision can be done in all hardware or in hardware-software, but ultimately it reduces to a simple yes/no comparison.

From the outcome of the comparison, three types of branches/jumps may be necessary in this process: 1) Data dependent data address generation, 2) Data dependent arithmetic decision, or 3) Conditional or unconditional jumps in program memory. Because all types of decisions or branches are best handled in different portions of the processor, the control for these branches should be put in the various portions rather than totally centralized.

2.3.6 Benchmarks

This section will contain a discussion on three algorithms and benchmarks necessary to accomplish the algorithms.

Pulse Classification, Mean and Variance Determination, and PRF Sorting algorithms will be developed primarily from the representation of the received signal given in equations (26) and (27). Certain properties of random variables and stochastic processes will be included only if it is necessary for completeness. The benchmarks will be developed for the main sections of the algorithms. Various processor setup steps will be excluded and only included if requisite for understanding.

2.3.6.1 Representation of the Received Signal

The signal processor receives a set of parameters from the receiver that are the receiver's best effort to characterize the incoming pulse. The parameters deviate from the exact set of parameters and parametric values because the receiver has finite capabilities to detect the pulse, may not detect all the "proper" parameters, and receives a noisy, corrupted signal which it further corrupts. Using equation (26), the received signal, $R(t)$, is represented by an $N + 1$ dimensional vector word, which is transformed by the function \hat{y} to approximate $R(t)$, that is,

$$R(t) \approx \hat{y}(t, b_1, \dots, b_n) \quad (28)$$

The $N + 1$ dimensional vector word is considered a pattern vector and the parameters b_1, \dots, b_n are random variables. Assuming $\hat{y}(t, b_1, \dots, b_n)$ satisfies the general conditions for a random variable, the $R(t)$ is also a random variable.

NOTE: In the signal processor, the b_i 's will be represented by digital words, which means that all probabilities, discussed henceforth, will be discrete probabilities. Furthermore, it is assumed that the b_i 's are independent and orthogonal, that is, uncorrelated. Although this is not strictly true, a set of b_i 's can be chosen where the b_i 's approach being uncorrelated.

2.3.6.2 Mean and Variance Determination

Algorithm: Assuming a complete statistical description of the noise at the receiver is known, the joint probability density function for the noise can be used. The pattern vector words can be represented as:

$$b_i = S_i + N_i \quad (29)$$

where S_i is the i th parameter representing the transmitted pulse and N_i is the noise on the i th parameter. Assuming the noise can be processed out or statistically removed, the job is to form estimates of the b_i 's on the basis of M observations, y_1, \dots, y_M of a given emitter-type.

Two helpful parameters in forming estimates for the b_i 's are the sample mean and variance which will be used later in the pulse classification algorithm.

The "sample mean" is simply the sum of the measurements divided by the number of observation. In terms of the b 's:

$$\text{Sample mean of } b_i = \bar{b}_i = \frac{b_{i1} + b_{i2} + \dots + b_{iM}}{M} \quad (30)$$

The sample mean, \bar{b}_i is the expected value of b_i . The "sample variance" is the measure of sum of the deviations of the individual observations from the expected value divided by the number of observations. In terms of the b 's,

$$\text{Sample variance of } b_i = \sigma_i^2 = \frac{(b_{i1} - \bar{b}_i)^2 + \dots + (b_{iM} - \bar{b}_i)^2}{M} \quad (31)$$

The sample variance, σ_i^2 is the second moment or the dispersion of b_i .

Benchmark: The mean and variance for the various parameters of the known emitter-types will be a priori data for the signal processor. These values will be the distillation of intelligence data. Pulses, not meeting the threshold criteria on AOA/DOA statistics, will have to build-up a sample mean and variance for AOA/DOA to permit easy classification. If a processor is able to add new categories or emitter-types, then the sample mean and variance algorithm or an equivalent will have to be included as an auxiliary processing task for all the signal parameters.

Equation (30) may be implemented directly in an iterative fashion such that 2 operations are necessary per iteration; however, equation (7) would require $2N-1$ additions, N multiplications and a division or a $1/M$ multiplication. To build up a statistic, it is often necessary to integrate a new b_i as the data comes.

A straight implementation would require $3N$ operations every new data point. Assuming N starts at 1 and goes to N , a total of $3N \left(\frac{N+1}{2}\right)$ operations would be required for N -observations. A different approach was explored and it can be shown that equation (31) can be reduced with the aid of equation (30) to

$$\sigma_i^2 = \frac{1}{N} \sum_{j=1}^N b_{ij}^2 - \bar{b}_i^2 \quad (32)$$

By this reduction, equation (32) can be used in an iterative procedure adding only 4 operations per iteration.

Below is a sample processing implementation of the sample mean and sample variance described by equation (30) and (32) respectively. Four values must be stored to set up the iteration loop and a loop counter test and iteration must be included for each iteration. Therefore, for N observation, $8N + 4$ operations must be performed.

SAMPLE PROGRAM FOR MEAN AND VARIANCE

```

SETUP  Enter C = 0, D = 0, N = 1, NMAX = NMAX
MEAN   C = C + BI (N)
       BIBAR = C/N           Comment: Sample Mean,  $b_i$ 
       BIB2 = BIBAR*BIBAR
       D = D + BIB2
       SIGMA2 = D/M
       SIGMA2 = SIGMA2-BIB2  Comment: Sample Variance,  $\sigma_i^2$ 
       IF N = NMAX, THEN STOP
       N = N + 1, JUMP MEAN
       STOP

```

2.3.6.3 Pulse Classification Algorithm

Algorithm: Assuming a sample mean and variance for each parameter b_j has been determined for J classes, the mean and variance may be used as a measure of the class, C_N against a new incoming vector word. A new data word y is received with parameters b_1, \dots, b_n . To determine the probability that y belongs to class, C_N , a mean-square error between each new b_i and the mean \bar{b}_{Ni} of class C_N is gotten and normalized with the variance, σ_{Ni}^2 , previously established for C_N . This mean-square error is often referred to as a "distance measure". The error is given by:

$$e_{Ni}^2 = \frac{(b_i - \bar{b}_{Ni})^2}{\sigma_{Ni}^2} \quad (33)$$

(Note: \bar{b}_{Ni} represents the sample mean of the i th parameter of class, C_N . Compare with b_{ij} which means the j th observation of the i th parameter of an unspecified class). For the entire data word y , equation (33) becomes

$$e_N^2 = \sum_{i=1}^n \frac{(b_i - \bar{b}_{Ni})^2}{\sigma_{Ni}^2}$$

This procedure assumes a multivariate normal distribution for the vector variable in each of the classes. We use the notion of swarm for the plot in measurement space of points representing the members of a single class. A multivariate normal swarm is very dense in the region of the class centroid and thins out in all directions. The normal swarm is a hyper-ellipsoidal distribution. The probability density function for the i th parameter is by belonging to the N th class is:

$$P_{ni} = \frac{1}{\sigma_{Ni} \sqrt{2\pi}} e^{-\chi_N^2/2} \quad (34)$$

where χ_N^2 represents a Chi-squared statistic and equals $\frac{(b_i - \bar{b}_{Ni})^2}{\sigma_{Ni}^2}$. Dropping the constant and extending equation (34), the probability density function of y belonging to C_N is

$$P_N = \exp(-e_N^2/2) \quad (35)$$

which also follows a Chi-squared statistic. The probability of Class C_N is now represented by P_N ; the probability of all J classes must be similarly determined. Finally, to yield the relative density of class C_N with respect to y versus the other J densities, the relative densities are determined by:

$$P_{rN} = \frac{P_N}{\sum_{j=1}^J P_j} \quad (36)$$

whichever P_{rN} is the largest is the class. However, a threshold function may be invoked at this point which will skew the determination. A class may be favored because it represents a larger threat or for whatever reason, or a "No class" determination may be made. A "No class" determination is valid only in a processor which can deal with unknown signals.

Benchmark: Unfortunately, no simple reduction is available for this algorithm. The processor must compute the mean-square error of each new \hat{y} versus every class, C_N . The mean-square error for each i th parameter requires three memory fetches to get \bar{b}_{Ni} , $\frac{1}{\sigma_{Ni}^2}$ and

b_i and 4 computational steps; a total of 7 operations are involved. With J classes and m parameters, a total of $7*J*m$ operations are necessary to make the computations plus indexing the loop counter which involves another $J*m$ operations.

The processor must calculate the probability that \hat{y} belongs to every class, C_N and normalize the probabilities. Assuming the exponential is a look-up function, 1 memory fetch and 2 computational steps are required for every class, plus one memory fetch per \hat{y} for the $\sum_{j=1}^J P_j$; a total of $3J + 1$ operations are required.

Lastly, the processor must determine which class has the highest probability as well as which classes have passed their threshold. A minimum of one memory fetch and two comparisons per class as well as one memory store per \hat{y} are required, a minimum total of $3J + 1$ operations are necessary. A maximum of $J-1$ memory stores is possible; therefore, the maximum total of $4J$ operations may be required. Below is a sample program.

SAMPLE PROGRAM FOR PULSE CLASSIFICATION

```

SETUP  EJ = 0, IMAX = IMAX, JMAX = JMAX, I = 0, J = 1,
       PYK = 0
ERROR  A = B (I) - BBAR (J, I)      Comment: a = (bi -  $\bar{b}_{ji}$ )
       C = A*A                      Comment: c = (bi -  $\bar{b}_{ji}$ )2
       D = C* SIGNA (J, I)          Comment: d =  $\frac{(b_i - \bar{b}_{ji})^2}{\sigma_{ji}^2} = e_{ji}^2$ 

       EJ = EJ + D
       IF I = IMAX, JUMP PROB
       I = I + 1, JUMP ERROR
PROB   E = - EJ/2 (Shift Right)
       PJ = MEM (E)                 Comment: Memory look-up for
                                     exponential: exp(E)
       PYJ = PJ*SUMPJ               Comment:  $P_r^J = \frac{P_j}{\sum P_j}$ 
       TEST = PYJ - PYK             Comment: Compare with previous
       IF TEST < 0, JUMP JCOUNT    "high" probability
       IF TEST = 0, JUMP JSTORE
       PYK = PYJ                    Comment: Replace with new
                                     "high" probability

JSTORE MEM = J                      Comment: Store CJ

JCOUNT IF J = JMAX, JUMP THRES
       J = J + 1, JUMP ERROR

THRES  TEST = PYK - T                Comment: Compare with
       IF TEST > 0, JUMP STORE      threshold

```

STORE At this point, the program will option to display the output (MEM), to determine a counter measure or do nothing if the test is passed. Otherwise, it will store the y for further processing.

2.3.6.4 PRF Sorting

Algorithm: as the incoming signal is received, a time-of-arrival number or word is associated with it. This TOA word is relative to an internal clock and demarks the beginning of the pulse. The primary purpose for the demarcation of the pulse is to develop multipulse statistics like the PRF/PRI of a specific emitter. By knowing the PRI and the time of arrival of the previous pulse, an ECM processor can anticipate its needs for expenditure of counter-measure resources.

The major problem for PRF sorting is multiple emitters of the same or similar type transmitting in close geometric proximity such that the AOAs cannot be resolved. The goal of the sorter is to pull apart the distinctive PRFs, either simple, staggered or jittered. The algorithmic flow is exceedingly simple; however, as the number of pulses to be sorted increases, the problem can become untenable.

The flow is, as follows:

1. Calculate the difference between all reasonable TOA combinations, that is,

$$\Delta_{ij} = \text{TOA}_i - \text{TOA}_j \text{ for all } j \neq i$$

where TOA_j represents the TOA of the i th pulse.

2. Compare the differences for a repetitive pattern, such as:

$$\Delta_{ij} = \Delta_{jk} = \Delta_{kl} = \Delta_{lm} = \dots$$

A tolerance must be included in this comparison so that the comparison is not overly sensitive to noise.

3. After a successful comparison of a given Δ_{ij} , a PRF can be declared and utilized. Utilization may range from simply preparing the countermeasure to developing histograms for beam width and scan pattern determination.

Benchmark: The benchmark described herein represents a "practical" approach to using the TOAs as they come to the PRF sorter. As the processor receives the m th TOA, it stores the data in memory, replacing an old TOA value. This approach represents a moving time window over which processing will be performed. Without this constraint, the processor would be saturated within a very few pulses. With small modification, this benchmark could be used as a batch process in which a large number of TOAs are saved and processed as a group.

The processor must update the memory pointer and fetch an "old" TOA for delta calculation. The delta calculation is performed N times, where N represents the average number of pulses received during the time window. Two operations are required per pass.

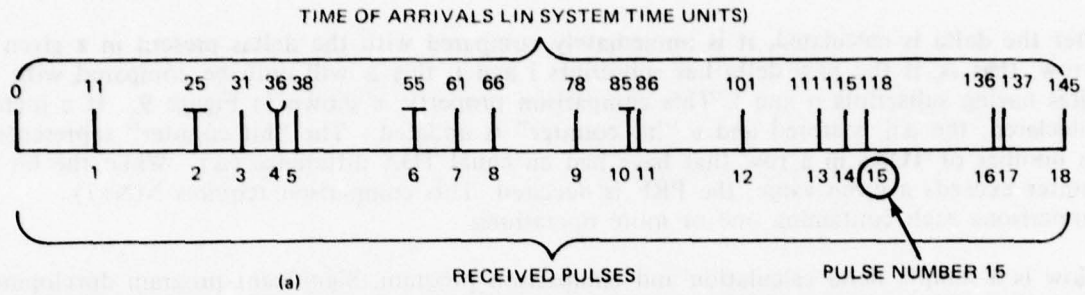
After the delta is calculated, it is immediately compared with the deltas present in a given Δ row, that is, if the new delta has subscripts i and j , this Δ will only be compared with deltas having subscripts n and i . This comparison property is shown in Figure 9. If a match is declared, the Δ_{ij} is stored and a "hit counter" is updated. The "hit counter" represents the number of TOAs in a row that have had an equal TOA difference (Δ). When the hit counter exceeds a given value, the PRF is declared. This comparison requires $N(N+1)$ comparisons each containing one or more operations.

Below is a sample delta calculation and comparison program. Significant program development is required to include how the bit counter is incremented or decremented, how a PRF is declared, and how the data is used for prediction.

A new instruction has evolved from this benchmark – the windowed compare. Because the use of absolute compare function would create a noise-sensitive process, a tolerance must be included to account for variations in the TOA measurements and the subsequent delta calculation.

SAMPLE PROGRAM FOR DELTA CALCULATION AND COMPARISON

<p>SETUP ENTER N = N, M = M, I = M-N-2 J = I-N-1, TOL = TOL TOA(M) = TOA</p>	<p>Comment: N is the average number of pulses received during the time window. M is the memory pointer. Comment: Store the new TOA</p>
<p>DELTA DELTA (I,M) = TOA(M)-TOA(I) IF I = M, STOP I = I + 1</p>	
<p>COMP TEST 1 = DELTA (I,M) - DELTA(J,I)+TOL J = J + 1 IF J = M-1, JUMP DELTA TEST 2 = TEST 1 - 2 *TOL IF TEST 1 > TOL, JUMP COMP IF TEST 2 < -TOL, JUMP DELTA IF TEST 1 < TOL + TEST 2 > - TOL, THEN Increment the hit counter; Store Δ_{im} Jump to COMP.</p>	<p>Comment: Comparison windows have been set-up will be replaced by new instruction.</p>



PULSE NUMBER (i)	TIME OF ARRIVAL	PULSE NUMBER (j)																	
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
1	11	X																	
2	25	14	X																
3	31	20	6	X															
4	36	25	11	5	X														
5	38	27	13	7	2	X													
6	55	44	30	24	19	17	X												
7	61	50	36	30	25	23	6	X											
8	66	55	41	35	30	28	11	5	X										
9	78	67	53	47	42	40	23	17	12	X									
10	85	74	60	54	49	47	30	24	19	7									
11	86	75	61	55	50	48	31	25	20	8	1								
12	101	90	76	70	65	63	46	40	35	23	16	15							
13	111	100	86	80	75	73	56	50	45	33	26	25	10						
14	115	104	90	84	9	77	60	54	49	37	30	29	14	4					
15	118	107	93	87	82	80	63	57	52	40	33	32	17	7	3				
16	136	125	111	105	100	98	1	75	70	58	51	50	35	25	21	18			
17	137	126	112	106	101	99	92	76	71	59	52	51	36	26	22	19	1		
18	145	134	120	114	109	107	90	84	79	67	60	59	44	34	30	27	9	8	X

(b)

LEGEND:

a) RECEIVED PULSES AND THEIR TIME OF ARRIVALS.

b) Δ_{ij} AND SLIDING TIME WINDOW.

SLIDING TIME WINDOW

78453-20

Figure 9. Comparison Properties

2.3.6.5 References

Feucht, Dennis: "Pattern Recognition: Basic Concepts and Implementations", *Computer Design*, Volume 16, No. 12, pp. 57-68.

Lundy, Duane: "SAW Filters Shrink Channelized Receivers," *Electronic Warfare*, Volume 9, No. 5, pp. 169-174.

Cooley, William W., and Lohnes, Paul R: *Multivariate Data Analysis*, John Wiley and Sons, Inc., New York, 1971, pp. 263-268.

Papoulis, Athanasios: *Probability, Random Variables and Stochastic Processes*, McGraw-Hill Book Company, New York, 1965, pp. 233-263.

Skolnik, Merrill I.: *Radar Handbook*, McGraw-Hill Book Co., New York, 1970, pp. 3-10 to 3-30.

Skolnik, Merrill I.: *Introduction to Radar Systems*, McGraw-Hill Book Co., New York, 1962, p. 54-56 and 461-462.

SECTION III

MULTIMODE CPU ARCHITECTURE

3.0 INTRODUCTION

In Section II, a baseline scenario was defined for future signal processing applications. The scenario was presented as a set of representative benchmarks. The benchmarks were chosen from previous Air Force procurements and in-house experience and are used to indicate the various processing and control structure necessary to properly handle the problem set. Table 4 is a brief compendium of the benchmarks and the data processing, data addressing, and control structures necessary to perform the benchmarks.

In this section, an attempt will be made to utilize Table 4 and discuss the impact of the processing needs on basic computing structures such as the control section, the ALU, the data addressing and the bus system.

3.1 PRIMITIVE COMPUTING STRUCTURE

Conceptually, the most basic computing structure must contain a control function, an arithmetic/logic function, and storage. All structures may be broken down to these fundamental structures. For the purpose of discussion, Figure 10 represents a primitive computing structure for handling signal processing. The control function is handled by an addressing unit and a micro-program/instruction memory. That memory controls the functioning of the arithmetics and storage, as well as, its own addressing unit, thereby creating a self-contained computer.

The arithmetic function is performed by the Register Arithmetic Logic Unit (RALU) and the multiplier. The RALU performs all the basic arithmetics; add, subtract, shift, and the basic logic functions, AND, OR, EXOR, COMPLEMENT. The multiplier performs a simple hard-wired multiply function on any two operands presented to it. The multiplier is an extension of the basic arithmetic function because the multiply function is generally required in signal processing.

The storage function (operand storage) is handled by the data memory and the register section of the RALU. The data memory has both permanent operand storage (i.e., ROM/PROM) and temporary storage (i.e., RAM). The structure shown in Figure 10 assumes that the addressing of operands (data addressing) is performed by the RALU or the controller.

Although Figure 10 shows a multitude of buses, a single bus can be conceived to handle all control and data informational transfers. The bus structure will be discussed at length in the next section.

This primitive computing structure has been presented as a basis for the following discussions. These discussions will expand the description of the elements in Figure 10, as well as, give the rationale for the specific embodiments of the elements based on the baseline scenario and architectural constraints.

Table 4. Benchmarks and Indicated Architectural Characteristics

Benchmark	Data Processing	Data Addressing	Control
FFT	Multiply Accumulate Complex Arithmetic	Array Indexing	Loop Counting
Coordinate Conversion	Double Precision Numerical Scaling	Tight Data Loops	Data Dependent Branches
CFAR	Bit Manipulation	Simple Addressing	Data Dependent Jumps
Cosine Transform	High I/O Rates To Memory and Outside World	High Addressing Rates	Loop Counting
Pulse Classification	Memory Table Lookup Variable Bit Length Data Words High Speed Arithmetic	Array Indexing	Data Dependent Branches

3.2 BUS SPEED, WIDTH, EFFICIENCY

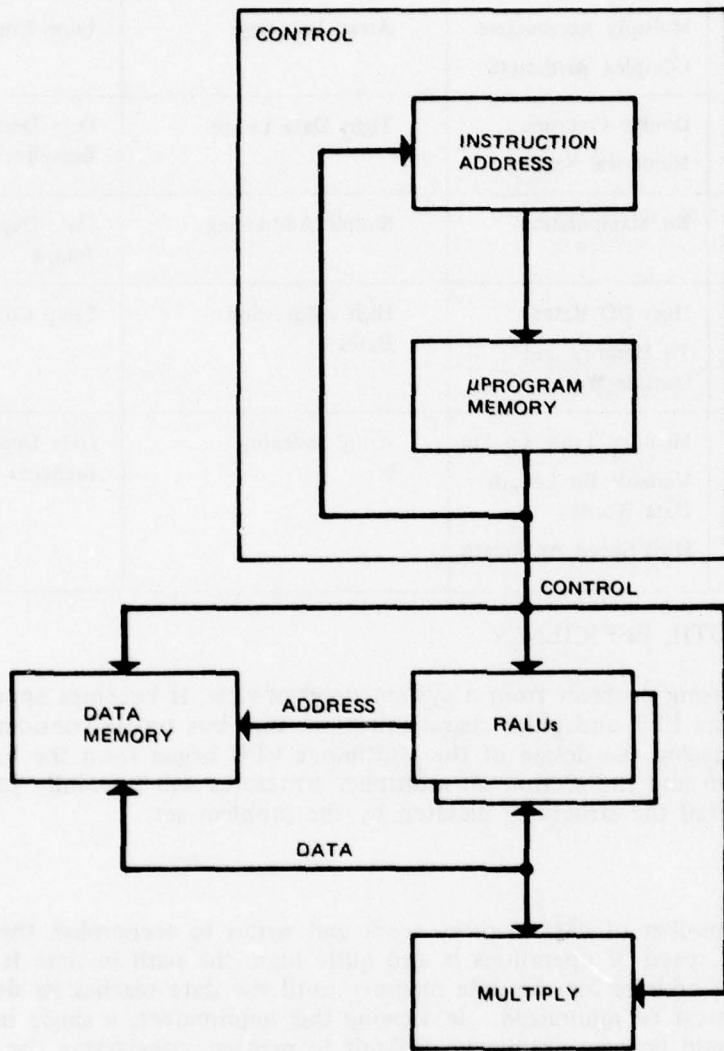
In viewing the signal processing problem from a system point-of-view, it becomes apparent for certain problems, such as the FFT and pulse characterization, that bus traffic considerations are paramount. For this reason, the design of the Multimode CPU began from the bus and proceeded out. This section and the section on multiplier structures will hopefully justify this decision, as well as, detail the structures dictated by the problem set.

3.2.1 Bus Speed

The FFT requires a great number of data memory reads and writes to accomplish the butterfly operation. Because the speed of operations is also quite high, the path in time from the generation of the read/write address for the data memory until the data reaches its destination or arrives from its source must be minimized. In viewing this requirement, a single bus for data addresses and data would become extremely difficult to manage, considering the high data flow required. It has been concluded that this path from address to data must be pipelined to provide maximum speed; therefore, a separate data address bus and a separate data bus is a necessity to handle the pipeline.

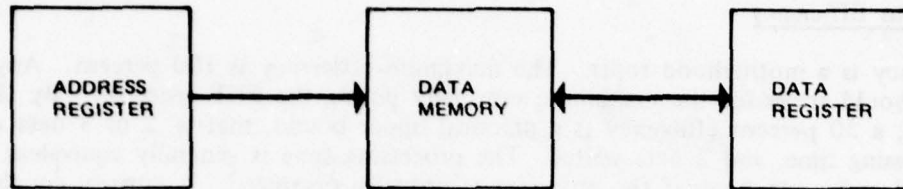
Furthermore, a minimum time path can be analyzed, as in Figure 11, which will give feasible estimates of the time to take a previously generated address from the address register to the memory, to fetch the operand from the data memory and to send operand to the data register. The time path, therefore, is

$$T = T_{\text{REG OUT}} + T_{\text{DRIVER}} + T_{\text{ADR BUS}} + T_{\text{ACC}} + T_{\text{DATA BUS}} + T_{\text{LATCH}}$$



78453-1

Figure 10. Primitive Structure



78453-2

Figure 11. Minimum Time Path for Address to Register

With current technology, the minimum time path can be from 50 to 100 μsec , depending on a number of factors, including IC drive, PC board techniques, memory selection, etc.

Using the above argument for separate data and data address buses, it has been concluded that a five bus system is necessary to maintain and support the data bus and data address bus requirements.

The five buses are:

- DATA
- DATA ADDRESS
- INSTRUCTION
- INSTRUCTION ADDRESS
- STATUS FLAGS OF PROCESSORS

Analysis shows that each of these buses must maintain a speed equivalent with the speeds of the data and data address bus, that is, the instruction address to microinstruction memory to instruction register path must be as quick as the data address/data path.

3.2.2 Bus Width

As stated before, the FFT presents the most challenging problem. This extends into the area of bus width. The FFT butterfly requires two or three complex data reads and two complex data writes be performed. Obviously, the bus could be structured so that the complex words are accessed as real quantities (2 per complex word). Such a bus would double the number of reads and writes necessary to accomplish an FFT butterfly, thereby doubling the time to set up the FFT independent of the multiplier.

The indicated conclusion is that a dual data bus system should be used so that a single read time is necessary to access and transmit a complex data word. Furthermore, the data words should be 16 bit real and 16 bit imaginary to allow processing gain without scaling which would require either additional processing steps or more hardware. Therefore, the data bus will be 32 bits wide to handle the complex data for the FFT. This size is also good if any double precision arithmetic is necessary. Coordinate conversion routines sometimes require expanded accuracy for positional fixes.

3.2.3 Bus Efficiency

Bus efficiency is a motherhood topic. The maximum efficiency is 100 percent. Any signal processor should strive for the maximum, especially during the FFT processes. By the FFT's very nature, a 50 percent efficiency is a practical upper bound, that is, 2 or 3 data reads, some processing time, and 2 data writes. The processing time is generally equivalent to the sum of read and write times if the processor is properly organized. A cursory conclusion at this point is if a practical upper bound of 50 percent efficiency is obtainable, why not have two FFT butterfly operations running concurrently and out-of-phase so that one is processing during the reads and writes of the other and vice versa? Thus, the bus efficiency can approach the maximum.

3.3 MULTIPLIER STRUCTURES

This section on multipliers has been included to discuss the impact of a multiplier special function unit on the speed and bus traffic of a signal processor. The multipliers described herein will be assumed to have 16 x 16 bit multiply capability and may be any of a number of available multiplier organizations, such as parallel, pipelined, or serial parallel.

The problem set will be those discussed heretofore; however, the FFT remains the most challenging problem. The actual design of the multiplier will not be included although its implementation greatly impacts the LSI-ability of the multiplier special function unit.

3.3.1 FFT Butterfly

To accomplish an FFT butterfly, the signal processor and its special function unit must fetch two complex data points (and possibly a complex rotation vector), perform a complex multiply and two complex adds, and store two complex data points. Figure 12 shows the actual operation of the butterfly.

However, to perform the complex operations described above, the current processors must perform all real operations. The complex multiply becomes four real multiplies and two real adds, and the complex adds become two real adds each. Thus, the optimum structure to perform the FFT butterfly would have four parallel real multiplier and two real adders performing the complex multiply, and four real adders performing the complex adds (see Figure 13).

All signal processors must emulate the FFT butterfly structure in Figure 13, either by furnishing all the hardware, by recursive use of a single multiplier, or the software. Assuming that the purely software method would be both clumsy and slow, only the first two methods will be discussed. Four multiplier structures will be discussed as means of accomplishing the the FFT butterfly.

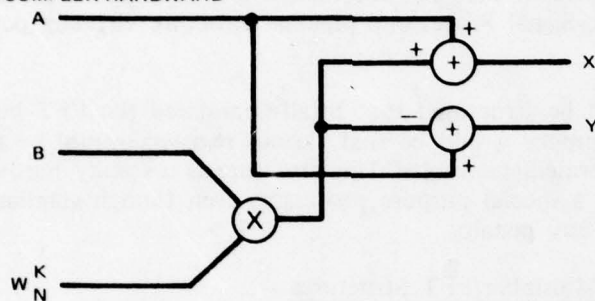
3.3.2 Multiplier/FFT Structures

The simplest structure is a single multiplier with two input latches to latch in the 16-bit operands, a 16 x 16 multiplier, and two 16-bit output latches to hold a double precision product. This multiplier could be constructed from the AMD 2 x 4 multiplier chips or the TRW 16 x 16 multiplier chip.

(a) COMPLEX ALGORITHM

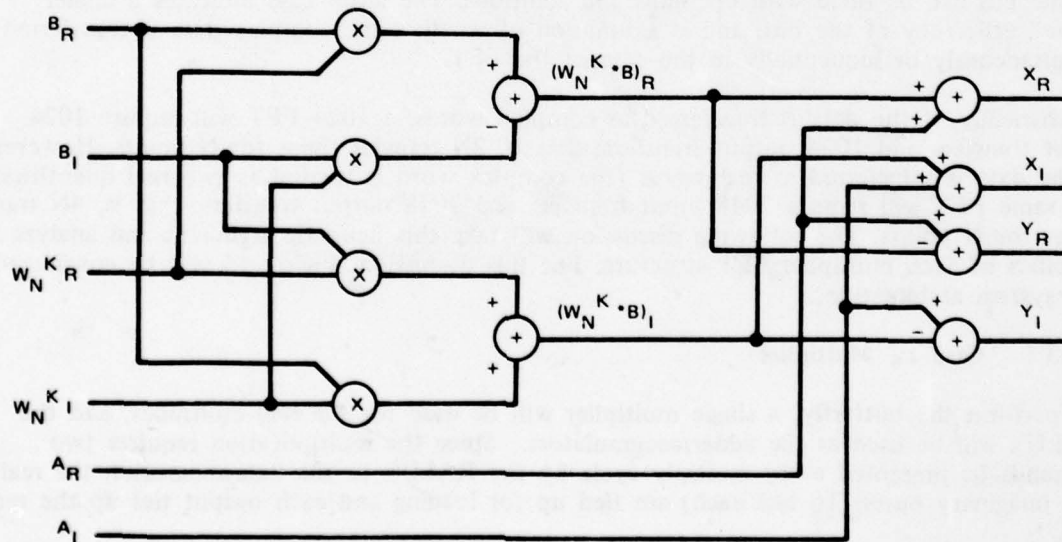
- FETCH A AND B
- FETCH W_N^K (IF NEW ROTATION VECTOR IS NEEDED)
- MULTIPLY $W_N^K * B = C$
- ADD $A+C = F$
- STORE F IN X
- SUBTRACT $A-C = G$
- STORE G IN Y

(b) COMPLEX HARDWARE



78453-3

Figure 12. Complex Butterfly



78453-4

Figure 13. Hardware Implementation of FFT Butterfly

An advancement from the simple structure is the addition of an accumulator that can handle addition of two 32-bit products; thereby performing the real or imaginary portion of the complex multiply. This structure is called a multiply/accumulator. Currently, a version is available from TRW that can handle 12 x 12 bit multiplication. Although the TRW product is insufficient, it is a step in the right direction.

A further advancement would include the holding registers necessary to perform the whole complex multiply without multiple operand fetches and stores. Figure 14 shows such a structure. The rotation vector and data point can be loaded directly into input latches; the four multiplies can be pipelined through the single multiplier; the partial products can be accumulated and held in latches; and the complex product can be outputted in a single clock time. No present product is known that can accomplish this structure on a single chip; however, the Raytheon Micro-Signal Processor's pipeline structure virtually performs this operation.

The final advancement would be structured that totally emulated the FFT butterfly structure in Figure 13. The only difference would be that various registers would be necessary to hold A, B, and W, as well as, intermediate results. This structure is a totally hardware approach; therefore, the unit would be a special purpose processor, even though standard multipliers could be performed without any penalty.

3.3.3 System Impact of Multiplier/FFT Structures

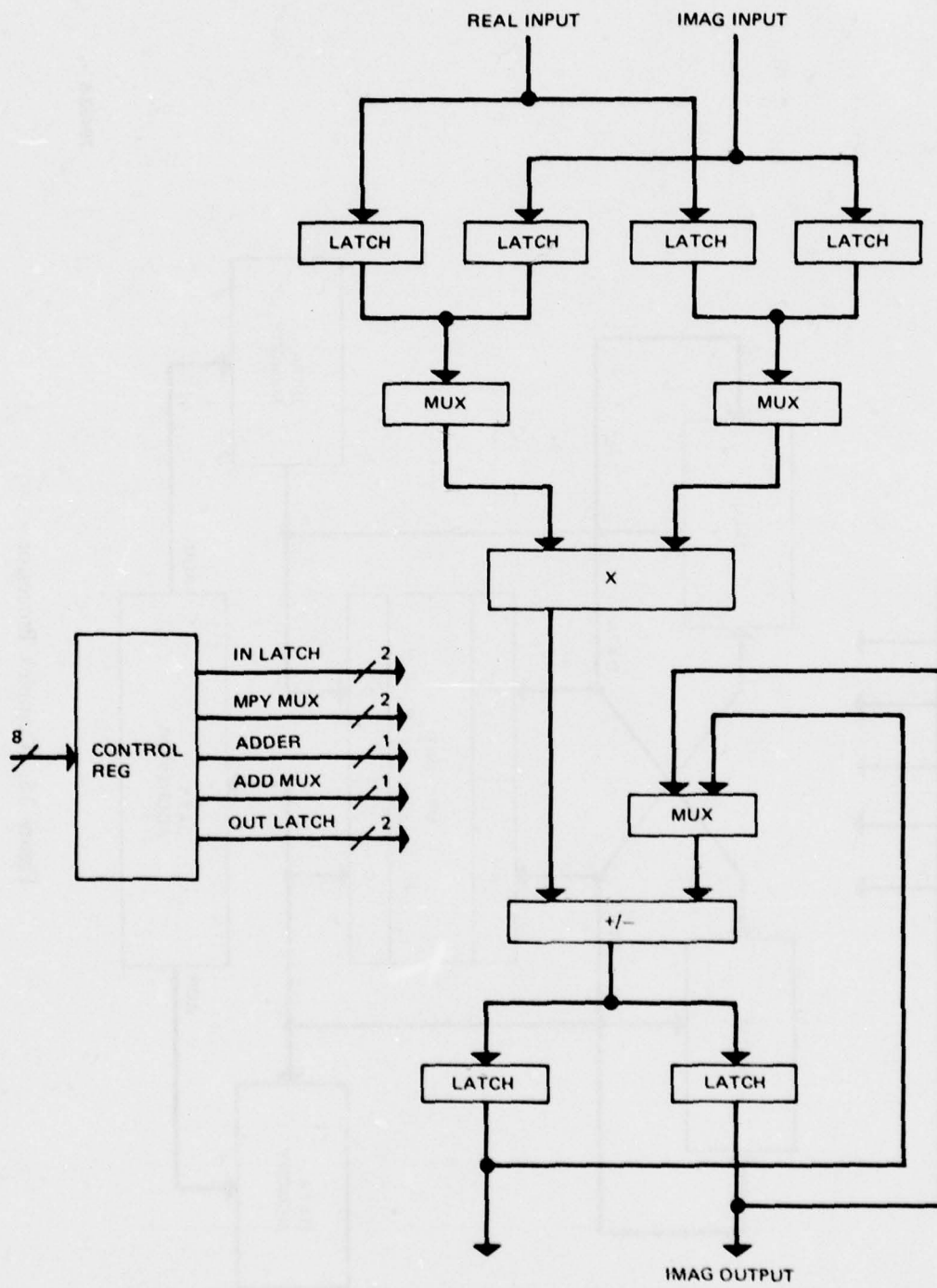
Each of the structures discussed above will be analyzed herein with regard to their impact on bus efficiency and speed. As discussed in Section 3.2.3, a goal of a processor should be 100 percent bus efficiency; however, this efficiency concept must be extended to include a statement about the types of bus traffic. Obviously, the bus can be filled with partial products and incompleated solutions (that is, shuffling intermediate data around to accomplish a task), or the bus can be filled with operands and solutions. The latter case indicates a higher "true" efficiency of the bus, and is a function of whether the complex data is transferred simultaneously or sequentially in the case of the FFT.

Heuristically, if the data is transferred as complex words, a 1024 FFT will require 1024 input transfers and 1024 output transfers; that is, $2N$ transfer times for N points. However, if the data is transferred as real words (the complex word is treated as two real quantities), the same FFT will require 2048 input transfers and 2048 output transfers; that is, $4N$ transfer times for N points. The following discussion will take this heuristic argument and analyze the specifics of each multiplier/FFT structure. For this discussion, Figure 15 will be considered the system architecture.

3.3.3.1 Case 1: Multiplier

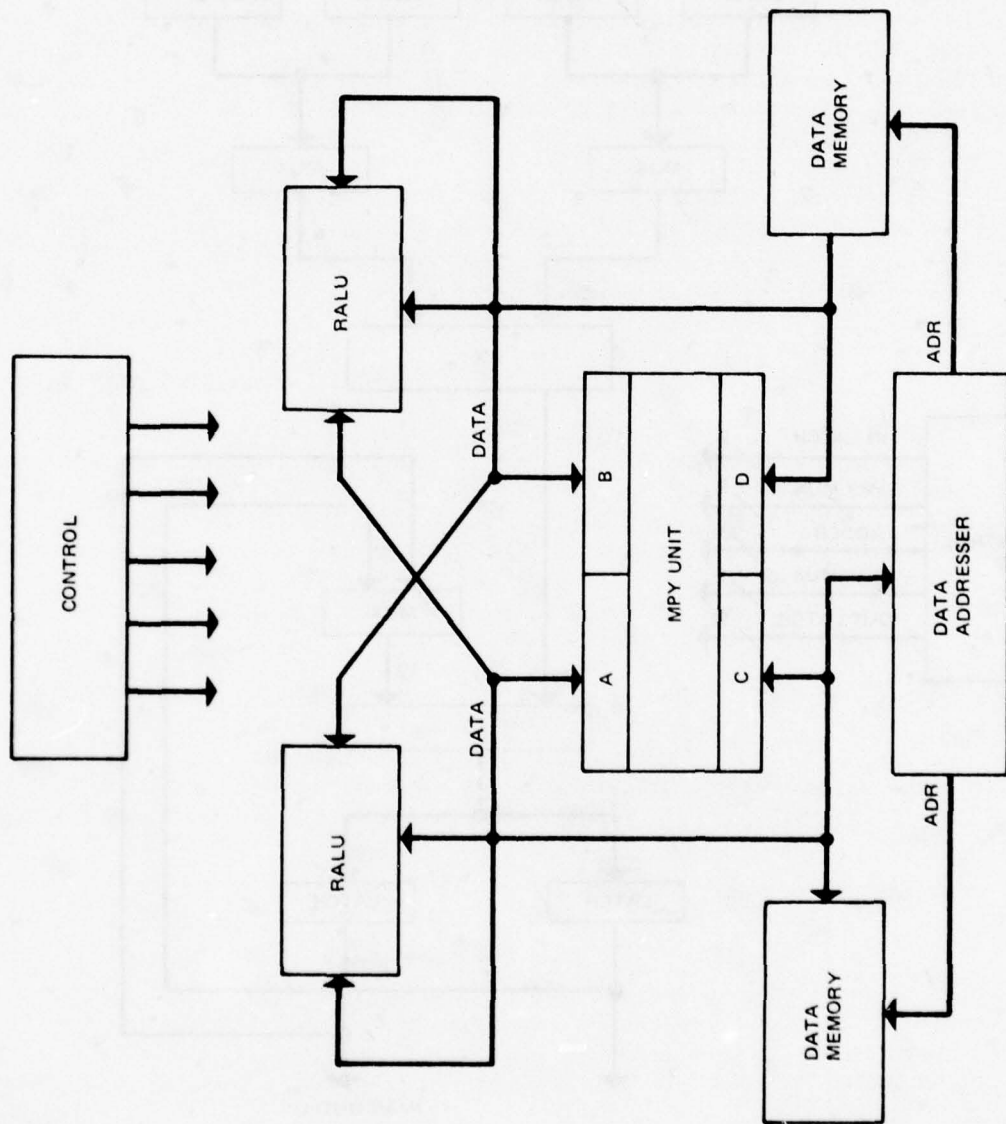
To perform the butterfly, a single multiplier will be used for the real multiplies, and the RALU's will be used as the adder/accumulators. Since the multiplication requires two operands be presented every multiply cycle by the RALU's or the data memories, the real and imaginary buses (16 bits each) are tied up for loading and each output ties up the real bus.

In addition to the bus traffic to load and unload the multiplier, two or three complex reads are necessary to set up the butterfly by putting the operands into holding register, that is, the registers on the RALU. Finally, two complex writes are necessary to store the output of the butterfly.



78453-5A

Figure 14. Multiplier with Accumulator and Holding Registers



78453-6

Figure 15. Complex Processor

Table 5 has been included to estimate the bus activity in clock cycles. The multiply time is assumed to be one or two clock cycles. From the cycle totals, the bus is busy about 70 percent of the time; however, two-thirds of the bus activity is shuffling data to and from the multiplier. Furthermore, overlapping of butterflies would be virtually impossible; therefore, the bus and multiplier must remain unused during part of the cycle. It is concluded that such a system would be inefficient in performing the FFT butterfly.

Table 5. Bus Activity as a Function of the Multiplier/FFT Structure

Operation Being Performed	Bus Activity in Clock Cycles							
	Case 1		Case 2		Case 3		Case 4	
	Bus Active	Bus Free	Bus Active	Bus Free	Bus Active	Bus Free	Bus Active	Bus Free
Complex Operand Reads	2/3	—	2/3	—	*	—	*	—
MPY Input	4	—	4	—	1/2	—	2/3	—
MPY Operation	—	4/8	—	4/8	1†	4/8	—	1/2
MPY Output	4	—	2	—	1	—	—	—
Intermediate Adds for Complex MPY	—	2	—	—	—	—	—	2
Complex Adds and Word Writes	2	—	2	—	2	—	2	—
Total Per Column	12/13	6/10	10/11	4/8	5/6	4/8	4/5	3/4
Total Cycles per Case	18/19 to 22/23		14/15 to 18/19		8/9 to 12/13		7/8 to 8/9	

*Complex words go directly to Multiplier

†Additional complex read during Multiply operation (does not increase total cycles)

3.3.3.2 Case 2: Multiply/Accumulator

As in Case 1, a single multiplier is employed, and the RALU is used as the operand holding registers; however, the intermediate adds necessary to complete one-half of the complex multiply are done in the accumulator.

Once again, the bus traffic is split between operand fetching and multiplier loading and unloading. As indicated in Table 5, the bus is busy about 60 percent to 70 percent of the time and 60 percent of the bus activity is the movement of operands to and from the multiplier. Overlapping of butterflies would again be quite difficult, and the bus and multiplier have idle time. Although the accumulator with the multiplier is an improvement over a simple multiplier, this system is still inefficient in performing the butterfly.

3.3.3.3 Case 3: Multiplier/Accumulator with Holding Registers

A single multiplier is used; however, the operand holding registers and accumulators are included so that the complete complex multiply can be done without intermediate data being placed on the data bus. The complex multiplier and multiplicand go directly to the multiplier holding registers, and the third complex word goes to the RALU's during the multiplier operation, thereby not increasing the total time to perform the butterfly.

Except for the movement of the complex product from the multiplier to the RALU so that the two complex adds can be done to finish the butterfly, all of the bus traffic is the fetching and storing of data in the data memories. The bus is active approximately 50 percent of the time; therefore, if two multiplier units of this type were employed, the overlapping of butterflies could be accomplished, resulting in approximately 100 percent bus efficiency. Using the overlapping process, the multipliers could be kept busy full-time.

This approach to the multiplier special function unit is a significant improvement over both cases 1 and 2. This system would be quite efficient in performing the butterfly operation.

3.3.3.4 Case 4: Multiplier/FFT

Multiple multipliers are used, all holding register for the three complex operands are in the unit, and all accumulators are included. Essentially, the rotation vector and the two complex operands are directly loaded into the multiplier unit, the complex multiply is performed, the two complex adds are performed, and the outputs are loaded back into the data memory.

All the data bus activity is dedicated to loading and storing operands. The bus is active 50 percent of the time, and as in case 3, 100 percent efficiency could be accomplished by overlapping in time if two multiplier units were employed.

Obviously, this approach represents the most efficient approach to performing the butterfly; however, this efficiency can only be accomplished with dedicated hardware. The system design must ultimately decide between the minimal differences between the performance of the units in case 3 and case 4.

3.4 COMPLEX PROCESSOR

A detailed review of the multiplier special function unit cases has revealed that the more sophisticated units - the multiplier/accumulator with holding registers and the multiplier/FFT - can permit extensions/modifications to the RALU structures that will impact the data addressing function. Although the modifications are minor, two different complex processors have been identified - one if the simpler multiplier units must be used, the other if the more complex units are available.

This section has been included to describe the processor architectures from a fairly high level. Within this section, the first complex processor to be discussed will have a multiplier or multiplier/accumulator, and the second will have the more sophisticated multiplier functions.

3.4.1 Processor I (See Figure 16)

3.4.1.1 Data Processors

This processor uses two real processors, or, more appropriately, RALU's to perform the complex arithmetic dictated by the problem set. Each real data processor is a 16 bit RALU, able to perform arithmetics, logicals, etc. in a single instruction cycle. Therefore, the two real processors can perform the full complex add or subtract function in a single instruction if they are worked in tandem.

3.4.1.2 Data Addresser

The data addresser is a single 16-bit processor RALU which must be able to add, subtract, increment and compare. In the configuration shown, the addresser can furnish two 16-bit addresses per clock cycle to the data memories; however, only one new address can be calculated during that period. This calculation limitation is not a hinderance for the problem set herein discussed. A third port on the data addresser is tied to one of the data buses so that a data word may be used as a data address such as in the case of a ROM table look-up.

3.4.1.3 Data Memories

The data memories will include both temporary and permanent storage, i.e. RAM and ROM. To support complex processing, one memory will be for the real operands; the other, for the imaginary operands.

3.4.1.4 Multiplier

The input latches are connected as shown in Figure 16. Because the complex multiply requires a multiplication of two real operands and two imaginary operands, the crisscrossing of the "real" bus to the imaginary processor and vice versa is necessary. The crisscrossing is also desirable if the processor is to be used in an array fashion.

The output latches hold the product of the input words until desired. The most significant bits are latched in the C Latch and attached to the real bus. This latch is the only one used in most cases. When double precision products are necessary, the D latch holds the least significant bits and is attached to the imaginary bus, thus, the imaginary bus becomes the lower bits bus when double precision arithmetic are being performed.

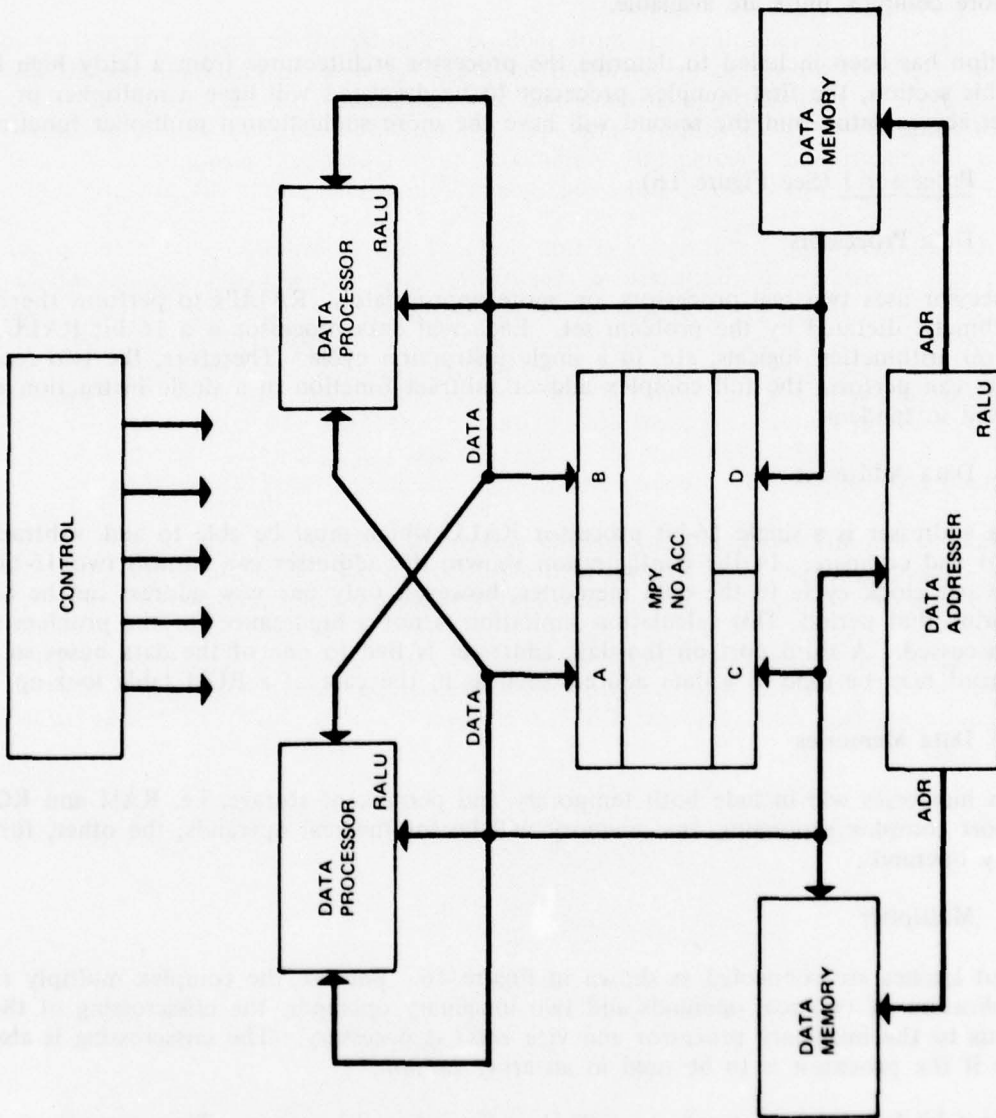


Figure 16. Processor 1

78453-21

All the latches, both input and output, are assumed to be independently latched.

3.4.1.5 Control

The control structure, as shown in Figure 17, contains a single Instruction Addressing Unit (which will be described in a subsequent section) that addresses the microinstruction memory. The microinstruction memory has a total of 109 control bits and a maximum of 4096 words. The control bits are, as follows:

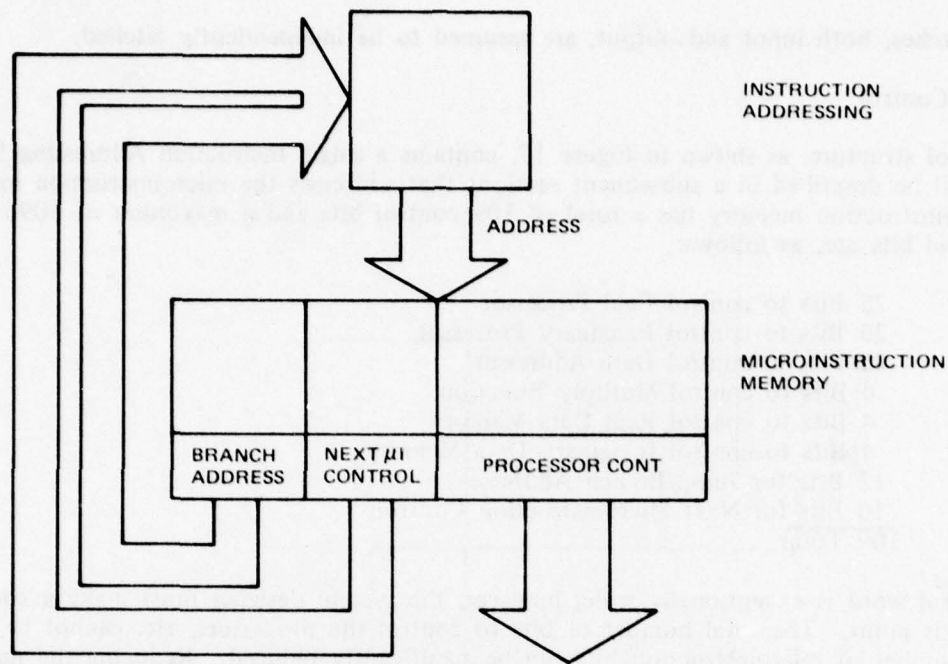
25 Bits to control Real Processor
25 Bits to control Imaginary Processor
25 Bits to control Data Addresser
4 Bits to control Multiply Function
4 Bits to control Real Data Memory
4 Bits to control Imaginary Data Memory
12 Bits for Jump/Branch Addresses
10 Bits for Next Microinstruction Control
<u>109 Total</u>

This control word is exceptionally wide; however, the system designer must make a compromise at this point. The total number of bits to control the processors, etc. cannot be lowered, but the number of microinstruction bits can be significantly reduced. Reducing the number of microinstruction bits, simply means that a high degree of decoding must be accomplished either within the processor or in an external ROM/PROM. The decoding operation takes time. The decision must be based on the time available. If speed is the goal, then the amount of decoding must be minimal. Thus, the control section here has opted for speed.

Because the microinstruction word is extremely wide, it is assumed that the microinstruction register is part of each function being controlled, i.e., the control registers are within the data processor, etc.

The Instruction Addressing (IA) unit contains the flag logic that is necessary. There are three sources of status or flag returns in the complex processor – the real and imaginary processors and the data addresses. Each of these processors can return four bits; this may be a problem for the flag logic provided on the IA unit. Expansion may be necessary in some cases; however, this is unlikely for the given problem set.

The control unit must be able to furnish microinstructions to the data processors and data addresser at the minimum instruction completion rate. Since these functions have been defined earlier in this section as having single clock instructions, the control unit must be able to supply instructions every clock cycle. If that instruction rate can be maintained, then no instruction buffer or FIFO register is necessary or even desirable. The buffer or FIFO causes problems in algorithms with a high degree of jumps such as the pulse classification task. Before a jump or branch can be accomplished, the FIFO must be cleared, or a fast-address-around loop must be included.



78453-7

Figure 17. Complex Processor Control Unit

3.4.2 Processor II (See Figure 18)

3.4.2.1 Data Processor/Addresser (DP/DA)

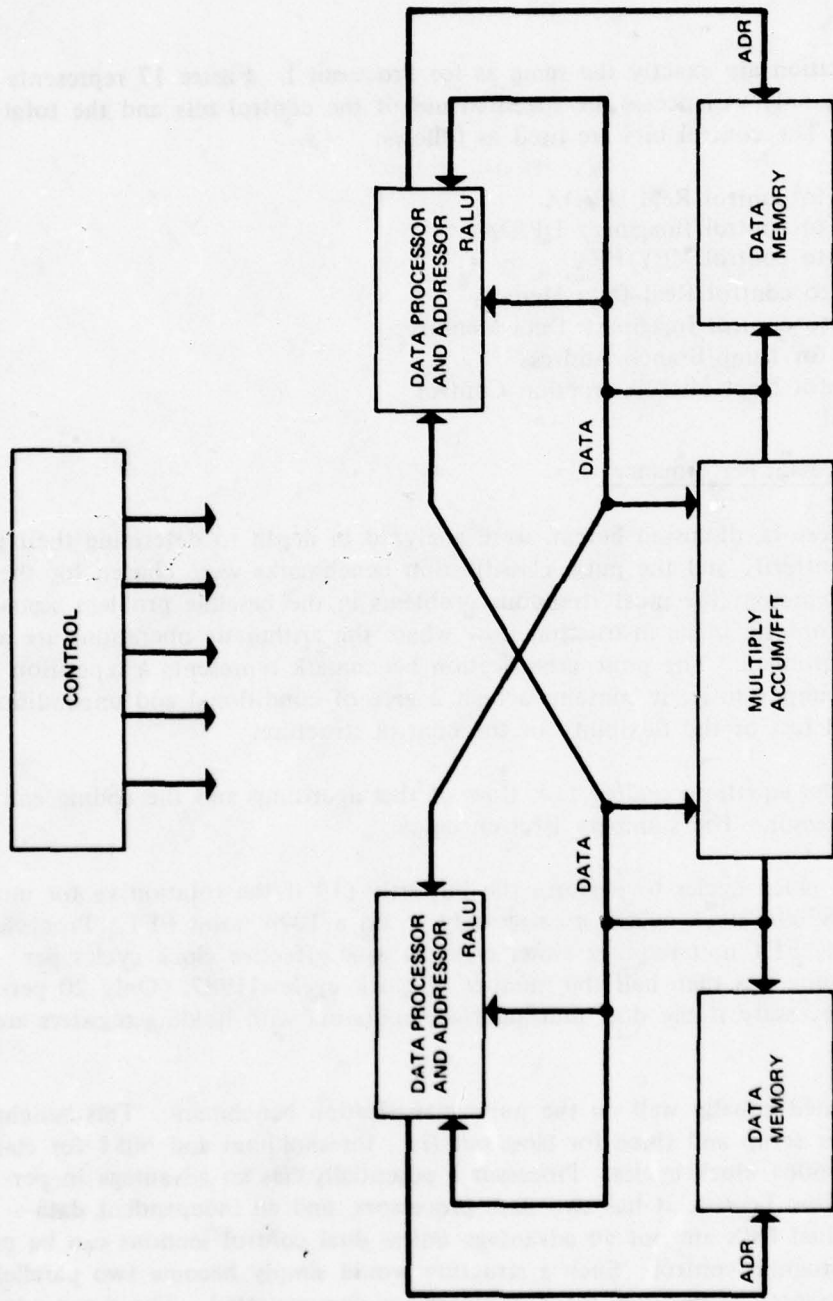
By providing a more sophisticated multiplier special function unit, the need for a separate data addressing unit is obviated because the ALU of the data processor is virtually unused during the FFT butterfly operation. The remaining problems in the benchmarks are much less difficult or strenuous from the ALU's point-of-view. In fact, the remaining problems require only one RALU or data processor and one data addresser.

The data processor/data addresser is explained in depth in a subsequent section. The structure is essentially the same as the data processor from Processor I with circuitry added to perform data address incrementing and with an additional data address register and port.

The dual function DP/DA is able to perform processing functions such as complex add or subtract and increment an address simultaneously or to calculate and furnish two 16 bit addresses every clock cycle. Furthermore, because the DP/DA functions share the same register stack, there exists an intrinsic ability to transfer data to the address port for a ROM table look-up.

3.4.2.2 Data Memory

Same as Processor I.



78453-8

Figure 18. Processor II

3.4.2.3 Multiplier/FFT

The multiplier for this processor is capable of performing fully complex arithmetic as well as real and double precision arithmetic.

3.4.2.4 Control

The principles of operation are exactly the same as for Processor I. Figure 17 represents the control structure. The only variance is the specified use of the control bits and the total number of bits necessary. The control bits are used as follows:

- 26 Bits to control Real DP/DA
- 26 Bits to control Imaginary DP/DA
- 10 Bits to control MPY/FFT
- 4 Bits to control Real Data Memory
- 4 Bits to control Imaginary Data Memory
- 12 Bits for Jump/Branch Address
- 10 Bits for Next Microinstruction Control
- 92 Total

3.4.3 Complex Processor Performance

The two complex processors, discussed herein, were analyzed in depth to determine their performance. The FFT butterfly and the pulse classification benchmarks were chosen for the analysis because they represent the most strenuous problems in the baseline problem scenario. The FFT is extremely orderly in its instruction flow where the arithmetic operations are a preponderance of the problem. The pulse classification benchmark represents a repetition of arithmetics, but, more importantly, it contains a high degree of conditional and unconditional jumps, which is a good test of the flexibility of the control structure.

Appendix A contains the equations and/or task flow of the algorithms and the coding and timing of the two processor. The summary is given below.

Processor I requires 17 clock cycles to perform the butterfly (19 if the rotation vector must be loaded); therefore, 89086 clock cycles are necessary to do a 1024 point FFT. Processor II with a single multiply/FFT unit requires either eight or nine effective clock cycles per butterfly; thereby, needing less than half the number of clock cycles-41987. Only 20 percent more cycles are necessary if the dual multiplier/accumulators with holding registers are employed.

Both processors performed equally well on the pulse classification benchmark. This benchmark requires seven cycles for setup and three for close-out (i.e., thresholding) and 6051 for classification. The total is 6061 clock cycles. Processor I potentially has an advantage in performing pulse classification because it has two data processors and an independent data address; however, the dual DP's are not an advantage unless dual control sections can be provided for testing and program control. Such a structure would simply become two parallel processors. Both processors can perform the benchmarks as demonstrated. The speed advantage of Processor II is purely a result of additional hardware, which is probably justified in the case of the FFT butterfly.

3.5 DATA PROCESSOR/DATA ADDRESSER

Within this section, a full description of the Data Processor (DP) and Data Addresser (DA) architectures, as well as, introductory words on the design rationale for the general structure will be included. These structures will not be discussed as integrated circuits although some reference may be included if a design rationale is only clear in the IC context. Specific IC trade offs will be in the technology section of this report.

3.5.1 Design Rationale

Early in the design effort, it was noted that similar structures for the DP and DA functions could be employed if the multiplier/FFT structure was considered independent of the Data Processor. Each function, DP or DA, has a need for a number of high-speed, on-chip registers and an Arithmetic/Logic Unit (ALU) structure. Because the general structures were similar, a more detailed look was warranted. Below is a capsule of the register and ALU needs of each function.

3.5.1.1 Registers

The Data Addresser, described as part of the complex processor section, is a highly utilized function requiring the same high speed that the DP requires. The problem set forces the signal processor to address data operands at a very high rate; therefore, it is incumbent on the processor to calculate its data addresses quickly, forcing the need for on-chip registers.

The registers must store the current address of the operand being fetched, the starting address of the operand string to be utilized, the maximum or ending address of the operand string, and the incremental values or delta addresses. An incremental value is used to determine the steps through the operand string, and there may be need for more than one incremental value if the addressing is complex. To further complicate the problem, if double indexing or higher indexing is advantageous, register space is necessary for all the start, maximum, current and delta addresses. To satisfy the double index need, a minimum of 8 is dictated, and 16 registers would be nice.

The register needs of the DP are very straight forward. Operand storage, intermediate results storage, and, depending on the multiplier special function unit, multiplier operand storage. In every algorithm coded to date, the maximum number of registers utilized has never exceeded six, even with the most inefficient multiplier structure.

A final comment on the registers is necessary. The registers should be Multiport RAM with two read-ports and one or two write-ports depending on the multiplier for ease of operand fetch and storage in the registers.

3.5.1.2 Arithmetic/Logic Unit

The DA function requires only the most basic arithmetics to be able to complete its tasks—

addition
subtraction
increment/decrement (± 1)

The ability to test addresses for the maximum must be available. The test requires simple subtraction and the generation of a test flag to force a jump/branch, or decrement in the loop counter.

The DP must have a sophisticated ALU with full arithmetics, logicals, and shifts. The generation of test flags for data dependent operations and signals for carry generate and propagate for extended precision arithmetic must be available.

3.5.1.3 RALU Structure

The conclusion drawn from the above discussion is that a RALU structure is indicated. The DA function forces the highest need for registers, and the DP requires the more sophisticated ALU; however, neither requirement forces an untenable deviation from the needs of the other function. If anything, the RALU is a slight overkill for each function.

3.5.1.4 Additional Comments

As discussed in Section 3.4, the RALU structure for the DP/DA will be controlled by a wide instruction word with little or no decoding on the chip. This constraint has been applied because it offers the highest speed and maximum flexibility in the timing cycles, thereby, allowing fast single cycles and multiple cycles if necessary.

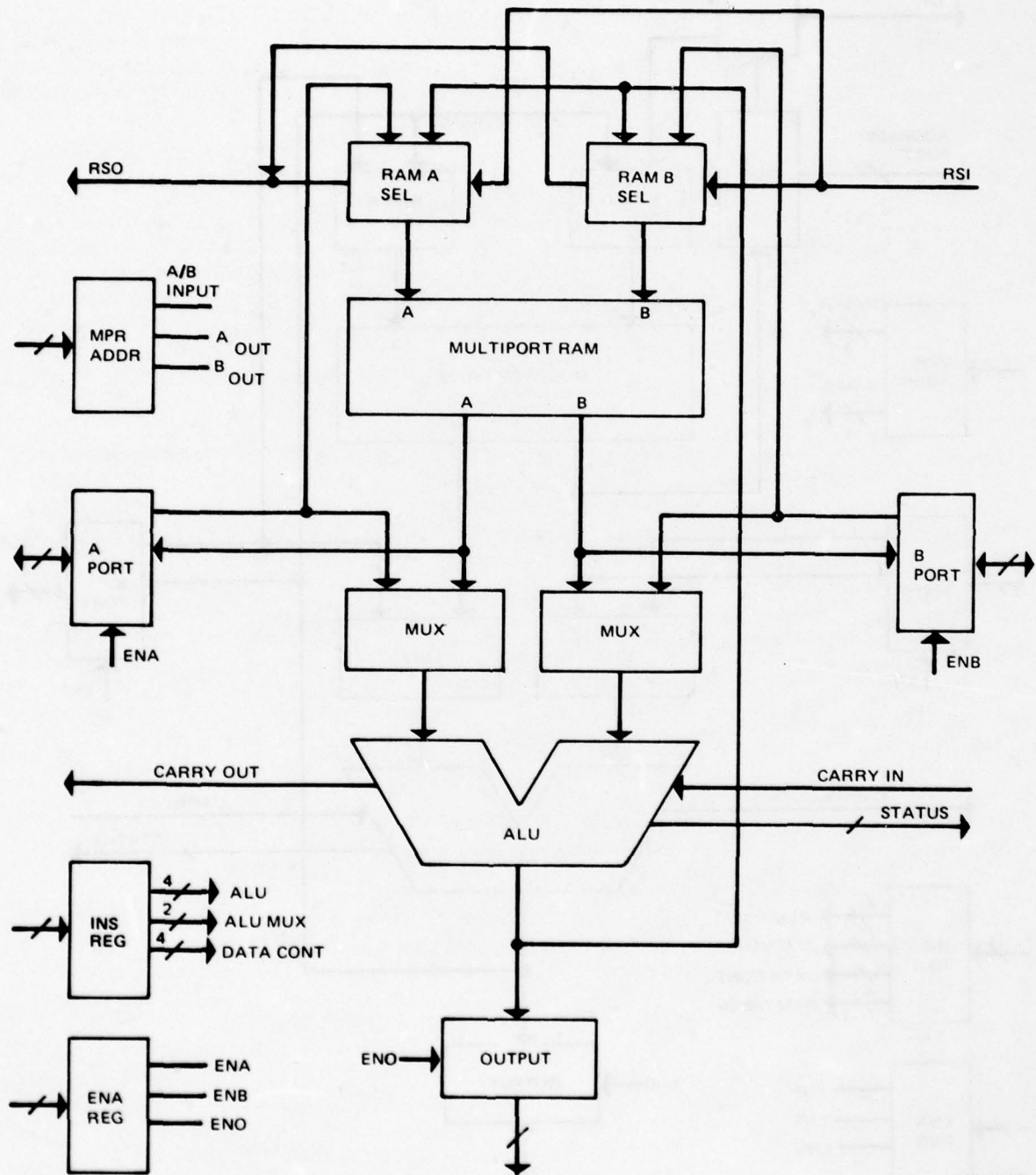
To minimize the total chip count of the signal processor, the instruction words are latched onto the chip and held in instruction registers. In other words, no external registers are needed for instructions. The rationale is simply that external registers are inefficient because their low gate-to-pin ratio requires many additional chips. By placing them on the RALU, the number of I/O pins on the RALU is unaffected, and the gate count is only slightly increased.

Finally, all the ports are latched and tristated to minimize external multiplexers. Since the data bus/data address bus are system limiters, it was concluded that the fewer the number of multiplexer, the faster the bus could operate.

3.5.2 Two DP/DA Structures

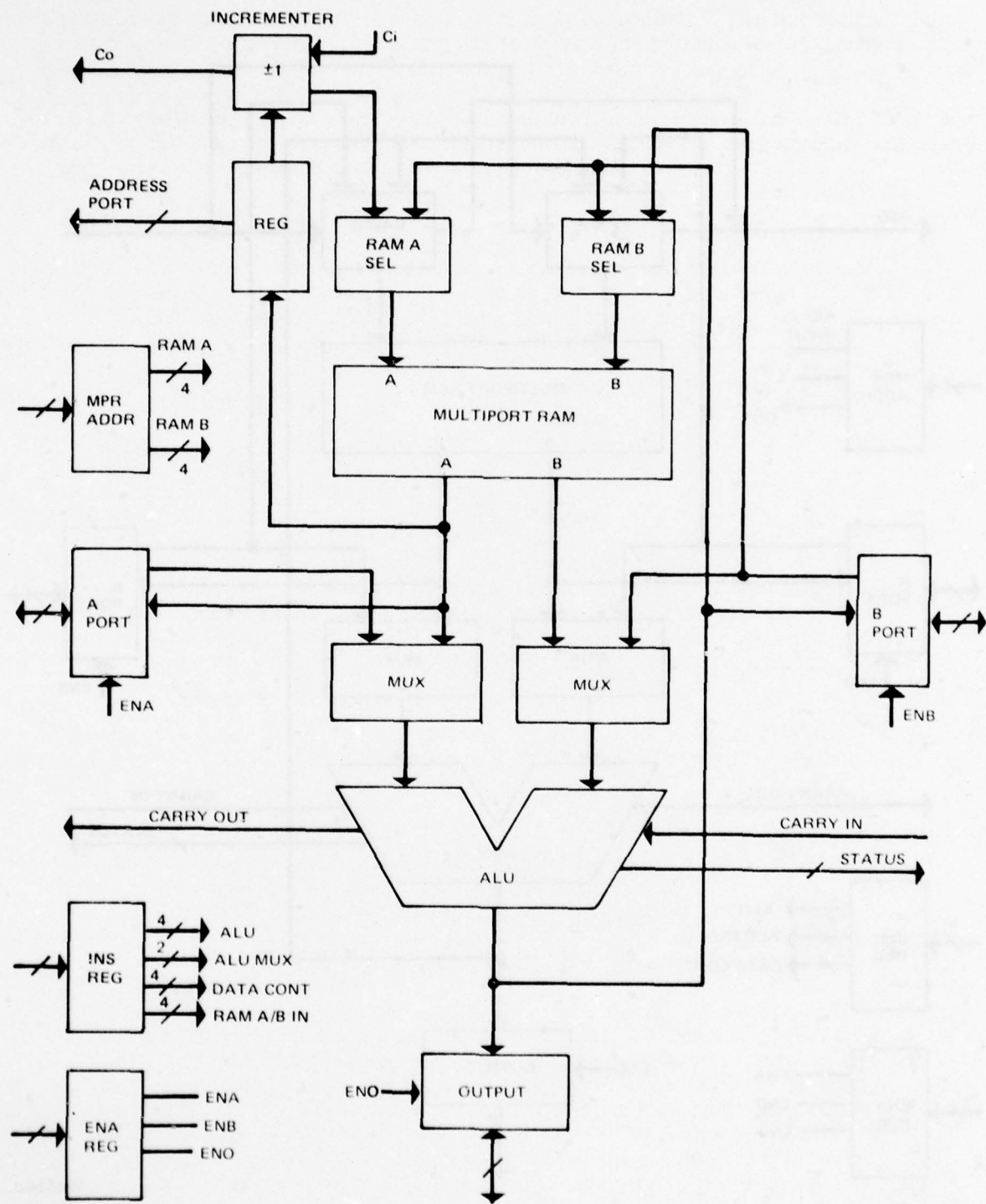
Depending on the multiplier special function unit, variations in the specific DP/DA architecture are indicated. The simpler multiply functions, discussed in the complex processor, required a whole unit be dedicated to data addressing. Each DP and DA unit has the RALU structure described above, that is, a full function ALU, a three Port MPR and 3 Bidirectional I/O ports (see Figure 19).

The more sophisticated multiply functions, also discussed earlier, minimizes the use of the ALU in the DP; thus, the functions of DP and DA can be combined because the DP/DA is used for addressing and calculating addresses during the FFT butterfly. By combining the functions, additional features are necessary on the DP/DA to support the addressing when the ALU & I/O ports are being used during processing. An address incrementer with increment/decrement and pass capability and an additional unidirectional port for addressing must be added. Furthermore, the MPR requires an additional write port so that addresses may be incremented and written back into the MPR simultaneously with data being processed in the ALU and being written back in the MPR (see Figure 20). Only in this case is a full four-Port MPR required.



78453-9B

Figure 19. Processor I RALU



78453-10A

Figure 20. Processor II RALU

3.6 INSTRUCTION ADDRESSING

The control function for the complex processor consists of a microinstruction memory and an instruction addresser. The instruction addresser (IA) includes a microsequencer, a loop counter, an interrupt control unit, and flag logic. The IA will furnish a 12 bit address to the microinstruction memory which will control the DP's and DA, set up to the IA for the next microinstruction, and provide the jump/branch addresses. The next sections will be devoted to explain the IA architecture.

3.6.1 Program Control Unit

The program control unit (PCU) is indigenous to all stored program computers and is often called the microsequencer (a la 2909). The PCU is shown in Figure 21. The heart of the PCU is the address multiplexer and register. Under the control of the IA instruction register, the flag logic, and the interrupt logic, the address MUX acts as a "traffic cop", selecting the next microinstruction address from 4 sources:

1. Program Counter
2. LIFO Stack
3. External Input
4. Interrupt Address.

The program counter generally contains the "next address" in its register. During normal operation, the program counter simply is incremented by 1 and steps through program. The output of address multiplexer is increment (actually +1, +2, or pass) and stored in the program counter. When a branch operation is being initiated, the program counter contents are fed to the LIFO Stack as the branch return address.

The LIFO (last in-first out) stage is a group of registers that are 12 bits wide which generally store the branch return address. The LIFO is a RAM when a branch return is necessary.

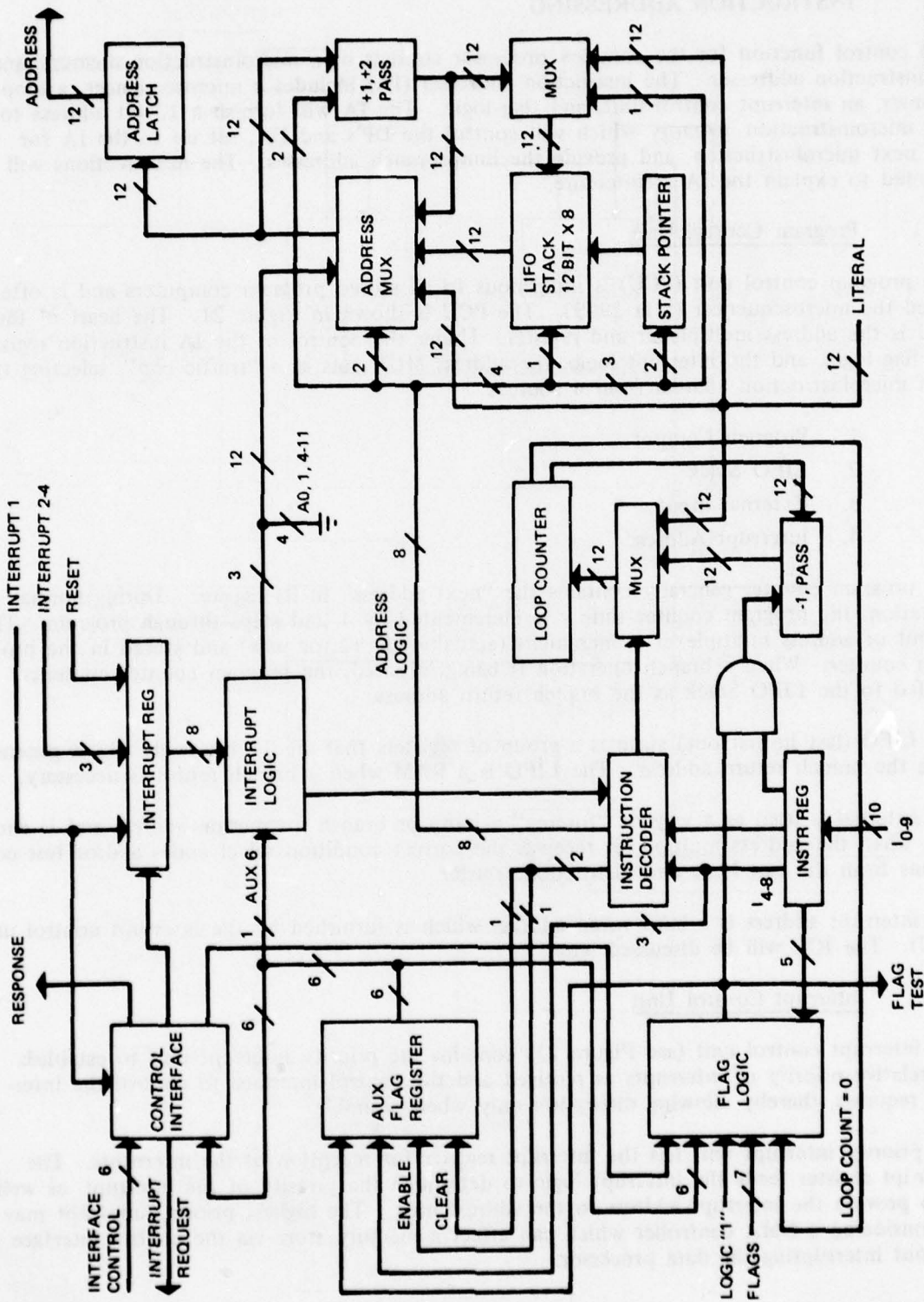
The external is used as a way of "forcing" a jump or branch instruction address and is chosen only when the address multiplexer receives the correct condition select codes and/or test conditions from the flag logic and instruction register.

The interrupt address is a hard wired address which is furnished by the interrupt control unit (ICU). The ICU will be discussed later.

3.6.2 Interrupt Control Unit

The interrupt control unit (see Figure 21) contains the priority interrupt unit to establish the relative priority of interrupts as received and the control interface to control the interrupt requests, thereby allowing disruption only when desirable.

The priority interrupt unit has the interrupt register for reception of the interrupts. The interrupt register feeds the interrupt logic to determine the priority of the interrupt, as well as to provide the interrupt address to the address mux. The highest priority interrupt may be considered a DMA controller which can affect a memory store via the control interface without interrupting the data processor.



78453-11A

Figure 21. Instruction Addressing Function

The control interface handles interrupt requests and provides control to the priority interrupt unit. It is controlled by the auxiliary flags and the high priority interrupt line which is reserved for DMA loading from the system I/O. The final function of the control interface is response to the higher level processors in an array configuration. In other words, the higher order control and response is handled by the control interface for array coordination.

3.6.3 Flag Logic

The flag logic (see Figure 21) is necessary so that test flags may be used to control the next address given to the microinstruction memory. The external test flags include the carry bit, overflow, sign and ALU equals zero received from any combination of the DP's and DA. Furthermore, the loop counter provides a zero indication which may be used to stop a "DO" loop. (Further discussion will be given in the loop counter section.) Auxiliary flags have been included to extend the limited input (seven flags) to the flag logic.

The flag logic and auxiliary flags control the loop counter's incrementer, the external flag test and the interrupt control unit; however, that control can partially be modified by the condition selects conditions furnished by the next instruction control word sent to the IA instruction register from the microinstruction memory.

3.6.4 Loop Counter

The loop counter (see Figure 21) provides a simple way to control the looping of repetitive routines, and it represents the only departure from the very fundamental control provided in most basic microprogrammable processors. The loop counter receives a literal from the external input which sets up the loop count. Each clock cycle, depending on the flag test and the IA instruction register, the count is either decremented or passed from the counter output to the counter input undisturbed; therefore, at the end of each pass through a routine, the beginning instruction of the routine is addressed and the loop count is decremented. When the count is zero and the end of the routine is reached, the loop is ended.

This structure is quite simple and could be replicated any number of times to allow automatic control of nested loops as in the FFT algorithm, pulse classification algorithm and any algorithm that requires a number of passes through a fixed routine. In the current structure only one has been included because LSI gate count constrains the number of loop counters that are advisable.

3.7 ARRAY PROCESSING

Either complex processor, discussed earlier, is suitable for us as an array processing element or controller in a parallel array multiprocessor. The rationale for array processing is simply to have a number of computers applied to a single task; thereby, multiplying the computation power. The multiple computer systems may be divided into two classes: (1) Single Instruction Stream/Multiple Data Stream (SIMD) systems, referred to as parallel processors and (2) Multiple Instruction Stream/Multiple Data Stream (MIMD) systems, called multiprocessors. Historically, signal processing problems have been proposed for parallel processors; however, data dependent algorithms, such as associative search and pulse classification, are extremely difficult.

Multiprocessors systems have a collection of relatively independent processors sharing a common memory and set of I/O devices. The processors must contend for access to the memory and I/O which makes the multiprocessor architecture slow for signal processing tasks, requiring high I/O rates. The ability to easily share data operands is a desirable feature of the multiprocessor systems.

By approaching the array processor problem from the point-of-view of the signal processing problem set, the parallel processor architecture with a limited ability to share and pass data between nearest neighbor processors is highly desirable. Heretofore, such an approach would have been limited by the sheer bulk of the array elements; however, current LSI technology affords new potential for a "mixed" approach.

The complex processors, shown in Figures 16 and 18 show that one data I/O port of each the real processor and the imaginary processor is removed from the data bus of the complex processor and freed for use in data transfer to the nearest neighbor array elements.

A port is also made available for data flow from a control processor element via the broadcast bus. Processor I requires that the broadcast bus be tied to the data buses to permit the proper data flow during multiplier operation. Processor II is able to free up the ports of the processors; therefore, the broadcast bus does not need to fight for contention with an internal array processor data bus.

3.7.1 Array Processor Element

A system of four array processor elements and a control element is shown in Figure 22 to represent a parallel array multiprocessor. One processor acts as a controller to this system, and the remaining four are configured as two 16-bit RALU's which provide arithmetic and logic capability for the processor. Associated with each RALU is a data memory consisting of both PROM and RAM. Each RALU is responsible for addressing its own memory. The RAM provides a total of 1K 32-bit words of storage for dynamic data, while the PROM holds 512 32-bit constants used in performing the FFT algorithm.

Each of the RALU's is independent of the other on that they may perform different instructions. This allows efficient complex number arithmetic to be performed. In executing algorithms involving complex values, real numbers are stored in one data memory and imaginary numbers in the other. A path is provided between the RALU's to allow transfer of data. Each of the RALUs provides one 16-bit bidirectional bus to a neighboring array processor so that interprocessor data transfers may take place. The real RALU provides a connection to the higher-order 16 bits of the system broadcast bus. The lower-order 16 bits are connected via transceivers to the imaginary RALUs memory data bus. The bus transceivers are controlled by a one-bit field in the microinstruction memory.

The multipliers are connected in parallel and have a bidirectional port to each memory. Their operation is alternated by the microcode which controls them. This is necessary because they are fully independent circuits, and it is fruitless to attempt to load or empty them simultaneously. The capabilities of the multipliers include the following: multiply, multiply accumulate, and butterfly in both real and complex formats; double precision scalar multiply.

The microinstruction memory supplies all instruction fields to the processor hardware. The fact that all elements of hardware can be controlled by a single microinstruction makes the array processor a horizontally micro-coded machine. This enhances its speed and makes each instruction very powerful.

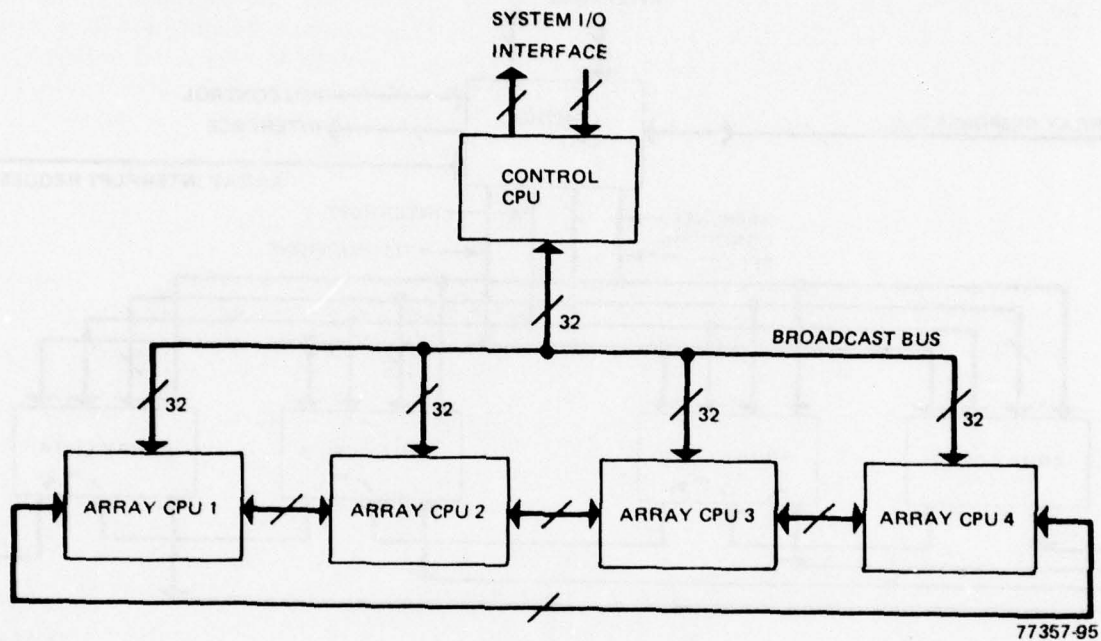


Figure 22. Parallel Array Multiprocessor

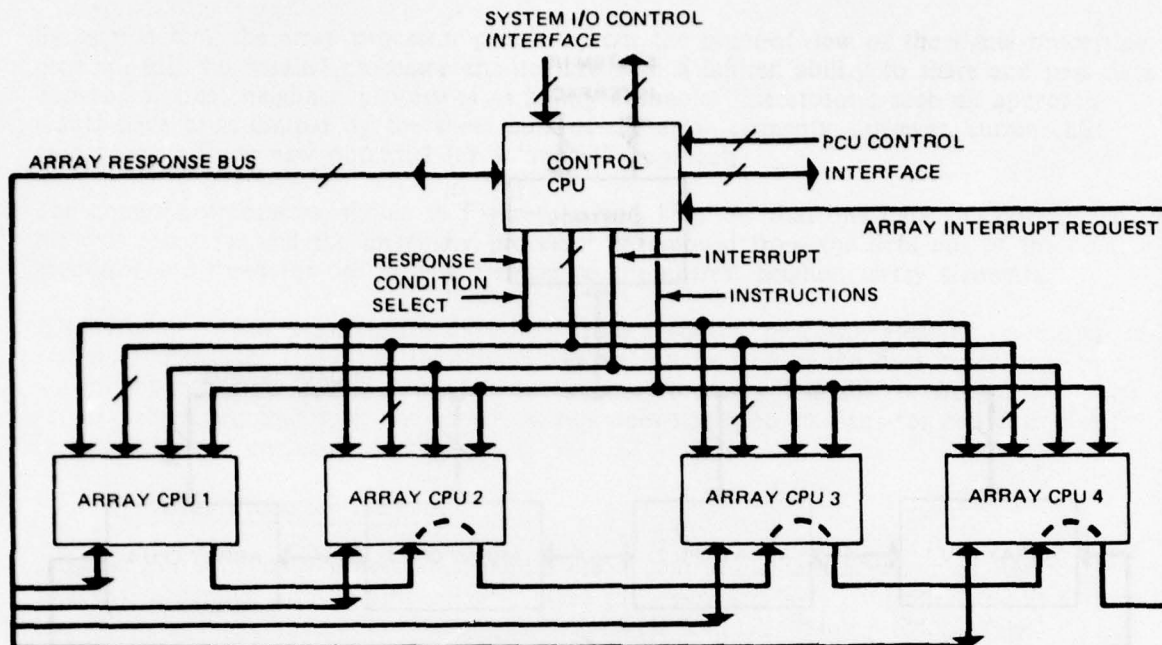
The processor's speed is enhanced further by the fact that the RALUs and the multipliers contain *instruction registers* that allow *instruction fetch and execution* to be overlapped.

The *microinstruction memory* is addressed by the program counter which is located in the controller. The microinstruction memory supplies a literal field to the controller which is generally used as a branch address. An alternate branch address can be determined from data received from the controller via broadcast bus. This is the mechanism by which the array processor can receive task assignments from the control processor. The controller has flag testing logic onboard and accepts up to eight flags from the RALU and multiplier chips. A total of 12 flags are available from the devices, however, so an FPLA should be used to combine some of the flags. The FPLA logic is controlled by a microcode field from the control PROM.

A specialized control interface is incorporated into the controller. The control interface is connected directly to the array control buses shown in Figure 23. The interface logic is illustrated in some detail in the IA discussion.

3.7.2 Parallel Array Multiprocessor

The efficiency of uniting the array processors to perform parallel tasks is dependent on their ability to operate synchronously. For this reason, all processors in the system operate from the same clock source. If they were not synchronized, complex and time consuming software routines would be required for intercommunication, and hardware would have to be provided to accomplish handshaking.



77357-96

Figure 23. Control Detail

The broadcast bus is used both for issuing commands to the array elements and for getting data into and out of the array. Its dual use is made practical by the fact that task initiation ties up the bus only for the amount of time required to issue a single program address to the responding array elements — one clock cycle. Efficient control of the processors in the array depends upon a mechanism for selectively issuing commands to the array elements and for determining their program status. The control structure indicated by Figure 23 allows this to be done.

The "instruction" control signal identifies whether or not the broadcast bus currently contains an instruction. For a processor element to accept an instruction from the bus, it must first be in a state of attention, either by having ended a previous task segment or by way of interrupt from the controller. The "interrupt" signal is used by the control processor to issue interrupts to the array. The control processor is able to determine which elements of the array are in a state of attention by means of a general purpose flag register which resides in each of the array processors. The controller may simultaneously sample the flag registers of the array elements by means of the "response" signal which is available from each element as shown in Figure 23. The flag registers contain a number of flags and any of them can be gated to the response bus by way of the "condition select" lines.

The controller accepts a single interrupt from the array. The interrupt line is daisy-chained throughout the array elements, and the assignment of priority is established by the way in which the chain is routed. As it is necessary for the control processor to determine the source of an interrupt, each array processor's flag register includes the interrupt flag.

The response logic described only operates when the controller is not issuing a command to the array (i.e., when the "instruction" signal is not asserted), so the controller cannot simultaneously examine flags and issue commands; as a practical matter, this is not a handicap. The reason for this is that the flag inspection logic has a dual use. Both instructions and interrupts to the array can be made conditional, so that it is possible to selectively apply them to the array. The response logic is instrumental to this purpose. The "condition select" lines control the condition by which each array processor determines whether or not an instruction or interrupt is intended for it.

One of the condition codes corresponds to "unconditional", that is, it specifies every element of the array. This is used when the entire array is to perform a parallel task. All but one of the remaining condition select codes specifies one of the flags in the array processors' flag registers; the "true/false" signal establishes whether the specified condition is the true or false state of the flag. It is thus possible to selectively issue commands to elements which, through previous program tasks, have set flags. The remaining condition code allows the controller to use the response bus to specify which array elements are selected; for this reason the response bus is bidirectional. The controller may then pick the responding elements by asserting the response line to which each is tied.

The control mechanisms described are extremely flexible and account for the ability to efficiently use the system in both parallel processor and multiprocessor modes.

SECTION IV

LSI TECHNOLOGY SUMMARY

4.0 INTRODUCTION

The status of LSI development is an everchanging scene. For a time, a given technology or several technologies will reign supreme in the marketplace, only to give way to new technologies or improved versions of the older technologies. This point notwithstanding, an attempt must be made to gather enough information about the state-of-the-art to determine whether a particular function is feasible as an LSI chip or must be made with a limited number of chips and chip types.

A survey of the technology has been made to get a rough picture of the present status of LSI technologies. From this survey, an attempt has been made to extract a list of macro-constraints which an LSI function can not exceed today or in the next one to two years. Included as the final section of this chapter are some methods and methodology for LSI development.

4.1 TECHNOLOGY SURVEY

This technology survey gives the present status of the technologies available for both custom LSI and memories. The current research in LSI technologies is to satisfy demands for greater function in the microprocessor area (custom LSI) and higher density and greater speed in all types of memories. The developments are related to economics: increased density, lower speed-power factors, larger wafers, and improved yield. The discussion will be separated into a section on custom LSI and on LSI/VLSI memories.

4.1.1 Custom LSI

The major characteristics of the current technologies that are available for custom LSI/VLSI applications are included in a number of brief description and are summarized in Table 6. Table 6 is an attempt to take the sometimes ambiguous data for the various technologies and alter it to some standard definition or measurement procedure; therefore, a description of the Table is included at the end of the technology discussions.

4.1.1.1 SiGate MNOS

The N-channel MOS uses the ion-implantation, SiGate and doped oxide technology, with a 100 crystal orientation process. The N-channel device with higher electron mobility and low threshold voltage means faster operation while using less power. At higher substrate doping, it allows the channel to be shorter, resulting in reduced input capacitance and reduced size. With its low power, high mobility, and packing density, NMOS, i.e. compatible and even desirable for custom LSI technology.

4.1.1.2 N-Channel Depletion-Enhancement Mode SOS-MOSFET

The NMOS/SOS evolved out of the conventional bulk SiGate NMOS approach where it is fabricated on the insulating sapphire substrate. The advantage over the bulk NMOS is observed by virtually eliminating the parasitic capacitance and by increasing surface carrier mobility which gives maximum current for a given geometry.

4.1.1.3 VMOS

The N-channel V-MOS transistor is formed along the slope of the V-groove, which is anisotropically etched into the surface of a silicon wafer. The process is a double-diffusion profile in the channel region under the gate, which effectively reduces the channel region to a micrometer. Compared with NMOS, VMOS technology saves about 40% in random logic area and lower speed-power product. This advantage makes VMOS attractive to be used in a broad range of memory devices.

4.1.1.4 DMOS

The planar-double-diffused MOS exhibits a short-channel characteristic which are obtained from a full-size device. The channel length is determined by the difference in lateral diffusion of two profiles. Effective channel lengths of less than 1 μm can be obtained independent of the photolithographic tolerances which limit channel length for conventional MOS fabrication. It appears that its performance advantage over a conventionally scale down device may be too small to make it worth considering at this time.

4.1.1.5 C^2L /MOS

The C^2L is a self-aligned silicon-gate CMOS technology where the gate completely surrounds the drain providing a transistor aspect-ratio which maximizes the transconductance-to-capacitance ratio thus allowing high speed on-chip. The C^2L device exhibits a factor of 3 improvement in packing density over standard CMOS and operates at frequency approximately 4 times faster than standard CMOS. The C^2L device requires 6 photomasks, one less than standard CMOS. In regard to custom C^2L LSI design, the only known source is not interested unless the volume is high (million units per year).

4.1.1.6 CMOS/SOS

The SOS/MOS technology evolved out of the conventional bulk-silicon approach. The silicon-on-sapphire (SOS) approach comes closest to these desirable features of high-speed performance at low supply voltages and with nanowatts of stand-by power dissipation. The MOS/SOS devices can be fabricated in a thin single crystal silicon film grown on the insulating sapphire substrate. The use of thin-film silicon virtually eliminates the parasitic capacitance which gives the highest speed with minimum power and circuit complexity. In addition, having the non-junction type isolation, it will improve its transient radiation resistance characteristics. Availability has been a consistent problem; however, for this technology.

4.1.1.7 First and Second Generation I^2L /MTL

First generation, integrated injection logic (I^2L) or merged transistor logic (MTL) is basically derived from direct couple transistor logic which utilizes a basic four-mask, double diffused bipolar process without junction isolation. Second generation I^2L /MTL gate is fabricated with

a new process/structure, which includes a matrix on the P^+ extrinsic base drive and implanted intrinsic base dose for the n-p-n transistor. While retaining the advantage of the first generation, it is designed to operate at greater speed with same injector current. The I^2L promises to plan an important role in LSI technology.

4.1.1.8 S^2L

The S^2L has a structure, topology, and characterization of integrated injection logic with a self-aligned double-diffused injector. The new structure, a lateral p-n-p transistor with effective submicron base width, can be realized, by using standard photolithographic techniques. The S^2L with higher injector efficiency and low parasitic capacitance results in a large fan-out capability, high speed and large noise margin. The packing densities are improved by factor of 2 over standard I^2L logic.

4.1.1.9 SFL

The substrate fed logic uses an approach designed primarily for LSI where high packing density and low power-delay is desired. The basic logic element is a multi-input, multi-output gate, formed in a single-base area by using several diffused collectors and several Schottky base contacts. It has been found that an overall improvement of 2.2 in packing density between SFL and I^2L technologies with the same tolerances can be obtained. It was noticed at maximum speed, SFL power dissipation is equal to standard I^2L logic.

4.1.1.10 SCHOTTKY I^2L

Schottky I^2L is a modified form of the substrate fed logic, differing from the earlier process in the extrinsic n-p-n base profile. Heavier boron doping in this region has lead to less charge storage so that minimum delay and power are reduced. The high performance of Schottky I^2L has been achieved with a structure designed for high yield by use of simple processing technique.

4.1.1.11 Up-Diffused I^2L

The "up-diffused" structure is fabricated in a fashion that Schottky diodes can be readily incorporated. With the addition of Schottky clamps between the collector and base of the n-p-n switching transistor, gate delay by factor of 5 and power-delay product by factor of 2 is achieved over standard I^2L . Another version is injected Schottky logic (ISL) currently under development by Signetics.

4.1.1.12 I^3L

The Isoplanar integrated injection logic (I^3L) technology emphasizes achieving high packing density and high performance by the use of various process innovation combined with topological design variation. A high performance has been achieved without the use of Schottky clamping, and the process is equivalent in complexity to any standard dual-layer metal bipolar technology. The packing density of I^3L is equal to NMOS technology, by using a two-level metal scheme.

4.1.1.13 Table 6 Description

Table 6 lists the bipolar and MOS technologies that are currently available or in development for custom LSI/VLSI applications. Table 6 was generated from the data received directly from various semi-conductor producers, from the literature search and from personal direct inquiries. The data specification supplied by semiconductor producers or journal reports are sometimes ambiguous and referenced to non-standard values. Therefore, data specifications were altered to a given standard value for ease comparison.

Table 6 contains 6 parameters which are most important for this technology survey study. They are as follows:

Gate Delay: For bipolar technology, a maximum intrinsic delay for a one and five-collector gate was listed. For MOS technology, a maximum intrinsic delay for fan out one and three was listed, at 5 volts power supply.

Power Dissipation Per Gate: It is static and dynamic power dissipation at nominal maximum frequency with +5 volts power supply. The nominal maximum frequency is defined as the average of maximum repetition rate at single and multiple load conditions.

Speed-Power Product: It is a product of gate delay times power dissipation per gate.

Gate Area per Square Millimeter: It is a random logic area with approximately 50% area assigned to interconnect and power busing.

Repetition Rate: It is a range of frequency of operation where the lower and upper end of the range is a function of the fan-out load.

All of the circuit technology listed in Table 6 are referenced to 5-7 um mask rules.

Table 6. Technology Survey

System Parameters Circuit Technology	Gate Delay Nanosec	Power Gate MWatts/ Gate	Speed Power Product Pico-Joules	Random Logic Gate Area Gates/ MM ²	Number of Mask	To Repetition Rate MHz
SiGate NMOS	8-30	.4-.5	4-12	100-150	6-7	8-30
DMOS	6-20	1.4-1.6	15-20	225	6	13-40
C ² L/MOS	6-40	1.25-2.5	15-50	270	6	6-40
NMOS/SOS	2.7-9	2.6-3	8.1-23.4	100-150	7	28-90
VMOS	5-20	.8-1	5-16	80-300	7	13-50
CMOS/SOS	3-20	1-2.5	7.5-20	150-250	7	11-80
FIRST GENERATION I ² L/MTL	25	.5	12.5	80-160	4	10
2ND GENERATION I ² L/MTL	4-8	5	20-40	60-120	6	30-60
S ² L	10	.4-5	4-50	170	5	25
SFL	20-30	.5	12.5	120-240	-	10
SCHOTTKY I ² L MODIFIED SFL	8	2-3	16-24	400	6	20
UP-DIFFUSED I ² L	2.5-3.5	5	12.5-17.5	100	6	70-100
ISL	2-5	3-7.5	15	100	6	50-125
I ³ L	4-5	5	20-25	250-300	6	50-62

4.1.2 LSI/VLSI Memories

There are several new and old LSI technologies that are competing for new generation of memories in range of 64 kilobits. Table 7 lists current memory devices and their performance. Charged couple device (CCD) memories with 65 kilobit level for block storage application, are serially accessible and slower and more difficult to use RAMs. The only reason to use CCD is the price advantage in order of two to one over RAMs. A VMOS device that has large potential density and low power consumption is available in 64K read only memory into 175 mils square chip. Another competitive technology is HMOS using scaled-down 2-um rule and high density, lower power MOS RAM.

In future, one or two years away, VLSI memories with 256K bit capabilities will emerge out of production lines. One of the problems in VLSI is the interconnection on the chip. This problem may be reduced by use of double-poly or three-layer metal interconnection in conjunction with an innovative logic.

Table 7. Memory Device and Performance

Device Type	Density/ Capability Bits	Speed NSEC	Process	Manufacturer
DYNAMIC RAMS	16K	150-300	N-MOS	FAIRCHILD
	16K	100	I ³ L	NTT
	65K	150-300	N-MOS	AMI
	65K	150	V-MOS	AMI
STATIC RAMS	4K	150	N-MOS	—
	4K	50	H-MOS	INTEL
	4K	55	V-MOS	AMI
	8K	150	N-MOS	MOSTEK
ROMS	64K	80	V-MOS	MOSTEK
	64K	250	V-MOS	AMI
	64K	300	H-MOS	INTEL
CCD	64K	200-500		FAIRCHILD

4.1.3 Figure of Merit

The complexity of the MOSFET and bipolar technology over the past several years has created the hard task of standardizing sensitive parameters. Those parameters are used for comparison in the LSI technology survey. One of the very important parameters is the power-delay product which indicates how much power is necessary for a given gate to operate at its maximum frequency at a given supply voltage and fan out condition. For example, a power delay product of I^2L exhibits linear relationship over extrinsic region (slow gate delay) and power dissipation using injector current times collector voltage swing (less than 1 volt). Obviously, the power-delay product parameter will be low and impressive. In order to be comparative with the rest of the LSI technologies in this survey, an intrinsic gate delay, which is a delay due to minority carrier charge-storage effect, and 5 volts supply voltage was used to determine the power-delay product. As for CMOS technology, where the only static power dissipation factor was used to generate low power delay product, a given nominal maximum frequency is included in power-delay product. Therefore, I^2L and MOSFET technologies can be easily evaluated and compared.

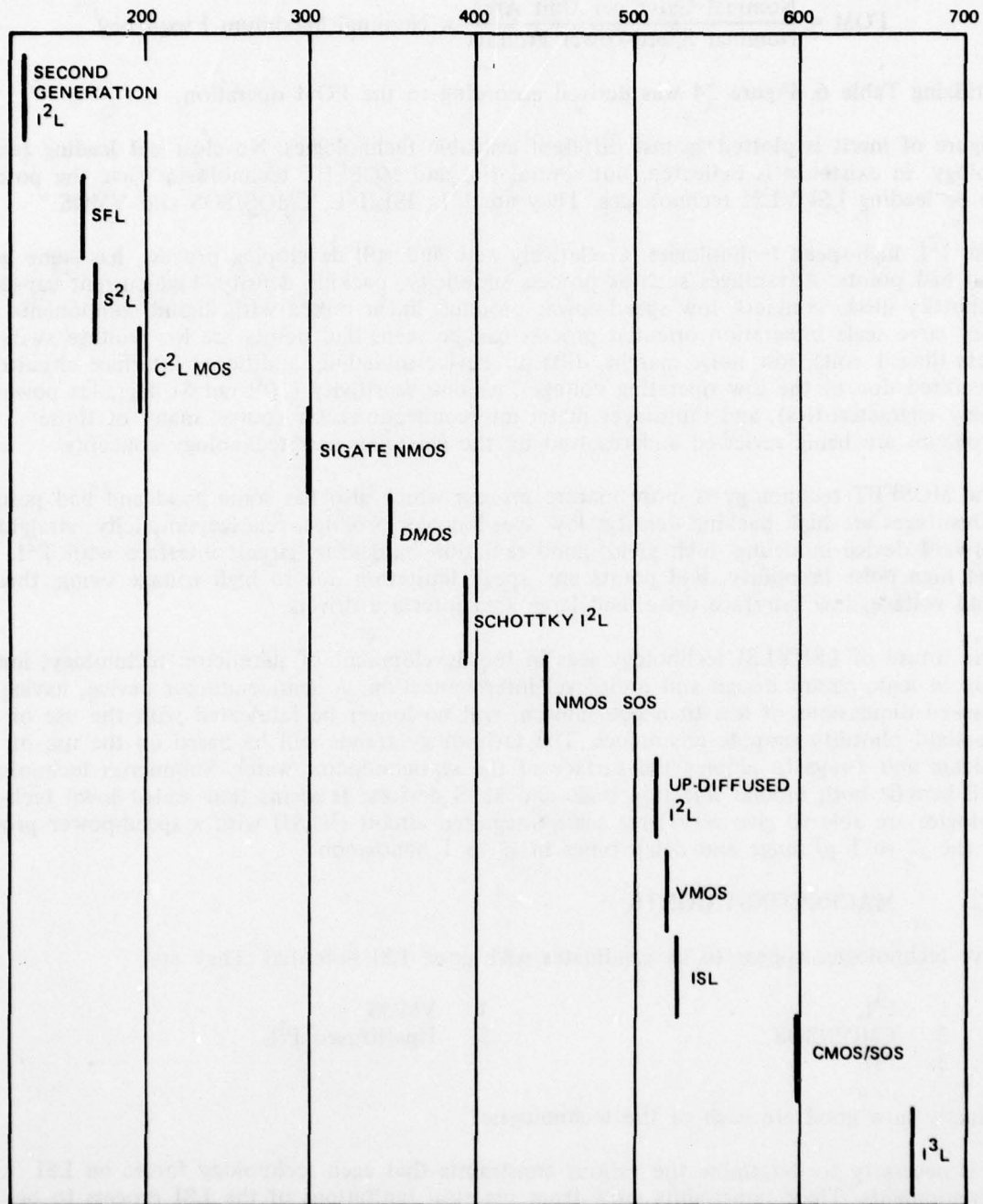
These factors are not the only ones that could or should be included to determine the relative merits of the technologies in Table 6. A number of system considerations should be included before a technology is chosen for a given application. A fuller discussion of this point is included in Section 4.3. However, a figure of merit will be defined using the speed-power product, gates per unit area, and maximum frequency as defined for Table 6. Since lower speed power products are preferable, this factor will go in the denominator. Higher gate densities and maximum frequencies of operation are more desirable; therefore, these factors will be in the numerator.

In an attempt to rate these technologies for custom LSI, a very simplistic approach was taken: Utilize the factors from Table 6 to create a figure of merit (FOM). Two factors are immediately discernible, as significant from an LSI point of view - speed-power product and gates per unit area.

Speed-power product has long been used as a measure of the "goodness" of a technology. It is used to measure technologies against constant speed-power lines on the now-famous gate-delay, gate-power chart. On that chart, the lower a technology's speed-power product, the better the technology is considered.

To evaluate LSI potential, a second factor must be added to the evaluation-gates per unit area. High gate packing density is crucial if a technology has any hope of approaching LSI/VLSI potential, because the integrated circuits will be smaller, thereby lowering the probability of failure due to surface defects on the wafer. From a system application point of view, if a lower-speed technology can provide sufficient parallelism of operation and can fit in a smaller circuit area, than a higher-speed technology, it may be more advantageous to go with the slower technology. The decision will be partially based on the true maximum speed of the slower technology. If the parallelism is too high or the safety margin in the performance of tasks is too low, the faster technology may be chosen. Thus, the maximum frequency of the technology must be included in any FOM.

FIGURE OF MERIT



78453-12

Figure 24. LSI Technology, vs Figure of Merit

Assuming the approach taken for Table 6 has provided some standardization in these factors, then the figure of merit will be of the form:

$$\text{FOM} = \frac{\text{Nominal Gates per Unit Area}}{\text{Nominal Speed-Power Product}} \times \text{Nominal Maximum Frequency}$$

Utilizing Table 6, Figure 24 was derived according to the FOM operation.

Figure of merit is plotted against different available technologies. No clear cut leading technology in existence is indicated, but several I^2L and MOSFET technologies have the potential to be leading LSI/VLSI technologies. They are I^3L , ISL/ I^2L , CMOS/SOS and VMOS.

The I^2L high-speed technologies, a relatively new and still developing process, has some good and bad points. Advantages such as process simplicity, packing density, high-current capacity, Schottky diode contacts, low speed-power product, linear mixed with digital components and very large scale integration oriented process can be seen. Bad points are low voltage swing (less than 1 volt), low noise margin, difficult device modeling, additional interface circuitry (required due to the low operating voltage), gamma sensitivity (10^6 rad Si degrades power-delay characteristics), and multilayer metal interconnections. Of course, many of those problems are being reviewed and resolved by the emerging new technology concepts.

The MOSFET technology is more mature process which also has some good and bad points. Advantages are high packing density, low speed power product, relative simplicity, straight forward device modeling, high yield, good radiation hardening, circuit interface with T²L logic and high noise immunity. Bad points are: speed limitation due to high voltage swing, threshold voltage, low interface drive, and large area interface drivers.

The future of LSI/VLSI technology lags in the development of submicron technology, innovation in logic circuit design and multilevel interconnection. A semiconductor device, having masked dimensions of less than one micron, will no longer be fabricated with the use of standard photolithographic techniques. The technology trends will be based on the use of e-beam and x-rays to pattern the surface of the semiconductor wafer. Submicron technology will benefit both bipolar injection logic and MOS devices. It seems that scaled-down technologies are able to give very large scale integrated circuit (VLSI) with a speed-power product in the .2 to 1 pJ range and delay times in .5 to 1 nanosecond.

4.2 MACROCONSTRAINTS

Five technologies appear to be candidates with good LSI potential. They are:

1. I^3L
2. CMOS/SOS
3. ISL
4. VMOS
5. Up-diffused I^2L

Exactly how good are each of the technologies?

It is necessary to determine the various constraints that each technology forces on LSI developments. These constraints grow from practical limitations of the LSI process to be used. In essence, one must assess the ground rules of each technology in the areas of:

- a. I/O Pin limits
- b. Power dissipation limits

- c. Level of integration limits
- d. On-chip/off-chip gate delays
- e. Interface compatibility
- f. Maximum chip sizes.

Without becoming tutorial, each of the above ground rule areas are simple reflections of a given technologies ability to handle a function with LSI.

Virtually without regard to technology, the maximum practical package size for dual-in-line packages seems to be 64 pins. Larger sizes have not become popular. Leadless packs may increase this number to 80 or more pins; however, power dissipation must be considered when dealing with leadless packages.

Interface compatibility is almost always assumed to be TTL voltage and drive levels at the interface. Since the bulk of presently available commercial circuits have TTL compatibility, it remains a good ground rule that TTL levels be maintained for interface compatibility. This ground rule presents some problems for the MOS technologies which operate over a wider range of voltage levels and do not provide as much sink capability with normal output buffers. CMOS/SOS and VMOS are each capable of meeting the voltage levels with no difficulty since each technology is now powered by 5 volts; however, the drive levels require much larger output drivers which increases surface area of the chip. The bipolar technologies require some modification of the output devices from their basic on-chip devices, but the difference is small.

Maximum chip size is dependent upon the surface defect density of the LSI process. Chip size, therefore, has a direct bearing on the yield of the process. Each technology has different tradeoff points where the chip size/yield curve becomes unprofitable. However, vendors are more comfortable in considering larger chips with their improved processing capability. A reasonable chip size limit is approximately 250 mils on a side, although the average size for LSI is approximately 170 to 200 mils on a side.

The limits of power dissipation, level of integration, and gate delay are the areas where the technology differ significantly. Using the data accumulated for Table 6, each of the five technologies – I^3L , CMOS/SOS, ISL, VMOS, and Up-Diffused I^2L – is capable of phenomenal levels of integration. The actually obtainable level of integration is lower than value predicted from Table 6 data because high functionality forces high on-chip interconnect and a large number of bonding pads. Table 8, therefore, has reduced the maximum values of gates by 60% to account for the interconnects and bonding pads. From the gate count, power dissipation levels were estimated, using the power dissipation extremes from Table 6.

In general, all the technologies are able to exceed 1000 to 1500 gates, CMOS/SOS, I^3L , and VMOS easily passing the 2000 to 3000 gate range. Power dissipation becomes the limiting factor on all of the technologies. A maximum for power dissipation should be 2 Watts or less. Although the dissipation of greater than 2 Watts can be handled with special packaging or cooling, the overall cost is generally prohibitive.

Thus, assuming this 2 Watt power dissipation limit for custom LSI, the practical levels of integration for the various technologies is, as follows:

- a. I^3L 400 to 2000 gates
- b. CMOS/SOS 800 to 2000 gates

Table 8. Comparison of LSI Candidates

Chip Size	100 X 100 Mils	200 X 200 Mils
Technology I ³ L		
Gates*	650 to 780	2575 to 3100
Power**	650mW to 3.9W	2.6W to 15.5W
Speed (Max)	4ns	4ns
CMOS/SOS		
Gates*	390 to 650	1550 to 2575
Power**	390mW to 1.63W	1.55W to 6.4W
Speed (Max)	3ns	3ns
ISL		
Gates*	260	1040
Power**	780mW to 1.9W	3 to 7.5W
Speed (Max)	2ns	2ns
VMOS		
Gates*	210 to 780	830 to 3100
Power**	170mW to 780mW	660mW to 3.1W
Speed (Max)	5ns	5ns
UP I ² L		
Gates*	260	1040
Power	1.3W	5.2W
Speed (Max)	5ns	5ns
*40% of maximum gate count indicated by Table 6 for each chip size – assumes high degree of inter-gate connections and bonding pads.		
**Depends on percentage of high-speed, high power gates.		

- c. ISL 270 to 700 gates
- d. VMOS 2000 to 2500 gates
- e. Up-diffused I²L 400 gates

The MOS technologies are definitely LSI candidates, and the bipolar can be if the lower speed functions are integrated in the very low power I²L and the high speed functions use the faster I²L variations, i.e., ISL and Up-diffused I²L.

The final area of limitation is gate delay – both on-chip and off-chip. In Table 8, all the technologies are capable of high on-chip maximum speeds; however, not shown on that Table is the fact that the off-chip delays for the bipolar technologies are 20% to 40% more than the on-chip, i.e., 3 to 7 nsec, and the off-chip delays for the MOS technologies are

more than 100% greater than the on-chip delays, i.e., 6 to 10 nsec. This off-chip gate delay may be critical in some system applications.

A summary of the macroconstraints for the technology says that the following parameters are limits for general LSI development:

- a. I/O pin limit — 64 for DIP, 80 or more leadless package
- b. Interface Compatibility — TTL voltage and drive levels
- c. Maximum chip size — 250 mils on a side, 200 mils practical
- d. Level of Integration Limit — 2000 gates
- e. Power dissipation limit — 2 watts
- f. On-chip gate delay — 2 to 5 nsec
- g. Off-chip gate delay — 5 to 10 nsec

4.3 THE TECHNOLOGY DECISION

For LSI to be effective in helping military systems perform their mission, the LSI must be chosen by balancing the system needs with the technological abilities of the LSI. For a system design approach to accomplish this balancing act, new methods are needed for analytically exploring design tradeoffs in the context of the multitude of LSI technological changes. This section will endeavor to discuss a methodology for selecting LSI from system needs. It should be noted, before any discussion begins, that every system requirement forces choices in technology which affects every other system requirement. Rather than capitulating to the seemingly insoluble *problem of system requirement interdependence*, it is hoped that the first order effects of technology on system requirements can be isolated so that the interdependence is manageable in our minds.

In the following section, system requirement categories will be presented along with the technology parameters that directly relate to the system requirement category as first order effects.

4.3.1 System Requirement Categories

4.3.1.1 Architecture

The architectural design of a system is to accomplish the system's mission with the technological tools available to the designer. The system architectural design is a trade-off process of allocating system functions between the hardware tools, the system programs (software) and the firmware use in microprogram subroutines. The overall system complexity can be reduced by selecting the proper hardware — firmware — software balance in the system.

From an LSI point-of-view, level of integration, gate delay, chip I/O, and testing have the most direct effect on the architecture.

4.3.1.2 Environment

The system is designed and required to be operational under various environmental conditions such as extreme temperature variations, humidity, vibration, shock, electromagnetic or nuclear radiation, high or low atmospheric pressure, etc. The chip packaging, the temperature range of the technology, radiation hardening limits and noise immunity may be used to decide if a technology can meet the environmental conditions it must operate in.

4.3.1.3 Physical Characteristics

The physical characteristics of a system are its weight, volume, power consumption and cooling requirements. Higher system speed generally reduces the physical dimensions of the circuit and packaging; however, these reductions result in greater heat generation and power dissipation necessitating improved cooling.

Parameters such as chip packaging, I/O, power supplies, and total chip power impact the system volume and weight. Power is impacted by the level of integration, gate dissipation, off-chip drives and the number of power supplies. Speed is impacted by the system architecture, level of integration, gate delay, number of I/O, off-chip gate delay etc.

Both the system enclosure and internal module designs are influenced by a number of system requirements. The enclosure is the buffer between the system and its operational environment, as well as supplying the cooling capability for the system. All the factors affecting the Environment and Physical Characteristic system requirements categories impact the system packaging.

4.3.1.4 Viability – Reliability, Availability, Maintainability and Survivability

A failure-tolerant system is designed to remain operational at some minimal performance level despite almost any malfunction. At the system level, the impact is redundancy in components, or at the subsystem level, the capability of diagnosing malfunction and reconfiguring system fault. Reliability, availability and maintainability of the system are directly affected by component and packaging technologies, circuit and subsystem design philosophy, and system architecture. Reliability is the probability that a system will perform its function for the duration intended. Availability is the system capability to be in operational condition whenever needed. Desirable system maintainability is to replace the faulty modules without significantly disrupting the system activity and keep down time to a minimum.

Survivability of the system is a protection of system hardware against nuclear effects; gamma and x-rays, neutron influence, and electromagnetic pulse.

The viability of a system is related to virtually all the technology parameters previously mentioned. The power dissipation, i.e., the level of integration and gate dissipation, and chip packaging are measures of the temperature the components will experience which is compounded by the environmental extremes. Reliability is greatly affected by the temperature of the components.

Maintainability is related to testability of the components, packaging and I/O pins. Availability is a measure of reliability and maintainability, that is, operational time to down-time. Survivability is related to noise immunity, input/output protection devices on the chip, radiation hardening, etc.

4.3.1.5 Cost

Systems are characterized by special environmental hardness and survivability requirements, size, weight, power constraints which require special design, manufacturing techniques and quality control. Consequently, the system cost is affected by these requirements. With advances in architecture, in LSI component manufacturing techniques, and design automation, the cost will remain relatively same but at the same time will increase the system performance.

4.3.2 Forced-Pair Comparison

The preceding section summarized, quite briefly, several system requirement categories which may be seen in the request for proposal for any major military system. The categories are an attempt to reflect the mission goals for the specific system. When the system designer begins the design of the system, he must prioritize the system requirement categories for the whole system, and often, for many subsystems and functions. After the requirements are prioritized, it may be seen that various functions could best be implemented by LSI, and more specifically, by new custom LSI. The LSI designer must now ascertain from the system designer what the system requirement priorities are before an intelligent decision can be made on the LSI technology to be used.

To aid the system and LSI designer, an empirical, and somewhat, subjective methodology has been developed to prioritize the system requirements. The methodology is called the forced-pair comparison. Using this method is fairly simple, and often, the results are startling to the designer. He will not realize his major limitations or requirements until he actually uses a forced-comparison of every category against all remaining categories. "Forced" means that a decision about that category in relation to the next category must be made.

The Method

The system requirement categories are enumerated by the system designer, such as

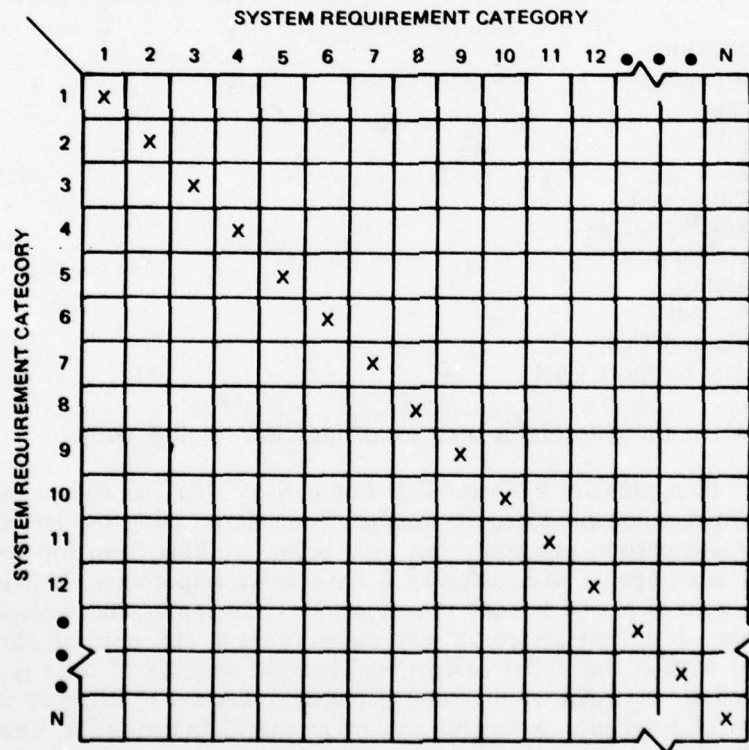
1. Architecture
2. Environment
3. Volume
4. Weight
5. Power
6. System Speed
7. System Packaging
8. Maintainability
9. Reliability
10. Survivability
11. Acquisition Cost
12. Logistic Support Cost

The number assigned to the category is used as an identifier at this point.

The system designer then prepares a Forced-Pair Comparison chart, as shown in Figure 25, which has the system requirement category number from above along the left vertical and top horizontal axis. The comparison procedure can now begin, working from top to bottom, row by row. Category 1 is compared with category 2 for relative importance. If 2 is more important than 1, a zero is placed in row 1, column 2 (1, 2) and a one is placed in (2, 1) as in Figure 26. Category 1 is then compared with category 3, 4, etc. until all the categories have been compared with all the other categories. Then the number of ones is counted across each row and entered to the right of the row. Category 1 has 6-1's, Category 2 has 7-1's, etc. From the count of total pins, a ranking can be arranged. In Figure 26, Categories 2, 3, 5, 7 and 11 are equally ranked. At this point, the procedure can be iterated to break the tie for equal ranking.

The system designer must then analyze his ranking with a final "reasonableness" test. Has the ranking procedure put various system requirement categories higher or lower in priority than they should be? Are some categories equally important, etc? The "reasonableness" test will reveal that the Forced-Pair Comparison method is somewhat subjective, but the method is useful in getting the system requirement categories in perspective, pointing out where the system tradeoffs should be made.

From the final ranking the most important IC parameters may then be discerned, thereby allowing the LSI designer to choose the proper LSI technology to perform the necessary function.



78453-13

Figure 25. Forced-Pair Comparison Chart

SYSTEM REQUIREMENT CATEGORY	SYSTEM REQUIREMENT CATEGORY												TOTAL POINTS	RANKING
	1	2	3	4	5	6	7	8	9	10	11	12		
1	X	0	0	0	1	1	1	1	1	0	0	1	6	6
2	1	X	0	1	1	1	0	0	0	1	1	1	7	1
3	1	1	X	1	0	1	0	1	0	0	1	1	7	1
4	1	0	0	X	0	0	0	0	0	1	1	0	4	9
5	0	0	1	1	X	1	1	0	1	1	0	1	7	1
6	0	0	0	1	0	X	0	1	0	0	1	1	4	9
7	0	1	1	1	0	1	X	1	1	0	0	1	7	1
8	0	1	0	0	1	0	0	X	1	0	0	0	3	11
9	0	1	1	1	0	1	0	0	X	1	0	0	5	8
10	1	0	1	0	0	1	1	1	0	X	0	1	6	6
11	1	0	0	0	1	0	1	1	1	1	X	1	7	1
12	0	0	0	1	0	0	0	1	1	0	0	X	3	11

78453-14A

Figure 26. Forced-Pair Comparison Example

4.4 REFERENCES

- [1] Paul Ou-Yang, "Double Ion Implanted V-MOS Technology", IEEE J., Solid-State Circuits, Vol. SC-12, pp. 3-9, Feb. 1977.
- [2] Alan Capell, "Process Refinements Brings C-MOS on Sapphire into Commercial Use", Electronics, pp 99-106, May 26, 1977.
- [3] Hung Chang Lin, Jack L. Halsor and Harry F. Benz, "Optimum Load Device for DMOS Integrated Circuits", IEEE J., Solid-State Circuit, Vol. SC-11, pp 443-452, Aug. 1976.
- [4] Michel J. Deccero and Thierry Laurent, "A Theoretical and Experimental Study of DMOS Enhancement/Depletion Logic", IEEE J. of Solid-State Circuits, Vol. SC-12, pp 264-270, June 1977.
- [5] Andrew G. F. Dingwall, Roger E. Stricker and Joseph O. Sinniger, "A High Speed Bulk CMOS C²L Microprocessor", IEEE J. Solid-State Circuits, Vol. SC-12, pp. 457-462, Oct. 1977.
- [6] Glen R. Madland, "The Future of Silicon Technology", Solid State Technology, pp 91-95, August 1977.
- [7] Bruce B. Roesner and Denis J. McGreivy, "A High Speed I²L Structure", IEEE Journal of Solid-State Circuits, Vol. SC-12, pp 114-118, April 1977.
- [8] John M. Herman, III, Stephan A. Evans and Ben J. Sloan, Jr., "Second Generation I²L/MTL: A 20 ns Process/Structure", IEEE Journal of Solid-State Circuits, Vol. SC-12, pp 93-100, April 1977.
- [9] F. Hennig, Hemraj K. Hingarh, David O'Brian and Peter W. J. Verhofstadt, "Isoplanar Integrated Injection Logic: A High-Performance Bipolar Technology", IEEE Journal of Solid-State Circuits, Vol. SC-12, pp 101-108, April 1977.
- [10] "The Multifacets of I²L", IEEE Spectrum, pp 29-36, June 1977.
- [11] J. L. Stone, "I²L: A Comprehensive Review of Techniques and Technology", Solid State Technology, pp 42-47, June 1977.

SECTION V

LSI DESIGN AND DEVELOPMENT

5.0 INTRODUCTION

In Section III, the design philosophy and architecture needs of a signal processing computer were presented and analyzed. Aside from memory, three major functional areas, Data Processing/Data Addressing, Instruction Addressing, and Multiplier/FFT, were analyzed in depth, and their architecture structures were chosen to meet the needs of the signal processor.

Within this chapter, the register level design of the DP/DA will be presented. The major architectural substructures of the RALU will be described, and the number of gates per structure will be estimated. From these estimates, the LSI development of such a chip will be analyzed in three technologies, CMOS/SOS, I²L, VMOS, and a practical approach for the development will be concluded.

The IA and multiplier/FFT functions will not be analyzed. They are beyond the scope of this contract, although a brief discussion of the IA will be included with only terse conclusions drawn.

5.1 REGISTER-LEVEL DESIGN DISCUSSION

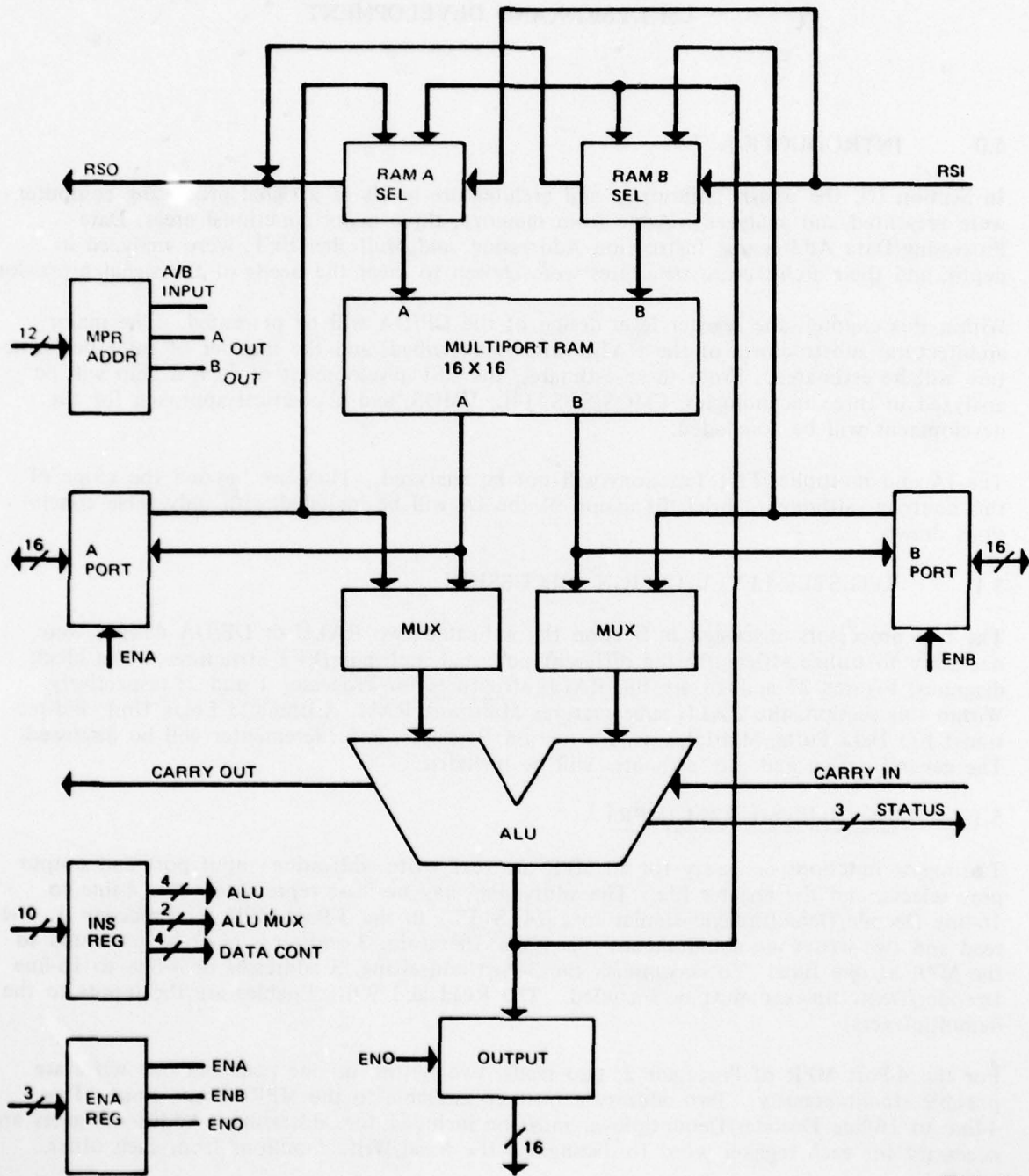
The two processors, discussed in Section III, indicated two RALU or DP/DA designs were necessary to utilize efficiently the different potential multiplier/FFT structures. The block diagrams, Figures 27 and 28 are the RALU structures for Processor 1 and 2, respectively. Within this section, the RALU substructures Multiport RAM, Arithmetic Logic Unit, Bidirectional I/O Data Ports, Multiplexers, Instruction Registers, and Incrementer will be discussed. The general design and gate estimates will be included.

5.1.1 The Multiport RAM (MPR)

The major functions necessary for an MPR are read/write addressing, input port and output port selects, and the register file. The addressing may be least represented as a 4-line to 16-line Decoder/Demultiplexer similar to a 74LS138. In the 3-Port MPR for Processor 1, one read and two-writes are simultaneously possible; therefore, 3 addresses must be presented to the MPR at one time. To accomplish the 3-Port addressing, 3 addresses or 4-line to 16-line Decoder/Demultiplexer must be included. The Read and Write Enables are the inputs to the demultiplexers.

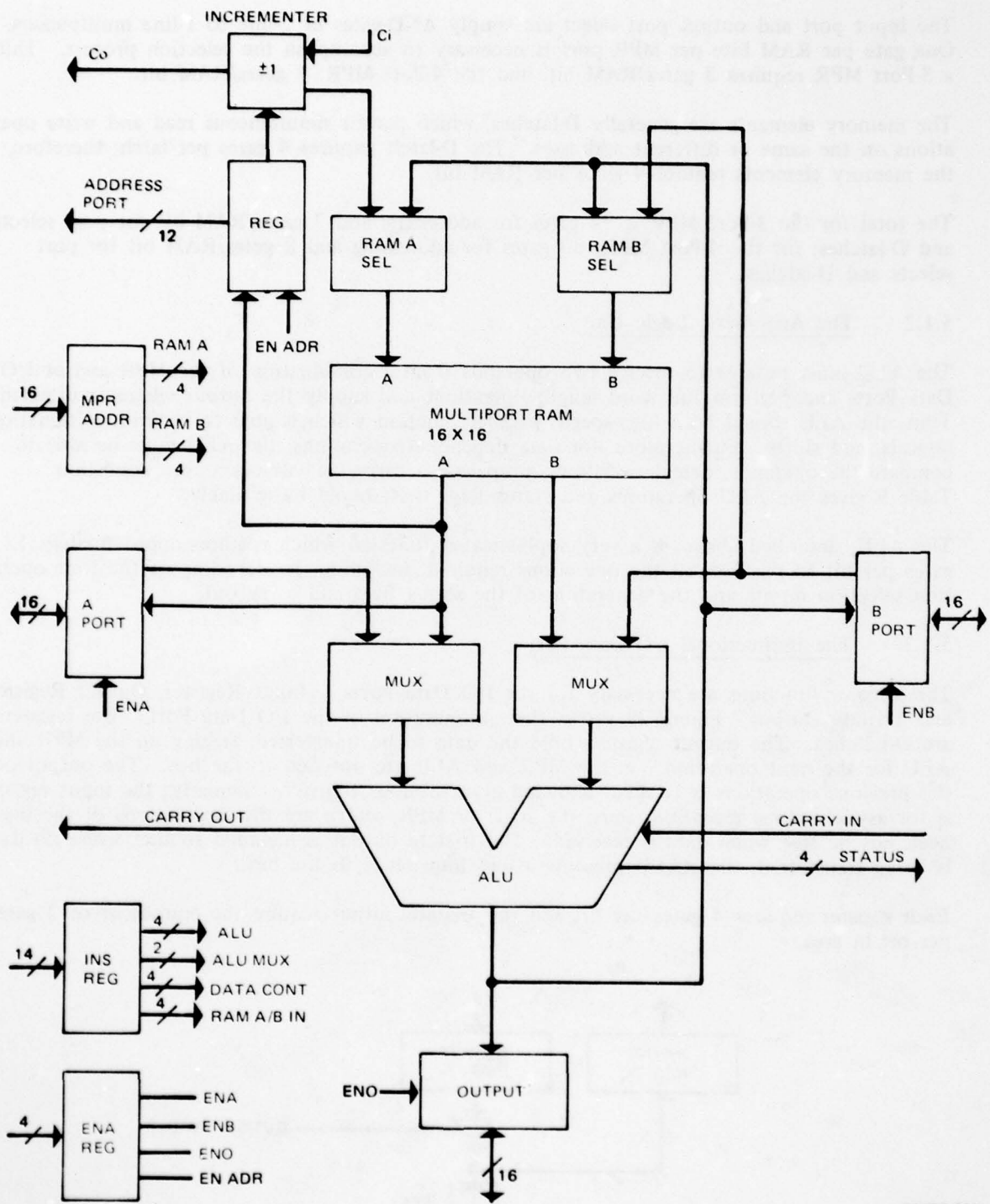
For the 4-Port MPR of Processor 2, two reads, two writes, or one read and one write are possible simultaneously. Two addresses must be available to the MPR at one time. Two 4-line to 16-line Decoder/Demultiplexer must be included for addressing. Additional gates are necessary for each register word to distinguish the Read/Write functions from each other.

For the 3-Port MPR, 24 gates/addresser is necessary. For the 4-Port MPR, 40 gates/addresser is necessary.



78453-30B

Figure 27. RALU (DP/DA) for Processor 1



78453-31A

Figure 28. RALU (DP/DA) for Processor 2

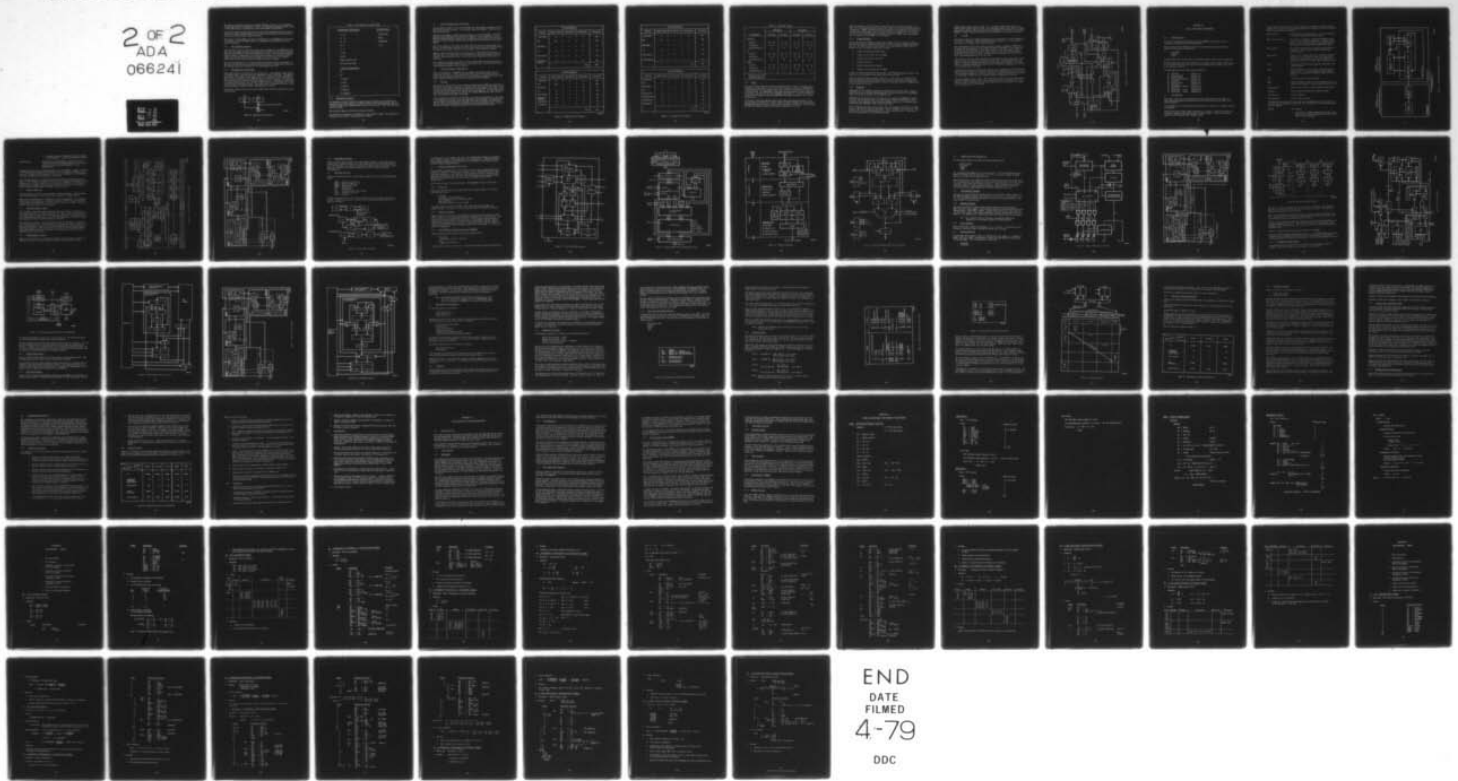
AD-A066 241 DATA SYS DIV, LITTON SYS, INC., VAN NUYS, CAL
MULTIMODE CPU DESIGN STUDY (U)
JUNE 1978 GARY L. MALLALEY
UNCLASSIFIED AFAL-TR-78-134

F/G 9/2

F33615-77-C-1158

N/L

2 OF 2
ADA
066241



END
DATE
FILMED
4-79
DDC

The input port and output port select are simply AND-gates or 2-line to 1-line multiplexers. One gate per RAM bite per MPR port is necessary to accomplish the selection process. Thus, a 3-Port MPR requires 3 gates/RAM bit, and the 4-Port MPR, 4 gates/RAM bit.

The memory elements are generally D-latches, which permit simultaneous read and write operations on the same or different addresses. The D-latch requires 4 gates per latch; therefore, the memory elements require 4 gates per RAM bit.

The total for the 3-Port MPR is 72 gates for addressing and 7 gates/RAM bit for port selects and D-latches; for the 4-Port MPR, 80 gates for addressing and 8 gates/RAM bit for port selects and D-latches.

5.1.2 The Arithmetic Logic Unit

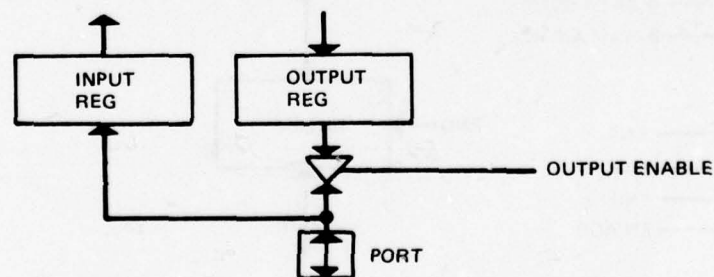
The ALU must be able to receive two operands from a combination of the MPR and/or I/O Data Ports and perform full word length operations and supply the output wherever directed. Thus, the ALU should be a high-speed, parallel function which is able to perform arithmetics, logicals, and shifts. Furthermore, for data dependent operations, the ALU must be able to compare the operands, detect overflows, propagate a carry, and detect a zero condition. Table 9 gives the ALU operations and status flags that should be available.

The ALU, described above, is a very sophisticated function which requires approximately 12 gates per bit to perform all the operations required, including the decoding of the four operation selection inputs and the generation of the status flags and carry out.

5.1.3 The Bidirectional I/O Data Port

Three major functions are necessary for the I/O Data Ports – Input Register, Output Register and Tristate Output. Figure 29 shows the configuration of the I/O Data Port. The registers are D-Latches. The output registers hold the data to be transferred, freeing up the MPR and ALU for the next operation, i.e. the MPR and ALU are not tied to the bus. The output of the previous operations is latched, allowing asynchronous transfer. Similarly, the input register is for asynchronous reception; thus, the ALU or MPR, which are the destinations of the input, need not be free when data is received. The tristate output is included so that when no data is being transferred, the RALU presents a high-impedance to the bus.

Each register requires 4 gates per bit and the tristate output require the equivalent of 2 gates per bit in area.



78453-32

Figure 29. Bidirectional I/O Data Port

Table 9. ALU Operations and Status Flags

<u>ARITHMETIC OPERATIONS</u>	<u>STATUS FLAGS</u>
A + B	CARRY OUT
A - B	ZERO
B - A	OVERFLOW
A + 1	A = B
A - 1	
A only	
B only	
Right Arithmetic Shift	
Left Arithmetic Shift	
<u>LOGICAL OPERATIONS</u>	
\bar{A}	
\bar{B}	
A AND B	
A OR B	
A NOR B	
A EXOR B	
A EXNOR B	

5.1.4 Miscellaneous Functions

The multiplexers within the RALU are extremely simple, requiring a simple AND gate for choosing the proper input signals to the MPR or ALU and some simple decoding. The rule of thumb on gate count is approximately one gate per bit per input. Thus a three-input mux requires 3 gates/bit.

The instruction registers are D-Latch and require 4 gates/bit.

The incrementer for Processor 2 (see Figure 28) is the simplest of adders. No sophistication is desired for this function. Seven gates/bit are required.

5.2 GATE ESTIMATES FOR THE RALU's

The discussion in section 5.1 gives the LSI designer the tools necessary to estimate the total gate counts of the two processors. From the gate counts, the feasibility of the function in LSI may be discerned.

Figure 30 is an attempt to estimate the gates necessary for the 16-bit RALUs. The RALU functions require 2828 gates and 3304 gates for Processor 1 and 2, respectively. In the MACRO constraints Section, a practical limit of 2000 gates was presented. The most logical approach is an 8-bit, bit-sliced RALU; thus, the RALU function can be made of two 8-bit RALUs.

Most of the functions on the RALU are simply reduced to one-half in size and gate count; however, the MPR Address Decoder and the Instruction Register must remain full-size because the same level of control is necessary, only the number of bits controlled is reduced.

Figure 31 reflects the 8-bit RALUs and their gate count. Both RALUs are well below the 2000 gate limit. The penalty paid for the duplication of control was minimal in this case; however, a similar conclusion cannot be drawn about other chips unless a full analysis is performed.

It is concluded at this point that, indeed, an 8-bit, bit-sliced RALU is the proper approach for the Data Processor/Data Addresser from an LSI point-of-view. In the next section, three LSI development approaches will be evaluated.

5.3 LSI DEVELOPMENT APPROACHES

Three LSI technologies – CMOS/SOS, I^3L , and VMOS, are reasonable choices to use to develop the 8 bit RALU's. These technologies will be analyzed in four areas: chip size, power, fundamental speed and availability. For completeness the analysis data for the 16-bit RALU's will be included primarily to further justify the bit-sliced approach.

5.3.1 Chip Size

The chip size of the RALU's has been estimated using the gates per MM^2 data from Table 6 Section IV. The estimate assumes that an average of 40% of the best-case gate density found in the Technology Survey is actually attainable because the high degree of interconnect of this function and the high number of I/O pins will limit the gate density. Even with this assumption, all three subject technologies are capable of exceeding 2000 gates in a 200 x 200 mil chip. The estimate also assumes a square chip.

Table 10 reflects the results of the technologies. Each chip size is specified in a range which is a manifestation of the high and low gate densities for the technologies. All three technologies are capable of producing a chip under 200 x 200 which will perform the 8-bit RALU function. I^3L has the best density, and VMOS has the most inconsistent density. The inconsistent density reflects conflicting information sources with different stories to tell.

RALU FOR PROCESSOR I						
FUNCTION	QUANTITY (BITS)		TOTAL ON RALU		GATES PER BIT	TOTAL GATES
I/O DATA PORT	16	X	3	X	10	480
MPR	256	X	1	X	7	1792
MPR DECODE	-	X	3	X	24	72
ALU	16	X	1	X	12	192
SELECT MUXES	16	X	4	X	3	192
INSTRUCTION REGISTER	25	X	1	X	4	100
					TOTAL	2828

RALU FOR PROCESSOR II						
FUNCTION	QUANTITY (BITS)		TOTAL ON RALU		GATES PER BIT	TOTAL GATES
I/O DATA PORT	16	X	3	X	10	480
MPR	256	X	1	X	8	2048
MPR DECODE	-	X	2	X	40	80
ALU	16	X	1	X	12	192
SELECT MUXES	16	X	4	X	3	192
INSTRUCTION REGISTER	25	X	1	X	4	104
ADDRESS PORT	16	X	1	X	6	96
INCREMENTER	16	X	1	X	7	112
					TOTAL	3304

78453-33

Figure 30. 16-Bit RALU Gate Estimates

RALU FOR PROCESSOR I						
FUNCTION	QUANTITY (BITS)		TOTAL ON RALU		GATES PER BIT	TOTAL GATES
I/O DATA PORT	8	X	3	X	10	240
MPR	128	X	1	X	7	896
MPR DECODE	-	X	3	X	24	72
ALU	8	X	1	X	12	96
SELECT MUXES	8	X	4	X	3	96
INSTRUCTION REG	25	X	1	X	4	100
					TOTAL	1500

RALU FOR PROCESSOR II						
FUNCTION	QUANTITY (BITS)		TOTAL ON RALU		GATES PER BIT	TOTAL GATES
I/O DATA PORT	8	X	3	X	10	240
MPR	128	X	1	X	8	1024
MPR DECODE	-	X	2	X	40	80
ALU	8	X	1	X	12	96
SELECT MUXES	8	X	4	X	3	96
INSTRUCTION REG	26	X	1	X	4	104
ADDRESS PORT	8	X	1	X	6	48
INCREMENTER	8	X	1	X	7	56
					TOTAL	1744

78453-34

Figure 31. 8-Bit RALU Gate Estimates

Table 10. Technology Analysis

TECHNOLOGY	PROCESSOR 1		PROCESSOR 2	
	16-bit RALU	8-bit RALU	16-bit RALU	8-bit RALU
CMOS/SOS				
Chip Size*	210 - 270	153 - 197	227 - 292	165 - 212
Power (watts)	2.8 - 7.1	1.5 - 3.8	3.3 - 8.3	1.7 - 4.4
R-R Add** Time (ns)	33	33	33	33
I³L				
Chip Size*	191 - 210	139 - 153	206 - 227	150 - 165
Power (watts)	2.8 - 14.1	1.5 - 7.5	3.3 - 16.5	1.7 - 8.7
R-R Add** Time (ns)	44	44	44	44
VMOS				
Chip Size*	191 - 369	139 - 269	206 - 399	150 - 290
Power (watts)	2.3 - 2.8	1.2 - 1.5	2.6 - 3.3	1.4 - 1.7
R-R Add** Time (ns)	55	55	55	55
*Measured in mils on a side				
**Register-to-Register Add				

5.3.2 Power

The power has been estimated from the gate dissipation data in Table 6. The estimate was normalized to include speed as a factor in the gate dissipation; consequently, CMOS/SOS has a gate dissipation in the low milliwatt range when it is normally reported in the microwatt or nanowatt range. For CMOS/SOS to reach signal processing speed, the supply voltage must be increased to 10 volts and the power dissipation simply goes up. In Table 10, VMOS has a lower power dissipation per gate than CMOS/SOS, but VMOS is assumed to have a lower speed potential.

I³L also has a wide power dispersion which truly reflects the power/speed diversity of the I²L technology, of which I³L is a variant. Unlike the MOS technologies, I³L could maintain maximum performance and have a power below the maximum of Table 10 if the lower speed data paths are carefully chosen and the I³L gates tailored for lower power.

The power prospects of I^3L are limited; however, I^3L is the worse power dissipator of the three. If a low operational speed is assumed, I^3L is under the 2 watt per chip limit discussed in Section 4.2. VMOS will definitely meet this requirement, and CMOS/SOS is probably able to handle this function under 2 watts if most of the gates are assumed to be "off" during most of the time. This assumption is reasonable since only the MPR words being addressed are "on"; thus, most of the MPR is "off" or inactive.

5.3.3 Fundamental Speed

For this discussion, the fundamental speed will be defined as the time required to perform a basis operation such as a register-to-register add, a compare, a register-to-register logical operation, etc. The register-to-register add is assumed to be a representative operation for the problem set and was analyzed in gate delays as follows:

- a. 2 delays for Instruction Register Setup
- b. 1 delay to read to data out of the MPR
- c. 1 delay to pass thru the ALU select
- d. 4 delays in the ALU
- e. 1 delay to pass thru the PR select
- f. 2 delays to store the data into the MPR

A total of 11 delays is required for this operation. The fundamental speed, therefore, is the product of the total number of gate delays and the gate delay time.

Table 10 includes the Register-to-Register Add Time. All the times were calculated using minimum gate delays and include no delay time estimate for interconnect path length. As seen in the Table, all these technologies are capable of high speed operation. CMOS/SOS has the best potential at this time.

5.3.4 Availability

CMOS/SOS, I^3L , and VMOS are probably best described as in the early stages of maturity, which means that each has been demonstrated with commercial products in the marketplace; however, wide lines of products are not available as yet.

CMOS/SOS has the longest history. The early days were rough, but CMOS/SOS is available from RCA, HP and Rockwell with new sources coming. Because CMOS/SOS has a good temperature range, high noise margin, radiation-hardening potential, etc., the military market is good, thus, the availability is sure to increase with time.

I^3L and VMOS are new stars on the horizon. I^3L is an extension of I^2L which is a simple process in concept, but difficult if high speed is desired. Fairchild is the only source. Other high speed I^2L variants are becoming available - ISL and Up-Diffused I^2L . Time will tell! For now, high speed I^2L (I^3L) has limited availability.

VMOS for LSI is solely sourced by AMI. It is a variation of NMOS which gives 1 um channel length using 4-6 um layout rules. As the photolithographic process improves, VMOS will improve in density without the heartaches that HMOS from Intel will have to go through. VMOS is a winner! Second-sourcing will come, but availability is very limited at this time.

5.4 IA CHIP

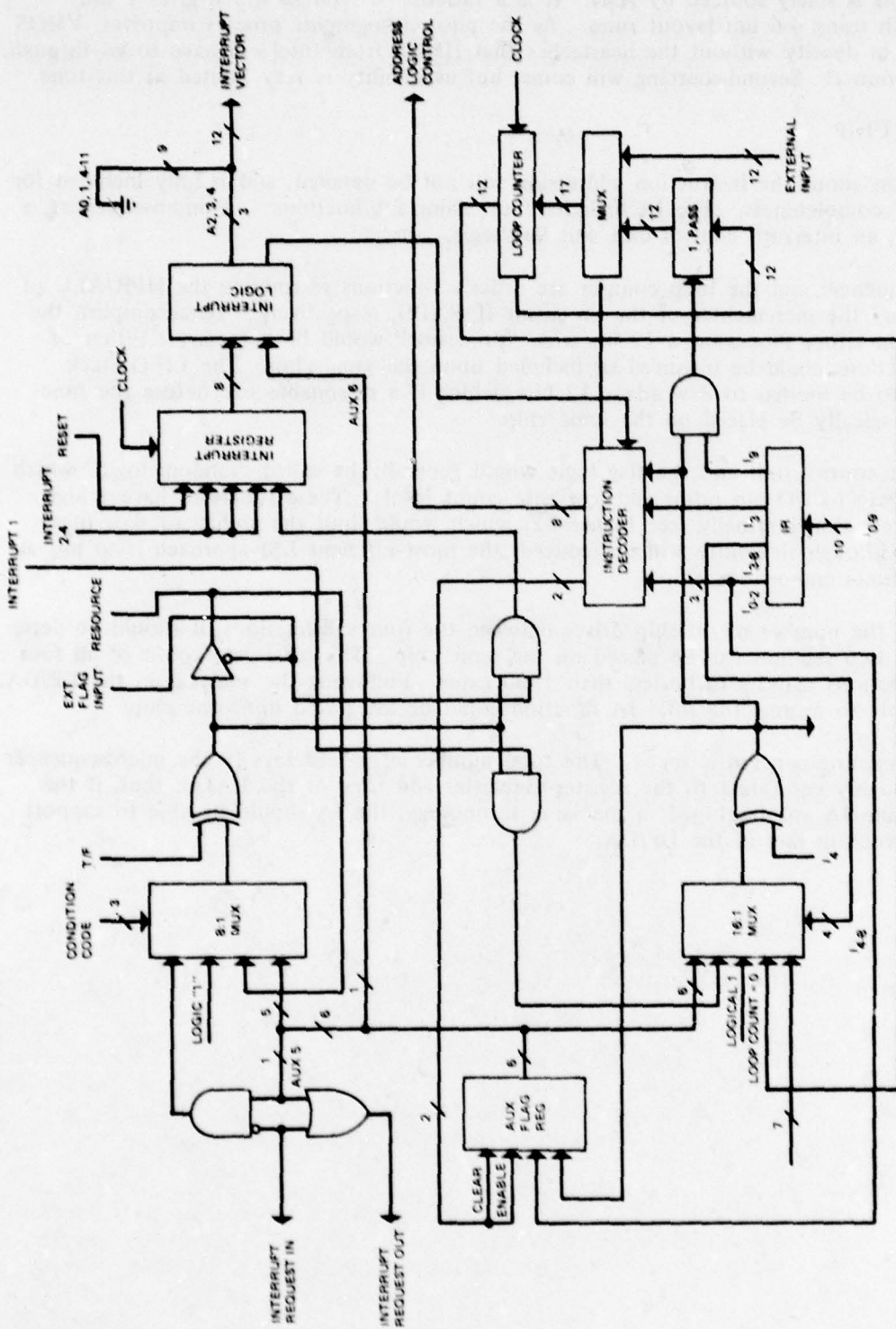
This discussion about the instruction addressing will not be detailed, and is only included for a measure of completeness. The IA includes four major subfunctions – a microsequencer, a loop counter, an interrupt control unit and flag logic.

The microsequencer and the loop counter are orderly functions resembling the MPR/ALU of the RALU and the incrementer of the Processor II RALU, respectively. To accomplish the IA function in either Processor, a 12 bit wide "processor" would be necessary. Either of these subfunctions could be bit-sliced or included upon the same chip. The LIFO Stack would have to be limited to 8 words x 12 bits, which is a reasonable size before the functions could logically be placed on the same chip.

The interrupt control unit and the flag logic would generally be called "random logic" which implies low gate to I/O pin ratios and low gate count totals. These functions have a high degree of interaction internally (see Figure 32) which would limit the ability to slice these functions. Although flexibility will be reduced, the most efficient LSI approach is to put all of these subfunctions on one chip.

To minimize the number of off-chip drives between the four subfunctions, it should be determined if the two sections can be placed on the same chip. The total gate count of all four major subfunctions appears to be less than 1500 gates. Following the analysis on the DP/DA, it is reasonable to assume the total IA function could be integrated onto one chip.

The final remaining concern is speed. The total number of gate delays in the microsequencer should be roughly equivalent to the register-to-register add time of the RALU, thus, if the DP/DA and the IA are developed in the same technology, the IA should be able to support the high instruction rate of the DP/DA.



78463 25A

Figure 32. Controller Without Microsequencer

SECTION VI
SIGNAL PROCESSOR COMPARISON

6.0 INTRODUCTION

As a part of the Multimode Central Processing Unit (MMCPU) Design Study Contract, a comparison of signal processors has been performed.

The main thrust of the comparison is based upon three benchmark problems which are applied to 3 microprocessor architectures,

Tracor/RCA
Raytheon
Litton

presented herein, these microprocessors are specifically suited to signal processing applications.

We shall briefly describe the system architecture as a tutorial. This description will then lead to the microprocessor and its application to the benchmark problems. Details can be found in the references, section 6.9.

The following sections of the study include discussion of:

- Definition (Section 6.1)
- Macro Computer (Section 6.2)
- Building Blocks (Section 6.3)
- Central Processing Unit (Section 6.4)
- Controller (Section 6.5)
- Microcomputer (Section 6.6)
- Benchmarks (Section 6.7)
- Comparison of Results (Section 6.8)
- References (Section 6.9)
- Benchmarks – Coding (Appendix B)
- Benchmarks – Timing (Appendix C)

6.1 DEFINITIONS

Since many users of data processing systems are not acquainted with the techniques and terms used in data processing, we shall briefly describe some of these as they relate to the system architecture.

It is the hallmark of a scientist that he define his terms, for only then can semantic confusion be eliminated.

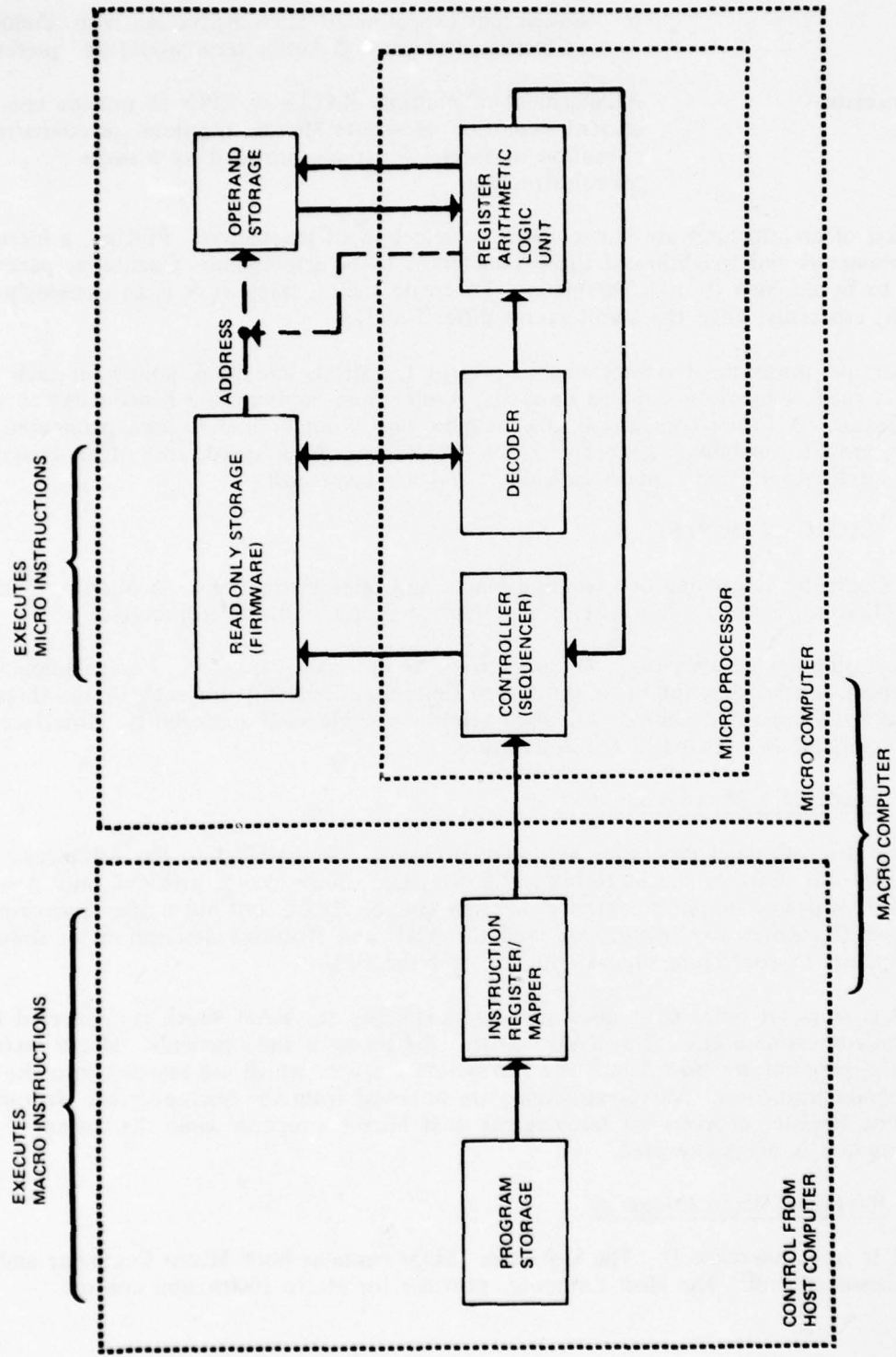
There are terms like "storage" which is preferred over "memory." However, the latter term is so heavily impressed in the literature that it is hard to change it. The term "instruction location" is more descriptive than "program counter."

Note the difference below between Macro and Micro, and the difference between Computer and Processor. Figure 33 shows a simplified definition block diagram. It shows the boundary lines for the purpose of the definitions.

- Macro Computer — Executes Macro and Microinstructions. It is the combination of the host computer with the Microcomputer.
- Host Computer Control — Provides Macro instructions, obtained from Program Storage and puts it into the Instruction Register. Through the Mapping it controls the Microcomputer. The feedback from the Microcomputer is not shown which simplifies the diagram.
- Microcomputer — Executes Microinstructions. It consists of the Microprocessor and Storage. Storage includes Firmware Storage and/or Operand Storage.
- Microprocessor — Consists of the Controller (Sequencer), Decoder, and Register Arithmetic Logic Unit (RALU). Often very limited Read Only Storage and Random Access Storage for Firmware and Operands respectively are provided within the Microprocessor.
- ALU — Arithmetic Logic Unit performs additions, subtractions, and logical operations.
- RALU — Register Arithmetic Logic Unit, often called Central Processing Unit (CPU). It contains Multi Port Registers (MPR) Multiplexers including shifter, Arithmetic Logic, and Control Decode.
- CPU — Central Processing Unit same as RALU.
- MPR — Multi Port Registers, Register stack with multiple access ports (addresses) and capable of multiple operations (read and write).
- Building Blocks — Blocks used to construct a Microprocessor or Microcomputer.
- Controller — In this context, the sequencing of Microinstructions.
- Data Processor — Processes operands.
- Data Addressing — Processes operand addresses.

As Large Scale Integrated (LSI) circuits progress to accommodate more circuits, more and more functions are included in a single chip. Thus, the dividing of functions becomes more and more difficult.

- Pipeline — Two meanings:
 - a. Arrangement of multiple Arithmetic Logic Units to provide execution of multiple Microinstructions concurrently. Similar to array processing.



78453-36

Figure 33. Definition Block Diagram (Simplified)

- b. Concurrent execution of Microinstruction with fetching of Microinstruction. A better term would be "prefetch."

Array Processor — Arrangement of multiple RALUs or CPUs to provide concurrent execution of several Microinstructions, or concurrent execution of several functions provided by a single Microinstruction.

Comparison of architectures are based upon the selection of parameters. Further, a hierarchy of the parameters and weighting of the parameters is to be established. Qualitative parameters are to be assigned to each parameter. To create such a framework is an extremely difficult task, especially when the architectures differ widely.

As an alternate approach, it is suggested to present the strong and weak points for each system. A final comparison is based upon the results from applying the benchmarks to each Microprocessor. A future comparison may include, but is not limited to such parameters as flexibility, growth capability, number of chips, technology, clock speed, cost of hardware/firmware/software, program support capability, and life cycle cost.

6.2 MACRO COMPUTER

A Macro Computer is capable of executing Macro and Microinstructions. A Microcomputer executes Microinstructions. A subset of the Microcomputer is the Microprocessor.

Signal processing applications use a Microprocessor as the main hardware. For completeness of this report, a brief description of the Macro Computer capability for each of the three manufacturers is presented below. All three vendors provide such a capability. Interface to the host computer is included in the description.

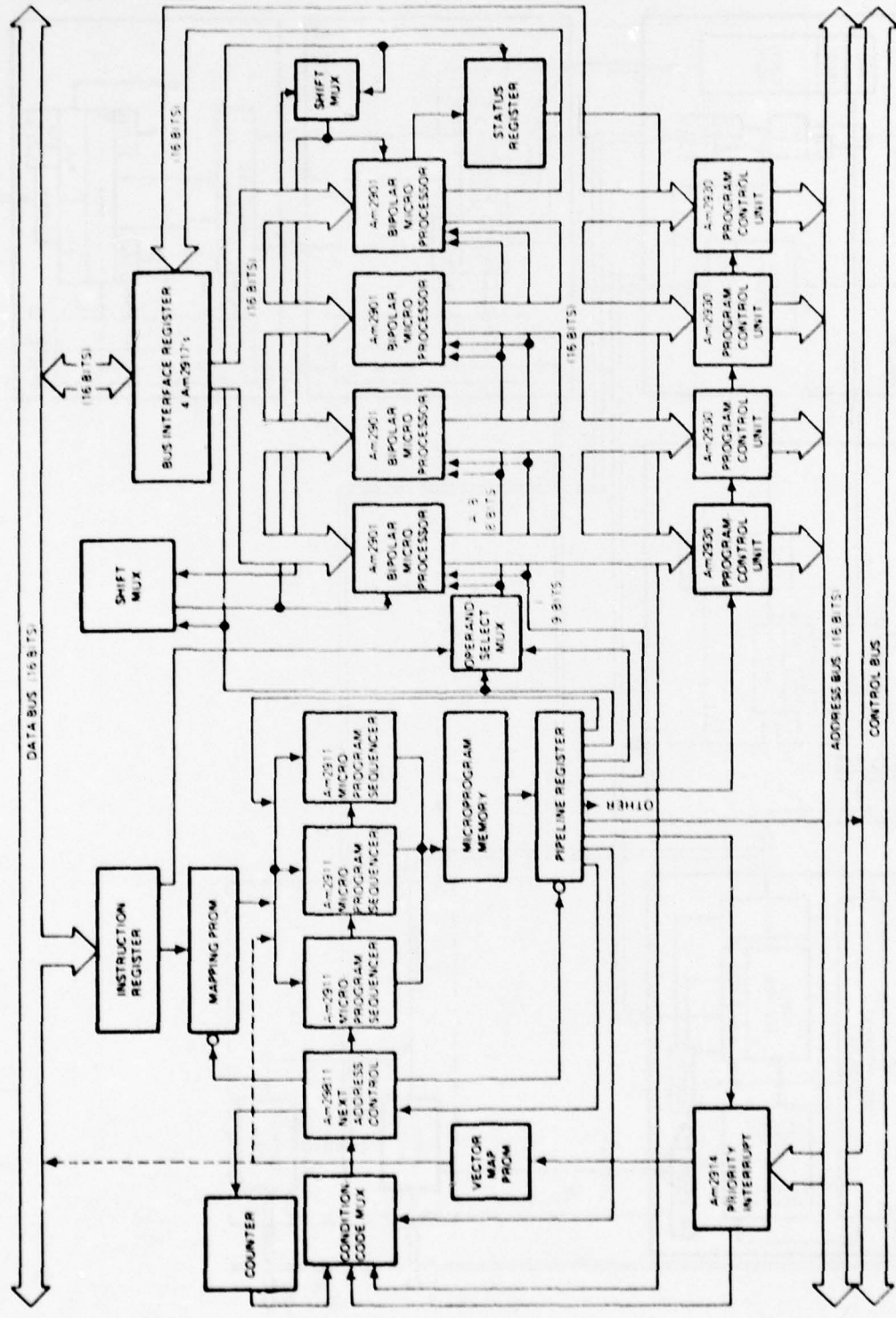
6.2.1 Tracor/RCA Macro Computer

Tracor provides a General Processing Unit (GPU) chip which is similar to the Advanced Micro Device AM 2901 or the Motorola MC 2901 chip. Since Tracor provided only description of GPU which is simply a central processing unit or RALU and not a Microprocessor nor a Macro Computer architecture, we used the AMD and Motorola descriptions of simple Microcomputers to conjecture similar structures for the GPU.

Figure 34 is from reference C; it does not show explicitly the RAM which is connected to the control/address/data bus. The RAM contains the program and operands. Macro Instructions of the program are loaded into the Instruction Register which are mapped into the Micro Program Sequencer. Microinstructions are retrieved from the Microprogram Memory. The Pipeline Register provides for fetching the next Microinstruction while the current Microinstruction is being executed.

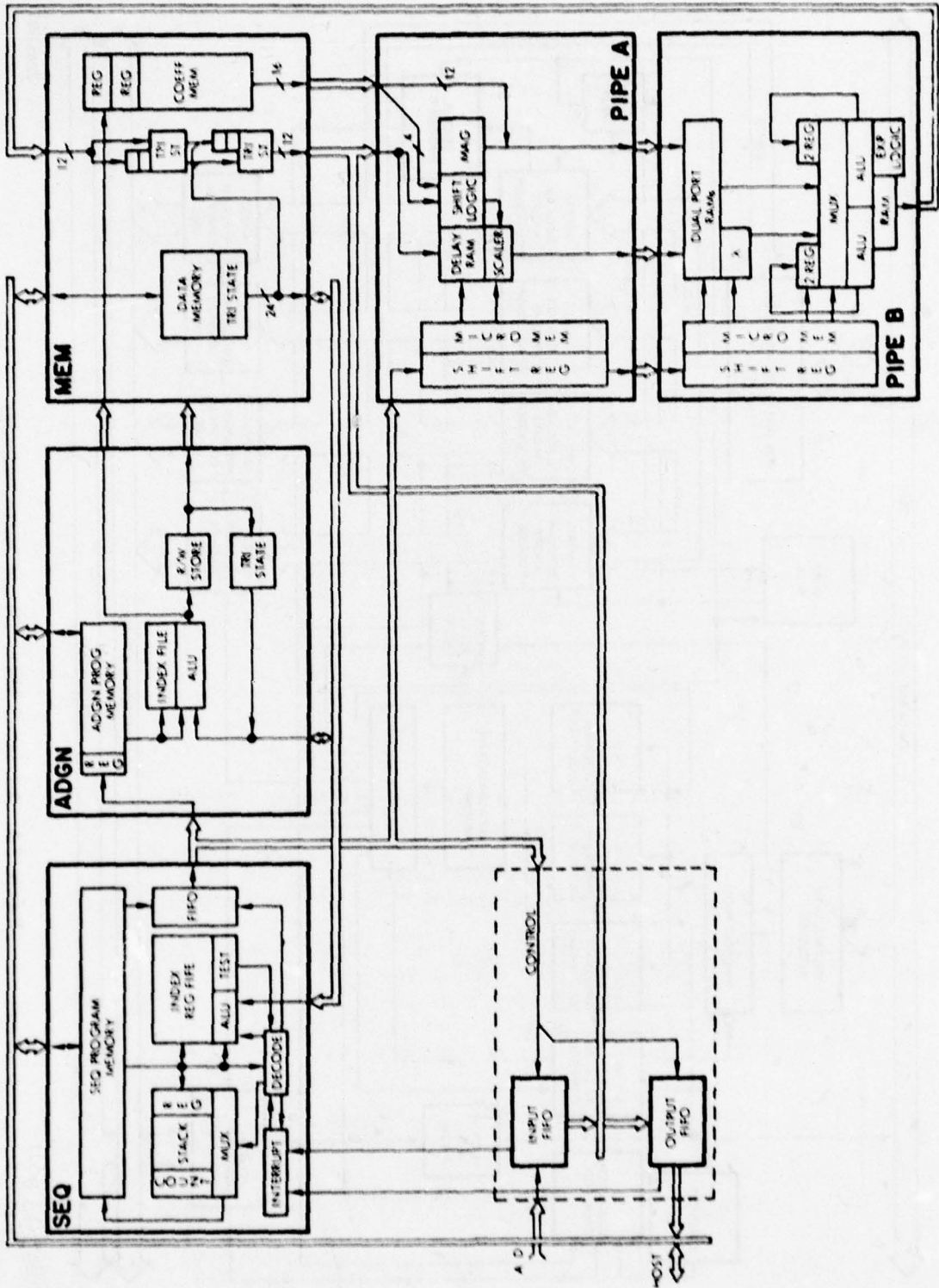
6.2.2 Raytheon Macro Computer

Figure 35 is from reference D. The Sequencer (SEQ) contains both Macro Computer and Microprocessor control. The Host Computer provides for Macro Instruction control.



78453-37

Figure 34. Advanced Micro Device Microcomputer



78453-38

Figure 35. Raytheon Macro Computer

6.2.3 Litton Macro Computer

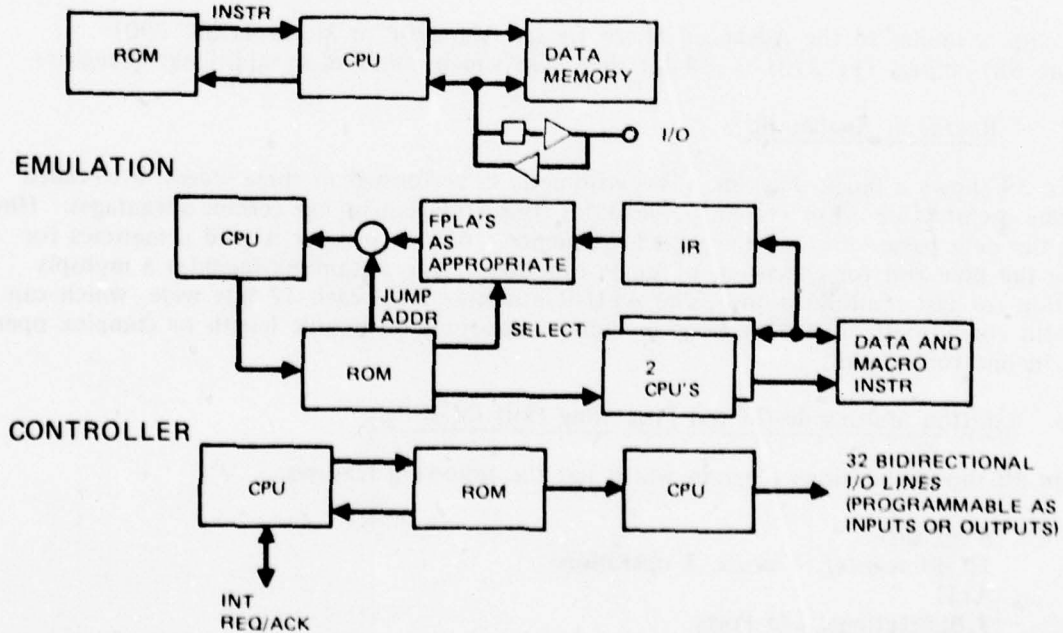
Figure 36 shows a typical example of the Litton Macro Computer. A CPU chip is used to provide for the controller function and for Data Addressing as well as Data Processing function. The Emulation part in Figure 36 provides for emulation of Macro Instructions. Macro Instructions are held in the Instruction Register (IR). The Controller executes the Microinstructions.

6.3 BUILDING BLOCKS

As shown in the previous section, a typical Microcomputer consists of the following building blocks.

- RAM – Random Access Memory
- ROM – Read Only Memory, or
- PROM – Programmable ROM
- MPY – Multiply Chip
- I/O – Input/Output
- CPU – Central Processing Unit, including
- ALU – Arithmetic Logic Unit
- CONT – Controller

Further, the operation of the interconnected building blocks is a function of the program or firmware. Program and firmware are produced using a programming language which is not subject to this report.



78453-39A

Figure 36. Litton Macro Computer

It is assumed the reader is familiar with most of the building blocks; therefore, a description can be omitted. However, the important blocks of a microprocessor – CPU and Controller – will be analyzed in further detail. Differences in the architecture of these blocks will show an effect on the benchmark problems.

6.4 CENTRAL PROCESSING UNIT (CPU)

The name CPU is misleading, but it is used in this report because the semiconductor manufacturers have adopted it. In this report, the CPU is a chip. The architecture of the CPU varies from manufacturer to manufacturer. It depends upon the chip size, number of gates, technology, and number of pins. The CPU under consideration may be classified bit slices. The slices are typical 4-bit and 8-bit wide and can be cascaded to provide 16 bit Microprocessors.

The references provide for detailed description. Only highlights are given in this report.

6.4.1 Tracor GPU

This chip is called General Processing Unit (GPU) and is shown in Figure 37. It has the following characteristics:

- 8 bit slice
- 16 registers, 3 ports/2 operations
- ALC (Arithmetic Logic Circuit, limited ALU)
- Input and Output
- Concatenation logic for any word length

This chip is similar to the Advanced Micro Device AM 2901 or Motorola MC 2901 (Figure 38) chips. The 2901 is a 4 bit slice chip which contains an additional Q register.

6.4.2 Raytheon Arithmetic

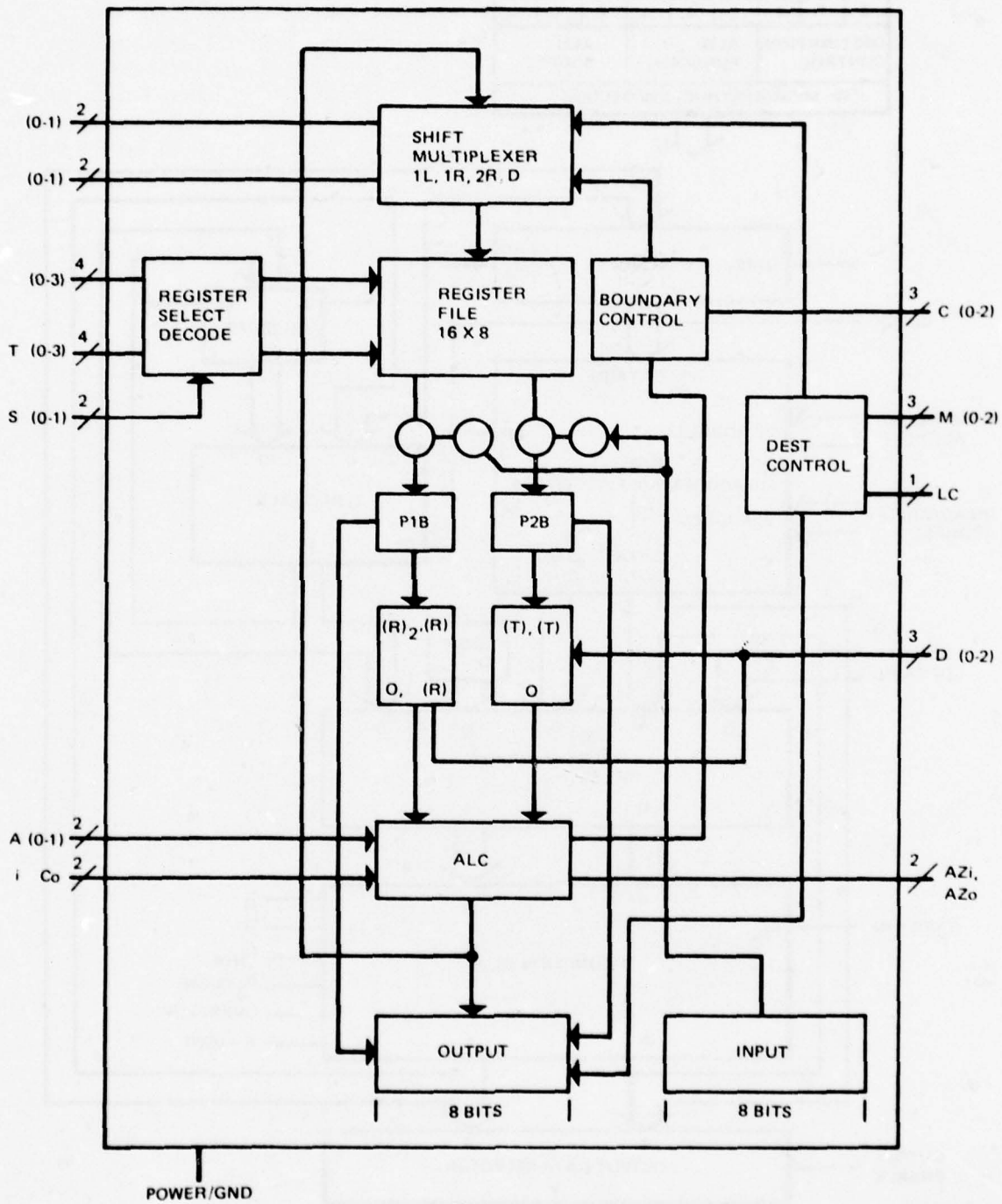
Figure 39 shows a block diagram. The arithmetic is performed in three stages, a so-called pipeline architecture. For certain applications, this arrangement has certain advantages. However, the data passes through the pipe in sequence. A time penalty is paid sometimes for filling the pipe and for execution of single functions. The arithmetic includes a multiply function for fast multiplications. The ALU is a double ALU each 12 bits wide, which can perform concurrently two operations including operations on double length or complex operands in one timing unit.

6.4.3 Litton Multimode Central Processing Unit (MMCPU)

Figure 40 shows the block diagram which has the following features:

- 8 bit slice
- 16 bit register, 4 ports, 3 operations
- ALU
- 3 Bidirectional I/O Ports

This chip can be used for two functions – Data Processing (DP), and Data Addressing (DA).



78453-40

Figure 37. Tracor GPU Block Diagram

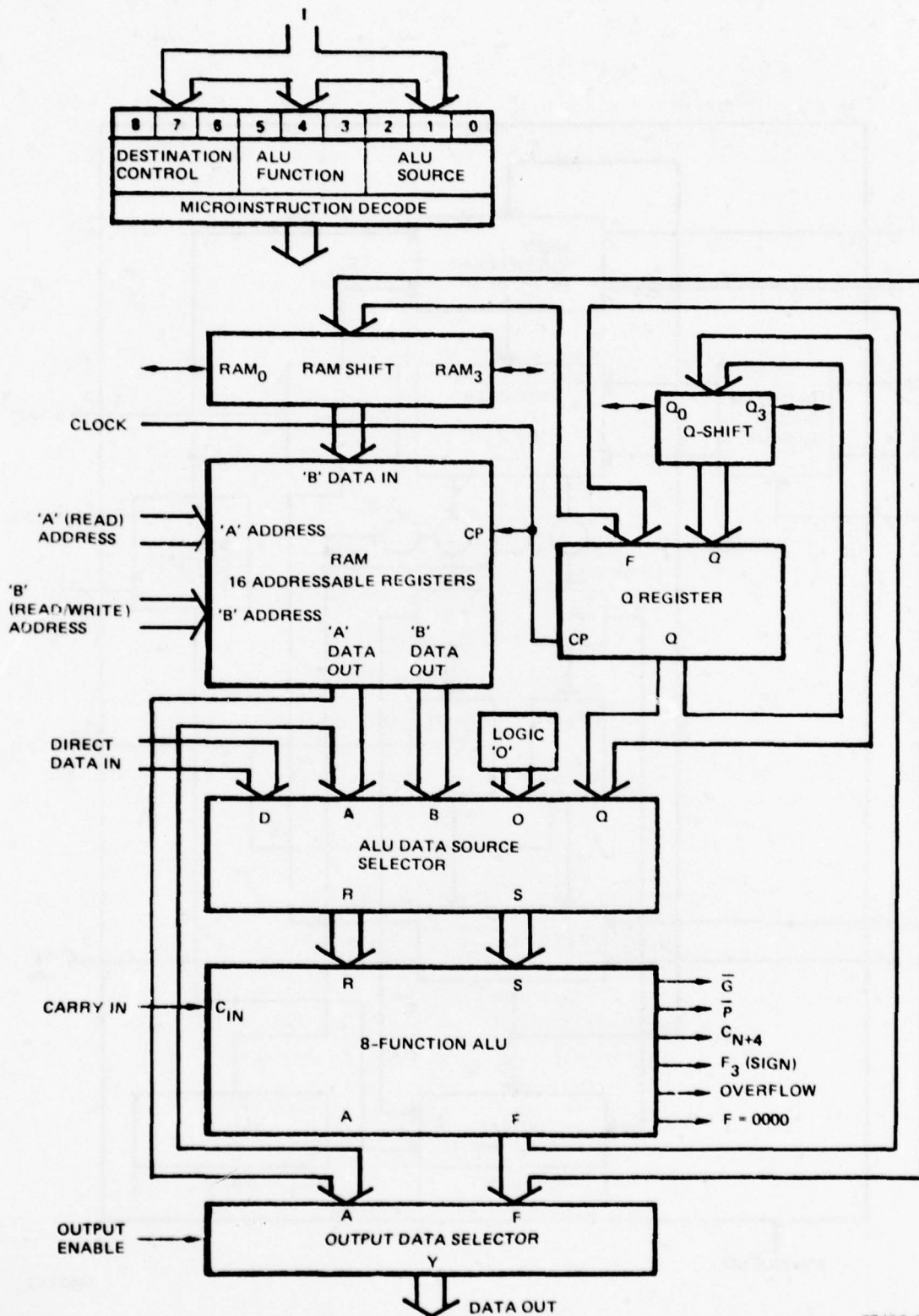
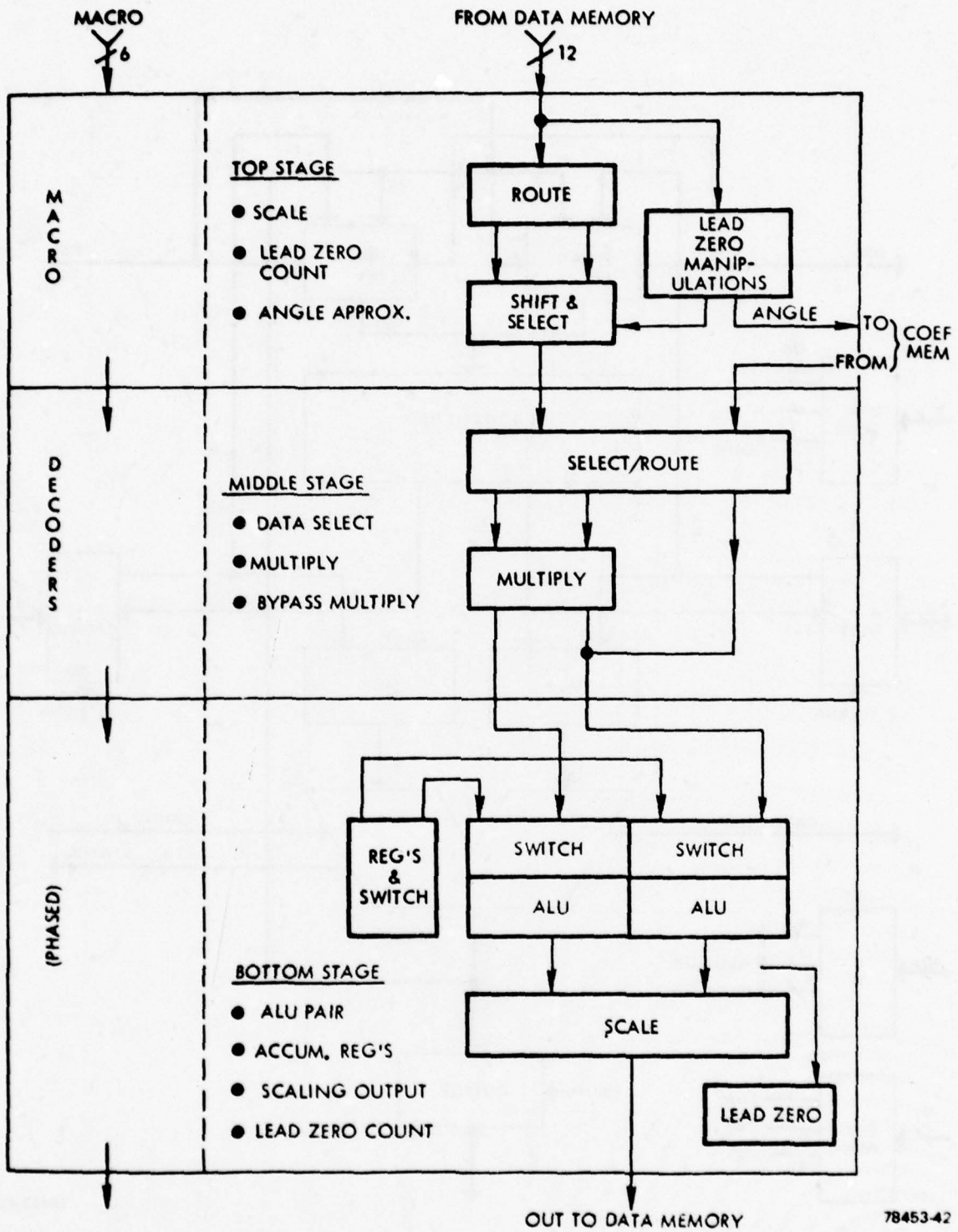


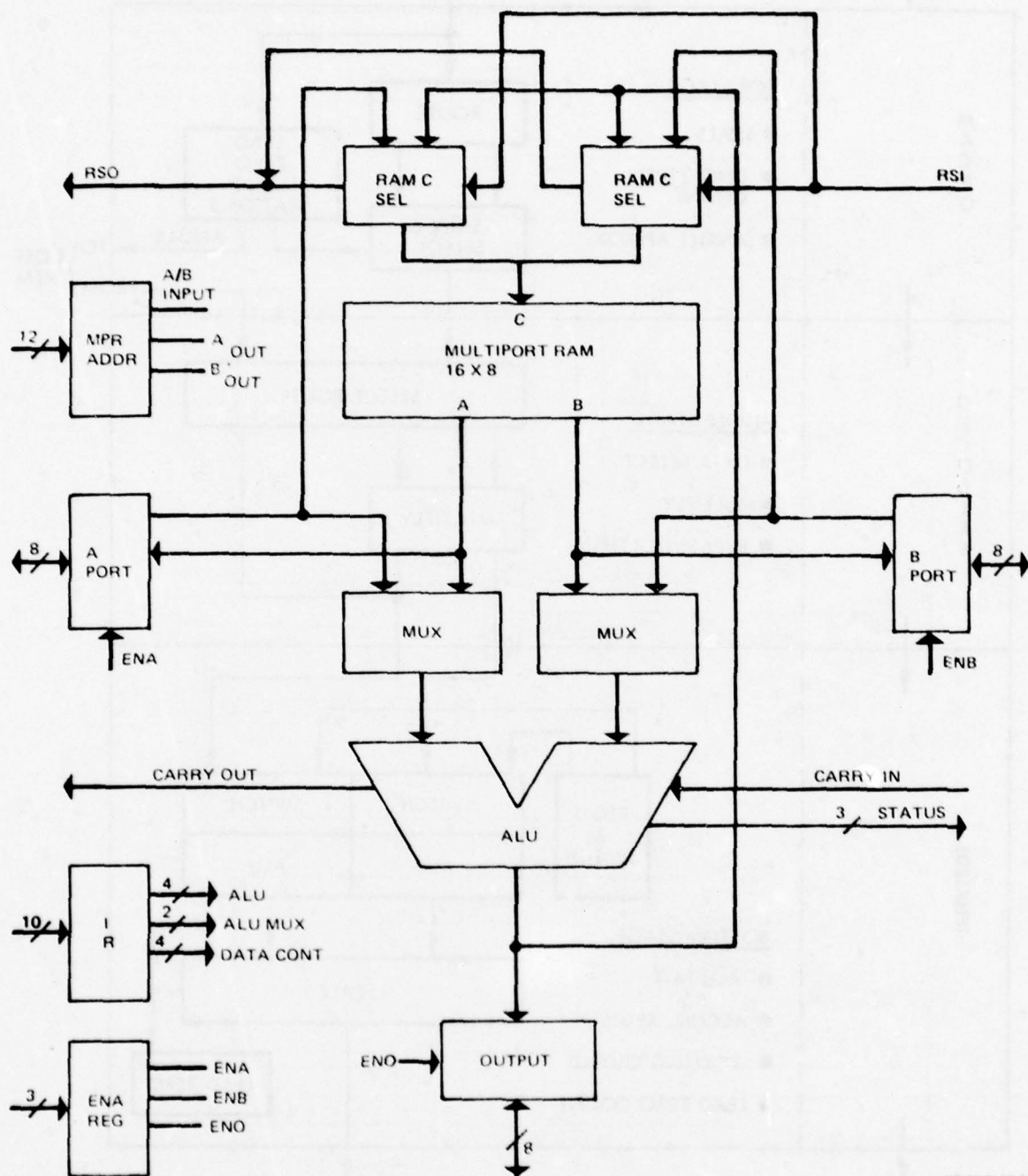
Figure 38. Advanced Micro Device Microprocessor Slice

78453-41



78453-42

Figure 39. Raytheon Arithmetic



78453-43A

Figure 40. Litton Multi Mode CPU (Used for DP/DA)

6.5 CONTROLLER OR SEQUENCER

To build a Microcomputer, one needs the following building blocks:

✓ CPU or RALU
Controller
ROM
RAM

The controller accesses ROM to fetch a Microinstruction. It decodes the Microinstruction to steer the CPU/RALU and other functions in the same time interval it updates the ROM address for accessing the next Microinstruction.

Advanced architectures use a register to hold the Microinstruction while the next Microinstruction is being assessed. Thus, instruction execution and next instruction access occurs at the same time. This is often called "pipeline operation." A more appropriate name would be "instruction prefetch." Thus avoiding confusion with "pipeline operation" referring to sequential operation through serially connected RALUs or RALUs in an array.

6.5.1 Tracor/MC2909 Controller

In absence of a Tracor Controller, the MC 2909 has been substituted. Figure 41 shows the MC 2909 Microprogram Sequence block diagram. It is a 4 bit slice and is cascadable. A 4 x 4 file with stack pointer and push/pop control provides for nesting subroutines. Direct inputs provide for N-way branches.

6.5.2 Raytheon Controller

The control function in the Raytheon Microcomputer/Microprocessor are distributed. Figure 42 shows a block diagram. The dotted line encloses the controller function with the following blocks - SEQ, PIPE A, PIPE B. A FIFO (first in first out) is used to shift control from block to block in this pipeline architecture. Details of the shift logic is shown in Figure 43. The PROM is used to decode the control code obtained from the shift registers.

Note: For consistency with our definitions, the Raytheon MACRO (see Reference D) is functionally equivalent to a Microinstruction.

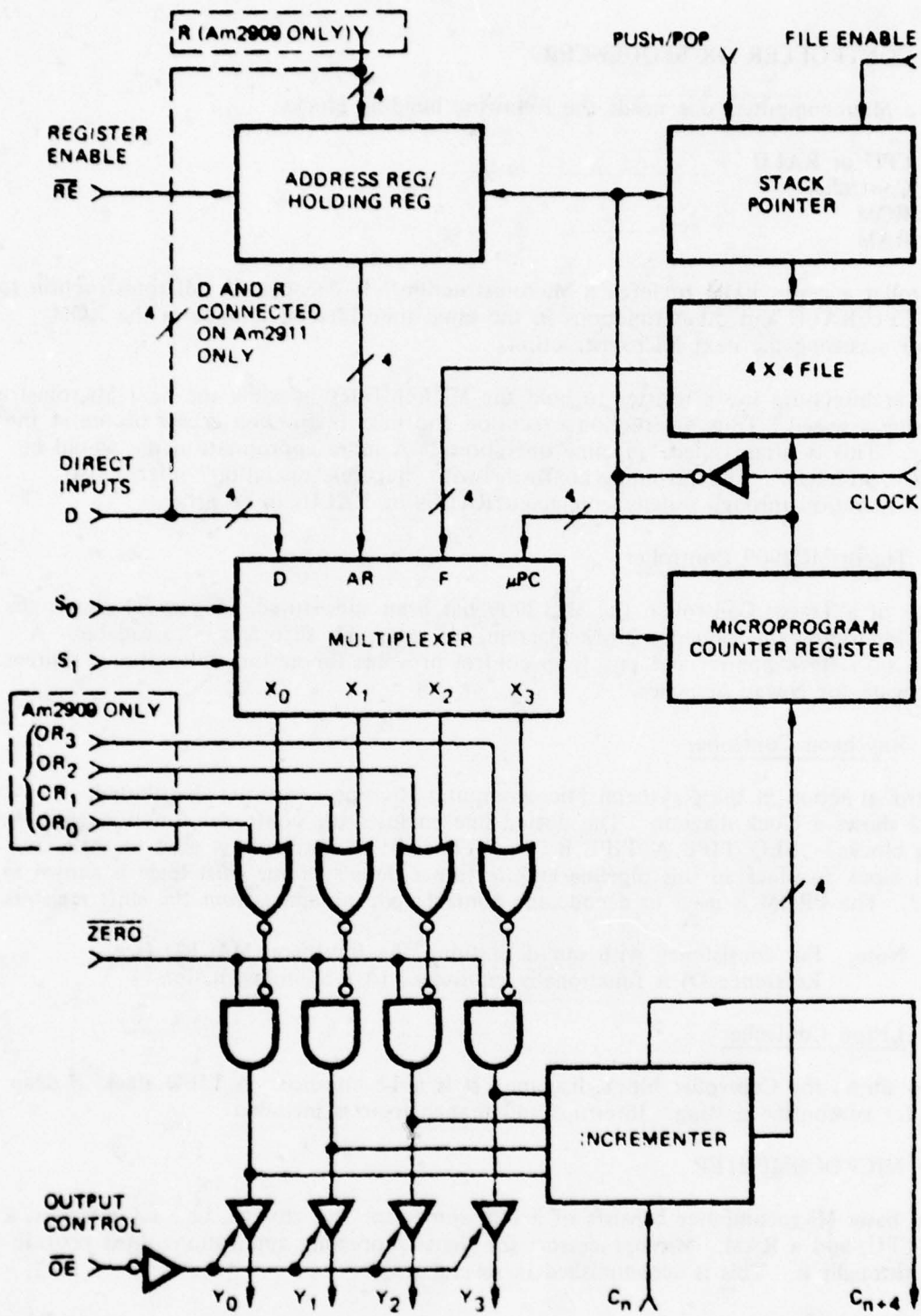
6.5.3 Litton Controller

Figure 44 shows the Controller block diagram. It is a 12 bit slice. A LIFO stack, 8 deep provides for subroutine nesting. Interrupt and branch logic is included.

6.6 MICROCOMPUTER

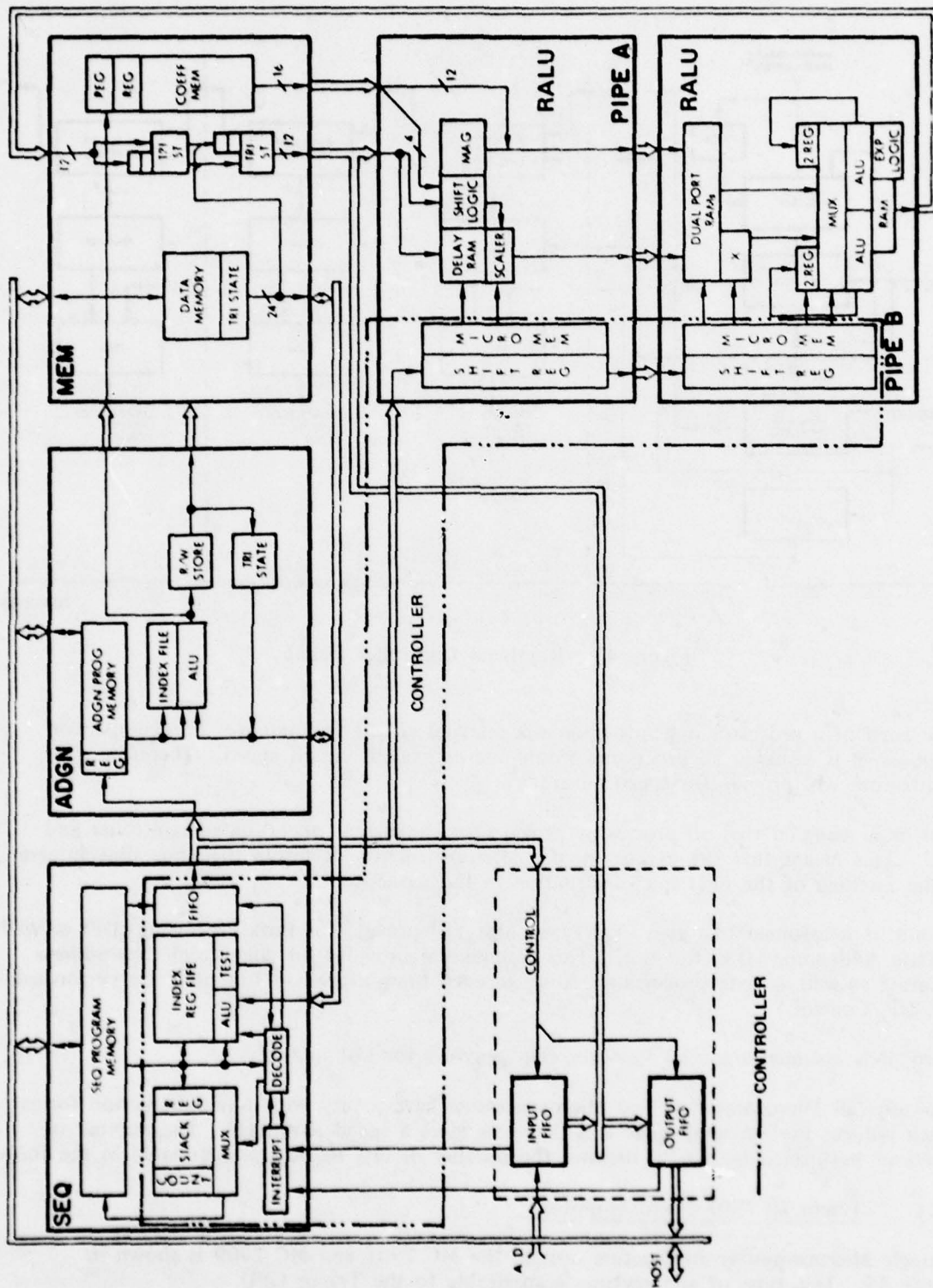
A typical basic Microcomputer consists of a Microprocessor and storage, i.e., a Controller, a ROM, a CPU, and a RAM. Microprocessors for signal processing applications must provide for high throughput. This is accomplished in several ways:

- technology
- architecture



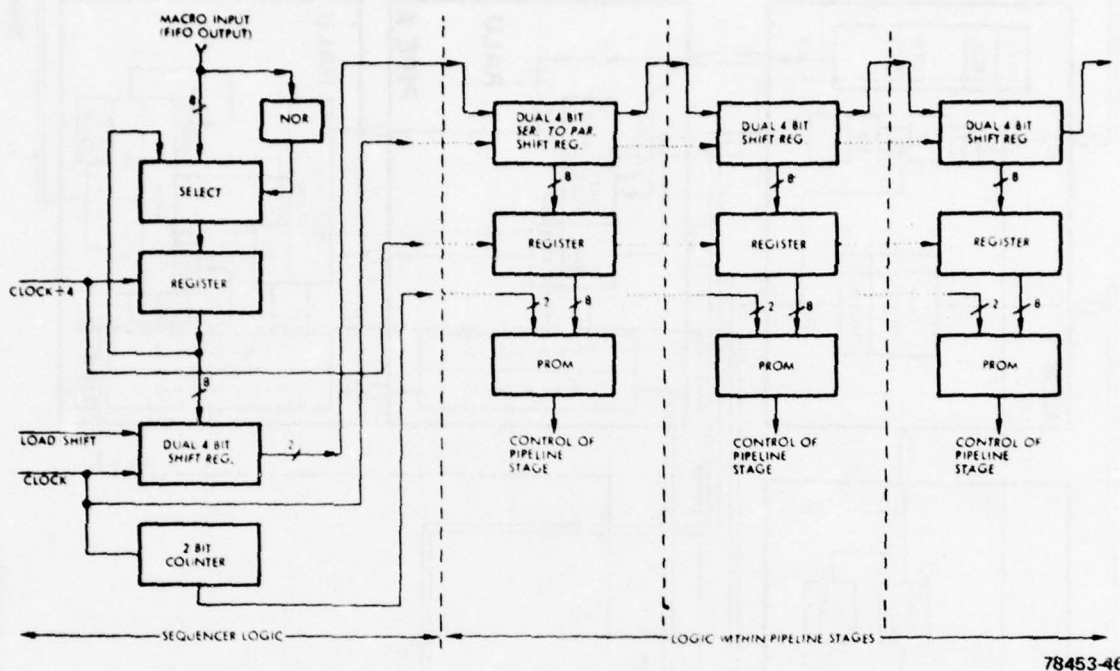
78453-44

Figure 41. Advanced Microdevice Controller



78453-45

Figure 42. Raytheon Controller



78453-46

Figure 43. Raytheon Controller Detail

The speed of a processor depends upon the selected circuit technology. For comparison purposes, it is assumed all processors would use equivalent circuit speed. Therefore, the architecture will provide for speed advantage.

First, it is assumed that all processors provide for concurrent operation of controller and CPU. This means that the execution of a Microinstruction occurs in the same time interval as the fetching of the next microinstruction in the sequence.

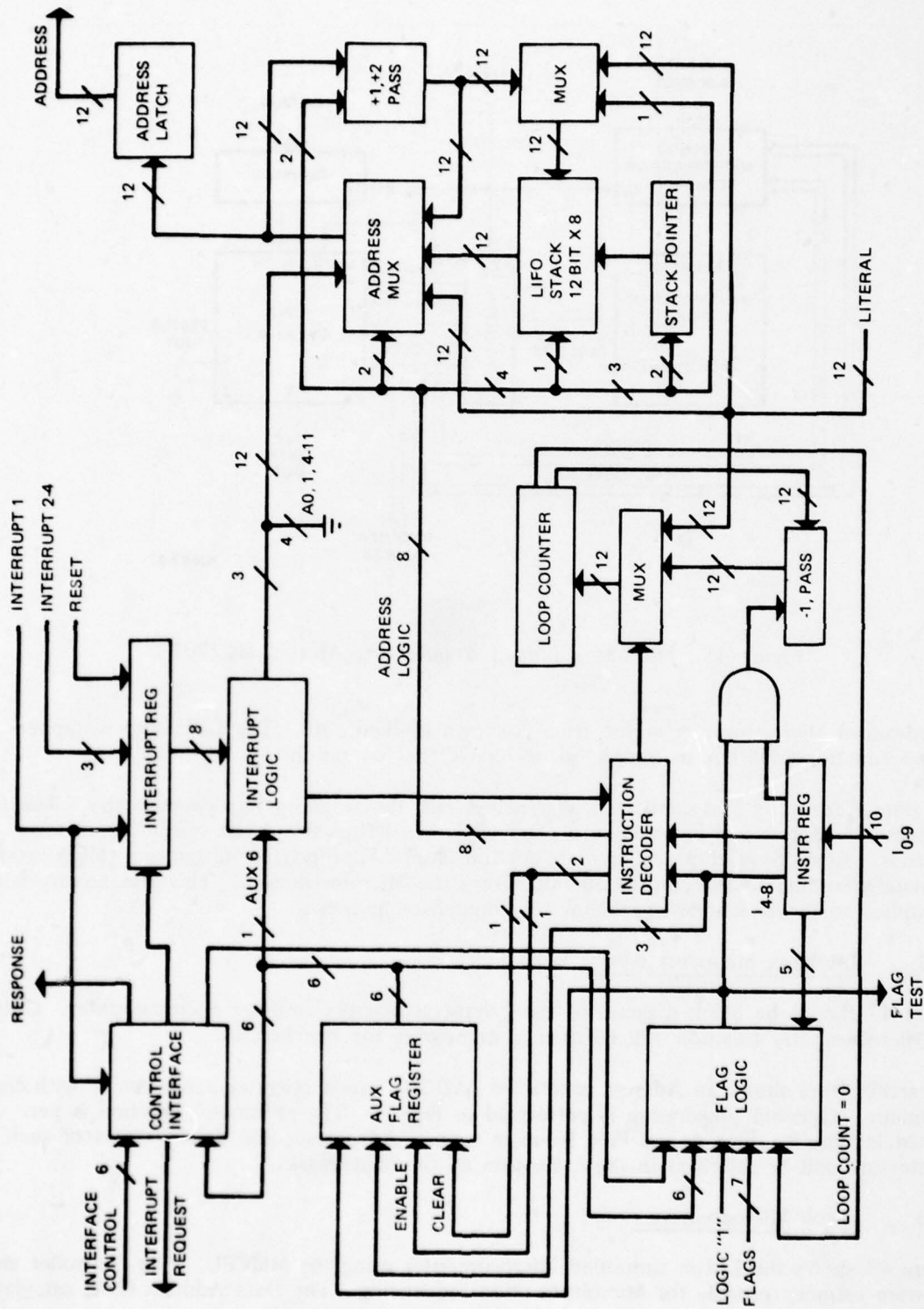
Second, it is assumed that two CPU type chips will provide for Data Processing (DP) as well as Data Addressing (DA) function. This architecture provides for concurrent data address updating as well as data processing. Note, in each time interval, 3 functions are performed - DA, DP, Control.

Third, it is assumed a special function chip provides for fast multiply.

Typically, all Microcomputers and Microprocessors have a very wide Microinstruction format which reduces the decoding logic and therefore gives a speed advantage. The number of functions performed by a CPU dictates the number of bits to be accommodated in the format.

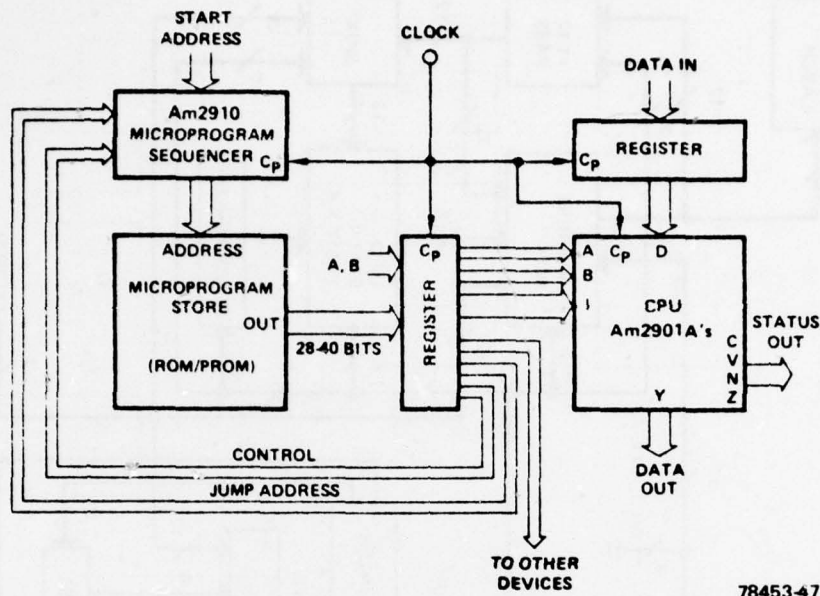
6.6.1 Tracor/MC2901 Microcomputer

A single Microcomputer architecture around the MC 2901 and MC 2909 is shown in Figure 45. This type of architecture is applicable to the Tracor GPU.



78453-25A

Figure 44. Litton Controller



78453-47

Figure 45. Microprogrammed Architecture Around MC2901's

An advanced Microcomputer architecture is shown in Figure 46. The GPU chip is applied to two functions, DP/DA to obtain further concurrent operation.

The Data Addressing (DA) and Data Processing (DP) functions operate concurrently. The DA provides for address and index computation while the DP performs the operation on the operands. A multiply chip is a special function chip. Appropriate multiplexers (MUX) route the data according to the control obtained from the Microinstruction. This architecture will be applied to the benchmark problems for comparison purposes.

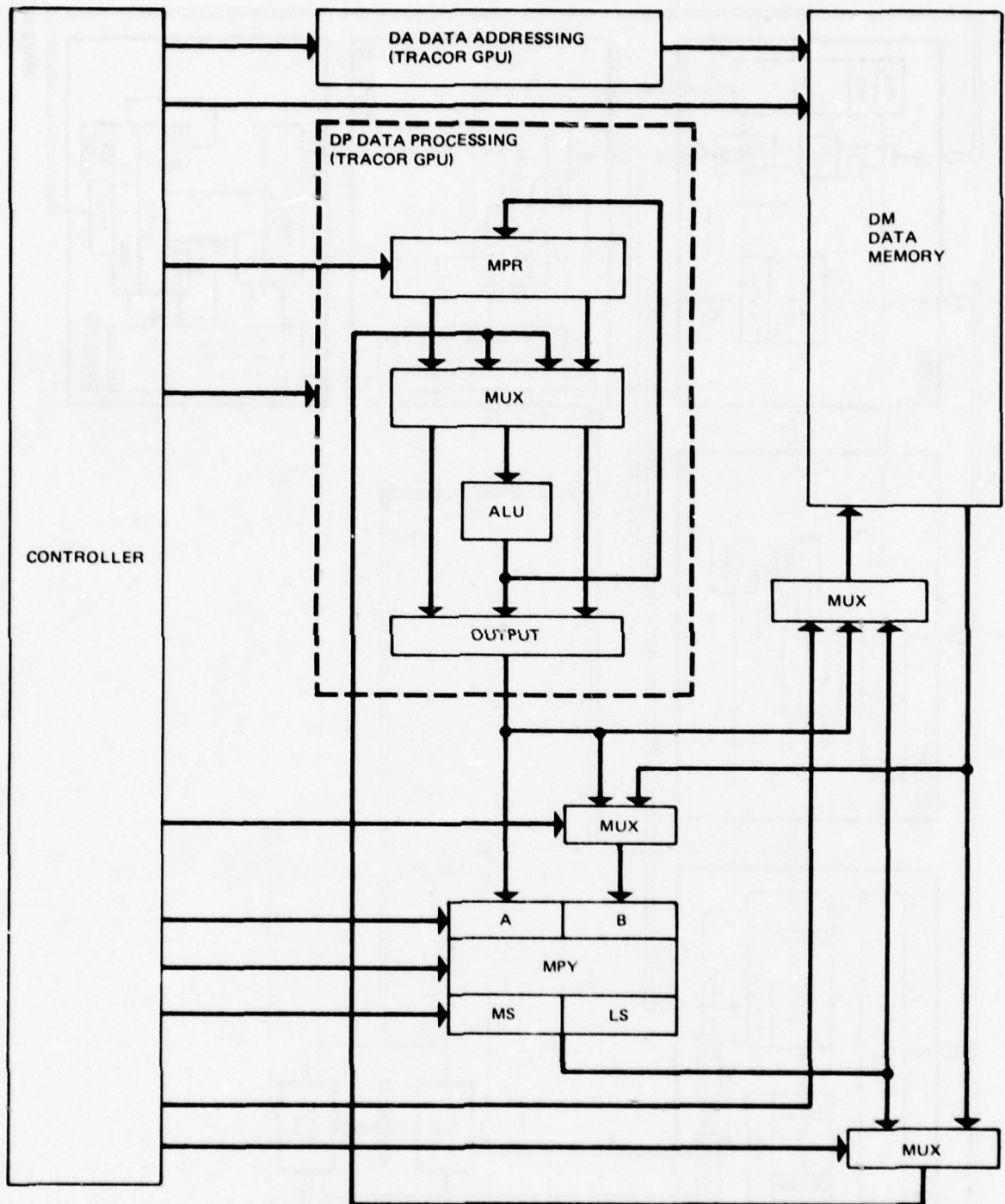
6.6.2 Raytheon Microcomputer

Figure 47 shows the block diagram of the Raytheon Macro Computer/Microcomputer. Only the Microcomputer function will be used in comparing the benchmarks.

The architecture shows an Address generation (ADGN) which operates concurrently with the arithmetic. Operand preparation is performed in Pipe A. The arithmetic function is performed in Pipe B. Pipe A and Pipe B are in series. Advantages and disadvantages of such an architecture will be reflected in the evaluation of the benchmarks.

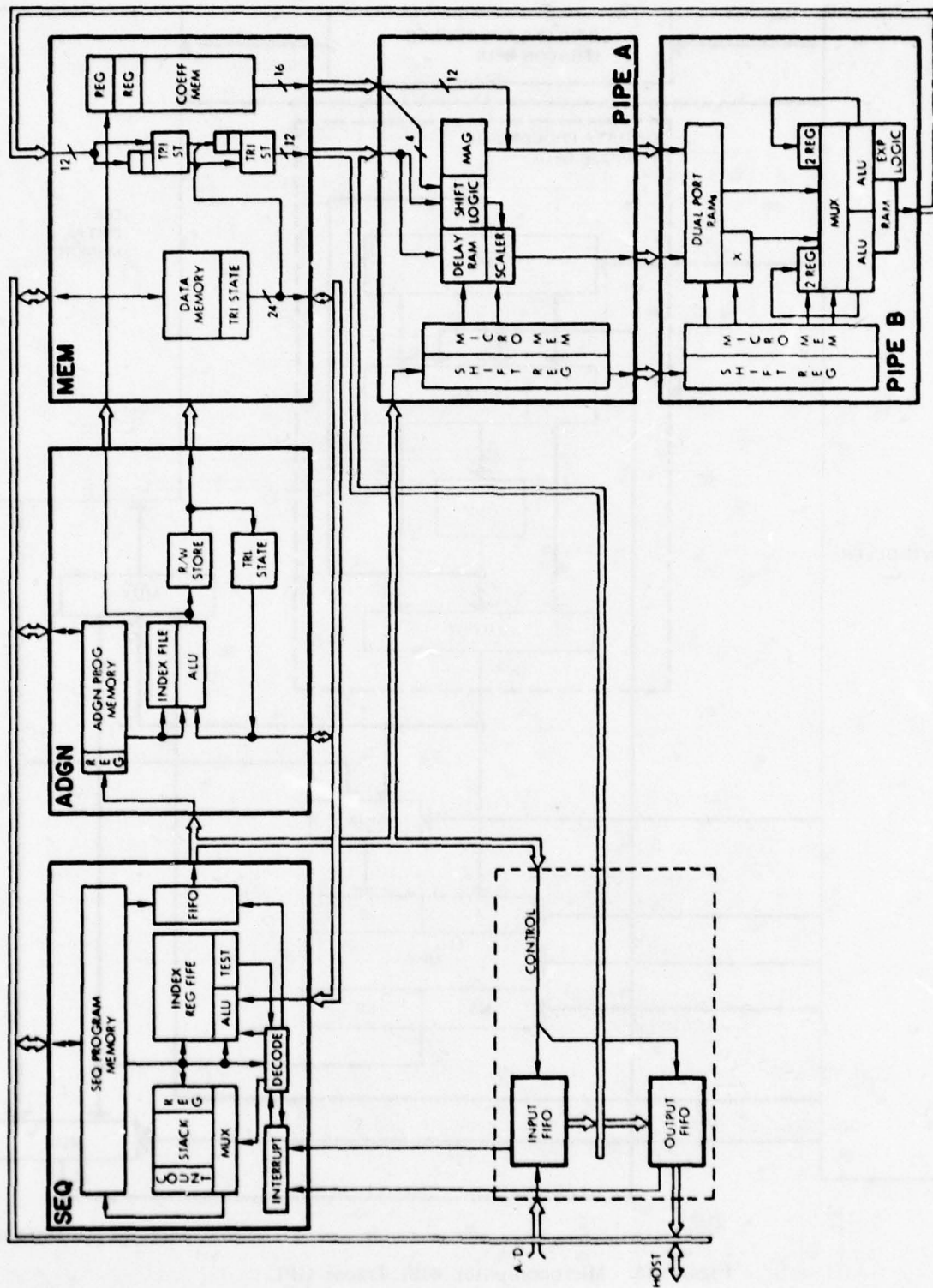
6.6.3 Litton Microcomputer

Figure 48 shows the Litton simplified Microcomputer using the MMCPU. The controller and firmware memory provide for Microinstruction sequencing. The Data Address (DA) calculates



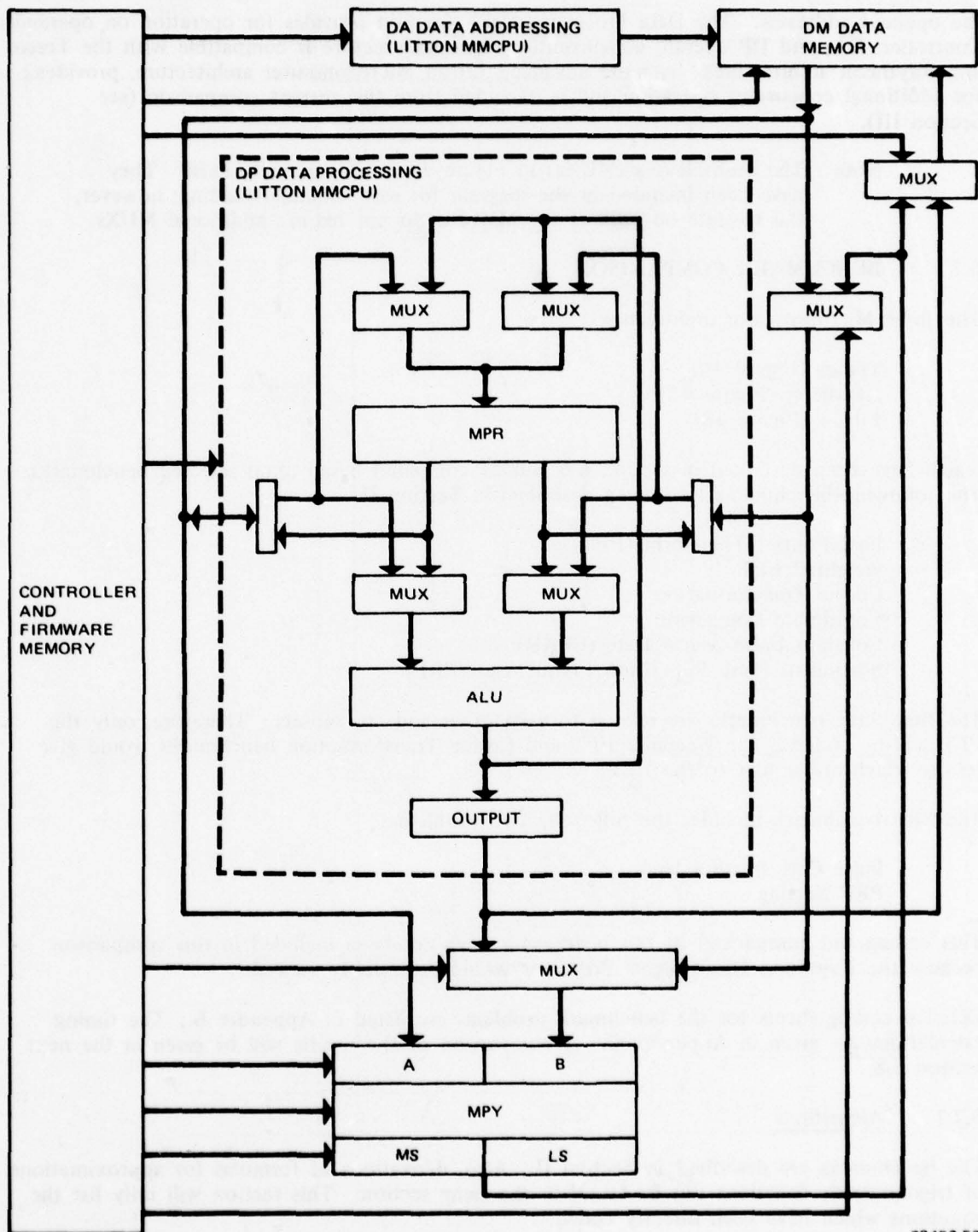
78453-48

Figure 46. Microcomputer with Tracor GPU



78453-49

Figure 47. Raytheon Microcomputer



78453-50

Figure 48. Litton Microcomputer

the operand addresses. The Data Processing (DP) function provides for operation on operands. Controller, DA, and DP operate concurrently. This architecture is compatible with the Tracor and Raytheon architectures. A more advanced Litton Microcomputer architecture, provides for additional concurrent operation and is excluded from the current comparison (see Section III).

Note: The multiplexers (MUXs) in Figure 48 do not actually exist. They have been included in the diagram for ease of understanding; however, the tri-state outputs of the MMCPU do not require additional MUXs.

6.7 BENCHMARK COMPARISON

The three Microprocessor architectures –

Tracor (Figure 46)
Raytheon (Figure 47)
Litton (Figure 48)

which have been described in section 6.6 will be compared based upon selected benchmarks. The following benchmarks have been described in Section II.

Fast Fourier Transform (FFT)
Weighted FFT
Cosine Transformation
Coordinate Conversion
Constant False Alarm Rate (CFAR)
Sorting of Pulse Repetition Frequencies (PRF)

The first three benchmarks are related to each other and are subsets. Therefore, only the FFT will be coded. The Weighted FFT and Cosine Transformation benchmarks would give results which are similar to the FFT.

The PRF benchmark includes the following two problems:

Pulse Classification, and
PRF Sorting

This coding and comparison of this benchmark have not been included in this comparison because the Raytheon Micro Signal Processor would be unfairly viewed.

Detailed coding sheets for the benchmark problems are listed in Appendix B. The timing calculations are given in Appendix C. A comparison of the results will be given in the next section 6.8.

6.7.1 Algorithms

The benchmarks are described in Section II. Also, derivations of formulas for approximations of trigonometric functions can be found in the same section. This section will only list the equations which have been directly coded.

Coding of isolated benchmarks can be misleading. The designer should always keep the total system in mind. Therefore, the benchmark may not give the total story. The programming strategy and style greatly depends upon the application. For example, buffer space may depend upon the coding of a single data point or the repetition of data points. In other words, a formula may be applied to one data point followed by the next data point. Alternatively, the first step of the formula may be applied to all data points followed by the next step in the formula. This will effect the indexing through the data base and affect the requirements for temporary storage as well as the throughput. Careful analysis influences the selection of an approach.

Small systems versus large systems can influence the coding. In small systems one can code routines in-line, i.e., with a minimal number of subroutines, jumps, calls, etc. This coding technique provides for high throughput at the expense of larger program (firmware) storage.

In larger systems, one would code routines as subroutines or call routines. This approach provides for efficient initialization of routines at the price of an overhead in calling the subroutines. Further, the subroutines may use registers which have to be saved at the entrance to the subroutine and must be restored before leaving the subroutine. The passing of parameters to the subroutines is provided by preassigned registers.

In total system programming, the initialization of subprograms, such as benchmarks, must be considered. This overhead is normally not included in benchmark problems and depends greatly upon the architecture.

6.7.2 Instructions and Timing

Conventionally benchmarks are compared upon the following parameters:

- Number of instructions – storage
- Operand and temporary storage
- Number of instructions executed – throughput
- Time to execute benchmark

The Tracor and Litton Microprocessors architecture are similar. Therefore, the analysis of Microinstructions are grouped together. The Raytheon is of a different type of architecture. Therefore, the total number of "MACRO" instructions will not be tabulated nor compared. The operand storage can be excluded from the comparison when assuming that the data bases for all Microprocessors are very similar. One should note that the total storage capacity in bits for the Raytheon architecture is smaller due to its 12 bit word length as compared with the 16 bit word length for the Tracor and Litton storage. Further, storage inefficiencies in the Raytheon architecture may occur due to its addressing structure. Always double words -12 plus 12 bits – are accessed by a single address.

The number of Microinstructions executed is significantly different from the Microinstructions in the program. This effect is due to the execution of loops in signal processing applications. The number of Microinstructions executed is proportional to the total throughput time. Thus, the time to execute a benchmark is a yardstick for comparison.

The Microinstruction execution time depends upon several parameters such as the logic speed which depends upon the selected technology. The speed of storage (memory) is reflected in

the execution time. Concurrent operation of Data Addressing and Data Processing functions allows a reduction in the total run time of certain algorithms. In algorithms, where the number of operand memory access is high, the reduction may be a factor of 2 or more. Pipeline or array processing may give additional speed advantage at the cost of additional hardware.

The clock speed in a Microprocessor depends upon the technology and the logic path through the logic. All these parameters make a comparison extremely difficult. Therefore, this study attempts to normalize the parameters for comparison purposes. This means that all architectures assume the same technology which includes storage speed as well as logic speed. The normalization factor is called "cycle." All comparisons are based upon the total number of cycles. The calculations of cycles is given below.

6.7.3 Tracor and Litton Microinstructions

The Microinstructions for the Tracor and Litton Microprocessors are very similar. The Microinstructions have been symbolized for the purpose of comparison in this report. Figure 49 shows the Microinstruction types. The types used for the benchmarks are classified into the following classes:

add, subtract, logic
multiply
square
shifts

M	-	MEMORY
M(X)	-	MEMORY, X - ADDRESS
M(IN)	-	MEMORY, IN - INDEX N ADDRESS
M(RN)	-	MEMORY, RN - REGISTER N ADDRESS
RN	-	REGISTER N, N - 0 TO E
RM	-	MULTIPLIER OUTPUT
MPYA	-	MULTIPLIER INPUT A
MPYB	-	MULTIPLIER INPUT B

78463-51

Figure 49. Microinstruction Symbols (Tracor/Litton)

Jump operations are assumed to be an option of each Microinstruction and operate in parallel with the above mentioned classes.

The difference between the Tracor and Litton list in Figure 50 is due to the architecture. The Litton multiport register file provides for 3 addresses as compared with the 2 addresses in the Tracor Microprocessor. Therefore, several Microinstruction or data manipulation options are provided in the Litton architecture which are not able to be duplicated by the Tracor architecture.

Two of the multiply operations have to be executed in two steps in the Tracor Microprocessor. This is due to the limited port logic in the General Processor Unit.

Figure 51 shows the Microinstruction timing in cycles. It is assumed that a Register to Register (R/R) operation is executed in one cycle which occurs concurrently with the access of the next Microinstruction. A jump instruction takes an additional cycle only when the jump has been executed. For example, the jump may depend upon the result of an operation; if the condition is false, then no jump takes place and no additional cycle is needed.

Storage is normally slower than the logic speed. For a simple approach, it is assumed that the speed factor is two. It is also assumed that a special multiplier chip requires two cycles to perform its operation.

Note: Generally, the multiplier speed is more than a factor of two slower than the CPU speed.

6.7.4 Raytheon Timing

The available documents show two versions of the pipeline architecture. One shows a Pipe A and a Pipe B, the other shows three stages in the pipeline. The overall timing of the pipeline is dependent upon the operand storage. For each clock period, one data word enters the pipe and one data word leaves the pipe.

During four clock periods, the memory reads from two addresses, a double word each and writes two double words into two addresses. Buffers coordinate the data flow as follows. AM use most significant 12 data bits of address "A." AL are least significant 12 data bits of address A.

Clock 1 - read AM, AL - AM to Pipe In - AL to buffer
CM from Pipe Out to buffer

Clock 2 - read BM, BL - BM to Pipe In - BL to buffer
DM from Pipe Out to buffer

Clock 3 - AL to Pipe In
- CL from Pipe Out - CM from buffer - write CM, CL

Clock 4 - BL to Pipe In
- DL from Pipe Out - DM from buffer - write DM, DL

Note: Address C and D are delayed by the sequencer to coincide with the data which has been delayed through the pipe.

TRACOR	LITTON
R1 = R1 + R2 (ADD, SUBTRACT, LOGIC)	R1 = R1 + R2
R1 = M + R2	R3 = R1 + R2
R1 = M + RM	R1 = M + R2 EITHER PORT
R1 = R2 + RM	R1 = M + RM
M = R1 + R2	R1 = M + M SAME LOCATION
M, R1 = R1 + R2 (WITH DA)	R1 = R2 + RM
RM = M * R2 SINGLE CLOCK/DELAY TIL PRODUCT	M = R1 + R2
R1, RM = M * R2 MULTICLOCK	M = R1 + R2
R1, RM = M * R2	M, R3 = R1 + R2
MPY B = R1	RM = M * R2 SINGLE CLOCK
M, RM = R2 * MPY B	R1, RM = M * R2 MULTICLOCK
MPY B = R1	R1, RM = M * R2
R3, RM = R2 * MPY B	M, R3, RM = R1 * R2
M = R1 * R1 SQUARE	R3, RM = R1 * R2
R1 = R1 * R1 SQUARE	M = R1 * R1 SQUARE
R2 = R1 * R1 SQUARE	R1 = R1 * R1 SQUARE
	R2 = R1 * R1 SQUARE
	R1 = M * M SQUARE
	RM = R1 * R2 SINGLE CLOCK
R1 = R1 SHR 1 OR 2	R1 = R1 SHR 1 OR 2
R1 = R1 SHL 1	R1 = R1 SHL 1
	R1 = R1 SHL 2

78453-52

Figure 50. Microinstruction Types (Tracor/Litton)

R/R		1 CYCLE
R/R	JUMP	2/1 CYCLE (JUMP/NO JUMP)
M/R		2 CYCLE
M/R	JUMP	3/2 CYCLE
MPY/R		2 CYCLE
MPY/R	JUMP	3/2 CYCLE
MPY/M		3 CYCLE
MPY/M	JUMP	4/3 CYCLE
R = REGISTER		
M = MEMORY		
MPY = MULTIPLY		

78453-53

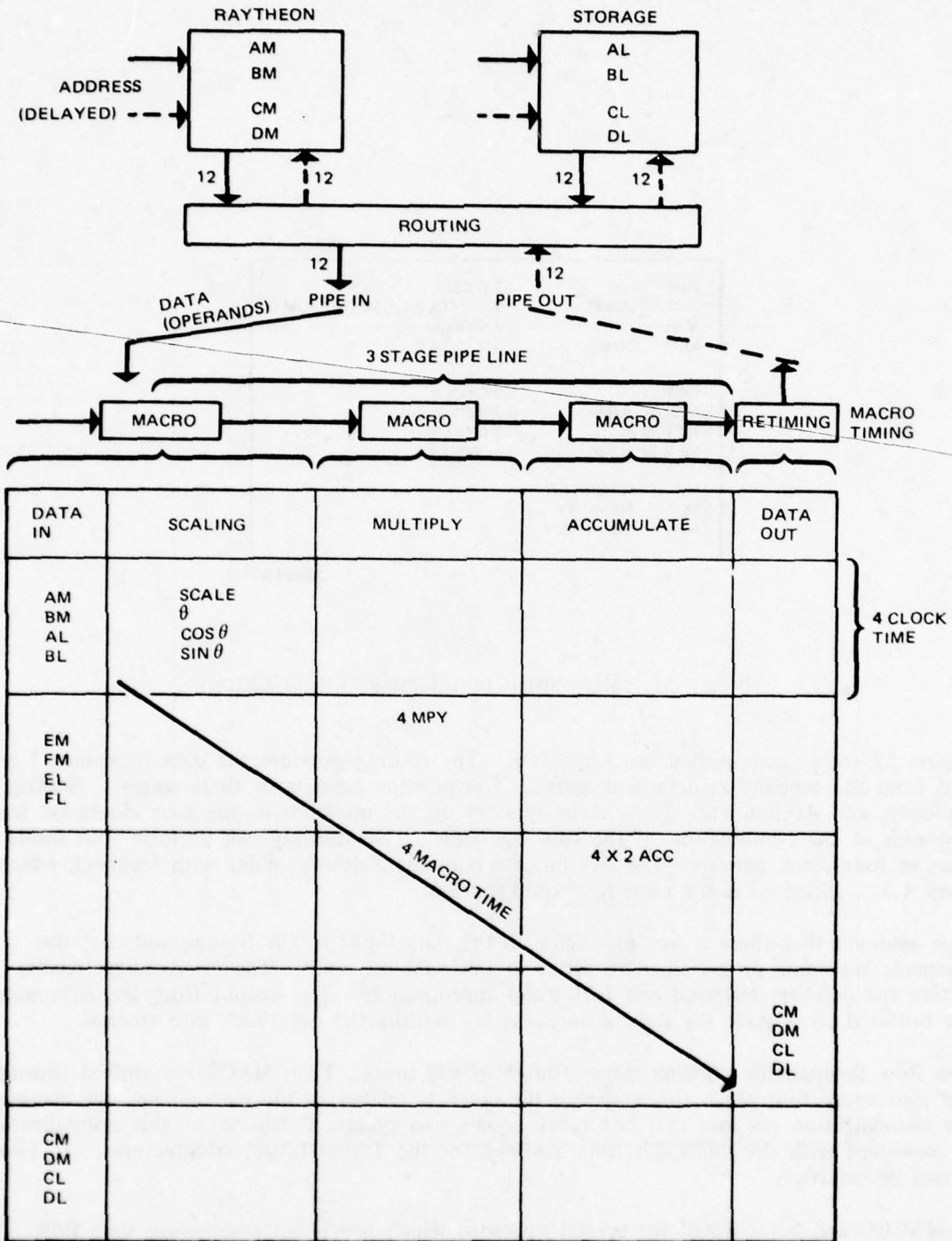
Figure 51. Microinstruction Timing (Tracor/Litton)

Figure 52 shows a simplified block diagram. The routing provides the data (operands 1 to and from the pipeline as described above. The pipeline consists of three stages – Scaling, Multiply, and Accumulate. Each stage operates on the operands during four clocks on four operands or the combination of the four operands. The Multiply can perform four multiplies in four clock periods. The Accumulate contains a double adder with feedback which gives 4 x 2 additions doing each four clock periods.

It is assumed that there is no time delay in the data input to the Scaling and that the operands are being programmed to arrive in the right sequence. The intermediate results within the pipe are buffered and forwarded appropriately. The output from the Accumulate are buffered to provide the right sequencing for writing the data back into storage.

The flow through the pipeline shows four MACRO times. Each MACRO is shifted through the pipe every four clock times. Since the clock is related to the data access, one assumes for normalization purpose that one clock equals two cycles. Furthermore, this normalization is consistent with the multiplier time assumed for the Tracor/Litton architectures, i.e., two cycles per multiply.

A MACRO can be repeated for several operands which provides a continuous data flow. The next MACRO in sequence may start immediately after the current MACRO has obtained the data. However, the data for the second MACRO (E/F in Figure 52) must be independent



78453-54

Figure 52. Raytheon Timing

of the data of the previous data output. Figure 52 shows a four MACRO delay (flush of pipe) when the data output (C/O in Figure 52) is to be used as a next data input.

Data set-up and address generation are assumed to operate appropriately.

6.7.5 Fast Fourier Transformation (FFT)

Coding of one butterfly for the FFT will be shown. The algorithms are shown on the coding sheets.

Appendix B1 shows the *Coding* for the Tracor architecture. The Litton architecture uses the same coding.

Appendix B2 shows the Raytheon *Coding*.

Timing calculations are shown in Appendix C1 for the Tracor/Litton architecture and in C2 for Raytheon. A 1024 point FFT was assumed which requires ten passes. The Tracor/Litton architecture uses a single accumulator and a single multiplier which is equivalent to a "real in-place FFT." The Raytheon architecture performs a complex in-place FFT. Adding hardware to the Tracor/Litton architecture to provide "complex" calculation will reduce the time by a factor of 2 to 4. (See Section III.)

Note, the results are tabulated in Figure 53.

BENCH MARK \ PROCESSOR	TRACOR	RAYTHEON	LITTON
FFT	153,600	40,948	153,600
COORDINATE CONVERSION POLAR TO RECT	68	16	65
RECT TO POLAR	222	16	212
CFAR BIT PACKED	63,499	-	63,499
NOT BIT PACKED	37,897	33,328	37,897

78453-55

Figure 53. Microprocessor Cycles for Benchmarks

6.7.6 Coordinate Conversion

There are two parts to the coordinate conversion.

- a. Polar to Rectangular
- b. Rectangular to Polar

Appendix B3 shows the Polar to Rectangular *Coding* for the Tracor architecture. The coding takes in account in which quadrant the angle is located. The trigonometric functions, sine and cosine, use an approximation for an angle of less than $\pi/4$. The sign of sine and cosine are selected according to the quadrant. The Litton coding differences are indicated and show a reduction of coding due to the multiply function.

Appendix B4 shows the Polar to Rectangular *Coding* for the Raytheon architecture. The simplicity of the coding is due to the built-in hardware function which directly provides the trigonometric function. Due to the hardware duality in the pipeline, two data points can be converted at the same time in a single macro. It should be noted that the coding assumes that the pipeline receives first quadrant data. If the angle must be tested, a severe penalty must be paid to perform one or more data-dependent branches.

Appendix C3 shows the Polar to Rectangular *Timing* for the Tracor and Litton architecture. The timing is for a single point. The angle is assumed to be in the second quadrant. The path through the coding is shown. The Litton architecture uses fewer instructions as indicated by the # symbol.

Appendix C4 shows the Polar to Rectangular *Timing* for the Raytheon architecture. Again, a single point conversion is assumed. Since 2 conversion per macros are performed, it is reflected in the timing calculation.

Appendix B5 shows the *Coding* for Rectangular to Polar conversion for the Tracor and Litton architecture. The square root (SQ) computation uses an estimation algorithm. Note, the coding tests for negative numbers which make the coding applicable as a general routine. Negative numbers are converted to positive numbers.

The calculation of the angle uses an approximation of the arc sine (\sin^{-1}) function. This function is similar to the sine function and differs only in the constants (see B3). A test of the coordinates is performed to determine whether the angle will be smaller or larger than $\pi/4$ in a quadrant. After calculation of the angle, the appropriate quadrant is being determined.

The algorithm requires divisions. Since the multiply chip does not contain the divide function, a separate approximation is being used which computes more than two bits of the quotient per iteration. Since the divide is a general subroutine, several tests are being made to determine whether the dividend is zero or the divisor is zero. Dividend, Divisor and Quotient are passed by Registers 1, 2, and 0 respectively. Other registers being used are appropriately saved/restored.

Appendix B6 shows the Rectangular to Polar *Coding*, for the Raytheon architecture. The angle is determined directly by the built-in trigonometric function assuming it will give the

appropriate quadrant. The R is determined by a multiplication and addition rather than by a square root. It takes advantage of the built-in hardware which provides trigonometric functions. Unlike the polar-to-rectangular conversion, the angle estimation hardware does not need quadrant information for this conversion. This function provides a real and significant performance improvement.

Appendix C5 shows the Rectangular to Polar *Timing* for the Tracor and Litton architecture.

Appendix C6 shows the Rectangular to Polar *Timing* for the Raytheon architecture.

6.7.7 Constant False Alarm Rate (CFAR)

The Constant False Alarm Rate (CFAR) is a sliding window CFAR benchmark. It assumes a 256 cell window. The range contains 4096 cells. Each cell assumes a six bit positive value from the A-to-D conversion stage.

Appendix B7 shows the CFAR *coding* for the Tracor/Litton architecture. The sliding window is defined by the Indices I1 and I2. The midpoint is I4. The threshold decision is one bit. Each 16 consecutive decisions are packed into one word (Index I5). The resulting word is stored by Index I3. Operation is in real time. The coding takes full advantage of the indexing capabilities of the Data Addressing function.

Appendix B8 shows the CFAR *coding* for the Raytheon architecture. The first macro accumulates the window, four bits at a time. The second MACRO updates the window for the next two cells; this is a pass over the whole range. After flushing the pipe, the next macro performs two decisions. Note, the decisions are not packed. The sign of each word represents the decision.

The operation is not in real time since the range has to be processed twice requiring large intermediate operand storage. If there would be a feedback from the accumulator into the multiplier, then the accumulator would be available internally to the pipe. In other words, the storing of the accumulation "S" would be eliminated. Each decision would be made from the internal accumulation with an accessed midpoint. The accumulator would be updated from 2 range cells. This scheme requires clever arrangement of the range cells. Program execution would alter between a pair of macros after 128 decisions each.

Appendix C7 shows the CFAR *Timing* for the Tracor/Litton architecture. Note, the processing is in real time and each 16 decisions are packed into a word.

Appendix C8 shows the CFAR *Timing* for the Raytheon architecture. Processing is not in real time and decisions are each in a separate word.

Appendix C9 shows the CFAR *Timing* for the Tracor/Litton architecture. This coding and timing assumes that each decision is stored in a separate word. Note, operation is still in real time, but coding is shorter and therefore the timing has been reduced significantly as compared with Appendix C7.

6.7.8 Microprocessor Cycles for Benchmarks

Figure 5-3 tabulated the microprocessor cycles for each benchmark and for each microprocessor architecture. A discussion of the results follows in the next section.

6.8 COMPARISON OF RESULTS

The comparisons of microprocessor architectures in this section are based upon the results presented in the section 5.8. The results are based upon the evaluation of three microprocessor architectures and its application on signal processing benchmarks. The evaluations are based upon available documentations. Some of the documentation may be voluminous but lacking of necessary details to allow sufficient analysis. Therefore, in many instances, assumptions have been made assuming the good guesses are right. The lack of details and consistency in the available documents may be due to the state in which the presented architectures were at the time of their publishing. This reflects in the appearance of the architectures to be conceptual rather than designed or being implemented.

An effort has been made to evaluate the architectures in the best light and to be rather optimistic than pessimistic. Parameters have been normalized to provide a fair comparison. Despite adverse circumstances, significant discoveries have been made. Further studies and subsequent comparisons may make use of these facts and a refinement of the analysis, evaluation and comparison of results may be achievable.

6.8.1 Architecture Comparison

The comparison of the Tracor, Raytheon and Litton microprocessor architectures showed the following:

- a. The Tracor and Litton architectures are very similar; the Raytheon architecture is different. Tracor and Litton architectures are readily enhanced to reflect an array processor architecture, similar to Raytheon's architecture.
- b. All three architectures have a controller/sequencer which fetches microinstructions and operates concurrent with microinstruction execution.
- c. All three architectures have data addressing/address generator hardware which operates concurrent with the data processing/pipeline.
- d. All three architectures have a multiply hardware as a special function. The Raytheon architecture reflects the State-of-the-Art "optimum" in this respect, i.e. *a multiplier followed by an accumulator*. That function reduces the data traffic on the data bus which is extremely important in signal processing. (See multiplier discussion in chapter 3)
- e. The Tracor/Litton architecture have a single Data Processor and operate on a single operand (data point) at a time. Raytheon has a parallel Pipeline which can operate on two operands (data points) at a time due to its dual data path and dual arithmetic. Therefore, more hardware provides for apparent higher throughput as compared with the Tracor/Litton architecture.
- f. It is anticipated that a Tracor/Litton array processor architecture provides a speed advantage of 2 to 4 over a single Data Processing architecture.

- g. The Litton CPU chip as compared with the Tracor GPU chip has more ports on the chip; and these ports are bidirectional. Thus, fewer microinstructions are required to route data, and more powerful microinstructions are provided in the repertoire. This is reflected in the benchmark coding. Furthermore, operation as a data addressing chip is enhanced because additional processing of literals is obtained.
- h. The Litton CPU has Multi Port Registers (MPR) with three addresses and performs three operations on all three addresses. Litton can read from two addresses and write into a third address. Tracor has a two address, three operation MPR. Tracor can read from two addresses; one of those addresses can be used to write data back into the MPR. The three address feature in the Litton CPU did not provide an advantage in the given benchmark coding; however, the advantage is most significant in Data Addressing.
- i. Special divide algorithm shows not a significant improvement over conventional algorithms due to a large overhead in determination of special cases of dividend and divisor.

6.8.2 Timing Comparison

Figure 54 is a summary of the benchmark comparison expressed in cycles (normalized). This figure also shows estimates for array architectures. The speed improvement of an array architecture over a single Data Processing architecture is assumed to be a factor 2 to 4.

BENCH MARK	PROCESSOR				
	TRACOR	RAYTHEON	LITTON	ARRAY 1/2	ARRAY 1/4
FFT	153,600	40,948	153,600	76,800	38,400
COORDINATE CONVERSION POLAR TO RECT	68	16	65	32	16
RECT TO POLAR	222	16	212	106	53
CFAR BIT PACKED	63,499	-	63,499	31,749	15,876
NOT BIT PACKED	37,897	33,328	37,897	18,948	9,474

78453-56

Figure 54. Microprocessor Cycles for Benchmarks

Figure 54 shows the following:

- a. Litton as compared with Tracor timing shows a small advantage due to the micro-instruction repertoire which requires fewer executions.
- b. Raytheon as compared with the Tracor/Litton Timing shows to be superior on the surface. Therefore, a detailed comparison follows.
- c. FFT – Raytheon is significantly better than Tracor/Litton timing. When compared with an array architecture, i.e., equivalent or less hardware, the timing is about equivalent.
- d. Coordinate Conversion/Polar to Rectangular – The array architecture timing is competitive with Raytheon.
- e. Coordinate Conversion/Rectangular to Polar – the array architecture is significantly slower than Raytheon. Raytheon's speed advantage is due to the hardware built-in trigonometric special functions to avoid data-dependent operations.
- f. CFAR – The Tracor/Litton bit packed algorithm is a factor of two slower as compared with the non-bit packed algorithm. The trade-off is speed vs storage requirement. Comparing the non-bit packed Raytheon timing with the Tracor/Litton timing shows about the same figures. However, an array architecture is much faster than the Raytheon timing. A bit packed Raytheon algorithm would be even slower because the algorithm necessitates data-dependent branches. This shows the Raytheon architecture is not geared for this type of application.
- g. Special functions such as trigonometric hardware provides a speed advantage.
- h. Divide function, if required, in the Raytheon architecture would be slow.
- i. Divide function should be incorporated into the multiply chip.
- j. Importance of the Data Addressing function has been shown in the CFAR benchmark and can be shown in the total FFT. Several index operations are performed concurrent with Data Processing.

6.8.3 Conclusions

- a. Special purpose processor is faster than general purpose processor, i.e., a processor which has built-in trigonometric functions.
- b. Special purpose processor is inflexible as compared with general purpose processor i.e. CFAR, divide, non-first quadrant angles.
- c. Raytheon computer is a type of array processor i.e., 2 data path in 2 pipe stages = 4 processor equivalent.
- d. Array processors are faster than single processors because of parallel processing.

- e. Divide in the Raytheon computer is extremely slow. Square root, if necessary, in the Raytheon computer has an unknown implementation.
- f. Raytheon should have feedback from the adder to the multiplier to improve its ability to process the CFAR.
- g. Multiply chip should include divide to significantly speed-up processing. How often the divide is needed is unknown?

6.9 REFERENCES

- a. Critical Item Development Specification for General Processor Unit, Performance/ Design Requirements and Technical Description. Appendix D, Preliminary CDRL E003. Prepared for Space and Missile System Organization Air Force System Command. Contract No. N0024-73-C-1131, Modification P00004, prepared by Tracor, Inc., 6500 Tracor Lane, Austin, Texas, 78721, 23 April 1974, Document No. T74-AU-9073-U.
- b. MC2901. Four-Bit Bipolar Microprocessor Slic, includes "using the MC 2901", "MC2909 Microprogram Sequencer", and "using the MC2909" by Motorola.
- c. Microprogramming ups your options in up-System Design, part 2, by Jim Bride and John Mick, Advanced Micro Device, EDN February 5, 1978, pages 53-61.
- d. High Speed Micro Signal Processor Study Final Report Draft. Prepared for Air Force Avionics Laboratory, United States Air Force, Wright-Patterson AFB, Ohio. Contract No. F33615-76-C-1339/CLIN:0003, Prepared by Raytheon Company, Missile Systems Division, Bedford Laboratories, Bedford Mass. January 1977. BR-9633.
- e. Initial Briefing to WPAFB/AFAL for High Speed Micro Signal Processor. Contract F33615-77-C-1224, AF Raytheon Company, Bedford, Massachusetts, 7 December 1977.
- f. Proposal for High Speed Microsignal Processor, Volume I - Technical. Submitted to Unit States Air Force, Air Force Systems Command, Aeronautical Systems Division/PPMEA, Wright-Patterson AFB, Ohio 45433. In response to solicitation No. F33615-77-R-1224. Prepared by Data Systems Division, Litton Systems, Inc., 8000 Woodley Avenue, Van Nuys, California 91409. 21 July 1977, MS 77357-1.
- g. Current Study Contract

SECTION VII

CONCLUSIONS AND RECOMMENDATIONS

7.0 INTRODUCTION

The attempt to design a single large scale integrated circuit, the Multimode CPU has revealed some interesting insights into digital signal processor design, LSI technology and the signal processing problem set. Analysis has shown that the Data Processing, Data Addressing and Instruction Address functions are all within the reach of LSI.

In this chapter, conclusions will be drawn and recommendations will be made in the area of the Benchmarks, the Architecture, the Technology, and the Comparison. The conclusions, presented herein, are supported in the preceding chapters.

7.1 CONCLUSIONS

7.1.1 Benchmarks

The problem set in Section II was included to be representative of the signal processing tasks required presently and through approximately 1990. The problem had to be bounded so that the MMCPU could be designed to span a wide range of applications and still be "specialized" enough to handle the unique requirements of the problem set. The tasks can be separated into high speed and low speed requirements. The FFT along with the weighted FFT and cosine transform are extremely high speed computation problems. The pulse classification algorithm requires a high speed computation but more importantly, a high speed data dependent testing capability. The other problems are lower speed and will not be discussed here.

The FFT and the classic Cooley-Tukey butterfly have a very orderly and repetitive arithmetic flow and a simple addressing scheme. From the analysis of multiplier structures in Section III it is concluded that the optimum processor structure for the FFT is a hardware special function unit which performs all the butterfly arithmetics. Furthermore, because the addressing requires simple additions and tests, a fairly simple, but high speed, general purpose RALU or CPU is required to support the special function unit.

The pulse classification algorithms is virtually at the other end of the processing spectrum. Although the operand set-up requires a repetitive set of adds and multiplies for calculating the distance measure, the bulk of the processing (about 23 percent by actual operation count) is involved in testing and selecting a branch path from the outcome of the test. It is concluded that a very sophisticated, high speed, general purpose CPU is required with a multiplier to support it.

The emphasis of the CPU and the special function unit (butterfly unit or multiplier) is reversed. Part of the objective of this program was to determine if a single architecture could accomplish this task. It is concluded that a single architecture would be somewhat inefficient

but a single CPU structure capable of handling the pulse classification problem can be defined which would be more than sufficient for the remaining tasks in the problem set.

7.1.2 The Architecture

To support the high I/O rate of the FFT butterfly, the addressing unit must supply addresses to the data memory so that operands can be read or written at a high rate; therefore, the bus system must support the FFT speed requirements. In Section III, the bus system was defined as a result of the problem set and became the prime concern for supporting the multipliers and RALUs. It is concluded from a top-down viewpoint that the processing system should be built around the maximum bus necessary for the job. By defining the bus first, the speed requirements for the processing elements or the speed limitations of the processor imposed by the bus are clearly established.

After the bus requirements were established, the multiplier structure was investigated in relation to the problem set. It is concluded that the processor structure is highly dependent on the multiplier structure as evidenced by the two DP/DA structures presented in Section III. For a maximization to the "general purpose" goals of the MMCPU, the Multiplier/FFT structure is preferred because it handles the high speed problems of the FFT, weighted FFT and Cosine Transform, as well as the other problems in Section II.

Array processing is generally a difficult procedure because bus transfers, resource sharing, etc. is difficult. The architecture presented was developed with a desire to expand via array processing so that the processing speed could be increased. The limitations of bus transfers, resource sharing, and timing were resolved by allowing only nearest neighbor intercommunication and several I/O Data Ports; furthermore, the processors must be operated in lock-step or time synchronism if the array approach will work with maximum efficiency.

7.1.3 The Architectural Comparison

Using the constraints of Section VI, several significant points can be concluded about the Tracor/RCA processor (GPU), the Raytheon processor (Micro-Signal Processor), and the Litton processor (MMCPU).

The GPU is similar to the 2901 processor. It is excellent for general purpose problems including emulation. The RALU structure provides great flexibility of operation; however, the limited number of I/O ports inhibits the bus interconnection flexibility necessary for signal processing. Furthermore, the I/O limitations make array processing virtually impossible because the data buses must be tied to each other, thereby, forcing a battle for bus usage.

The Micro-Signal Processor is designed specifically for signal processing problems. It will handle the FFT, weighted FFT, and Cosine Transform well because the pipeline structure is oriented toward the FFT. It resembles the multiplier/accumulator with holding registers discussed in Section III. The major weakness of the architecture is the lack of provision for data dependent operations; therefore, EW problems are beyond its scope. Any data dependent testing must be done in the sequencer, and the result must wait for the FIFO to clear before it can be implemented. The primary problem is the overdependence on the pipeline for all arithmetics.

The MMCPU is similar to the GPU and the 2901 RALUs; therefore, it is extremely capable of the general purpose problems. It has enough ports available to give it signal processing flexibility in a single or array configuration. The major limitation to this processor is the multiplier. Current commercially available multipliers are not responsive to the FFT needs. Without such a multiplier/FFT unit, this processor is greatly limited for the FFT type problem; however, the EW problems are well within reach.

A final point should be made. The pipeline arithmetics of the Raytheon processor along with the MMCPU for general processing and data dependent operations would be a powerful processor configuration.

7.1.4 The Technology and the MMCPU

In section 5.3, CMOS/SOS, I³L, and VMOS were analyzed, and each is capable of high gate count LSI. Although the 1500 to 1750 gate 8-bit RALUs appear to be a limit for the technologies, any of these technologies could perform well now or in the near future.

The final question remains. Is the MMCPU chip concept feasible? If so, in what context? To answer this question; the gate count must be estimated. Using the RALUs as a basis, the microsequences and loop counter could be accomplished by using the MPR/ALU functions of the RALUs. Approximately 200 additional gates would be necessary in the MPR so that it could perform as a 16 word by eight bit register file and as an 8 word by 12 bit LIFO stack.

The interrupt control unit, flag logic, and instruction addressing instruction decode would have to be included on the single chip. Another 500 gates would be necessary. Because the IA made is significantly different than either the DP or DA mode, additional gates would be necessary to permit multiple modes at the I/O data ports as well as to allow 12 bit instruction addresses to be generated instead of 8 bit data I/O. Lastly, the internal chip buses would have to be structured to accommodate 8 bit data and 12 bit instruction addresses. The total estimate for an MMCPU is between 2300 and 3000 gates.

From a commercial point of view, the single chip is inefficient, requiring a significant amount of the chip to be unused in various modes. Unused portions of a chip are costly because when the unused portions of the chip are stripped away, the chip is smaller, the yield is higher and the cost is lower. However, from a military point of view, a single chip type may offset the cost of unused portions of the chip. A single chip type reduces the number of types that must be supplied. Lower life cycle cost can be aided by such reductions.

A single chip might be advisable to the military; unfortunately, the problem set requires high speed gates be utilized to perform the FFT and EW problems. As previously discussed, higher speed means higher gate dissipation. Power is the major limitation. The total chip power dissipation would be greater than 3 Watts for any of these technologies. The only way the MMCPU chip would be feasible is if the unused functions on the chip were not powered. Such a scheme is possible, but generally not practical.

It is, therefore, concluded that the MMCPU chip concept is not feasible in today's technology. A two chip type system — one a DP/DA RALU chip, the other an IA controller chip — would satisfy the needs of the complex processors discussed in Section III.

At the present rate of increase in technology, the single chip concept remains two to three years away for high speed applications. Lower speed chips are possible today which may mean that a lower speed version could be developed and utilize the array processing concept to perform the higher speed problem.

7.2 RECOMMENDATIONS

7.2.1 Electronic Warfare

The Electronic Warfare problem was briefly discussed in Section II as one of the benchmarks for the MMCPU. The ultimate solution was not presented and is not totally known. The solution given herein is a fairly simple-minded approach, assuming all the emitter data parameters are the same word length. In truth, the word lengths are greatly different, and weighting would be necessary to "standardize" the word lengths for processing.

The pulse classification algorithm has a very repetitive distance-measure calculation using the parameters. Array processing should be explored as a means of greatly increasing the speed of the simple calculation via parallelism. A possible solution is simple hardware for the calculation and an MCCPU for probability comparisons. More study of architectures is needed in this area.

7.2.2 Array Processing

Array processing has been presented herein in a very limited manner. The approach given is a cross between the full parallel processor and the multiprocessor. The major area of application for the nearest-neighbor approach is very structured problems such as signal processing. The concept needs more study in two areas. 1) Slower processors are possible if more arraying can be efficiently done. 2) Processor speed increases may be possible without obviating software for the slower processor because one processor simply works twice as fast as its predecessor, thereby doing the work of two. Both areas seem quite fruitful.

7.2.3 Demonstration of MMCPU

A practical demonstration of the MMCPU to demonstrate the concept and to study the arraying possibilities in EW and other applications is necessary to "prove the concept." The demonstration processor could be built of 2900 series parts very easily because the MMCPU is very similar to the RALU's and microsequences of that series. The necessary speed could not be simulated but the function could be proven.

7.2.4 VMOS Technology

Lastly, the VMOS technology should be carefully watched as a potential LSI signal processing technology. Although it is an NMOS variation, many of the temperature range problems seem to be ameliorated. With the possibility of miniscule channel lengths using standard geometry rules, this technology has the potential of outdistancing every available technology in the LSI field.

APPENDIX A

TIMING BENCHMARKS FOR COMPLEX PROCESSORS

TASK: PERFORM BUTTERFLY FOR FFT.

Algorithm:

A + B are input points

X + Y are output points

$$TR = BR*CO - BI*SI$$

$$TI = BR*SI + BI*CO$$

$$XR = AR + TR$$

$$XI = AI + TI$$

$$YR = AR - TR$$

$$YI = AI - TI$$

Alternate Algorithm:

$$TR_1 = BR*CO$$

$$XR_1 = AR + TR_1$$

$$TR_2 = BI*SI$$

$$XR = XR_1 - TR_2$$

$$TI_1 = BR*SI$$

$$XI_1 = AI + TI_1$$

$$TI_2 = BI*CO$$

$$XI = XI_1 + TI_2$$

$$YR_1 = AR - TR_1$$

$$YR = YR_1 + TR_2$$

$$YI_1 = AI - TI_1$$

$$YI = YI_1 - TI_2$$

PROCESSOR 1

Task 1: FFT Butterfly

Coding:	Timing (in cycles)
R1 = M(B)	2
R2 = M(C)	(2) if necessary
R3R = R1R*R2R	2
R3I = R1R*R2I	2
R4R = R1I*R2*	2
R4I = R1I*R2R	2
R3 = R3 + R4	1
R0 = M(A)	2
M(X) = R0 + R3	2
M(Y) = R0 - R3	2
	<hr/>
	17 (19)

Total Timing:

4097 Butterflies require loading of A and B

1023 Butterflies require loading of A, B, and C – the new rotation vector

$$\begin{aligned}\text{Total Cycles} &= 17 \times 4097 + 19 \times 1023 \\ &= 89086 \text{ cycles}\end{aligned}$$

PROCESSOR 2

Task 1: FFT Butterfly

Coding:	Timing (in cycles)
MPYL1 = M(C)	(1) if necessary
MPYL2 = M(B)	1
MPYL3 = M(A)	1
COMPLEX MPY = L1*L2	2
COMPLEX ADD L3+CMPLY	1
COMPLEX ADD L3-CMPY	1
M(X) = CADD	1
M(Y) = CSuB	1
	<hr/>
	8/9

Total Timing:

4097 Butterflies require loading of A and B

1023 Butterflies require loading of A, B, and C – the new rotational vector

$$\text{Total Cycles} = 8 \times 4097 + 9 \times 1023$$

$$= 41983$$

TASK: PULSE CLASSIFICATION

Algorithm:

ERROR

RO, = M(BI) get bi
RI = MC(BJI) get bji
R2 = RO - R1
R2 = R2*R2 (bi-bji)²
R2 = M(SJI)*R2 (bi-bji)²
R3 = R3 + R2: IO = IO + 1 : JUMP ERROR IF IO NE I1
R4 = R3 shift right EJ = -EJ/2
R5 = M(R4) Memory look-up exp (R3)

[Could calculate the exponential]

R5 = R5*M (SPJ) M(SPJ) = Pj

ALu = R5 - R6: JUMP JCOUNT IF ALu 0

R6 = R5 M(I5) = I2, I5 = I5 + 1 Store Cj

JCOUNT :JUMP THRES IF IR = JMAX

I2 = IR+1 :JUMP ERROR

THRES ALU = R5 - M(T); PC = PC+2 IF ALU 0

M(T) is the threshold

:JUMP ERROR

PROCESSOR I AND II

Task: Pulse Classification

Coding:

Timing (in cycles)

CLR MPR	1
I0 = 0	1
I1 = IMAX	1
I2 = JSTART	1
I3 = JMAX	1
I4 = JDELTA	1
I6 = THRESHOLD	1
	<hr/>
	7
ERROR R0 = M(I0) : I5 = 2*I0	2
R1 = M(I5 + I2 + 1)	2
R2 = R0 - R1	1
R2 = R2*R2	2
R2 = R2*M(I5 + I2 + 1)	3
R3 = R3 + R2; I0 = I0 + 1:JUMP ERROR	1/2
	<hr/>
IF I0.NE.I1	11/12
I5 = -R3 Shift Right	2
R2 = M(I5)	2
R2 = R2*M(I2)	3
ALU = R5-R6 : JUMP COUNT IF ALU 0	2/3
	<hr/>
	9/10
JCOUNT R6 = R5 :I6 = I2	2
:JUMP THRES IF I2 = IMAX	1/2
:I2 = I2	2
	<hr/>
	5/6
THRES ALU = R5 - M(I2 + I6) :SKIP IF ALU 0	2/3
:JUMP ERROR	1
	<hr/>
	3/4

STORAGE ROUTINE - END OF ALGORITHM

TOTAL TIMING:

SETUP - 7 cycles

CLASSIFICATION

Assuming 100 possible classes

ERROR ROUTINE

Assuming 4 parameters to calculate distance

4 total passes per class

3 require jumps
1 requires no jump

$$\text{Total per class} = 11 \times 1 \times 12 \times 3 = 47$$

$$\text{TOTAL} = 100 \times 47 = 4700 \text{ cycles}$$

PROBABILITY ROUTINE

Each class requires 14/13 cycles depending on jumps.
Assuming 50% require jumps
Final pass requires 2 jumps

$$\text{TOTAL} = 14 \times 50 + 13 \times 50 + 1 = 1351 \text{ cycles}$$

THRESHOLD ROUTINE

Entered only once per classification

$$\text{TOTAL} = 3 \text{ cycles}$$

$$\text{TOTAL} = 7 + 4700 + 1351 + 3 = 6061 \text{ cycles}$$

APPENDIX B
BENCHMARKS - CODING

- 1 FFT, Tracor/Litton
- 2 FFT, Raytheon
- 3 Coordinate Conversion A, Tracor/Litton
Polar to Rectangular
- 4 Coordinate Conversion A, Raytheon
Polar to Rectangular
- 5 Coordinate Conversion B, Tracor/Litton
Rectangular to Polar
- 6 Coordinate Conversion B, Raytheon
Rectangular to Polar
- 7 CFAR, bit packed, Tracor/Litton
- 8 CFAR, non bit packed, Raytheon

BI - FFT, Tracor/Litton Coding

1. Benchmark: FFT 1024 Points
2. Algorithm:

$$\begin{aligned} \text{TR} &= \text{CO} * \text{BR} - \text{SI} * \text{BI} \\ \text{TI} &= \text{SI} * \text{BR} + \text{CO} * \text{BI} \end{aligned}$$

$$\begin{aligned} \text{XR} &= \text{AR} + \text{TR} \\ \text{XI} &= \text{AI} + \text{TI} \end{aligned}$$

$$\begin{aligned} \text{YR} &= \text{AR} - \text{TR} \\ \text{YI} &= \text{AI} - \text{TI} \end{aligned}$$

3. Coding:

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
	R1 = M(CO)	
	R5 = R1 * M(BR)	

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
	R2 = M(SI)	
	R6 = R2*M(BI)	
	R6 = R5 - R6	TR
	R5 = R2*M(BR)	
	R7 = R1*M(BI)	
	R7 = R5 + R7	TI
	R3 = M(AR)	
	R4 = M(AI)	
	M(XR) = R3 + R6	
	M(XI) = R4 + R7	
	M(YR) = R3 - R6	
	M(YI) = R4 - R7	

4. Remarks:

- DA operation is transparent to DP operation
- Coding is for one butterfly
- For 1,024 Point FFT there are 10 passes

<u>Pass</u>	<u>Number of CO/SI</u>	<u>Number of Butterflies/CO.SI</u>
1	1	512
2	2	256
3	4	128
4	8	64
.	.	.
.	.	.
10	512	1

- A,B are inputs of butterfly
X,Y are outputs of butterfly

Second pass uses the following:

$$\begin{array}{l}
 \text{For } (CO,SI)_1 \quad A1 = X1 \quad \text{and} \quad B1 = X2 \\
 \quad \quad \quad \quad A2 = X3 \quad \quad \quad B2 = X4 \\
 \quad \quad \quad \quad \text{etc} \\
 (CO,SI)_2 \quad A1 = Y1 \quad \quad \quad B1 = Y2 \\
 \quad \quad \quad \quad A2 = Y3 \quad \quad \quad B2 = Y4 \\
 \quad \quad \quad \quad \text{etc}
 \end{array}$$

Note: DA requires multiple indexing for operand access.

- e. The reordering of final results is not included because Data Addressing can perform this operand access in reversed order with no penalty.

B2 – FFT, RAYTHEON CODING

1. Benchmark: FFT 1024 Points

2. Algorithm:

$$\begin{aligned} XR &= BR + AR \times CR - AI*CI \\ XI &= BI + AR*CI + AI*CR \\ YR &= BR - AR \times CR + AR*CI \\ YI &= BI - AR*CI - AI*CR \end{aligned}$$

3. Coding:

Data In	Scaling	Multiply	Accumulate	Data Out	Comments
A1 AR B2 BR AI BI	CR CI				Repeat 512 × 10 PASS
Flush		M1 = AR*CR M2 = AI*CI M3 = AR*CI M4 = AI*CR			
			S1 = BR + M1 . YR = S1 + M2 S2 = BR - M1 . XR = S2 - M2 S3 = BI + M3 . YI = S3 - M4 S4 = BI - M4 . YR = S3 + M4		
				XR YR XI YI	

4. Remarks:

- a. Coding is for one butterfly.
- b. The reordering of final results is not included.

B3 - COORDINATE CONVERSION A, TRACOR/LITTON CODING

1. Benchmark: Polar to Rectangular

2. Algorithm:

$$X = R \cos \theta$$

$$Y = R \sin \theta$$

3. Coding:

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
	R0 = 0	quadrant indicator
	R1 = M(θ)	
	R2 = $\pi/2 - R1$	$\pi/2 - \theta$
	ALU = R1 - $\pi/2$ IF < 0 JUMP SIN	$\theta - \pi/2$ R1 = $\pi/2 - \theta$
	R0 = R0 + 1	
	R2 = -R1	$\theta - \pi/2$
	R1 = $\pi - M(\theta)$	$\pi - \theta$
	ALU = -R1 IF < 0 JUMP SIN	$\theta - \pi$ R1 = $\theta - \pi/2$
	R0 = R0 + 1	
	R1 = R1 - $\pi/2$	$\theta - \pi$
	R2 = $3 \pi/2 - M(\theta)$	$3 \pi/2 - \theta$
	ALU = -R2 IF < 0 JUMP SIN	$\theta - 3\pi/2$ R1 = $3\pi/2 - \theta$
	R0 = R0 + 1	
	R2 = -R1	$\theta - 3\pi/2$
	R1 = $2\pi - M(\theta)$	
SIN:	RE = 2	Set up 2 passes
MP:	R1 = R1 * R1	θ^2
	R3 = M(K4) + R1	Z0
	MPYB = R1	Litton:
	R3 = MPYB * R1	R3 = R1 * R3
	R3 = R3 + M(K3)	Z1
	MPYB = R1	Litton:
	R3 = MPYB * R3	R3 = R1 * R3
	R3 = R3 + M(K2)	Z2
	MPYB = R1	Litton:
	R3 = MPYB * R3	R3 = R1 * R3
	R3 = R3 + M(K1)	Z3, (sin) cos
	RE = RE - 1 IF ZERO JUMP SIGN	
	R4 = R3	Save sin θ
	R1 = R2 JUMP MP	for cos

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
SIGN:	ALU = R0	IF ZERO JUMP POL
	R3 = -R3	
	R0 = R0 - 1	IF ZERO JUMP POL
	R4 = -R4	
	R0 = R0 - 1	IF ZERO JUMP POL
	R3 = -R3	
		cos = - sin
		sin = - sin
		cos = sin
POL:	MPYB = M(R)	} Litton: R5 = M(R) M(X) = R5*R3 M(Y) = R5*R4
	M(X) = R3*MPYB	
	M(Y) = R4*MPYB	
	END	

4. Remarks:

- $\pi/2$ etc are constants, literal operands
- Sine subroutine uses approximation
- Cosine uses second pass through sine subroutine
- Coding included quadrant determination of θ .

B4 - COORDINATE CONVERSION A, RAYTHEON CODING

- Benchmark: Polar to Rectangular Coordinate Conversion
- Algorithm:

$$X = R \cos \theta$$

$$Y = R \sin \theta$$

3. Coding

Data In	Scaling	Multiply	Accumulate	Data Out	Comments
$\theta 1$	Cos $\theta 1$				
$\theta 2$	Cos $\theta 2$				
R1	Sin $\theta 1$				
R2	Sin $\theta 2$				
		X1 = R1 Cos $\theta 1$ X2 = R2 Cos $\theta 2$ Y1 = R1 Sin $\theta 1$ Y2 = R2 Sin $\theta 2$			
				X1 X2 Y1 Y2	

4. Remarks:

Coding does not include quadrant determination of θ .

B5 - COORDINATE CONVERSION B, TRACOR/LITTON CODING

1. Benchmark: Rectangular to Polar

2. Algorithm:

$$R = \sqrt{X^2 + Y^2}$$

$$\theta = \sin^{-1} \frac{|Y|}{R}$$

$$\theta = \frac{\pi}{2} - \sin^{-1} \frac{|X|}{R}$$

$$0 \leq \frac{Y}{R} \leq \frac{\sqrt{2}}{2}$$

$$\frac{\sqrt{2}}{2} < \frac{|Y|}{R} \leq 1$$

Special Purpose Divide Algorithm:

$$Z = X/Y$$

$$\text{Registers } R1/R2 = R0$$

$$\frac{Z}{2} = X \left(-\frac{Y}{2} - 1 \right) + (-Y - 1)^2 \frac{Z}{2}$$

Assume user provides test that $|X| < |Y|$

- (1) If $X = 0$ then $Z = 0$ (even if $Y = 0$), END
- (2) If $Y = 0$ and $X > 0$ then $Z = \max$, END
- (3) If $Y = 0$ and $X < 0$ then $Z = -\max$, END
- (4) If $Y = -\max$ then $Z = \overline{X} + 1$, END
- (5) If $Y > 0$ then complement Y and X
- (6) If $Y \geq -0.5$ then shift left, "count" until in range
- (7) $K_X = X \left(-1 - \frac{Y}{2} \right)$ note $Y < 0$
- (8) $K_Y = (-1 - Y)^2$
- (9) $Z_0 = K_X$
- (10) $n = 0$ set iteration counter
- (11) $Z_{n+1} = K_X + K_Y Z_n$




- (12) $n = n+1$ If $n \neq 7$ JUMP 11
 (13) $Z = 2 Z_n$
 (14) Z shift right until "count" (6 above) = 0
 (15) END

When using index registers, then

R1 = M (11)
 R2 = M (12)
 M(13) = R0

3. Coding

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
	R0 = M(X) } R0 = R0*R0 } Litton	Calculation of R
	R1 = M(Y) } R1 = R1*R1 } R0 = M(Y)*M(Y)	
	R1 = R1 + R0 CALL SQ	
	M(R) = R0	
	and	
SQ:	M(T) = R2	Save R2
	R0 = 0	$n0 = \sqrt{R1}$
	R2 = R1' IF EQUAL JUMP SQOT	Check for 0
	ALU = R2 IF NEG JUMP POS	Check for NEG
	R1 = R2 + 1	2' COMPL
POS:	R2 = 0	Scaling
	R0 = R1 - 0.5625 IF NEG JUMP SQ1	If < 0, 5652
	R2 = R2 - 1	
	R0 = R0 SHR 2 JUMP SQ4	
SQ1:	R0 = R0 + 0.5 JUMP SQ3	
SQ2:	R1 = R1 SHL 1	SHL 2
	R1 = R1 SHL 1	Scaling
	R2 = R2 + 1	
	R0 = R1 - 0.0625	
SQ3:	ALU = R0 IF NEG JUMP SQ2	If < 0.0625
SQ4:	M(U) = R2	Save Scaling
	R2 = - 13	Iter count
	M(V) = R1	Save Value
	R0 = 0.56	Approx

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
SQ5:	R1 = R0 X R0 R0 = R0 - R1 R0 = M(V) + R0 R2 = R2 + 1 R2 = M(U) ALU = R2 R2 = R2 - 1	Z^2 $Z - Z^2$ + N Scaling
SQ6:	R0 = R0 SHR 1 R2 = R2 - 1	IF NEG JUMP SQ5 IF NEG JUMP SQ7 IF ZERO JUMP SQOT IF ZERO JUMP SQ6 JUMP SQOT
SQ7:	R0 = R0 SHL 1	
SQOT:	R2 = M(T) RETURN	
	M(θ) = 0 R2 = M(R) R0 = 0 R1 = M(Y) R3 = R1' R1 = R3 + 1	Calculation of θ Center IF ZERO JUMP END IF ZERO JUMP QUAD IF NEG JUMP POS
POS:	CALL DIV ALU = R0 - $\sqrt{2}/2$ CALL ARC SIN	R1/R2 = R0 > $\pi/4$ R0 = AVG
QUAD:	R1 = M(X) R2 = M(Y) M(θ) = R0	IF NEG JUMP Q2 IF NEG JUMP Q4 JUMP END ++ 
Q2:	R2 = M(Y) M(θ) = $\pi - R0$	IF NEG JUMP Q3 JUMP ENG - + 
Q3:	M(θ) = $\pi + R0$	JUMP END -- 
Q4: END	M(θ) = $2\pi - \theta$	+ -
C2:	R1 = M(X) R3 = R1' R1 = R3 + 1	IF ZERO JUMP C3 IF NEG JUMP POS
POS:	CALL DIV CALL ARC SIN	
C3:	R0 = $\pi/2 - R0$	JUMP QUAD
DIVIDE:	R0 = R1 R4 = M(I2) ALU = R2	R0 = R1/R2 X = 0 IF NOT ZERO JUMP D2 Y \neq 0

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
	ALU = R1	IF NEG JUMP D1
	R0 = MAX	JUMP END
D1:	R0 = -MAX	JUMP END
		Y = 0
D2:	R0 = R1 + 1	
	R2 = R2 + 1	IF OV JUMP END
		- X Test Y = -MAX. Z = - X
	ALU = R2	IF NEG JUMP D3
	R1 = R1 + 1	
	R2 = R2 + 1	
D3:	M(S) = R3	
	R3 = 1	
		For NEG Y
D4:	ALU = R2 + 0.4999	IF NEG JUMP D5
	R3 = R3 + 1	
	R2 = R2 SHL 1	JUMP D4
		Test Y ≥ -.5 Bring in range
D5:	M(X) = R4	
	M(Y) = R5	
	R4 = R2 SHR 1	
	R4 = R4 + (-MAX)	
	MPYB = R1	Litton: R4 = R4*R1
	R4 = R4*MPYB	
	R5 = R2 + (-MAX)	
	R5 = R5*R5	
	R0 = R4	
	M(N) = R6	
	R6 = 7	
		Counter
D6:	MPYB = R5	Litton: R0 = R0*R5
	R0 = R0*MPYB	
	R0 = R0 + R4	
	R6 = R6 - 1	IF NOT ZERO JUMP D6
	R0 = R0 SHL 1	
		2Z
D7:	R3 = R3 - 1	IF ZERO JUMP D8
	R0 = R0 SHR 1	JUMP D7
D8:	R3 = M(S)	
	R4 = M(K)	
	R5 = M(Y)	
	R6 = M(N)	
END:	M(I3) = R0	
ARC SIN:	R0 = R0*R0	
	R1 = M(K4) + R0	
	MPYB = R0	Litton R1 = R0*R1
	R1 = MPYB*R1	
	R1 = R1 + M(K3)	
	MPYB = R0	R1 = R0*R1
	R1 = MPYB*R3	
	R1 = R1 + M(K2)	
	MPYB = R0	R1 = R0*R1
	R1 = R1 + M(K1)	
		RETURN

4. Remarks:

- a. Arc sine subroutine uses same approximation algorithm as sine but different K values
- b. Divide subroutine user approximation
- c. Coding includes quadrant determination
- d. Angle θ is calculated and not determined by table look-up.

B6 - COORDINATE CONVERSION B, RAYTHEON CODING

1. Benchmark: Rectangular to Polar Coordinate Conversion

2. Algorithm:

$$\theta = f(X, Y) \quad X = R \cos \theta \quad Y = R \sin \theta$$

$$R = \sqrt{X^2 + Y^2} = R \cos^2 \theta + R \sin^2 \theta$$

3. Coding:

Data In	Scaling	Multiply	Accumulate	Data Out	Comments
X1 X2 Y1 Y2	$\theta 1 \cos \theta 1$ $\theta 2 \cos \theta 2$ $\sin \theta 1$ $\sin \theta 2$				
		X1 cos $\theta 1 = A$ Y1 sin $\theta 1 = B$ X2 cos $\theta 2 = C$ Y2 sin $\theta 2 = D$			
			R1 = A + B R2 = C + D		
				$\theta 1$ $\theta 2$ R1 R2	

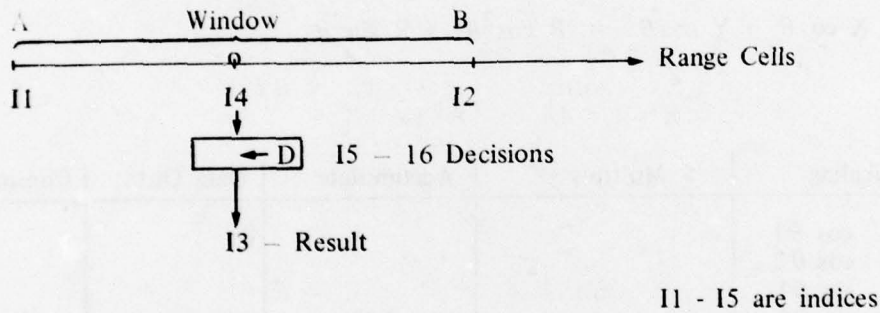
4. Remark:

Angle θ is determined by table look-up via the hardware in the scaling stage.

B7 - CFAR, BIT PACKED, TRACOR/LITTON CODING

1. Benchmark: Sliding window CFAR
2. Algorithm:

- a. $S1 = \sum_0^{256} X_i$
 - b. $R1 = S1 \times K/257$
 - c. $D1 = X_{128} - T1$ pack into 16 bit word
 - d. $S2 = S1 + X_{257} - X_0$
- Repeat b - d



3. Coding:

<u>Labels</u>	<u>Instructions:</u>	<u>Comments</u>
CFO:	R0 = 0	For Sum
CFI:	I1 = 256	
	R0 = M(I0 + I1) I1 = I1 - 1	IF NOT ZERO JUMP CFI
	I1 = 0	
	I2 = 257	
	I3 = 0	
	I4 = 128	
	I5 = 0	
CF2:	I3 = I3 + 1	IF I5 ≠ 0 JUMP CF3
	ALU = 257 - I3	IF ZERO JUMP END
	I5 = 16	4096:16 + 1
	R1 = 0	Temp
	R2 = 1	Bit Mask

<u>Labels</u>	<u>Instructions:</u>	<u>Comment</u>
CF3:	R3 = R0*M(K) R3 = R3 - M(I0 + I4) I4 = I4 + 1 IF NEG JUMP CF4	T = K*Sum T - X
CF4:	R1 = R3 and R2 R2 = SHL 1 I5 = I5 - 1 R0 = R0 + M(I0 + I2) I2 = I2 + 1 R0 = R0 - M(I0 + I1) I1 = I1 + 1 JUMP CF2	Sum + B Sum - A
	END	

4. Remarks

- Threshold $K/257$ is a constant, not a literal.
- Range cells are 6 bit unsigned quantities.
- 256 cells of 6 bits, accumulated requires 14 bit accumulator.

B* - CFAR, NON BIT PACKED, RAYTHEON CODING

1. Benchmark: Sliding window CFAR

2. Algorithm:

$$S1 = \sum_1^{256} X \quad S2 = S1 + X_{257} - X_1$$

$$T1 = K*S1 \quad D1 = X_{127} - T1$$

$$T2 = K*S2 \quad D2 = X_{128} - T2$$

3. Coding

Data In	Scaling	Multiply	Accumulate	Data Out	Comments
X1 X2 X3 X4	-				Repeat 64 times
X1 X257 X2 X258		-			Repeat 2048
NOP			ACC = X1 + X2 + X3 + X4		

Data In	Scaling	Multiply	Accumulate	Data Out	Comments
NOP			$S1 = ACC$ $S2 = S1 - X1 + X257$ $ACC = S2 - X2 + X258$		
NOP				S1 S2	
S1 X127 S2 X128					Repeat 2048
		$T1 = S1 * K$ $T2 = S2 * K$			
			$D1 = X127 - T1$ $D2 = X128 - T2$		
				D1 D2	

4. Remarks

- a. Window contains an even number of cells. Therefore center is off by 1/2 cell.
- b. Constant is 1/256 of threshold.
- c. Coding may require double length arithmetic to accommodate the summation of 256 range cells of 6 bits each.

APPENDIX C

BENCHMARKS – TIMING

- 1 FFT, Tracor/Litton
- 2 FFT, Raytheon
- 3 Coordinate Conversion A, Tracor/Litton
Polar to Rectangular
- 4 Coordinate Conversion A, Raytheon
Polar to Rectangular
- 5 Coordinate Conversion B, Tracor/Litton
Rectangular to Polar
- 6 Coordinate Conversion B, Raytheon
Rectangular to Polar
- 7 CFAR, bit packed, Tracor/Litton
- 8 CFAR, non bit packed, Raytheon
- 9 CFAR, non bit packed, Tracor/Litton

C1 – FFT, TRACOR/LITTON TIMING

1. Benchmark: Fort Fourier Transformation 1024 Points
2. Program

Cycles	Instructions executed
2	R1 = M(CO)
3	R5 = R1*M(BR)
2	R2 = M(SI)
3	R6 = R2*M(BI)
1	R6 = R5-R6
3	R5 = R2*M(BR)
3	R7 = R1+M(BI)
1	R5 = R5+R7
2	R3 = M(AR)
2	R4 = M(AI)
2	M(AR) = R3+R6
2	M(AI) = R4+R7
2	M(BR) = R3-R6
2	M(BI) = R4-R7
30	

3. Cycles calculation:

For 10 passes, 512 iterations per pass

$$\begin{aligned} \text{Total} &= 10 \text{ passes} \times \frac{512 \text{ iterations}}{\text{pass}} \times \frac{30 \text{ cycles}}{\text{iteration}} \\ &= 153,600 \text{ cycles} - \text{Tracor/Litton} \end{aligned}$$

4. Remarks:

- a. Total cycles excludes set-up
- b. Program executed as an In-Place FFT with single accumulator and multiplier.
- c. Complex In-Place FFT will reduce cycles by a factor of 2 to 4.

C2 - FFT, RAYTHEON TIMING

1. Benchmark: Fort Fourier Transformation 1024 Points

2. Program

$$1 \text{ MACRO XR, YR} = f(\text{AR, BR})$$

3. Cycles calculation:

2 cycles/clock, whole pipeline is tied to "external" memory access where other architectures may have "internal" memory access that one factor

$$\text{MACRO TIMING} = 4 \text{ clocks (MACRO Clock)} = 8 \text{ cycles (normalized)}$$

$$\begin{aligned} \text{MACROS} &= 512 \frac{\text{MACROS}}{\text{pass}} \times 10 \text{ passes} + \frac{3 \text{ MACROS}}{\text{push}} \\ &= 5,120 + 3 = 5,123 \text{ MACROS} \end{aligned}$$

$$\text{Total} = 5,123 \text{ MACROS} \times \frac{8 \text{ cycles}}{\text{MACRO}} = 40,984 \text{ cycles} - \text{Raytheon}$$

4. Removals:

Program executed as a Complex In-Place FFT which takes advantage multiple accumulation and multiplications.

C3 - COORDINATE CONVERSION A, TRACOR/LITTON TIMING

1. Benchmark: Polar to Rectangular

2. Program: Single Point R, θ to X, Y

Assume $R > 0$, θ in second quadrant

<u>Cycles</u>		<u>Instructions executed</u>	
1		R0 = 0	
2		R1 = M(θ)	
1		R2 = $\pi/2 - R1$	
1		ALU = $R1 - \pi/2$	ALU > 0 NO JUMP
1		R0 = R0 + 1	
1		R2 = -R1	
2		R1 = $\pi - M(\theta)$	
2		ALU = -R1	ALU < 0 JUMP SIN
<u>11</u>			
1	SIN:	RE = 2	
2	MP:	R1 = R1 * R1	
2		R3 = M(K4) + R1	
1 #		MPYB = R1	
2		R3 = MPYB * R3	
2		R3 = R3 + M(K3)	
1 #		MPYB = R1	
2		R3 = MPYB * R3	
2		R3 = R3 + M(K2)	
1 #		MPYB = R1	
2		R3 = MPYB * R3	
2		R3 = R3 + M(K1)	
1	$\frac{2}{22}$	RE = RE - 1	IF = 0 JUMP SIGN
1		R4 = R3	
2		R1 = R2	JUMP MP
<u>24</u>			
1	SIGN:	ALU = R0	NO JUMP
1		R3 = -R3	
2		R0 = R0 - 1	JUMP POL
2	POL:	MPYB = M(R)	
2		M(K) = R3 * MPYB	
2		M(Y) = R4 * MPYB	
<u>10</u>		END	

3. Cycles calculation:

$$\text{Total} = 11 + 24 + 22 + 10 = 67 \text{ cycles} - \text{Tracor.}$$

Subtract 3×1 cycles for Litton; 64 cycles - Litton

4. Remarks:

- Array processor reduces this time by factor of 2 to 4
- This example shows average case time

C4 - COORDINATE CONVERSION A, RAYTHEON TIMING

1. Benchmark: Polar to Rectangular
2. Program: Single Point R, θ to X, Y
2 conversions per MACRO
3 MACROS to flush
3. Cycles calculation:

$$\text{Total} = \frac{4 \text{ MACROS}}{2 \text{ conversions}} \times \frac{8 \text{ cycles}}{\text{MACRO}} = \frac{16 \text{ cycles}}{\text{conversion}} - \text{Raytheon}$$

4. Remarks:

For multiple conversions, flush of pipe becomes less significant in calculation of total cycles.

C5 - COORDINATE CONVERSION B, TRACOR/LITTON TIMING

1. Benchmark: Rectangular to Polar
2. Program: Single Point X,Y to R, θ
Assume X < 0, Y > 0 - second quadrant

<u>Cycles</u>	<u>Instructions executed</u>
2 } #3	R0 = M(K)
2 } #3	R0 = R0 * R0
2 } #3	R1 = M(Y)
2 } #3	R1 = R1 * R1
2	R1 = R1 + R0 CALL SQ
2	M(R) = R0
<u>12</u>	END
2	SQ: M(T) = R2
1	R0 = 0
1	R2 = R1 NO JUMP
2	ALU = R2 JUMP POS
1	POS: R2 = 0
2	R0 = R1 - 0.5625 JUMP SQ1
2	SQ1: R0 = R0 + 0.5 JUMP SQ3
1	SQ3: ALU = R0 NO JUMP
2	M(U) = R2
1	R2 = -13
2	M(V) = R1
1	R0 = 0.56
<u>18</u>	
2	SQ5: R1 = R0 * R0
1	R0 = R0 - R1

<u>Cycles</u>	<u>Instructions executed</u>
2 2	R0 = M(V) + R0
2 1	R2 = R2 + 1 JUMP SQ5
(12 × 7) + 6	
2	R2 = M(U) NO JUMP
1	ALU = R2 JUMP SQOT
3	SQOT: R2 = M(T) RETURN
6	

Subtotal for R = 12 + 18 + (12 × 7) + 6 + 6
= 12 + 18 + 84 + 6 + 6 = 126 cycles - Tracor
subtract 2 = 124 cycles - Litton

<u>Cycles</u>	<u>Instructions executed</u>
2	M(θ) = 0
2	R2 = M(R) NO JUMP
1	R0 = 0
2	R1 = M(Y) NO JUMP
2	R3 = R1 JUMP POS
2	POS CALL DIV
1	ALU = R0 - $\sqrt{2}/2$ NO JUMP
2	CALL ARL SIN
3	QUAD R1 = M(*) JUMP Q2
2	Q2 R2 = M(Y) NO JUMP
3	M(θ) = π - R0 JUMP END
22	END
1	DIV R0 = R1 NO JUMP
2	ALU = R2 JUMP D2
1	D2 R0 = R1' + 1
1	R2 = R2' + 1 NO JUMP
2	ALU = R2 JUMP D3
2	D3 M(S) = R3
1	R3 = 1
2	D4 ALU = R2 + 0.4999 JUMP D5
2	D5 M(Y) = R4
2	M(Y) = R5
1	R4 = R2 SHR 1
1	R4 = R4 + (-MAX)
1 #	MPYB = R1
2	R4 = R4 * MPYB
1	R5 = R2 + (-MAX)
2	R5 = R5 * R5
1	R0 = R4
2	M(N) = R6
1	R6 = 7
28	D6 MPYB = R5
1 #	R0 = R0 * MPYB
2	

<u>Cycles</u>	<u>Instructions executed</u>
1	R0 = R0 + R4
1	R6 = R6 - 1 JUMP D6
5 × 7 = 35	R0 = R0 SML L
2	R3 = R3 - '1 JUMP D8
2 D8	R3 = M(S)
2	R4 = M(*)
2	R5 = M(Y)
3	R6 = M(N) RETURN
11	
1 ARC SIN:	R0 = R0 * R0
2	R1 = M(K4) + R0
1 #	MPYB = R0
2	R1 = MPYB * R1
2	R1 = R1 + M(K3)
1 #	MPYB = R0
2	R1 = MPYB * R3
2	R1 = R1 + M(K2)
1 #	MPYB = R0
3	R1 = R1 + M(K1) RETURN
17	

Subtotal for $\theta = 22 + 28 + (5 \times 7) + 11 + 17$
 $= 22 + 28 + 35 + 11 + 17 = 113$ cycles - Tracor
 $\# = 22 + 27 + 28 + 11 + 14 = 102$ cycles - Litton

4. Cycles calculation:

Total = subtotal R + Subtotal $\theta = 126 + 113 = 239$ cycles - Tracor
 $\# = 124 + 102 = 226$ cycles - Litton

5. Remarks

- a. Time varies depending on the quadrant of X and Y.
- b. This example shows average case time

C6 - COORDINATE CONVERSION B, RAYTHEON TIMING

1. Benchmark: Rectangular to Polar
2. Program: Single Point X, Y to R, θ
 2 conversions per MACRO
 3 MACROS to flush

3. Cycles calculation

$$\text{Total} = \frac{4 \text{ MACROS}}{2 \text{ conversion}} \times \frac{8 \text{ cycles}}{\text{MACRO}} = \frac{16 \text{ cycles}}{\text{conversion}} - \text{Raytheon}$$

4. Remarks

For multiple conversions, flush of the pipe becomes less significant in calculation of total cycles.

C7 - CFAR, BIT PACKED, TRACOR/LITTON TIMING

1. Benchmark: Sliding window CFAR

2. Program: Assume 4,096 range cells
256 cell window

<u>Cycles</u>		<u>Instructions executed</u>		
1		R0	=	0
1	CF1:	+1	=	256
3		R0	=	M (I0 + I1) I1 = I1-1 IF NOT ZERO JUMP CF1
<u>4</u>				
1		I1	=	0
1		I2	=	257
1		I3	=	0
1		I4	=	128
1		I5	=	0
<u>5</u>				
1 2 2	CF2:	I5	≠	0 NO JUMP/CF3
1 1		I3	=	I3 + 1
1 <u>2</u>		ALU	=	257 - I3 NO JUMP/END
1 <u>5</u>		I5	=	16
1		R1	=	0
1		R2	=	1
3 3	CF3:	R3	=	1 R0 * M(K)
3 3		R3	=	R3 - M (I0 + I4), I4 = I4 + 1, JUMP CF4
1 1	CF4:	R2	=	SML 1 I5 = I5 - 1
2 2		R0	=	R0 + M
3 3		R0	=	R0 - M JUMP CF2
<u>18</u> 14				
× 15				
= 210				
<u>228</u> × 256				
58,368				

3. Cycles calculation:

$$\begin{array}{r}
 \text{Total} = \\
 4 \times 256 = \quad 1 \\
 \quad \quad \quad 1,024 \\
 \quad \quad \quad 5 \\
 \quad \quad \quad \underline{58,368} \\
 \quad \quad \quad 59,403 \text{ cycles - Tracor/Litton}
 \end{array}$$

4. Remarks:

- a. Algorithm includes packing of every 16 threshold decisions into one word
- b. Processing is in real-time, single pass.

C8 - CFAR, NON BIT PACKED, RAYTHEON TIMING

1. Benchmark: Sliding window CFAR

Program	Assure	4,096 range cells
		256 cell window
MACRO		64 times
MACRO		2,048 times
FLUSH		3
MACRO		2,048 times
FLUSH		4
		<u>4,166</u>

3. Cycles calculation

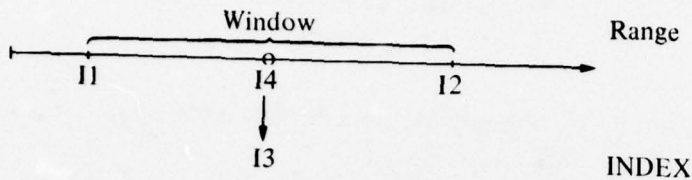
$$\text{Total} = 4,166 \text{ MACROS} \times \frac{8 \text{ cycles}}{\text{MACRO}} = 33,328 \text{ cycles - Raytheon}$$

4. Remarks:

- a. Each threshold decision is in a separate word.
- b. Two decision per MACRO.
- c. Processing is not in real-time. A complete sweep of all range cells is required before processing begins.
- d. Interim result requires 4096 words of temporary storage.
- e. Accumulation of 256 cells (window), each of a 6-bit unsigned quantity, into a 12 bit accumulator may be a problem.
- f. Bit packed CFAR would require more MACROS and would be significantly slower.

C9 - CFAR, NON BIT PACKED, TRACOR/LITTON TIMING

1. Benchmark: Sliding Window CFAR
2. Program: Assure 4,096 range cells
 256 cell window



<u>Cycles</u>	<u>Instructions executed</u>
1	CF0: R0 = 0
1	CF1: I1 = 255
3	R0 = M(I0 * I1) I1 = I1 - 1 IF ≠ JUMP CF1
<u>4</u> × 256	
1	I1 = 0
1	I2 = 255
1	I3 = 0
1	I4 = 127
2	RF = M(K)
<u>6</u> 1 2	CF2: ALU = 257 - I3 IFO JUMP END
3	M(I3), R3 = R0 * RF I3 = I3 + 1
2	R0 = R0 + M(I0 + I2) I2 = I2 + 1
3	R0 = R0 - M(I0 + I1) I1 = I1 + 1 JUMP CF2
<u>9</u> × 4096	END

3. Cycles calculation:

$$\begin{array}{r}
 \text{Total} \\
 4 \times 256 = 1,024 \\
 9 \times 4,096 = 36,864 \\
 \hline
 37,888 \\
 + 9 \\
 \hline
 37,897 \text{ cycles - Tracor/Litton}
 \end{array}$$

4. Remarks

- a. Algorithm uses one word for each threshold decision.
- b. Processing is in real-time, single pass.