AD-A032 180 AIR FORCE AVIONICS LAB WRIGHT-PATTERSON AFB OHIO F/G 9/2
THE UNIFIED PROCESSOR, A MICROPROGRAMMABLE PROCESSOR.(U)
SEP 76 R E BARRERA
UNCLASSIFIED AFAL-TR-75-148 NL

1 OF 1
AD A032180

END

DATE FILMED
1-77

AD A032180

AFAL-TR-75-148

Fb (12)

THE UNIFIED PROCESSOR, A MICROPROGRAMMABLE PROCESSOR

INFORMATION MANAGEMENT BRANCH
SYSTEM AVIONICS DIVISION

SEPTEMBER 1976

DDC
RECEIVED
NOV 18 1976
B

TECHNICAL REPORT AFAL-TR-75-148
FINAL REPORT FOR PERIOD AUGUST 1972 THROUGH MARCH 1975

Approved for public release; distribution unlimited

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

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 technical report has been reviewed and is approved for publication.

Ralph E Barrera

RALPH E. BARRERA
Project Engineer

FOR THE COMMANDER

James M. Riley

JAMES M. RILEY, MAJ, USAF
Chief, System Technology Branch
System Avionics Division

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

AIR FORCE - 20 OCTOBER 76 - 100

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: AFAL-TR-75-148	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): The Unified Processor, A Microprogrammable Processor	5. TYPE OF REPORT & PERIOD COVERED: Final Technical Report. Aug 72 ~~thru~~ Mar 75.
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s): Ralph E. Barrera	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Air Force Avionics Laboratory, Systems Avionics Division (AA), Wright Patterson Air Force Base, Ohio	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62204F-2003-04-08
11. CONTROLLING OFFICE NAME AND ADDRESS: Air Force Avionics Laboratory (AFSC), System Avionics Division (AA), Wright Patterson AFB, Ohio	12. REPORT DATE: September 1976 13. NUMBER OF PAGES: 79
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved For Public Release; Distribution Unlimited.

2003 04

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)

Microprogramming; Processor; Avionics; Computers; Microprocessors; Architecture

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report presents the design of a microprogrammable processor called the Unified Processor (UP). First a brief history of microprogramming is given followed by a description of a present day processor. The architecture of the Interpreter is presented along with several deficiencies that were present in its design. A brief discussion of the microprogramming format of the UP is presented followed by an in-depth description of the UP. The features of the UP that are covered include general registers, special registers, internal bussing,

CONTINUED

over

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

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

011670

microprogram addressing, and the Barrel Switch. Examples of how each of the various features can be used is included in each section. Finally, the UP is evaluated and recommended changes for a future design presented.

FOREWORD

This Final Engineering Report was prepared by the Air Force Avionics Laboratory (AFSC), System Avionics Division, Information Management Branch, Wright-Patterson AFB, Ohio. The work was accomplished under USAF Project 2003 entitled "Avionic System Design Technology," Task 04 entitled "Modular Processors." The work was administered under the direction of Mr. R. Barrera, Air Force Avionics Laboratory, AFAL/AAM, Wright-Patterson AFB, Ohio.

This report covers work conducted from August 1972 through March 1975 and was submitted by the author March 1975.

The author, Mr. R. Barrera, is grateful for the help and contributions received from Messrs. John Camp and Fred Schurr, associates at the Avionics Laboratory in fabricating and debugging the Unified Processor. In addition he wishes to thank Mr. Dewey E. Brewer also of the Avionics Laboratory for his overall guidance and encouragement during the initial study and design and Mr. Robert Davis of the Burroughs Corp. who spent many hours explaining the hardware and programming of the Interpreter.

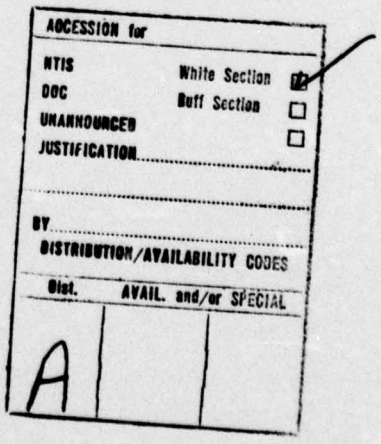


TABLE OF CONTENTS

SECTION			PAGE
I	INTRODUCTION		1
	1	Background	1
	2	Statement of the Problem	4
	3	Approach	4
II	THE INTERPRETER		6
	1	Architecture	6
		a. Logic Unit	6
		b. Control Unit	10
		c. Memory Control Unit	12
		d. Control Store	14
	2.	Interpreter Deficiencies	16
III	MICROPROGRAMMING		19
	1.	Language	19
	2.	Example	21
IV	UNIFIED PROCESSOR		25
	1.	General Registers	25
	2.	Barrel Switch	28
		a. Barrel Switch Operation	35
		b. Barrel Switch Control	39
	3.	Microprogram Addressing	41
	4.	Memory/Device Addressing	50
	5.	Micromemory Control Partitions	54
	6.	Internal Bussing	57
	7.	Programmable Instruction Decoder	62
	8.	Input/Output Registers	65
	9.	Special Registers	67
V	CONCLUSION		69
	1.	Summary	69
	2.	Recommendations	69
		a. Micromemory	69
		b. B Register	70
		c. Barrel Switch	70

PRECEDING PAGE BLANK-NOT FILMED

TABLE OF CONTENTS (Cont'd)

SECTION	PAGE
APPENDIX A - Unified Processor I/O Channel	71
APPENDIX B - Unified Processor Micromemory Implementation	75
BIBLIOGRAPHY	79

LIST OF ILLUSTRATIONS

FIGURE		PAGE
1	Basic Computer Structure	2
2	Interpreter Block Diagram	7
3	Logic Unit Block Diagram	8
4	Control Unit Block Diagram	11
5	Memory Control Unit Block Diagram	13
6	Micromemory Word Formats	15
7	Nanomemory Word Format	15
8	Unified Processor Control Fields	20
9	Unified Processor Block Diagram	26
10	Multiple Shifter	31
11	Single Level 8-Bit Barrel Switch	32
12	Single Level 8-Bit Barrel Switch Control	33
13	Two Level 8-Bit Barrel Switch	34
14	Shifting Examples	37
15	Shift Amount Register Block Diagram	40
16	Timing of CSAR Instruction	40
17	SAVE Successor Timing	45
18	JUMP Successor Timing	47
19	EXEC Successor Timing	49
20	CALL Successor Timing	51
21	MPM Addressing System Block Diagram	52
22	Type I Microinstruction Control Fields	55
23	Instruction Timing Diagram	57
24	X-Bus Multiplex Structure	58
25	Y-Bus Tri-State Structure	61

LIST OF ILLUSTRATIONS (Cont'd)

FIGURE		PAGE
26	Normal Fetch-Decode-Execute Cycle	63
27	Programmable Instruction Decoder Operation	64
28	I/O Channel Block Diagram	72
29	Input Data Transfers Timing Diagram	73
30	Output Data Transfers Timing Diagram	74
31	UP Micromemory	76

SUMMARY

This report presents the design of a microprogrammable processor called the Unified Processor (UP). First a brief history of microprogramming is given followed by a description of a present-day processor. The architecture of the Interpreter is presented along with several deficiencies that were present in its design. A brief discussion of the microprogramming format of the UP is presented followed by an in-depth description of the UP. The features of the UP that are covered include general registers, special registers, internal bussing, microprogram addressing, and the Barrel Switch. Examples of how each of the various features can be used is included in each section. Finally, the UP is evaluated and recommended changes for a future design presented.

SECTION I

INTRODUCTION

1. BACKGROUND

The structure of the computer has remained constant since 1830 when Charles Babbage described his Analytical Engine. This structure shown in Figure 1 is comprised of four basic units: the Arithmetic Unit that performs all the necessary calculations, the Memory that holds both the instructions and data, the Input/Output Unit that ties the computer to the world, and the Control Unit that produces the signals to synchronize the operations of the other units. Although Babbage had defined the necessary components of a computer he was never able to construct one because of the lack of suitable technology.

By the 1940's technology had advanced sufficiently that a working computer could be built. The first, made with relays, was the MARK I. It was followed closely by the ENIAC, which was the first computer to use vacuum tubes, and then UNIVAC I, the first commercially produced computer.

Since the introduction of UNIVAC I in 1951 there have been few changes to the basic units of the computer. As technology advanced the units were made to operate faster but their circuits remained unchanaged. There have been several suggested alternatives to the system presently being used but most of these have proved of little worth because of either limited applications or insufficient technology. One alternative

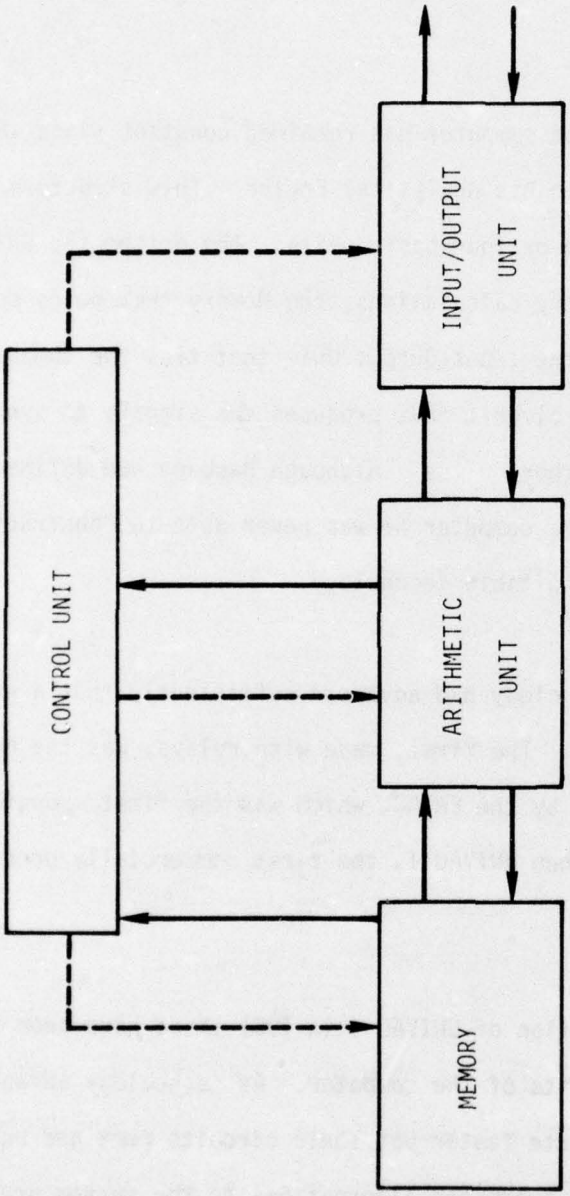


Figure 1. Basic Computer Structure

that had been considered an academic curiosity, for the latter reason above, was the use of microprogramming.

The Arithmetic Unit, Input/Output Unit, and the Memory have almost since their inception been very systematically designed. The flow of data within these units is easy to follow and the design or modification of an existing unit is relatively easy to accomplish. The same cannot be said of the Control Unit because its design tends to be of random logic groupings. It was this ad hoc, nonsystematic approach to designing the Control Unit that concerned M. V. Wilkes and caused him in 1951 to put forth the scheme of microprogramming. Wilkes observed that every machine level instruction was really a combination of several less complex commands or machine states. For example an add instruction was composed of the following sequence of machine states.

MACHINE LEVEL	MICRO LEVEL
ADD R1, R2	- ADDRESS R1 & R2 AND GATE ONTO ALU BUSSES
	- SET ALU FUNCTION TO ADD
	- ADDRESS DESTINATION REG R2
	- CLOCK RESULTS INTO REG R2

The idea of microprogramming was that instead of allowing a designer to create the series of states using gates and flip-flops the necessary control signals would be stored in a memory. Wilkes' original concept was to have each bit of a word from micromemory control a unique gate. This has been modified in most implementations by allowing some decoding between the micromemory and the logic to be controlled. This has not

produced many limitations because there are several operations that tend to be mutually exclusive such as add and subtract.

Microprogramming was not widely accepted commercially until IBM introduced the Model 360 computer in 1963. The cost of memory had been too high to make implementing the Control Unit with Memory feasible. Since the late 1960's the cost of high-speed memory has dropped to about one or two cents per bit, making microprogramming very cost-competitive with the previous method. Presently most computers being introduced use microprogramming in the Control Unit with a small amount of decoding. This has produced computers that are easier to understand and reasonably simple to modify.

2. STATEMENT OF THE PROBLEM

With microprogrammable processors becoming so economically feasible much interest has been generated concerning their potential applications. It has been proposed that these processors could be used to emulate another processor, thereby taking advantage of software that had already been written. Another possible area is the matching of processor to application by tuning the microprogram. This report presents the results of an effort to design a microprogrammable processor that would provide the necessary facilities for studying the potential applications of microprogramming.

3. APPROACH

The approach used in designing the microprogrammable processor was to first examine carefully an existing microprocessor. This study was

performed by writing several microprograms for a Burroughs designed microprocessor called an Interpreter. The programs were analyzed and a list of deficiencies was compiled.

The next step was to design a processor that incorporated the desired features. The result of this effort is the Unified Processor which represents an enhancement of the Intepreter.

The Interpreter along with its deficiencies is described in Section II of this report. Section III contains a brief explanation of the microprogramming language used on the Unified Processor (UP) and Section IV provides a complete explanation of the UP hardware. Finally Section V discusses some problem areas that have been identified with the present UP design and possible solutions to them.

SECTION II

THE INTERPRETER

The Interpreter is a microprogrammable processor that was built by the Burroughs Corp. The block diagram in Figure 2 shows information flow between major registers of the Interpreter. The implementation of the Interpreter necessitated the partitioning of these components into specific function modules.

The four major modules that comprise an Interpreter are the Logic Unit (LU), the Control Unit (CU), the Memory Control Unit (MCU), and the control store which is comprised of the Micro Memory (MM) and the Nano Memory (NM). A description of how each of these units function is given in the following section.

1. ARCHITECTURE

a. Logic Unit

The LU performs the required shifting, arithmetic, and logic functions as well as providing a set of scratch pad registers and data interfaces to and from external devices and memories. A functional block diagram of the LU is shown in Figure 3.

Registers A1, A2, and A3 are functionally identical. Each temporarily stores data and serves as a primary input to the adder. Selection gates permit the contents of any A register to be used as one of the inputs to the adder. Any combination of A registers can be loaded simultaneously from the B-Bus.

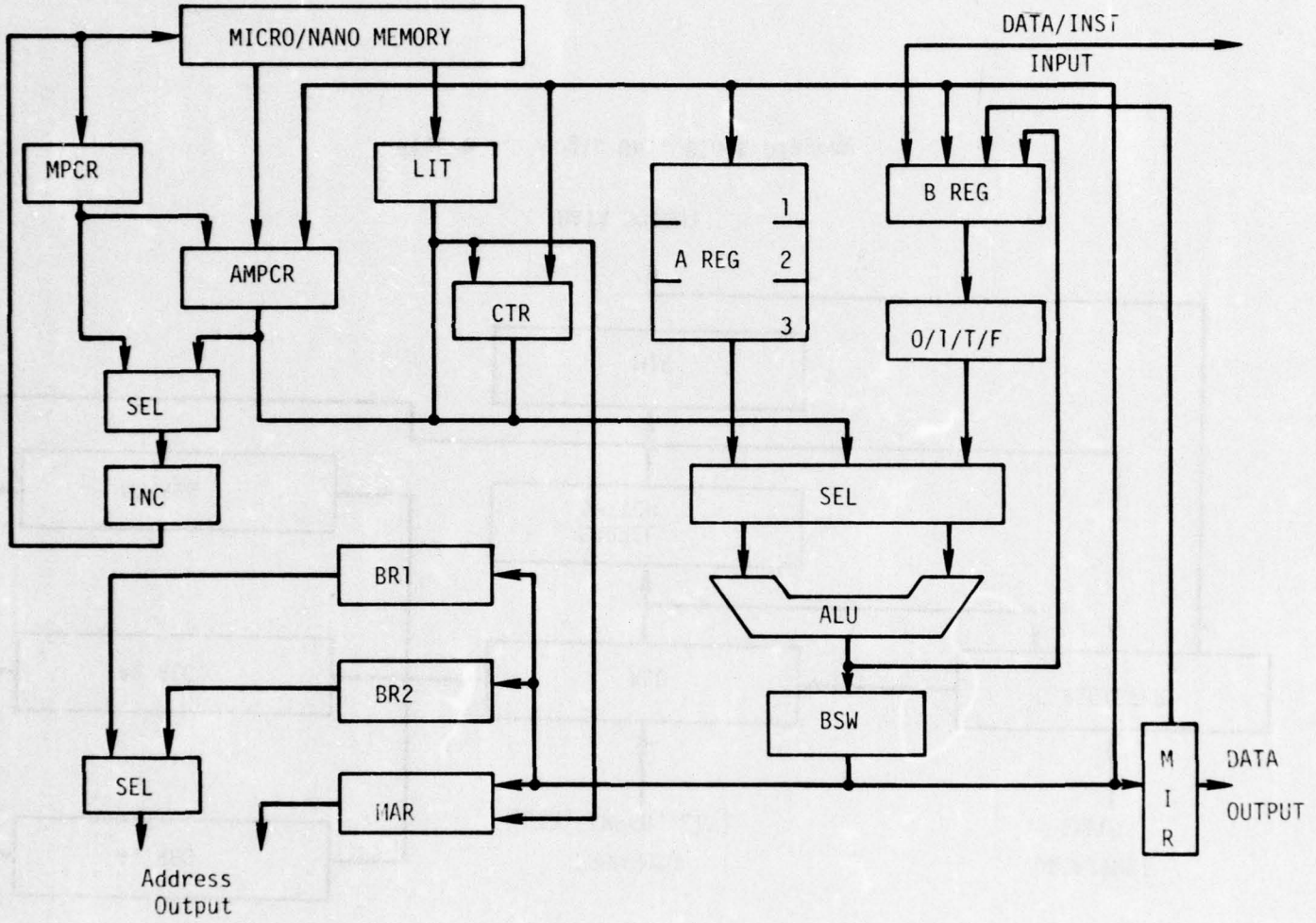


Figure 2. Interpreter Block Diagram

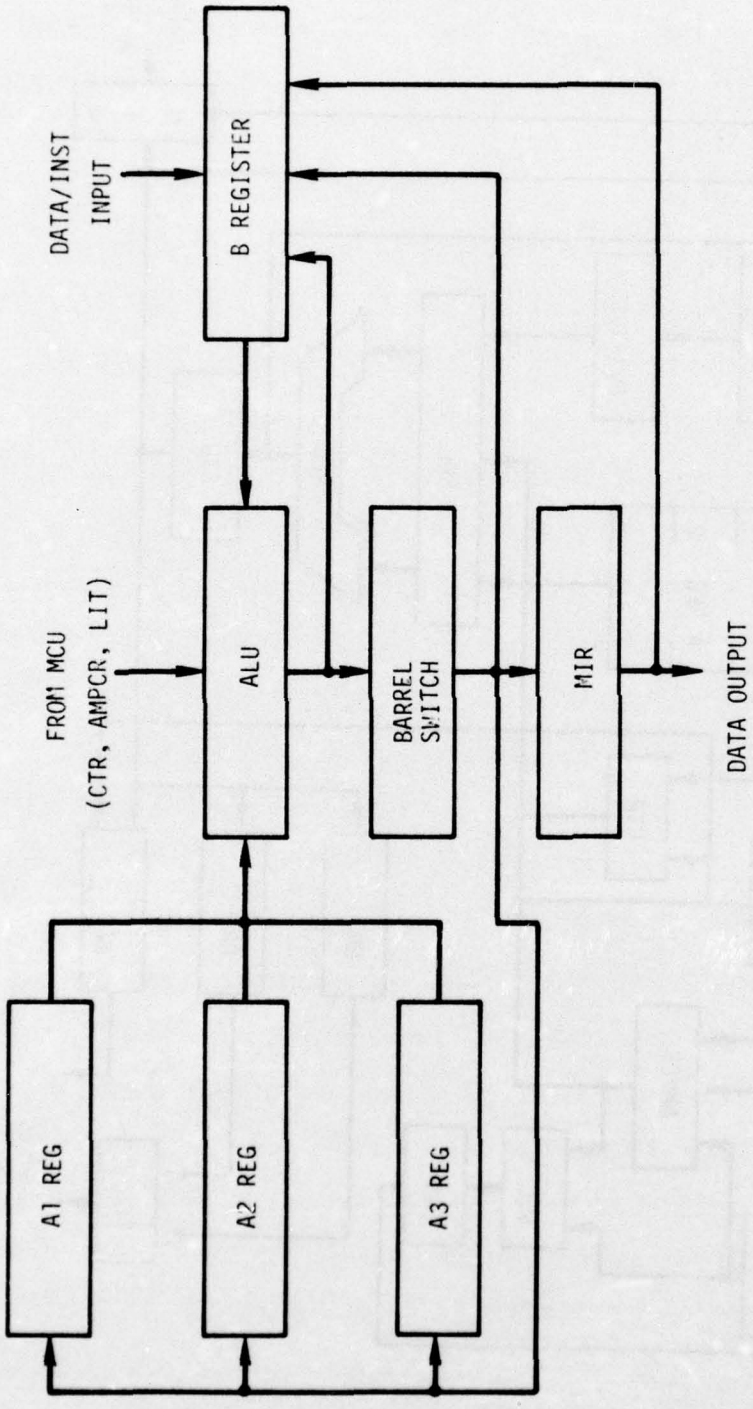


Figure 3. Logic Unit Block Diagram

The B register is the input buffer. It serves as the second input to the adder and can also collect certain side effects of arithmetic operations. The B register may be loaded from a number of locations.

The output of the B register has true/complement selection gates which are controlled in three separate sections: the most significant bit, the least significant bit, and all the remaining central bits. Each of these parts is controlled independently and may be either all zeros, all ones, the true contents or the complement (ones complement) of the contents of the respective bits of the B register. The operation of these selection gates affects only the output of the B register. The contents remain unchanged.

The Memory Information Register (MIR) primarily buffers information being written to main system memory or to a peripheral device. It is loaded from the B-Bus and its output may be sent to the Input/Output Unit, to the B register, or to the data input of the MPM or Nanomemory for programmatic loading.

The adder in the LU is a modified version of a straightforward carry look-ahead adder. Inputs to the adder are from selection gates which allow various combinations of the A, B, and Z inputs. The A input is from the A register output selection gates and the B input from the B register true/complement selection gates. The Z input is an external input to the LU and can be CTR, LIT, or AMPCR registers from the Memory Control Unit.

Using various combinations of inputs to the selection gates, any two of the three inputs can be added together, or can be added together with an additional "one" added to the least significant bit. Also, all binary Boolean operations between the A and B and between the B and Z adder inputs and most of the binary Boolean operations between the A and Z adder inputs can be done.

The Barrel Switch is a matrix of gates that shifts the parallel output of the ALU any number of places to the left or right, either end-off or end-around, in one clock time. Data passing through the Barrel Switch is put on the B-Bus from where it may be loaded into any selected register.

b. Control Unit (CU)

The CU contains a condition register and the logic necessary for testing the selected condition, a register for controlling shift operations in the LU, and part of the control register used for storage of some of the control signals to be sent to the LU.

Major sections of this unit shown in Figure 4 are: The Shift Amount Register (SAR), the condition register, part of the control register, the MPM content decoding, and part of the clock control. The functions of the SAR and its associated logic are:

(1) To load shift amounts into the SAR to be used in the shifting operations.

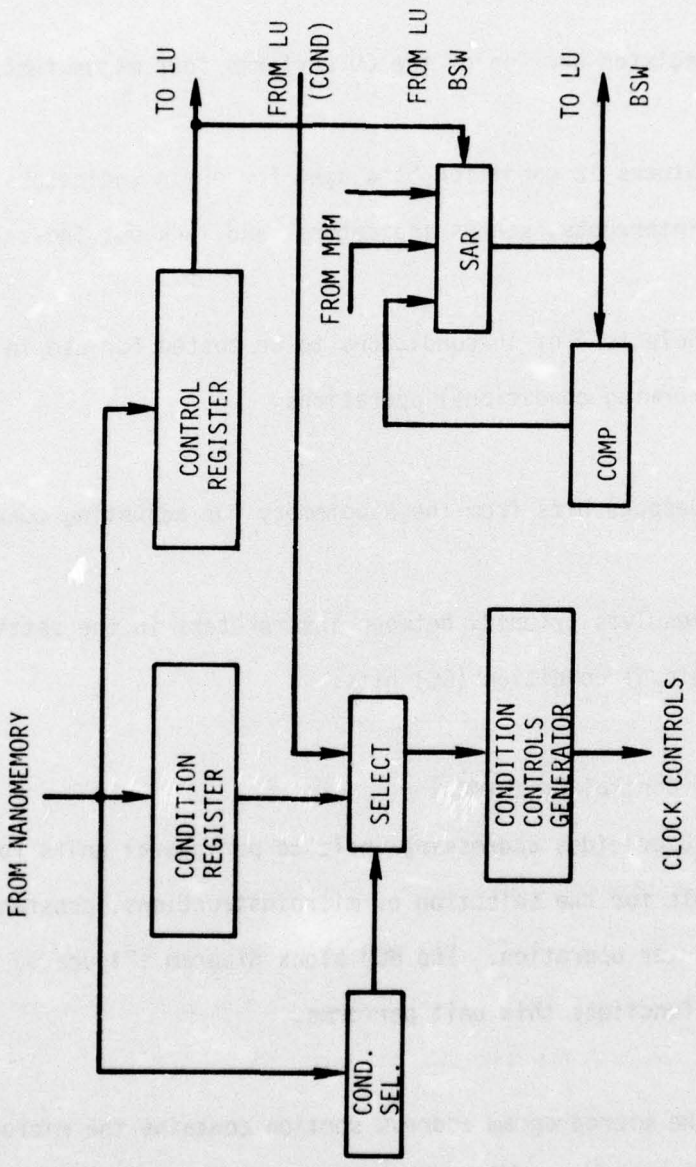


Figure 4. Control Unit Block Diagram

(2) To generate the required controls for the Barrel Switch shift operation indicated by the controls from the Nanomemory.

The condition register section of the CU performs four major functions:

(1) Stores 12 condition bits used for error indicators, interrupts, status indicators, and lock out indicators.

(2) Selects 1 of 16 conditions to be tested for use in performing conditional operations.

(3) Decodes bits from the Nanomemory for adjusting conditions.

(4) Resolves priority between interpreters in the setting of global condition (GC) bits.

c. Memory Control Unit (MCU)

The MCU provides addressing logic to peripheral units for data accesses, controls for the selection of microinstructions, constants storage, and counter operation. The MCU block diagram (Figure 5) shows the three major functions this unit performs.

(1) The microprogram address section contains the microprogram count register (MPCR), the alternate microprogram count register (AMPCR), the incrementer, the microprogram address control register, and associated control logic.

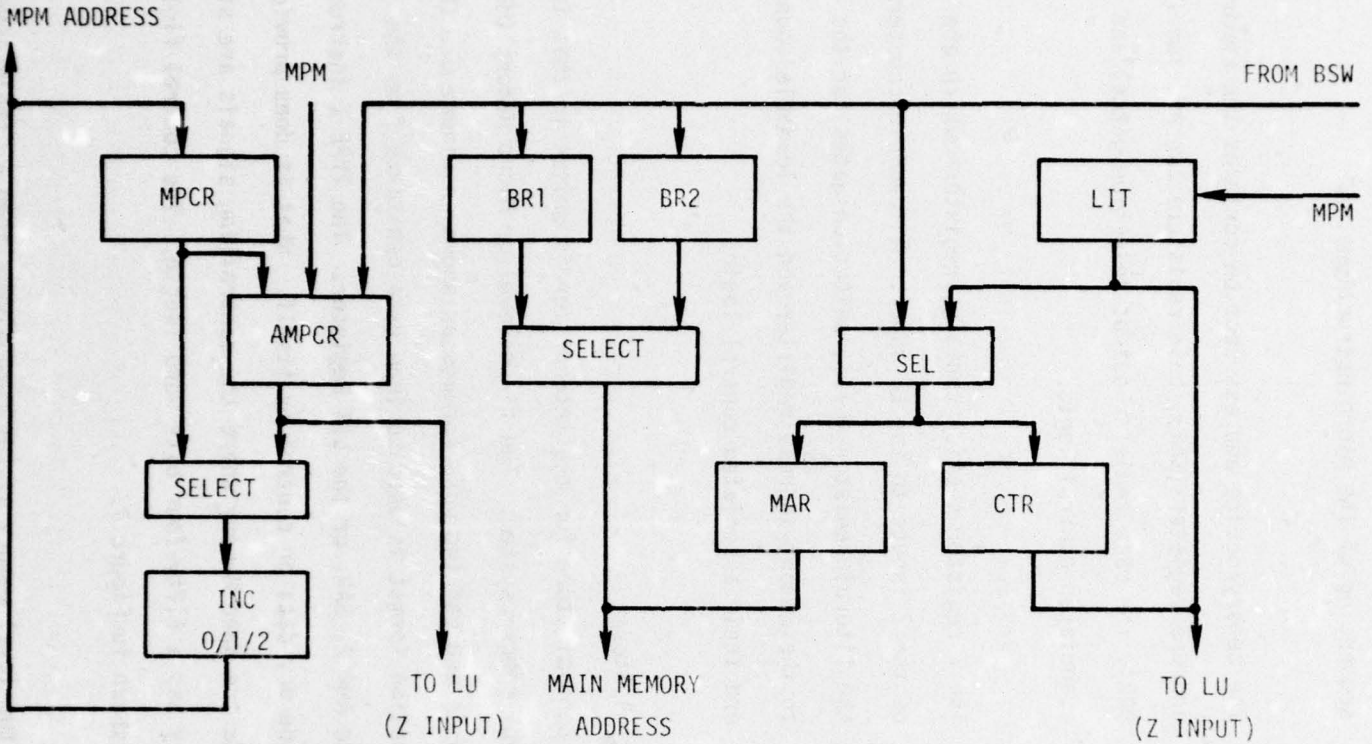


Figure 5. Memory Control Unit Block Diagram

The output of the incrementer addresses the MPM for the sequencing of the microinstructions.

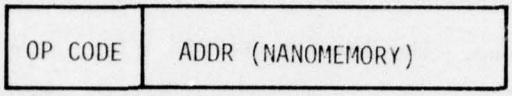
(2) The memory/device address section contains the memory address register (MAR), base registers one and two (BR1, BR2), the base register output selection gates, and the associated control logic.

(3) The Z register section contains registers which are two of the Z inputs to the LU adder: A loadable counter (CTR), the literal register (LIT), selection gates for the input to the memory address register and the loadable counter and their associated control logic.

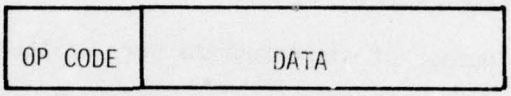
d. Control Store

The control store for the Interpreter is unique in that it uses a two-level memory system. The first level or Micro Memory (MM) is sixteen bits wide and has two word formats as shown in Figure 6. The TYPE II instruction format is used to load data constants from the MM into either the AMPCR, SAR, or the LIT Register. The TYPE I instruction does not provide any data or control by itself. What it does provide is an address into the Nano Memory where the controlling signals are stored. The Nano Memory has a fifty-four bits word length. The control fields of the NM are shown in Figure 7.

The two-level memory system runs slower than a single level system would because two memory accesses must be made for every TYPE I



TYPE I INSTRUCTION



TYPE II INSTRUCTION

Figure 6. Micromemory Word Formats

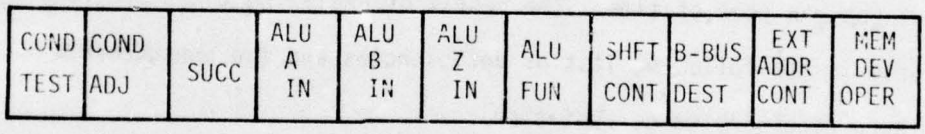


Figure 7. Nanomemory Word Formats

instruction. The first provides the Micro instruction and the second the Nano control word. The benefit derived from the two memories is that a unique fifty-four bit control word need be stored only once.

2. INTERPRETER DEFICIENCIES

A substantial number of microprograms were written on the Interpreter in order to arrive at a complete list of deficiencies in its architecture. The programs include an emulation of a 24-bit general purpose navigation computer, an emulation of a special purpose flight control computer, several I/O drivers, and some special purpose routines such as multiply, floating point operation, etc. The largest microprogramming effort that has been performed was the partial emulation of a Burrougns B6500. This required 2000 microinstruction and 1000 nanoinstruction and about one man-year of time. The result of analyzing these programming efforts is the following list of deficiencies and the enhancements believed necessary to correct them.

a. *General Purpose Register.* Three registers were supplied on the Interpreters and only one could be used as an input to the ALU at any one time. There were several occasions when temporary data had to be stored in core memory because the three registers were being used. The number of general registers was increased to eight, thereby allowing more processing within the registers and less of the time consuming memory operations. Another improvement was to allow two of the eight registers to be selected at one time. This allowed register to register operations to be performed without the extra instruction that had been necessary to move data to the B register prior to ALU operations.

b. Standard Arithmetic Logic Unit. The ALU function set provided on the Interpreter did not contain some very essential operations. These included ones complement and twos complement subtraction as well as decrement. To correct this a 74181 ALU integrated circuit (IC) was used. This IC provides all the arithmetic operations (addition, subtraction, increment, decrement) and logical operations (AND, OR, NAND, NOR, Complement, Exclusive OR, and Equivalence) that are required.

c. The addressing structure of the Interpreter did not provide a convenient way of implementing a program counter. There was no way of reading the present address or incrementing it, therefore the normal practice was to use a general register. To correct this problem a ninth general register that could be incremented was added. This register can be selected for addressing in the same manner that the Base Register and Memory Address Registers are selected.

d. A problem involved in writing complex programs was how to conveniently nest loops or save return addresses when branching to subroutines. The approach used in the B6500 emulation was to save them in core memory. This required not only more instruction but considerably more time. A hardware stack was added to the microprogram addressing system to save return addresses.

e. Emulated Instruction Decode. It was found in the emulations performed that the greatest decrease in execution time could be realized if the instruction decode routine was made extremely fast. By relying

strictly on microprogramming it was impossible to provide good performance over a variety of instruction formats. To obtain the desired speed and flexibility a hardware instruction decoder was used. It is capable of decoding an instruction and providing the proper microprogram address within one clock (100 nanoseconds).

f. Internal ALU Bussing. The bussing provided on the Interpreter provided little flexibility in how data was supplied to the ALU. To correct this a bus was provided on each of the ALU ports and all registers were tied to both. This allowed operations such as A1 - A2, CTR - A6, A3 - LIT, etc. to be performed.

SECTION III

MICROPROGRAMMING

1. LANGUAGE

Before entering into a discussion of the hardware of the Unified Processor it is necessary to describe briefly the language used to program the UP. No attempt will be made to present all the mnemonics of the language but instead a brief description of the different control fields will be given. The control fields of the UP are shown in Figure 8 and in general one entry from each field may be used to form a complete microinstruction.

a. Condition Test. Selects one of 32 conditions for testing and controls whether the ALU and/or the external operations are to be condition or unconditional.

b. Condition Adjust. Sets and resets one bit flag registers. These flags are used mainly by the programmer to store the results of tests.

c. Successor. Two successors are selected on every instruction. The first one is used if the tested condition was true and the second if the condition was false. The successors include the following: STEP, SKIP, JUMP, WAIT.

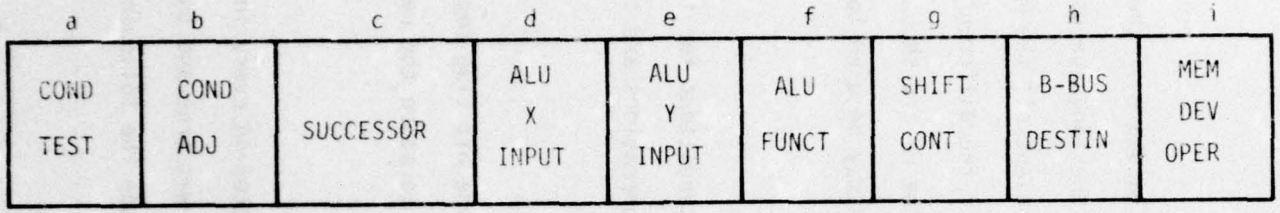


Figure 8. Unified Processor Control Fields

d. ALU X Input. Selects which register will be connected to the X-Bus. The registers include the A, B, PCR, CTR, LIT, and AMPCR. This field also has the controls for the 0/1 T/F network.

e. ALU Y Input. Controls the access to the Y-Bus by the A registers, PCR, LIT, CTR, and AMPCR.

f. ALU Function. Determines which of sixteen arithmetic/logic functions will be performed.

g. Shift Control. Controls the shift direction in the Barrel Switch. It provides for right or left end-off or right end-around shifts.

h. B-Bus Destination. This field controls what registers will be loaded with the output of the Barrel Switch. One A register may be selected and in addition the B register, MIR, AMPCR, MAR, BR, and CTR may be selected in any combination.

i. Memory/Device Operation. Determines what I/O operation will be performed and at the same time selects the source of the address.

2. MICROPROGRAM EXAMPLE

Below is a program that multiplies two thirty-two bit words together and arrives at a sixty-four bit product. The following assumptions are made at the start of this program.

a. Numbers are in sign magnitude form where the sign bit is the most significant bit.

b. Positive numbers have a zero sign bit and negative numbers a one.

c. The multiplier is a register A3 and the multiplicand is in register B.

d. The least significant part of the product will be in A3 and the most significant part along with the sign in B.

```
1.  A3 XOR B; IF LC1
        % Reset LC1.

2.  BOTT = A2; IF MST THEN SET LC1
        % Check sign bits of A3 and B if they are
        % different then set LC1.  Put magnitude of B
        % into A2.

3.  BOOO = B; LCTR
        % Clear B register, load LITERAL into COUNTER.

4.  31 = LIT; 1 = SAR
        % 31 is used as a loop counter below.  This
        % instruction and the previous one together load 31 into
        CTR.
        % 1 is loaded into SAR to allow for 1 bit shifts.
```

```
5.   A3 R = A3; SAVE
     % Shifts least bit of multiplier off.

5.   IF NOT LST THEN BOTT C=B SKIP  ELSE STEP
     % Tests least bit of A3 before it is shifted off.  If
     % it was a "0" then shift B circular into B and
     % skip next instruction.

7.   A2 + BOTT C=B
     % If least bit was a "1" then add multiplicand
     % to B and shift circular into B.
     % NOTE both instructions 6 and 7 put the new
     % least bit of the product into the most significant bit.
     % Also the old least bit that was transferred to
     % A3 is removed (BOTT).

8.   A3 OR BTOO/R = A3, INC; IF NOT COV THEN JUMP ELSE STEP
     % Combine the new least bit of B with A3 and
     % shift right.  Increment counter and test for overflow.
     % If overflow then multiply complete if not JUMP to
     % instruction 6.

9.   B R=B
     % Shift off the last least bit of the product.

10.  IF LST THEN A3 + B100 = A3
     % If least was true then set the most bit of A3.
```

```
11.   IF NOT LC1 THEN BOTT = B SKIP ELSE STEP
      % If LC1 was not set then signs were the same and
      % product sign will be positive.

12.   BITT = B
      % If LC1 was set then product is negative.

13.   END
```

SECTION IV

UNIFIED PROCESSOR

The Unified Processor represents an enhanced version of the Interpeter by the elimination of the deficiencies described in Section II.2. Figure 9 is a register level block diagram of the UP showing the data transfer paths. The following sections describe the specific features of the UP and the reasons they were selected.

1. GENERAL REGISTERS

Many processors, especially for avionics application, have been built using only one general purpose register called the accumulator. All arithmetic/logical operations that were to be performed had to be between a memory location and the accumulator. This put a severe limitation on the programmer because all intermediate results had to be stored in memory. For most machines the result would be a minimum of four extra memory cycles.

	Operation	Mem Cyc
	FETCH "STO" INST	1
STO TEMP		
	WRITE TEMP DATA	1
	FETCH "LDA" INST	1
LDA TEMP		
	READ TEMP DATA	1

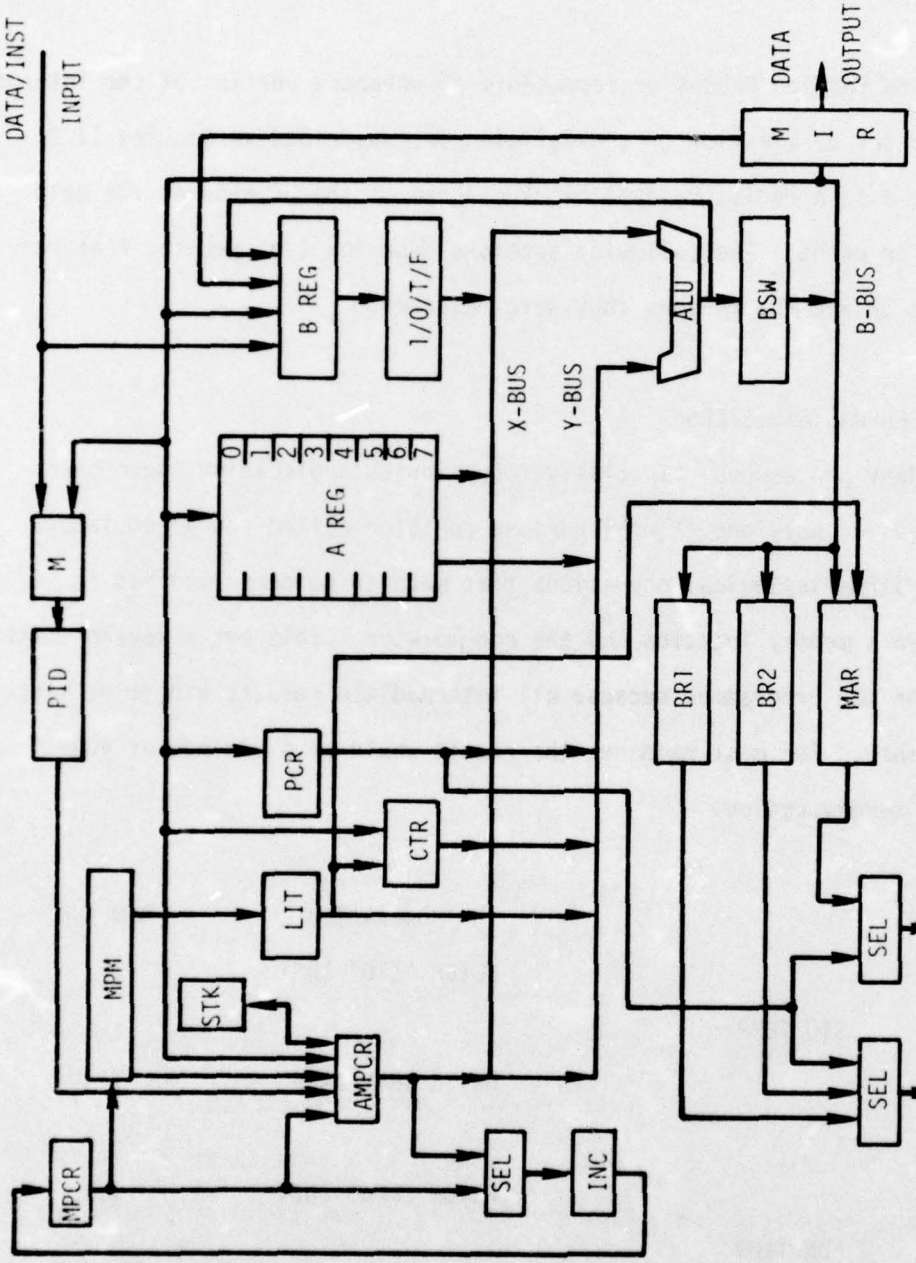


Figure 9. Unified Processor Block Diagram

The reason for the single accumulator machines was the requirement for simple logic, small size and low power. The cost of adding additional registers and the controlling logic was prohibiting.

As technology improved and integrated circuits became available the problems of size and cost were partially solved. A 16-bit register that once required eight printed circuit cards with discrete components could now be built from four ICs. Not only had the parts cost been lowered but, possibly more important, labor time had been reduced making the final product more reliable at a reduced cost.

Recent advances in integrated circuit technologies have made available an MSI device called a Multiport Register File. A typical example is the Fairchild 9338 which has eight 1-bit registers with one input port and two output ports. This allows the designer to put either general purpose registers into a processor by merely adding sufficient ICs for the work length.

With many of the new processors being developed having several general purpose registers it was necessary to include some in the Unified Processor in order to maintain a near real-time emulation ability. Although some machines have sixteen or even thirty-two general registers, only a few of the registers get frequent use. Eight registers were selected as an optimum between availability of ICs and programmer usage. It must be noted that if an emulation of a machine having sixteen or more general registers is desired, then all sixteen will have to be mapped into the UP. The eight most frequently used registers would of course use the hardware registers.

The eight general registers A_0, A_1 ... A_7 may be loaded one at a time from the B-Bus which is the output of the Barrel Switch. There are two output ports on the register file, one connected to the X-Bus, the other to the Y-Bus. Their addressing is completely independent, allowing any two registers to be selected for output to the ALU. This arrangement permits register to register operations such as:

1. $A_1 * A_3 \rightarrow A_7$
2. $A_7 * A_4 \rightarrow A_1$
3. $A_7 * A_2 \rightarrow A_7$

NOTE: *can be any of the 16 possible ALU operations to be performed in one micro instruction clock. (100 ns when in a normal instruction stream.)

The 9338 chip presently costs $6.00 each ($192.00 for eight 32-bit register) and has a read access time of less than 30 ns, making it as fast as most TTL gating circuitry would be. To write into a register requires a setup time of approximately 15 ns on both address and data and a clock of 10 ns. This means that the register file costs very little in speed for the vast improvement in programming ease it gives over the single accumulator.

2. BARREL SWITCH

In many types of information processing such as communication, emulation, compiling, character manipulation, etc., it is necessary to

shift or rotate a data word several places. The operation is normally performed one shift left or right per instruction. The procedure takes one to two memory cycles each for position so that on a 10 position shift 10 cycles (about 10 microsec) would be required.

The need for a means of shifting data rapidly is probably best demonstrated when considering an emulator. The original machine may have two half word counters hardwired together to give the effect of one large register but still allowing the programmer the ability of incrementing or loading each half independently. A system like this would be useful in a paging type operation where the most significant counter defines the page and the least significant counter identifies the particular entry. The original machine would take one clock period to increment or load either half of the register. In an emulation to load or increment the upper half of the word would require the following steps.

NOTE: MS is multiple shifts per clock and SS is single shift per clock.

STEP	MS	SS
ROTATE Reg Left 8 places	1	8
Increment/Load Reg	1	1
ROTATE Reg Right 8 places	1	8
TOTAL	3	17

For this particular example the multiple shift emulator takes three times as much time as the original machine but it is still almost six times better than the single shift machine.

The multiple shifter or Barrel Switch is a device that allows the input to be shifted end-off right or left or right end-around n-1 places in one clock where n is the width of the registers. One implementation of a multiple shifter is shown in Figure 10.

The single level switching system in Figure 11 will provide the necessary gating to perform right and left end-off shift and right end-around shifts on eight bits of data. The control of the shifter is simple with three lines determining the amount of the shift and eight lines for blanking control (fill with zeros). The states of the controls for the different shifts are shown in Figure 12. The circuit could be built from eight 8 to 1 multiplexers such as the 74151.

The single level system may be satisfactory for an eight-bit word operation but if the word is longer (16, 24, 32 bits) then a significant problem arises. The multiplex IC must have N inputs for an N bit word system. Although there are 16 to 1 multiplexers available there are no standard 24 to 1 or 32 to 1 ICs.

To avoid the problem of using non-standard ICs the multiplexing was divided into two levels. This allows a total word length 64 bits without going beyond the standard 8 to 1 multiplex IC. A two-level 8-bit switching system is shown in Figure 13. This circuit performs the same function as the singles level system and uses the same shift amount and blanking controls. One extra benefit of the two level system is that in this case the circuit could be built from four (4) 74153 DUAL 4 input

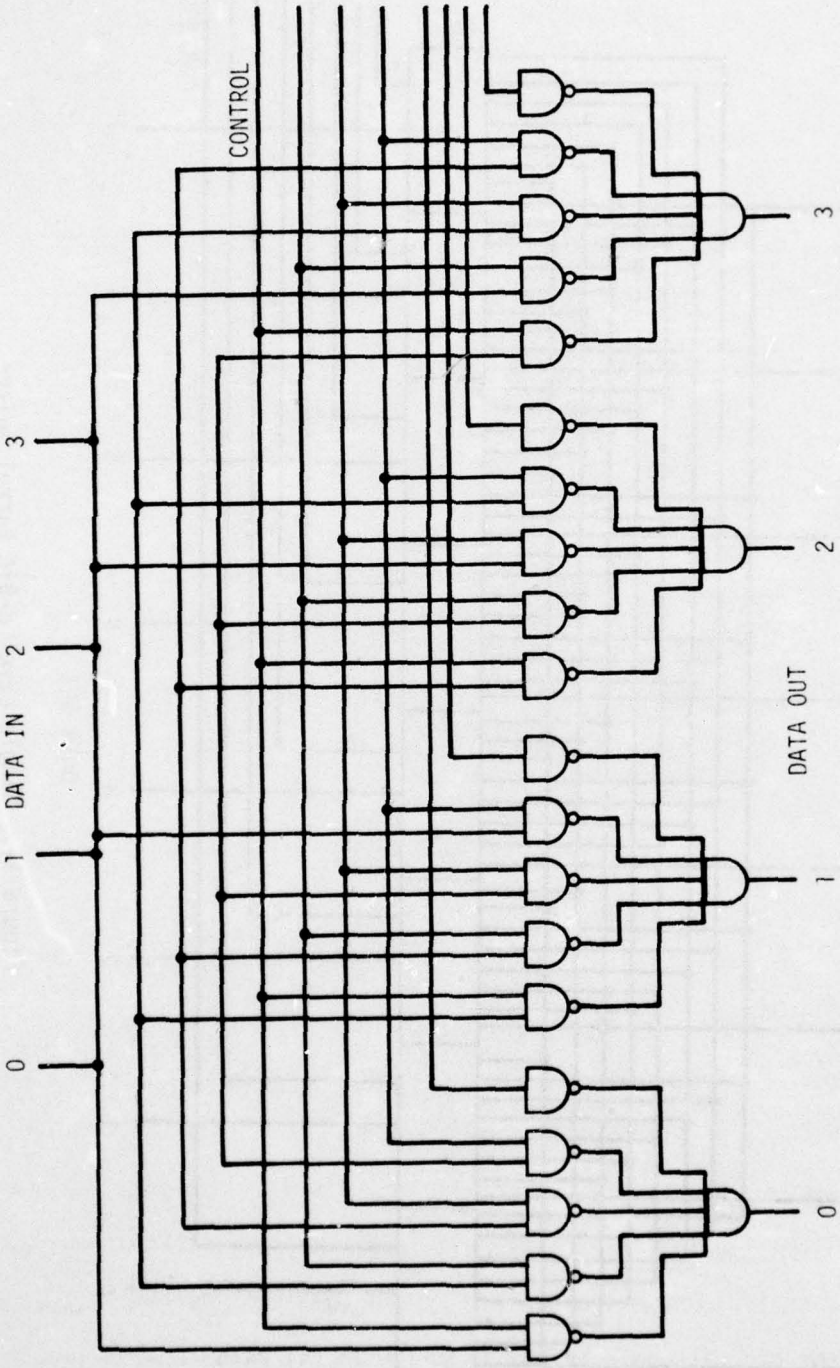


Figure 10. Multiple Shifter

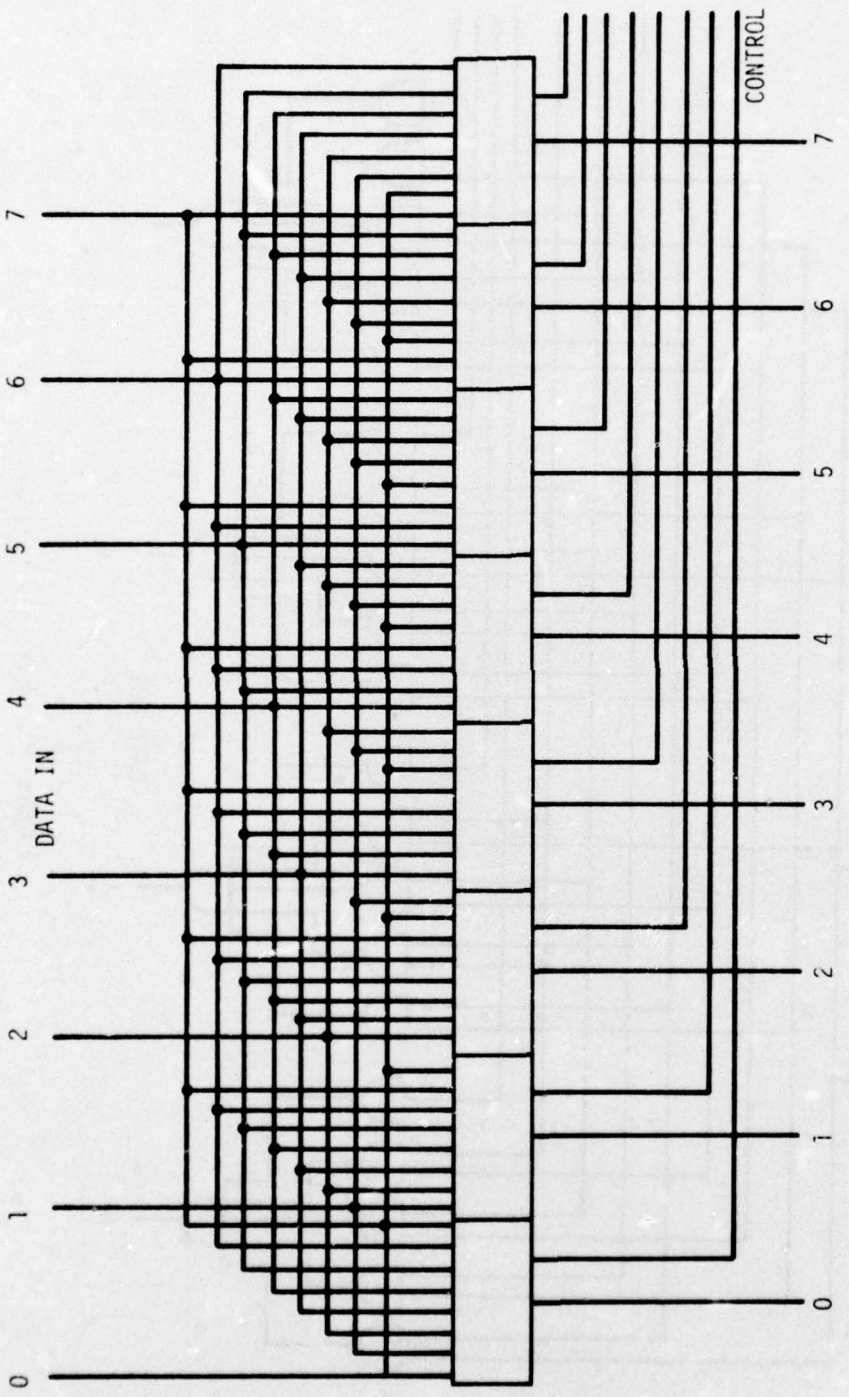


Figure 11. Single Level 8-Bit Barrel Switch

LEFT END OFF

SHFT	A	B	C	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7
0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1	1	0
2	0	1	0	1	1	1	1	1	1	0	0
3	0	1	1	1	1	1	1	1	0	0	0
4	1	0	0	1	1	1	1	0	0	0	0
5	1	0	1	1	1	1	0	0	0	0	0
6	1	1	0	1	1	0	0	0	0	0	0
7	1	1	1	1	0	0	0	0	0	0	0

RIGHT END OFF

SHFT	A	B	C	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7
0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	1
2	1	1	0	0	0	1	1	1	1	1	1
3	1	0	1	0	0	0	1	1	1	1	1
4	1	0	0	0	0	0	0	1	1	1	1
5	0	1	1	0	0	0	0	0	1	1	1
6	0	1	0	0	0	0	0	0	0	1	1
7	0	0	1	0	0	0	0	0	0	0	1

RIGHT END AROUND

SHFT	A	B	C	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7
0	0	0	0	1	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	0	1	1	1	1	1	1	1	1
3	1	0	1	1	1	1	1	1	1	1	1
4	1	0	0	1	1	1	1	1	1	1	1
5	0	1	1	1	1	1	1	1	1	1	1
6	0	1	0	1	1	1	1	1	1	1	1
7	0	0	1	1	1	1	1	1	1	1	1

Figure 12. Single Level 8-Bit Barrel Switch Control

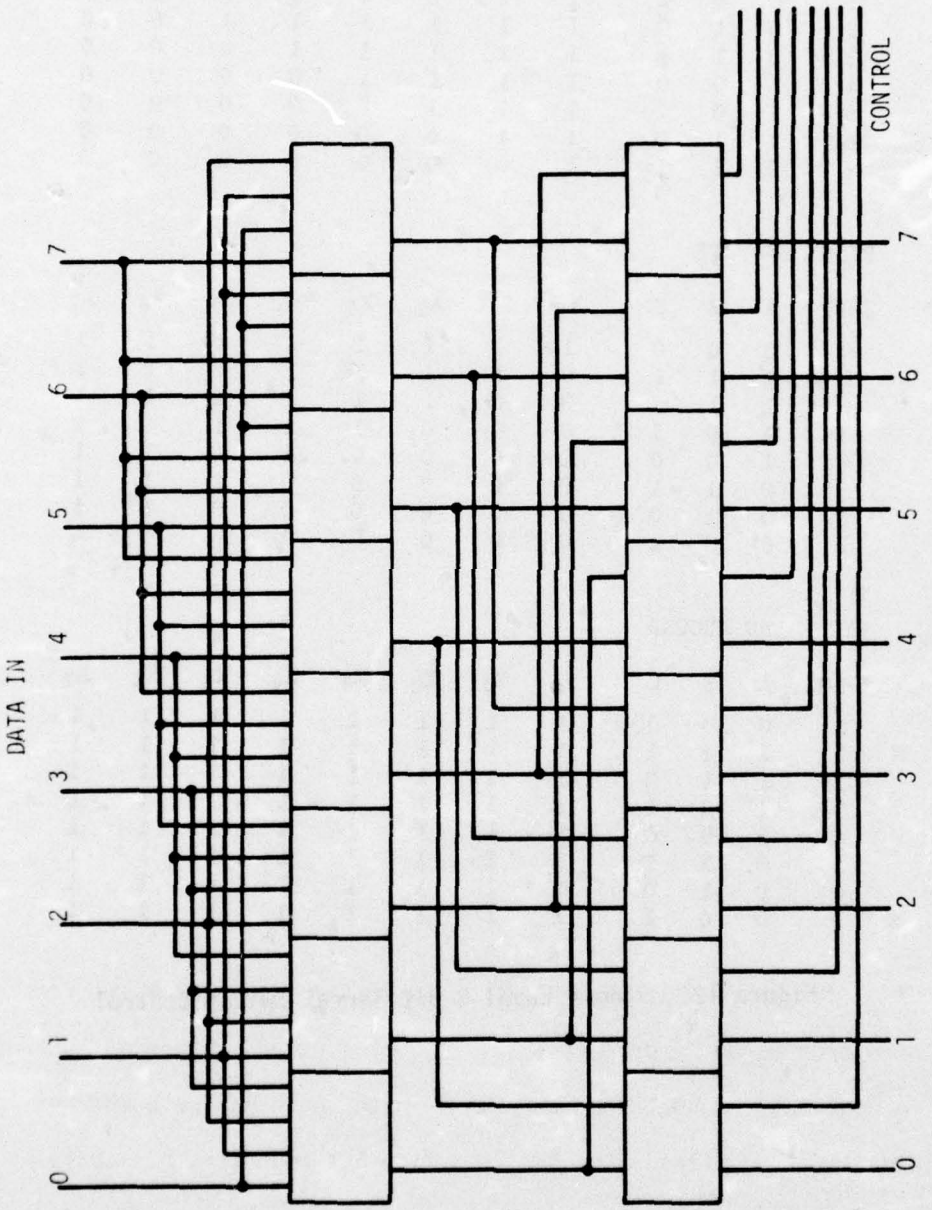


Figure 13. Two Level 8-Bit Barrel Switch

multiplexers and two 74157 QUAD 2 input multiplexers, a savings of two packages over the single level.

A modified version of the two level system shown in Figure 13 was used for the 32 bit shifter in the Unified Processor. The first level uses 8 to 1 multiplexers and the second level uses 4 to 1 multiplexers. Because of an 8-bit modularity requirement the first level required nine ICs instead of the normal eight. The total package count was 52.

a. Barrel Switch Operation

The specific Barrel Switch that has been implemented in the Unified Processor consists of thirty-six 8 to 1 multiplexers for the first level and thirty-two 4 to 1 multiplexers for the second level. The first level has four more multiplexers than a minimum system in order to ease the interconnection of eight-bit wide modules. Another advantage is that the blanking control logic is simpler. With nine columns the blanking of each column proceeds in an orderly progression dependent upon the amount of shift. If only eight ICs were used then a more complicated series of controls would be necessary in order for the less orderly blanking required.

The Barrel Switch is controlled from two sources. The first is the Microprogram Memory bits [37, 38]. The second source of control is from the Shift Amount Register (SAR) which contains the number of positions to be shifted. The SAR is a five-bit register that contains the actual value of the amount to be shifted for right end-off and right end-around shifts. For left end-off shifts the value in the SAR is the twos complement of the actual shift amount.

TWOs complement = $(\overline{DATA}) + 1$

EXAMPLE:

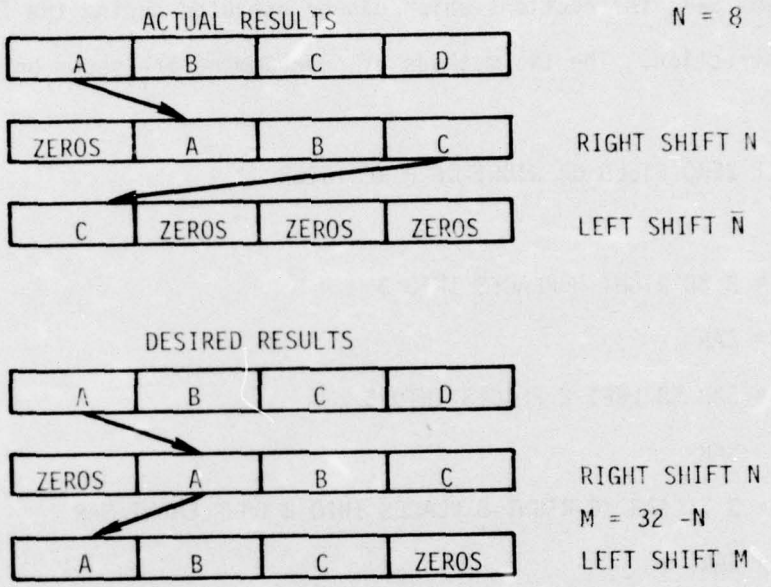
	SAR				
	0	1	2	3	4
RIGHT END OFF					
23 Places	1	0	1	1	1

	SAR				
	0	1	2	3	4
LEFT END OFF					
23 Places	0	1	0	0	1

Using standard 74 series TTL circuits the present Barrel Switch can shift data in about fifty nanoseconds. This includes the time necessary for the SAR controls to settle and the data to propagate through both levels of the switch.

During normal programming it is often desired to shift data right end-off n places and then left end-off n places or right end-around n places and then returned. The first procedure is a blank field n bits long on the right of a word and the second allows an easy means of testing the nth bit of a word. Because left shifts expect the twos complement of the shift amount in the SAR the desired results would not be produced if a right shift - left shift or right circular - TEST - right circular were executed. Figure 14 shows the desired results and the actual results that would be obtained. There are two ways to get the desired results. The first way is to insert an extra instruction

RIGHT SHIFT - LEFT SHIFT



RIGHT CIRC. - TEST - RIGHT CIRC

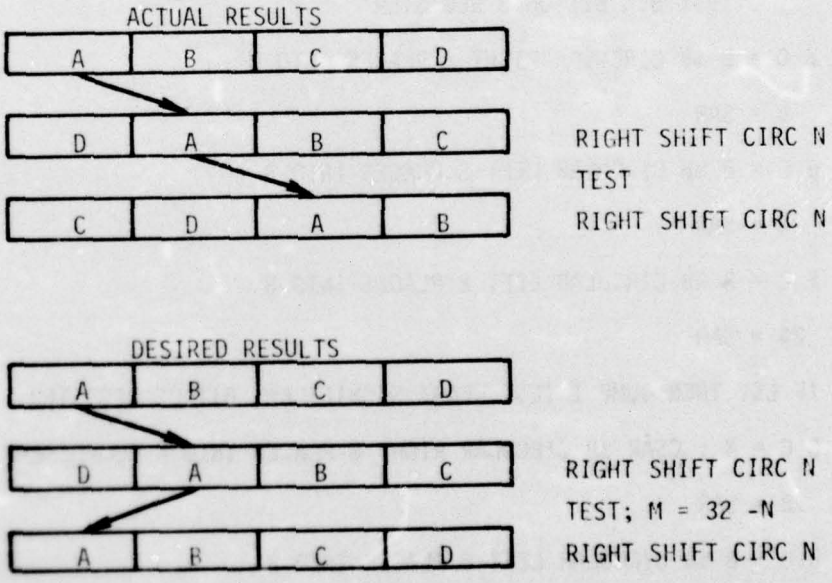


Figure 14. Shifting Examples

(TYPE II SAR LOAD) into the stream to modify the SAR contents, but this of course requires an extra clock. A better way is to use the CSAR (Complement SAR) instructions which can be executed during the first shift instruction. The two methods of programming are shown below:

```
8 BIT ZERO FIELD OR RIGHT OF B REGISTER

B R = B %B RIGHT 8 PLACES INTO B
  8 = SAR
B L = SAR %B LEFT 8 PLACES INTO B
 24 = SAR
B R = B ; CSAR %B RIGHT 8 PLACES INTO B COMPLEMENT SAR
  8 = SAR
B L = B %B LEFT 8 PLACES INTO B
      TEST 8th BIT OF B REGISTER
B C = B %B CIRCULAR RIGHT 8 PLACES INTO B
  8 = SAR
B C = B %B CIRCULAR LEFT 8 PLACES INTO B
 24 = SAR
B C = B %B CIRCULAR LEFT 8 PLACES INTO B
 24 = SAR
IF LST THEN JUMP % TEST LEAST SIGNIFICANT BIT OF REGISTER
B C = B : CSAR %B CIRCULAR RIGHT 8 PLACES INTO B COMPLEMENT SAR
  8 = SAR
B C = B %B CIRCULAR LEFT 8 PLACES INTO B
IF LST THEN JUMP % TEST LEAST SIGNIFICANT BIT OF B REGISTER
```

NOTE: The CSAR instruction is executed at the end of the second phase of the instruction therefore it does not affect the instruction in which it appears. The 8 = SAR is a Type II instruction and is executed prior to the completion of the previous instruction.

b. Barrel Switch Control

The Shift Amount Register along with MPM [57, 58] provide the necessary information for controlling the Barrel Switch. Figure 15 is a general block diagram of the SAR and its support logic.

The SAR may be loaded from one of three sources. The method that is most used is a load from the Microprogram Memory with a Type II instruction. This requires the programmer to insert a constant into memory to be loaded in the proper sequence so that the necessary shifting may take place. As mentioned previously there are several times in programming that it is desirable to shift information temporarily and then return it to its original position. To accomplish this sequence of operations it is necessary to provide the original shift amount and then provide it twos complement for the second shift. The necessary logic to produce the twos complement of the present SAR is provided so that the process requires only the CSAR input selected and an SAR clock. This operation is executed at the end of Phase II and therefore does not affect the present instruction but is complete for the next instruction to use. A programming example and timing chart is shown in Figure 16. The "*" in the timing diagram shows when the operation is performed. The time between clocks is used only for controls to stabilize. The

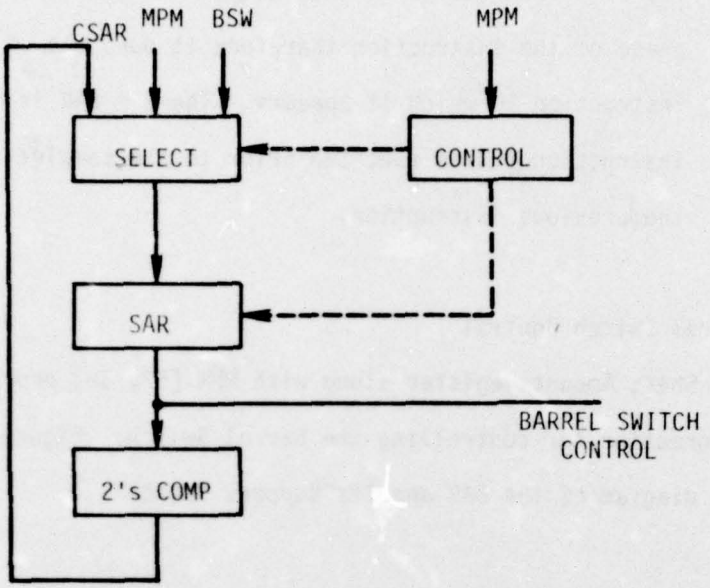


Figure 15. Shift Amount Register Block Diagram

```
1.  A1 R = A1; CSAR          % PRODUCE AN 8-BIT
2.  8 = SAR                  % ZERO FIELD RIGHT
3.  A1 L = A1                % JUSTIFIED IN A1
```

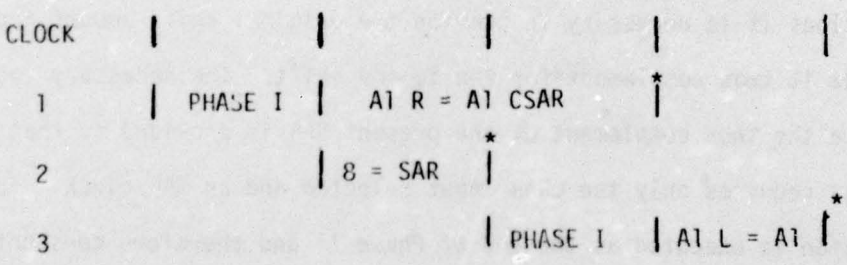


Figure 16. Timing of CSAR Instruction

delay is execution of instruction one is accomplished by inhibiting the clock used by Type I instructions.

The third source for the SAR is the least used but probably the most powerful. Loading the SAR from the Barrel Switch allows the programmer to dynamically determine the shift amount based upon present conditions. A possible application for this operation is in doing scaling for floating point arithmetic. The amount of the shift is dependent upon the value of the exponent and providing an easy way of entering the exponent into the SAR can greatly increase the speed of these instructions.

3. MICROPROGRAM ADDRESSING SYSTEM

The control Unit of a microprogrammable processor is in reality a complete processor in itself. This microprocessor works within a somewhat limited environment, taking its input from the conditions of the Logic Unit and I/O Unit and giving as its output, commands to these units. It must be capable of executing a series of instructions (addressing MPM), testing various conditions (AOV, MST, LC, etc.), and issuing commands. The program executed by the microprocessor is called Firmware and is written in much the same way as standard software is. With all programs there is a need to be able to progress sequentially through a series of instructions (STEP) or to execute instruction not in a sequential manner as a loop or jump to a subroutine. To provide these options to the microprogrammer several successor commands are available. Each instruction must have a successor command so that the location of

the next instruction can be determined. Listed below are the eight possible successors available in the Unified Processor:

	NEXT MPCR	NEXT EMPCR	MPMADDR
WAIT	MPCR	AMPCR	MPCR
STEP	MPCR+1	AMPCR	MPCR
SAVE	MPCR+1	AMPCR	MPCR
SKIP	MPCR+2	AMPCR	MPCR
JUMP	AMPCR+1	AMPCR	MPCR
EXEC	MPCR+1	AMPCR	AMPCR
CALL	AMPCR+1	MPCR	MPCR
RETN	AMPCR+2	AMPCR	MPCR

WAIT

The WAIT successor is used mainly for synchronization between the processor and the external world. When a read from memory operation is performed it is necessary for the processor to wait for the new data before it proceeds with the calculations. Between the read operation and the entering of data several instructions may be executed so that meaningful work can be accomplished during this time. The WAIT instruction does not really turn off the processor but instead increments the present address (MPCR) by zero. This means that the same instruction is executed several times. As every instruction has two successors (TRUE & FALSE) it is obvious that a condition can be tested and as long as it is FALSE the WAIT successor is used. When it becomes TRUE the

other successor is used. Some examples of how the WAIT successor is used are shown below:

Synchronizing Processor and Memory

```
MR                                  % START MEMORY READ
IF RDC THEN BEX STEP ELSE WAIT      % WAIT FOR MEMORY TO
                                    % RESPOND THEN INPUT DATA
```

Single Instruction Loop

```
LCTR                                % LOAD THE COUNTER WITH
1000 = LIT                          % ONE'S COMP OF 1000.
INC, IF NOT COV THEN MW WAIT ELSE STEP
% INCREMENT COUNTER, IF NO OVERFLOW THEN WRITE
% INTO MEMORY AGAIN.  ELSE STEP.
```

STEP

In microprogramming as in other programming the instructions are set in an orderly sequence so that most instructions will have a STEP successor. The step successor cuases the MPCR to be incremented by one so that the next instruction is taken from the next word in memory. When STEP is used by itself (no logic or external operation) the equivalent of a NO-OP is performed.

SAVE

SAVE is used mainly to create the top end of a loop. The present contents of the MPCR are clocked into the AMPCR (SAVED) and the MPCR is incremented by one (STEPED). After this operation is complete a sub-phase cycle is initiated that writes the AMPCR into the stack. The

AMPCR is clocked at the trailing edge of Phase I (reference Figure 17) so that it is ready for the next instruction to use if desired. Below are some uses of the SAVE instruction.

```
   0 = PCR                          % RESET PROG CNT
 LIT = A1, MIR; IF SAI              % PUT ASCII A INTO
  65 = LIT                          % A1 & MIR; REST SAI
 LIT = CTR; SAVE                    % SET COUNTER FOR 26 LOOPS
  25 = LIT                          % TOP OF LOOP
  MW, INC.                          % WRITE, INCREMENT COUNT
  AL + 1 = A1, MIR; IPCR            % INC. A1 & MIR, NEXT LET
  IF AIS THEN STEP ELSE WAIT        % WAIT FOR MEM COMP
  IF COV THEN STEP ELSE RETN        % TEST FOR DONE
```

This program writes the alphabet in ASCII form into memory starting at location zero. The loop that is executed consists of the last four instructions. Note that the SAVE successor has put the address of the LIT = CTR instruction into the AMPCR but when the RETN successor is executed it branches to AMPCR+2 which makes MW, INC the top of the loop.

SKIP

The SKIP successor performs exactly the same way as the STEP except that the MPCR is incremented by two instead of one. This allows the programmer to create longer instruction. An example would be to load register A1 from A6 and A2 from B only if a program flag (LC) is set.

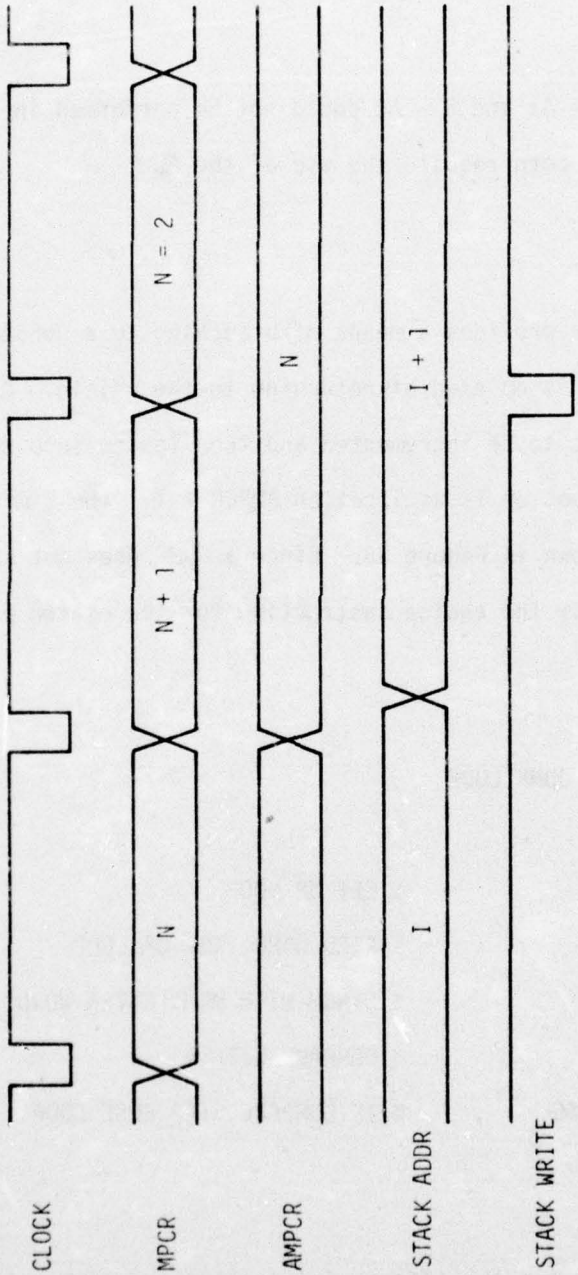


Figure 17. SAVE Successor Timing

```
IF LC THEN A6 = A1; STEP ELSE SKIP
B = A2
STEP
```

Note the operation A6 = A1 and B - A2 could not be performed in the same instruction since they both require the use of the ALU.

JUMP

The JUMP successor provides a means of branching to a nonsequential instruction when there is no need of returning to the origin. A JUMP causes the AMPCR output to be incremented and then loaded into the MPCR so that the next instruction is at location AMPCR + 1. The timing for this instruction is shown in Figure 18. Since a JUMP does not affect the AMPCR it is normally the choice instruction for the bottom of a loop.

Example of a SAVE-JUMP LOOP

```
SAVE                        % SET UP LOOP
MR                          % READ WORD, TOP OF LOOP
WHEN RDC THEN BEX           % SYNCH WITH MEM, ENTER WORD
A1 EQV B, IPCR              % COMPARE WITH A1
IF NOT ABT THEN JUMP        % IF COMPARE STEP ELSE LOOP
```

EXEC

The EXEC successor has the effect of inserting one instruction into the normal sequence. The MPCR is incremented by zero (WAIT) for one clock while the AMPCR is used as the MPM address source. This condition

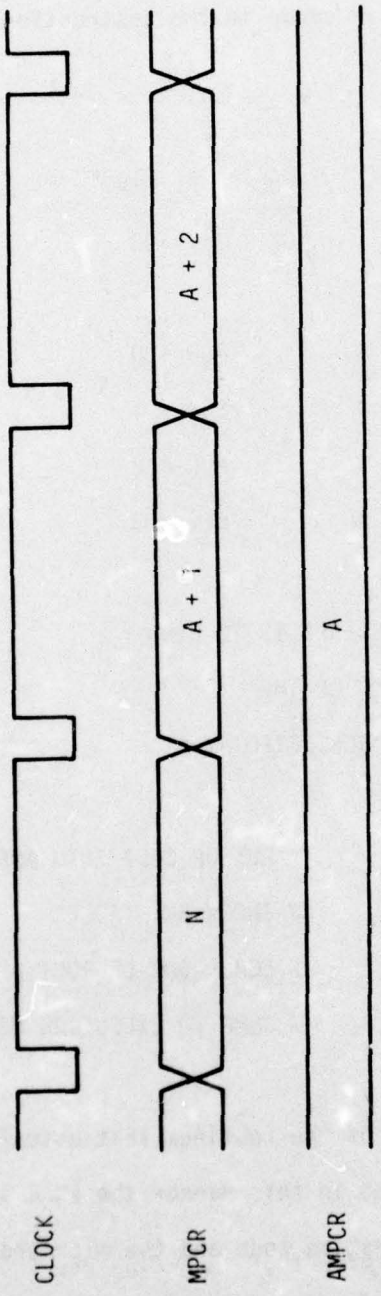


Figure 18. JUMP Successor Timing

is maintained for only one clock and is then returned to the normal sequence (reference Figure 19). The EXEC instruction can be used to make the MPM act as a ROM decoder as shown in the instruction decode program below.

```
% INST        ADD     SUB     MUL     DIV
% OP-CODE     00      01      02      03

TABLE:   MADD-1 = AMPCR     %       ADD = 0
         MSUB-1 = AMPCR     %       SUB = 1
         MMUL-1 = AMPCR     %       MUL = 2
         MDIV-1 = AMPCR     %       DIV = 3

DECODE:  % THE DECODE ROUTINE EXPECTS TO FIND
         % THE OP-CODE PORTION OF THE
         % INSTRUCTION RIGHT JUSTIFIED BY A1

           A1 + LIT = AMPCR      % PUT OP CODE INTO AMPCR
           TABLE = LIT           % INDEX BY "TABLE"
           EXEC                  % LOAD ADDR OF ROUTINE INTO AMPCR
           JUMP                  % JUMP TO EXECUTION ROUTINE
```

MADD, MSUB, etc. are the addresses of the routines that actually perform the ADD, SUB, etc. operations. Used in this manner the EXEC instruction provides a mapping between the operation code and the microprogram routine that performs it.

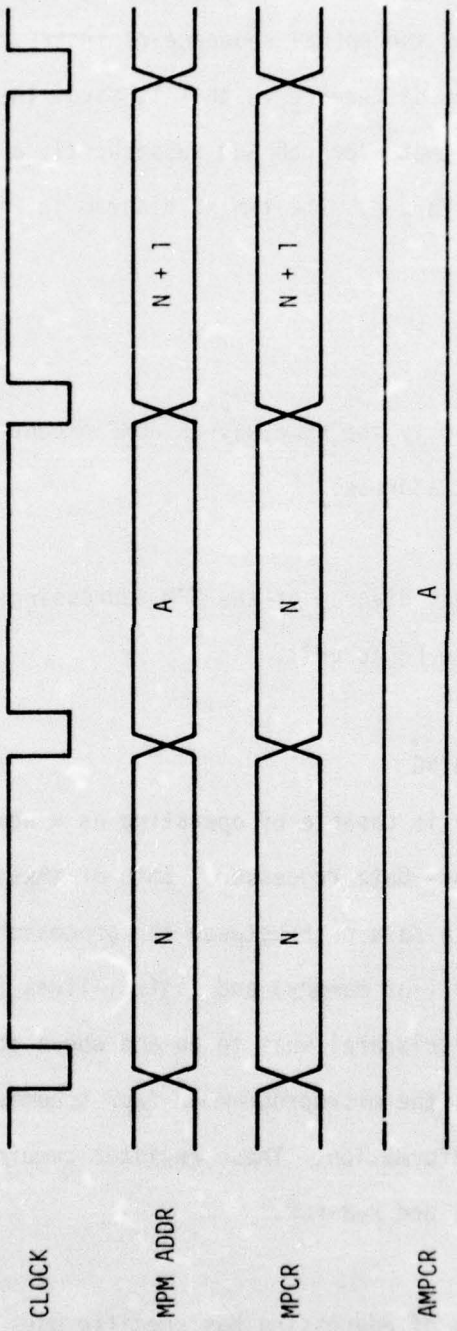


Figure 19. EXEC Sucessor Timing

CALL

The CALL successor, as with the JUMP, provides the microprogrammer a means of branching out of the normal sequence of instruction. The CALL does have an important difference in that it saves the present MPCR in the AMPCR. This means that a branch and subsequently a return to the calling program can be performed. The timing diagram in Figure 20 shows the operation of a CALL.

RETN

The RETN operates exactly the same way as JUMP except it uses the AMPCR plus two as the next address.

Figure 21 shows a block diagram of the MPM addressing system and its interconnections to the Logic Unit.

4. MEMORY/DEVICE ADDRESSING

The Unified Processor is capable of operating as a Real-Time Controller or a general purpose Data Processor. Both of these applications, of course, require a data path between the processor and device. In addition, address lines (for memory) and control lines (for devices) are required to tell the peripheral what to do and where to do it. The Unified Processor provides the microprogrammer four schemes for outputting address/control information. These register combinations are BR1-MAR, BR2-MAR, BR1-PCR, and PCR-PCR.

Each of the four ways of addressing has specific uses for which it is best suited. Examples of the various ways are shown below.

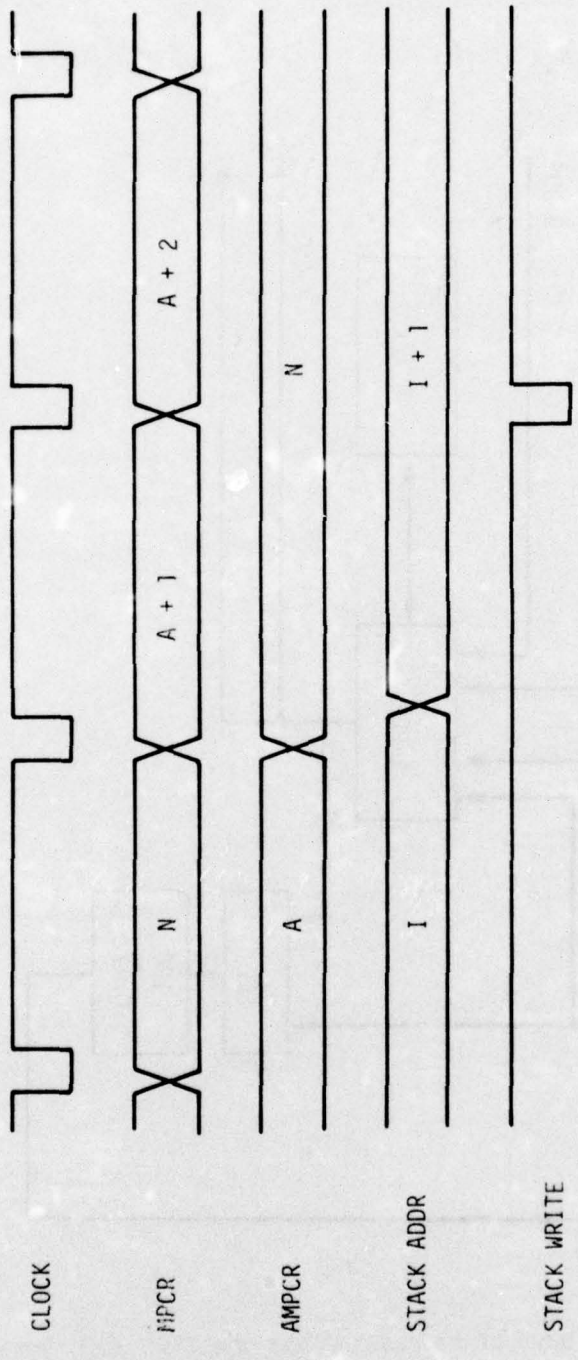


Figure 20. CALL Successor Timing

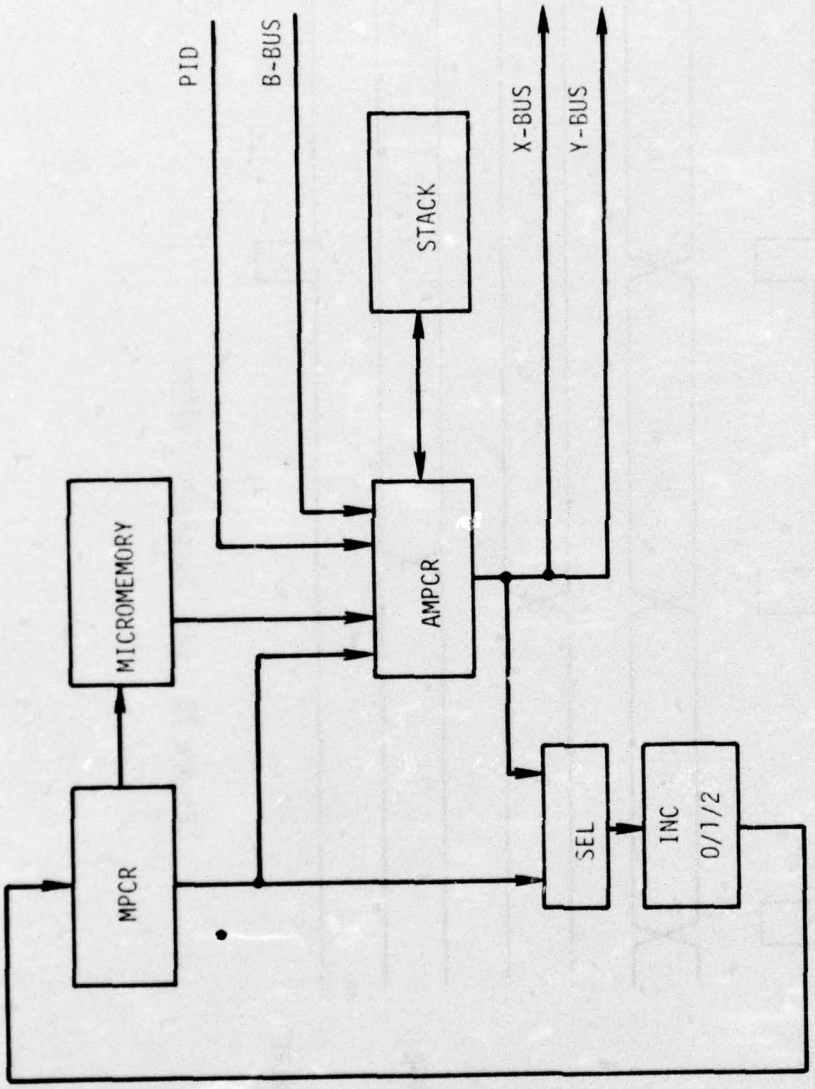


Figure 21. MPM Addressing Block Diagram

BR1-MAR. This form is available for all read or write operations. It is particularly useful when addressing a particular memory location or when controlling a device. The normal procedure would be to use BR1 to address the peripheral and the MAR to provide the control bits. This combination is also useful for supplying data addresses in an emulation since they tend to be nonsequentially located in memory.

BR2-MAR is the same as BR1-MAR and is included to allow the programmer the freedom of using one Base Register to address devices and the second for memory paging.

BR1-PCR. This form is available only on memory read or write and function operations. It provides the capability of using a Base Register to select a particular page or top of a table and then an eight-bit counter to step through the desired area.

PCR-PCR. This form is more in line with the conventional program counter. When using this mode the low order sixteen bits of the PCR are supplied to the I/O Unit as on address.

The three registers BR1, BR2, and MAR are each eight bits long. Their position in the UP is such that their major function is as an address buffer. This is because they cannot be used as temporary storage since there is no way to read them. The registers may be loaded

with the eight least significant bits of the Barrel Switch, or the MAR may be loaded from the Literal Register. Operations such as:

LMAR; A6 = BRl

which loads the MAR from the Literal Register and at the same time loads BRl from register A6 are permissible.

The fourth address register, the PCR, differs from the other three several ways. It is usable as a full thirty-two bit general purpose register with the same restrictions as the A registers have. The PCR is loaded from the B-Bus which is the output of the Barrel Switch. It can be used as a source of data for the ALU on either the X-Bus or Y-Bus. The PCR may also be used as a counter to keep track of the number of cycles in a loop or to step through a table in memory. Not only can it be incremented independently of all other operations it can also be tested for a carryout of the sixteenth or thirty-second bit.

5. MICROMEMORY CONTROL PARTITIONS

The Micromemory which is sixty-four bits wide supplies the necessary control lines to set the state of the processor at every clock. The memory is divided into several fields (see Figure 22) to provide independent control of different functions. The particular scheme used was selected to provide maximum flexibility of the processor's resources with minimum hardware decoding. The entries within a field represent mutually exclusive controls such as right shift and left shift. Any combination of elements from different fields can be used to give a valid microinstruction even though it may not be meaningful.

MPM DITS	
1	TYPE I or II
2-6	Condition to be tested
7	Select TRUE or NOT TRUE of condition
8	Conditional or Unconditional LU operation
9	Conditionl or Unconditional Ext operation
10-13	Condition Adjustment (set-reset)
14-19	Select TRUE and FALSE Successors
20-26	Selects X-Bus data source
27-31	Selects Y-Bus data source
32	Byte Carry Control
33-36	ALU Control
37-38	Shift Operation
39-42	Select A Register distination from B-Bus
43-46	Select B Register input source
47	MIR input enable
48-49	AMPCR source select
50-52	Address register input enable
52-56	Counter controls
57-58	SAR Source select
59-63	External Memory/Device operation select
64	Programmable Instruction Decoder enable

Figure 22. Type I Microinstruction Control Fields

A microinstruction is read from the Microprogram Memory (MPM) every clock. All instructions fall into one of two categories as determined by bit one of the instruction. If MPM[1] is a zero the instruction is a Type I class and the remaining bits are to be interpreted as controls.

If MPM[1] is a one then the instruction is a TYPE II class which is used for leading constants into selected register. MPM bits 3, 4, and 5 determining which register or combination of registers will be loaded.

MPM[3] AMPCR Control

MPM[4] LITERAL Register Control

MPM[5] SAR Control

Not all control fields from a single microinstruction are executed at the same time. The instruction is divided into two phases with phase one execution occurring on the next clock and phase two delayed one clock. Figure 23 shows the two-phase timing for several instructions. Note that phase one of instruction M+1 overlaps phase two of instruction M; therefore, even though each instruction requires two clocks to complete its execution with the overlap the average is one clock per instruction.

The operations during phase one of an instruction occur in the following sequence. At clock n the address for instruction M is applied to the MPM. After the completion of the memory access time (typically 40-60 n sec) instruction M is available. MPM [2-19] are applied directly to the control logic to determine the address of the next microinstruction.

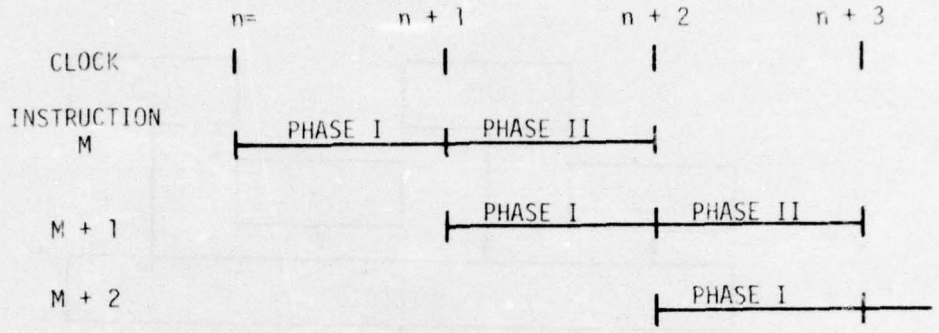


Figure 23. Instruction Timing Diagram

The remainder of the MPM bits are applied to a buffer. At clock n+1 the next (M+1) address is loaded into the MPCR; also MPM [20-64] are loaded into a buffer. During this phase (Execution Phase) the sources to the ALU and the ALU operation are set up. The registers that are to be loaded will have their clock lines enabled so that on clock n+2 the result will be loaded into the selected destination.

6. INTERNAL BUSSING

There are three main busses in the UP that are used for providing information to the ALU (X-Bus, Y-Bus). The X-Bus is a standard multiplex network that allows selection of any one register to be connected to the ALU. The bus is implemented using 74153 4 line to 1 line Data Selector ICs. The configuration is shown in Figure 24.

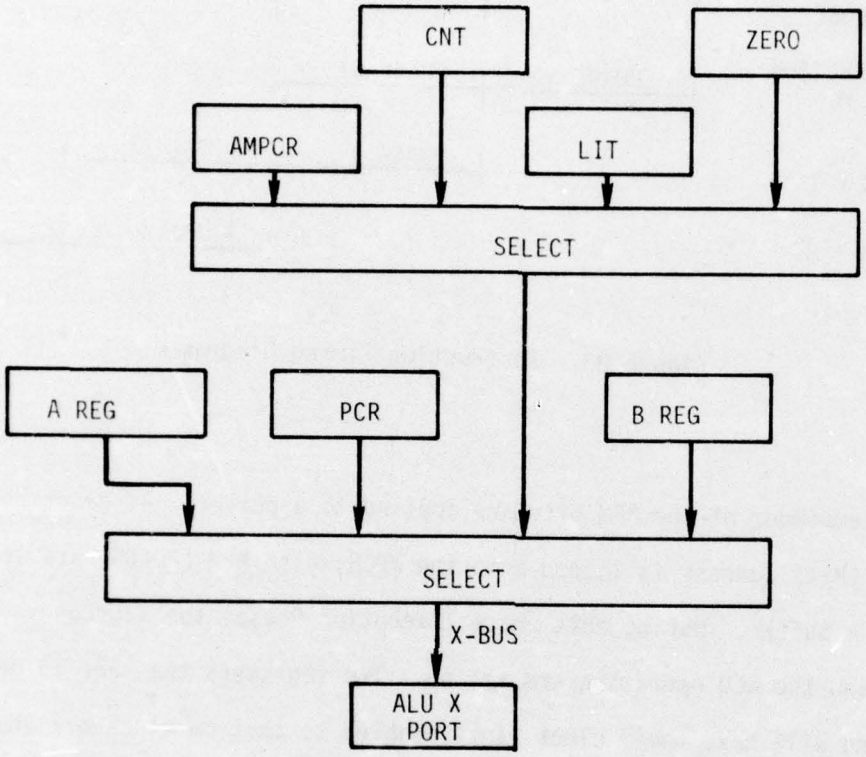


Figure 24. X-BUS MULTIPLEX STRUCTURE

The Y-Bus differs from the X-Bus in that it uses a Tri-State concept. Each register is connected to the bus with a Tri-State Buffer such as a 74126 Quad Bus Buffer Gate with Tri-State outputs. The gates have the normal on and off states and in addition have a third state where both output transistors are turned off, providing a very high impedance shunt on the bus. Because of the fixed nature of the X-Bus it was necessary to use a system on the Y-Bus that allowed future expansion of the system without necessitating a redesign of the control section. A possible application of this bussing structure is when using an external function and gating the result directly onto the Y-Bus. This has the effect of providing a second input port, and since the ALU and BSW are nonmemory asynchronous circuits the data can be clocked into any register in one instruction time.

An example of this feature is shown below where an external SINE function generator is attached to the UP. The angle in degrees is located in register A7 and the resulting angle in radia ns is to be placed in register A7. Note it is assumed that the SINE generator requires less than one clock to obtain a result.

```
LIT = BR1, MAR          % LOAD SINE GEN ADDRESS
SINEADDR = LIT          %
A7 = MIR; FW1           % WRITE ANG IN DEG TO SINE GEN
0 = A7                  % RESULTS PLACED IN A7
```

The complete structure of the Y-Bus is shown in Figure 25. Note that a multiplexer is still used on this bus for the LIT, CNT, and AMPCR registers. This was done to relieve unnecessary loading on the Tri-State bus.

The third bus in the UP is much simpler than either the X-Bus or Y-Bus. The B-Bus is used to distribute the output of the ALU as modified by the Barrel Switch to the various registers in the processor. The bus is actually hardwired to all the registers (input port of the A registers) and a clock is applied to those registers that are to be loaded. This allows for loading of more than one register at a time. An example of how this may be used is shown below. The routine sends the ASCII characters A to Z to the Line Printer continuously.

```
        LPADDR = LIT            % ADDRESS THE LINE
        LIT = BR1, MAR          % PRINTER
LOOP2:  LIT = A1,               % LOAD REGISTER A1, WITH ONE
        OCTA-1 = LIT            % LESS THAN OCTAL VALUE OF ASCII A
        LIT = A,2; SAVE         % LOAD REGISTER A2 WITH THE
LOOP1:  OCTZ = LIT              % OCTAL VALUE OF ASCII Z
        A1+1 = A1, MIR          % INC A1 AND LOAD MIR
        DW1, A1 EQV LIT         % WRITE AND TEST FOR Z
        LOOP2-1 = AMPCR         % A-Z LOOP COMPLETE
        JUMP                    % START AGAIN AT LOOP 2
```

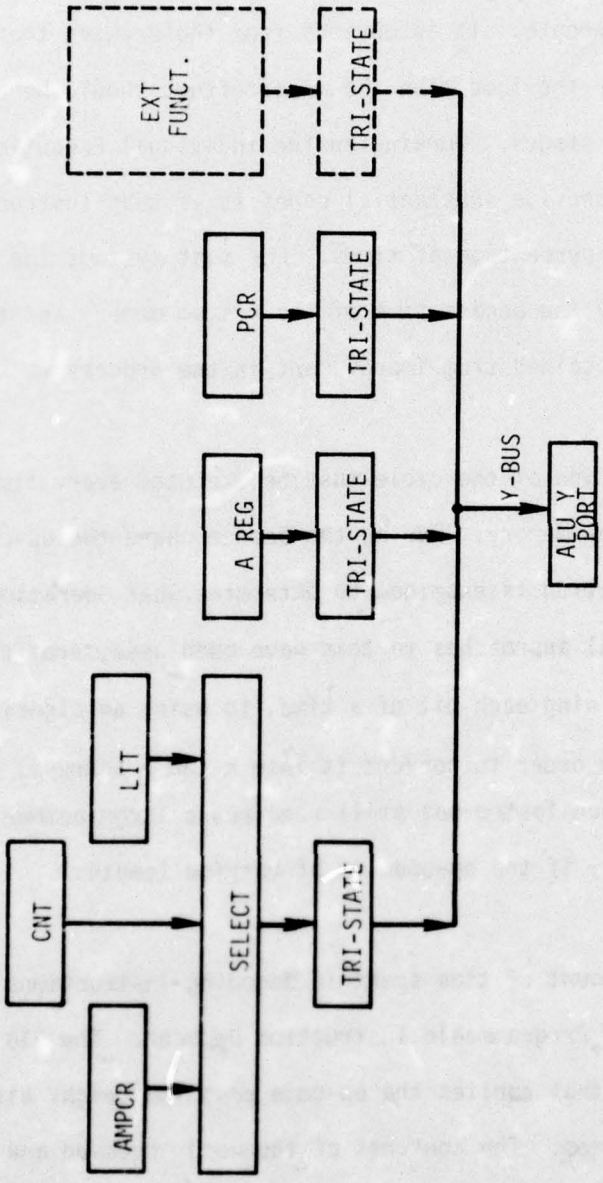


Figure 25. Y-Bus Tri-State Structure

7. PROGRAMMABLE INSTRUCTION DECODER (P/D)

During normal operation as a general purpose process a specific sequence of operations will be performed. Figure 26 shows the cycle of Fetch, Decode, and Execute. It is obvious from the drawing that if it is desired to shorten the loop time the major effort should be placed in the Fetch and Decode stages. Working on the individual Execution routines would only provide substantial benefits if that instruction were executed a high percentage of times. For most systems the Fetch time is controlled by the access time of the system memory and therefore no benefits can be obtained from improvement in the processor.

The Decode position of the cycle must be executed every time and it is not bound by system memory. During the Decode phase the op-code portion of an instruction is examined to determine what operation is to be performed. Several approaches to this have been used, from the very slow process of examining each bit of a time, to using an algorithm to modify the op-code in order to convert it into a table address. The second approach is much faster but still requires a large number of instructions, especially if the op-code is of varying length.

To reduce the amount of time spent in decoding instructions the UP was designed to use a Programmable Instruction Decoder. The PID is a random access memory that applies the op-code position (eight bits) to the memory address lines. The contents of the word accessed are used as a microprogram address to get to the proper Execution routine. The present implementation of the PID is with Read Mostly Memory having a write time of more than one millisecond and a read time of 45 nanoseconds.

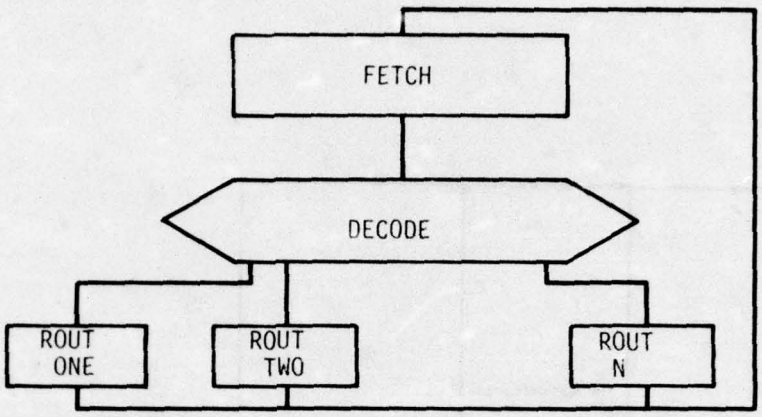


Figure 26. Normal Fetch-Decode-Exec Cycle

The memory is arranged as fourteen bits by 256 bits but could be increased if more address lines were brought through the mask. Only twelve bits of the memory are used for addressing the 4K Micromemory with the remaining two bits being testable as DC1 and DC2. This allows the programmer to use the same routine for several instructions with minor tailoring being performed based on the condition of DC1 and DC2. The PID can also be loaded from the B-Bus which allows a prefetch and storage in an A register of an instruction prior to its being required.

All information that is sent to the PID must first pass through a hardware mask. The mask is used to select the eight op-code bits from the thirty-two bit instruction and from a continuous eight bit PID address. An example of the PID in operation is shown in Figure 27. An instruction is read from core memory and applied through the mask to the PID. In Figure 27 this is an ADD instruction and therefore accesses the PID location AAD. The content of this location MADD is the address of

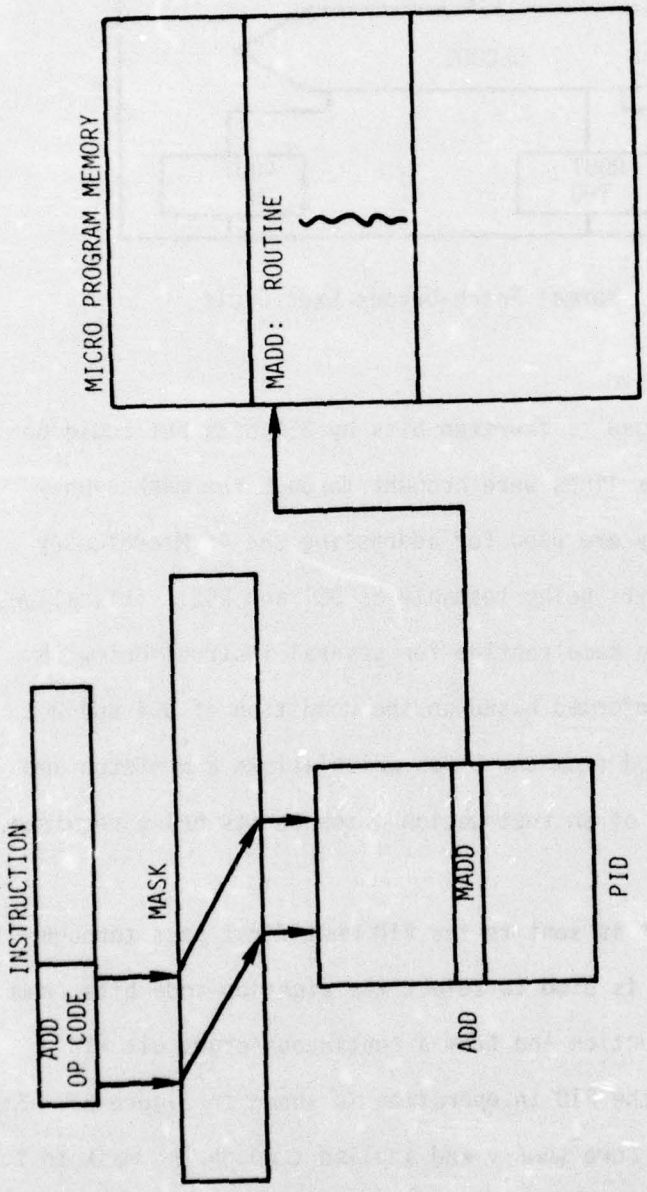


Figure 27. Programmable Instruction Decoder Operation

the Micro Add routine and can be used directly to jump and begin execution of the MADD routine. A programming example of the use of the PID to decode a simple instruction set is shown below.

```
MR4                          % READ NEXT INSTRUCTION
WHEN RDC THEN PID            % LOAD INTO AMPCR
JUMP                         % JUMP TO ROUTINE
```

8. INPUT/OUTPUT REGISTERS

The Unified Processor is completely buffered from the outside world by the B registers for incoming data and the Memory Information Register (MIR) for data output. The entry and exit of data through these registers is completely under the control of the microprogram and not directly tied to any I/O operation being performed.

The B register, in addition to providing input data buffering, is usable as a general purpose register. It is connected to the X-Bus through a series of gates called the I/O/T/F. The purpose of this gating network is to provide the programmer control of the form the data in the B register will take when put on the X-Bus. The network, which is divided into three sections - the most significant bit, the least significant bit, and all the rest, modifies the B register output in the following manner.

T	TRUE	Date passed unaffected (Data out = B)
F	FALSE	Date complemented (Data out = B)
1	ONE	Output set to one (Data out = one)
0	ZERO	Output set to zero (Data out = zero)

An example of the usefulness of this feature is shown in the multiply example given in Section V.

In addition to being loadable from the Input Bus the B register may also be loaded from the Barrel Switch, the MIR, or the ALU. Several ORed combinations of these are also available as shown below.

ALU	output of ALU
BSW	output of Barrel Switch
EXT	External data
MIR	Output of MIR
ALU or BSW	
ALU or MIR	
BSW or EXT	
BSW or MIR	
EXT or MIR	
ALU or BSW or MIR	
BSW or EXT or MIR	
4 BIT CARRY	Carry out of 4th BIT
8 BIT CARRY	Carry out of 8th BIT
4 BIT CARRY or MIR	
8 BIT CARRY or MIR	

A fifth input to the B register is available and is very useful when working with BCD numbers or 8-bit characters. The carryout of the fourth bit or eighth bit of the ALU is complemented, duplicated, and shifted three places right and then loaded into the B register.

BIT	0	1	2	3	4	5	6	7
CARRY	C_0	-	-	-	C_4	-	-	-
BC4	0	0	$\bar{C}_0$	$\bar{C}_0$	0	0	$\bar{C}_4$	$\bar{C}_4$
BC8	0	0	$\bar{C}_0$	$\bar{C}_0$	0	0	0	0

The MIR is also usable as a general register but, because of its location, its use is not always straightforward. The MIR is loaded from the B-Bus and outputs either to the B register or an external memory or device. It should be noted that loading the MIR does not initiate an output operation.

9. SPECIAL REGISTERS

There are three registers available to the microprogrammer that have not yet been discussed. Their function is unique to each register and therefore each will be described separately below.

LITERAL (LIT) register is loadable only from the microprogram memory and provides a means of entering constants. The register is bits long and usable for generating mask, entering characters for use in a compare, or for entering a looping limit. A sample program that illustrates the use of the LIT is shown abelow. A string of characters is read until an A is found

```
SAVE                    % START OF LOOP
65 = LIT                % CHAR "A" INTO LIT
DR                      % READ A CHARACTER
IF RDC THEN BEX         % INPUT TO B REG
B EQV LIT =             % PERFORM EQV OPERATION
```

```
IF ABT THEN STEP ELSE JUMP % IF AN "A" IS FOUND THEN % GO ON.  IF NOT
                           % "A" THEN LOOP BACK.
```

COUNTER (CTR) register is sixteen bits long and is loadable from either the B-Bus or the LITERAL register. The primary function of the CTR is to provide a convenient means of controlling the number of times a loop will be executed. Data presented to the CRT is automatically complemented (ones complement) before it is loaded into the CRT. The contents of the CTR can be incremented by using the INC command and an overflow condition is testable with the COV condition.

```
LCTR; SAVE                      % LOAD COUNTER FROM LIT (3)
3 = LIT; COMP 8 = SAR           % LOAD 3 INTO LIT; 8 (LEFT) INTO SAR
LOOP:  DR; AL L = A1            % READ A CHAR; SHIFT A1 L 8 PLACES
IF RDC THEN BEX                 %
A1 or B = A1; INC               % PACK CHAR. INC CTR
IF COV THEN STEP ELSE RETURN    % READ FOUR CHAR.
```

SHIFT AMOUNT REGISTER (SAR) is a five bit register that is uniqe in that it is the only register that the microprogrammer can write into but not read from. The SAR is loadable from the B-Bus, the Microprogram Memory (using a Type II instruction) and the complement (Twos Complement) of itself. The SAR is used to hold the shift amount for the Barrel Switch. For right shifts the contents represent the actual number of bit positions that will be shifted. On left shifts the contents are the twos complement of the shift amount. A more complete description of the operation of the Barrel Switch is given in Section IV.2.

CHAPTER V

CONCLUSIONS

1. SUMMARY

All of the deficiencies that were noted in the Interpreter have been eliminated. The Unified Processor is more than just a rework of the Interpreter. It is instead a new design that used the Interpreter as a baseline design.

A gate level design of the UP has been completed and will be implemented using standard bipolar integrated circuits. This will provide a powerful laboratory tool for use in studying microprogramming techniques.

2. RECOMMENDATIONS

After the completion of the design of the Unified Processor it was noted that several areas could be changed to provide a more efficient machine. The major areas are discussed below.

a. Micromemory

The present UP used a sixty-four bit memory for its control store. It becomes obvious after writing several test programs that seldom are more than one-third of the control fields used for a particular instructor. The remaining two-thirds are in their "no-change" state. To make more efficient use of memory but yet not reduce flexibility or increase decoding complexity the memory could be divided in two. This would introduce a third instruction type and a complete study would have to be made to determine what controls each instruction had.

b. B Register

The B register should be removed and its function taken over by the A registers. The B input network would be put on the A register with the exception of the internal input. This source would be connected through a buffer to the tri-state Y-Bus. The O/1/T/F should be used on the A register output that goes to the X-Bus.

c. Barrel Switch

The Barrel Switch provides a fast means of shifting a parallel word but requires an enormous amount of circuitry. A better compromise between complexity and speed would be if only shifts in increment of four were available. This would reduce circuitry complexity by one-half. If a shift is desired that is not a multiple of four the Barrel Switch would be used to provide an initial shift and then a standard shift register would be used for the additional one, two, or three shifts. The maximum using this system would be four clock, one for the Barrel Switch and three for the shift register.

APPENDIX A

UNIFIED PROCESSOR I/O CHANNEL

All information transfers between the UP and the outside world are through a buffered parallel channel. A block diagram of the channel showing its double buffering is shown in Figure 28. The parallel data and address lines are attached to all devices/memories and the determination of who accepts the information is made by the interface.

When a memory operation is being performed the sixteen address bits directly define the particular word to be worked with. For a device operation there is no need for an address so the sixteen lines can be used as control bits to define the particular function to be performed. Input and output operations are very similar as the timing diagrams in Figures 29 and 30 show. The main difference is that for an input operation only the B Buffer is loaded and during an output operation the MIR Buffer is loaded. The only indication an interface has that an operation is an input or output is from the read/write line.

This channel is used only for programmed, controlled data transfers and can operate at a maximum of 2,500,000 words/second.

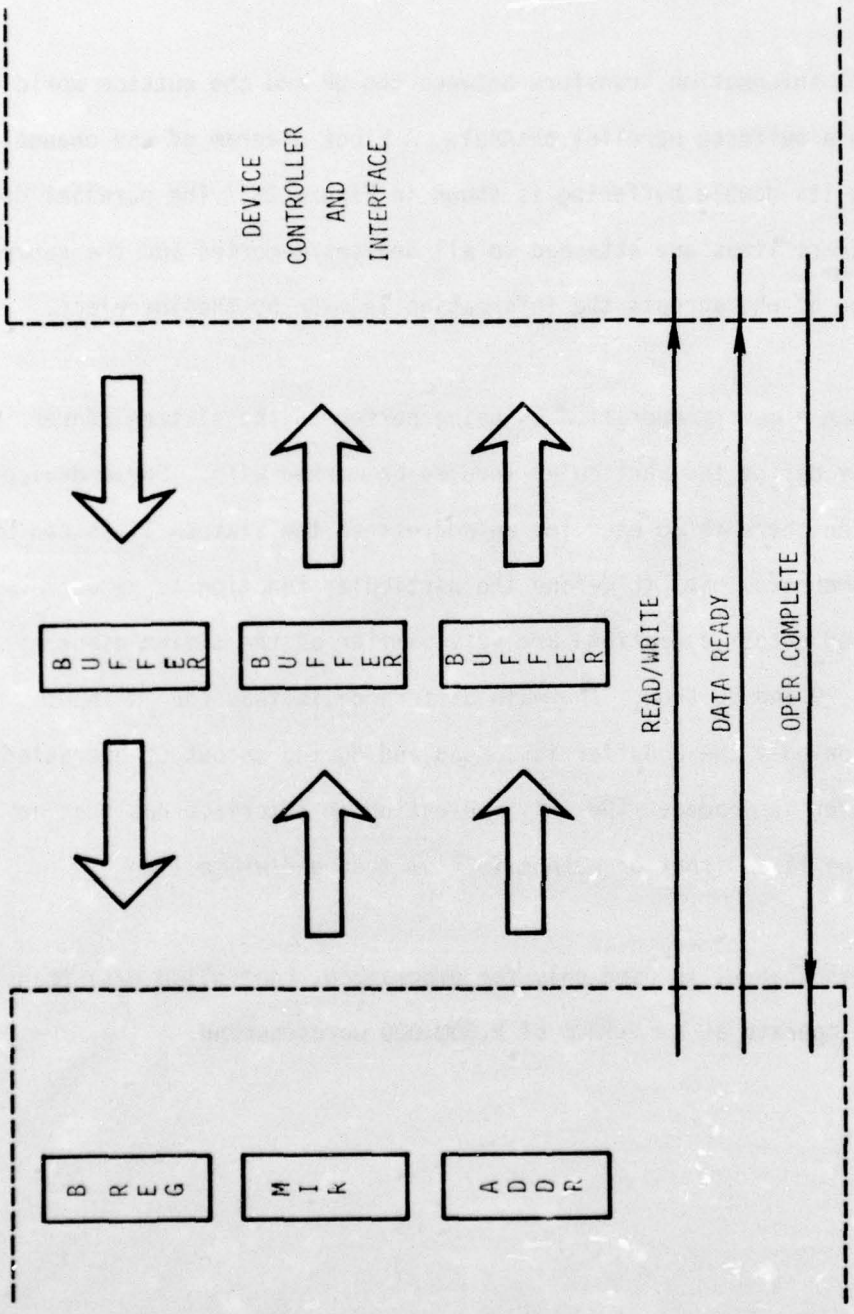
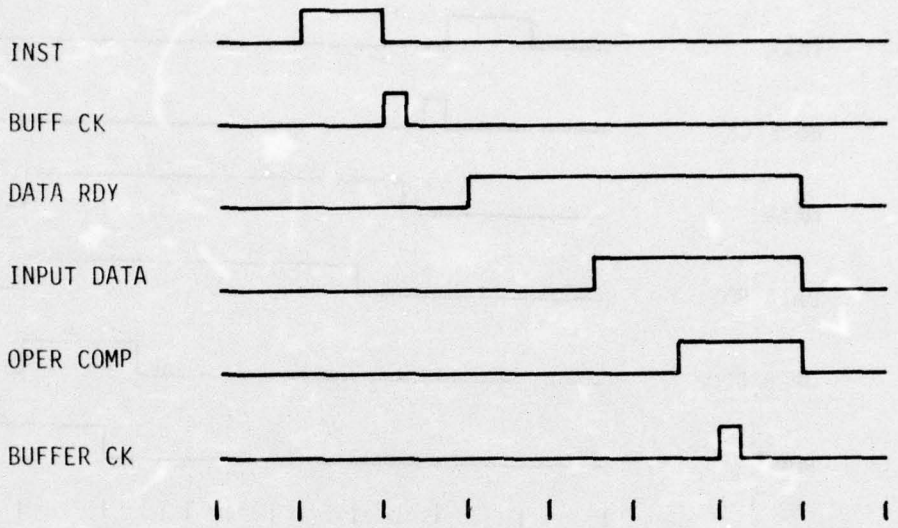


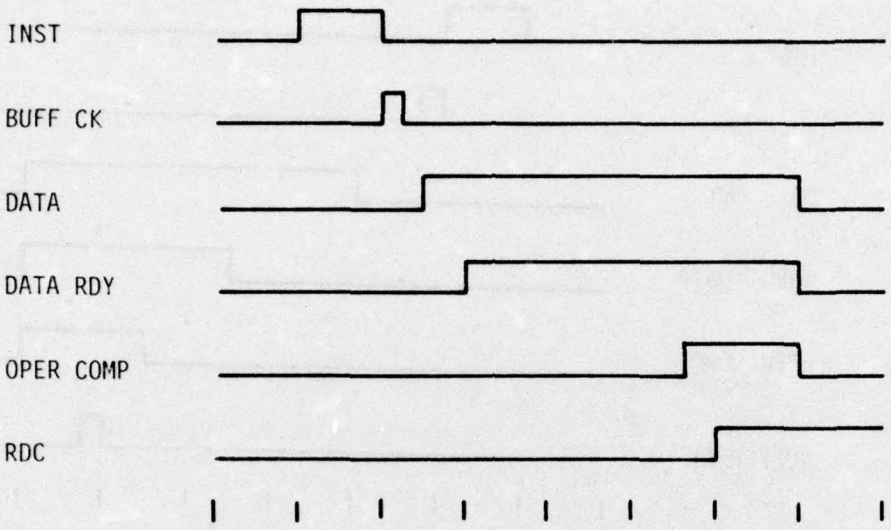
Figure 28. I/O Channel Block Diagram



NOTES:

1. Buffer clock is only for Address Buffer.
2. Address is stable on output lines approximately fifty nanoseconds prior to DATA RDY goint TRUE.
3. DATA RDY will remain true until OPER COMP goes TRUE.
4. INPUT DATA must be stable on input lines fifty nanoseconds prior to OPER COMP going TRUE.
5. OPER COMP must remain TRUE until DATA RDY goes false and it must then also return to false.

Figure 29. Input Data Transfer Timing Diagram



NOTES:

1. Data is stable on output lines approximately fifty nanoseconds prior to DATA RDY going TRUE.
2. DATA RDY will remain TRUE until the OPER COMP lines goes true.
3. The OPER COMP line must return to FALSE when DATA RDY goes FALSE.
4. The RDC is set by the interface and will remain that WAY UNTIL RESET BY THE MICROPROGRAM.

Figure 30. Output Data Transfers Timing Diagram

APPENDIX B

UNIFIED PROCESSOR'S MICROMEMORY IMPLEMENTATION

The memory used to store the control signals for the UP is 4096 words of sixty-four bits each. A new word is assessed each clock period, which is 100 nanoseconds; therefore, the memory has an access time less than fifty nanoseconds. To obtain a memory of this size and speed it was necessary to go to integrated circuit memory chips. The original design by Burroughs Corp. used a Fairchild 93410 which is one bit by 256 words. The chip is TTL compatible and has a maximum access time of forty-five nanoseconds but it would require 1024 of these chips. The micromemory would have taken more volume and power than the remainder of the processor.

To achieve a memory with higher density, lower power consumption, and reasonable cost without sacrificing speed it was necessary to go to the MOS family of ICs. A new pseudo static MOS chip, the AMS 7011, proved to have the desired features. The 7001 has a maximum access time of fifty nanoseconds and each chip is one bit by 1024 words; therefore, only 256 chips are necessary for the entire memory.

The structure of the memory is shown in Figure 31. During normal execution the MPCR provides a new address every clock. The new control word is read from the Random Access Memory (RAM) and applied to the 64 control lines going to the UP. There are times when it is desirable to change the microprograms, possibly when emulating a new machine. To accomplish this, two input sources are available to the programmer.

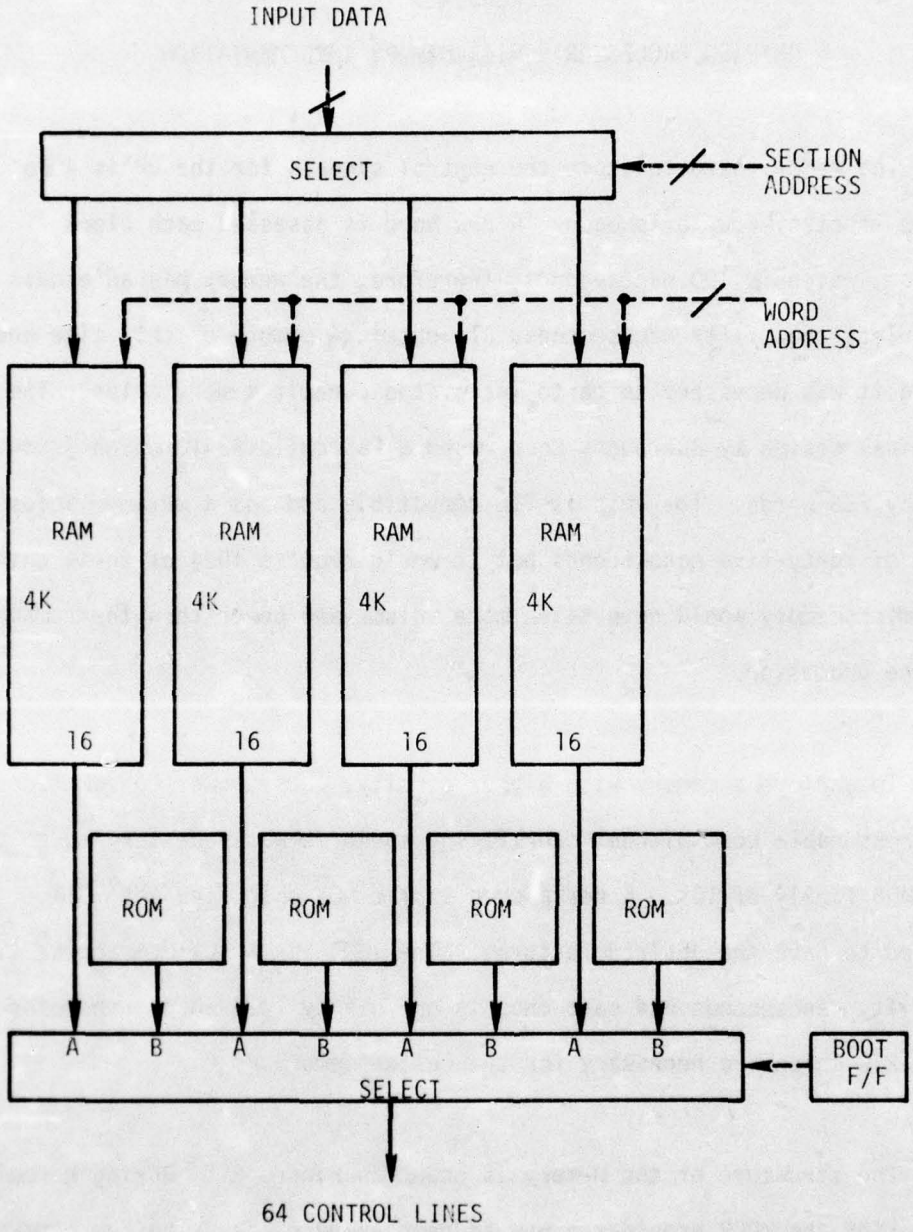
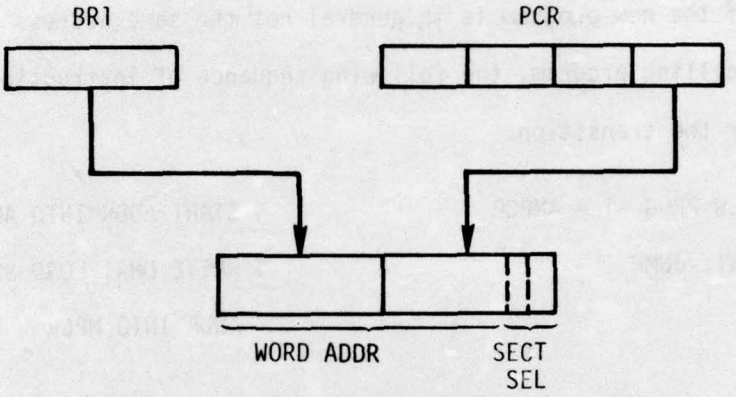


Figure 31. UP Micromemory

The first source is the least significant sixteen bits of the MIR. To load the memory the programmer put the desired 16-bit control word in the MIR and the section and word address in the BR1/PCR. The form the address takes in these registers is shown below.



After the data and address have been loaded an LDM instruction is performed. This causes the UP to stop for one clock while the memory address lines are switched from the MPCR to the BR1/PCR. The word is then written into memory and the control returned to the UP.

The second data source is from a floppy disk that is used to store microprograms. When the programmer desires to load a program that is on the floppy disk he must perform the following sequence of operations:

a. Check status of floppy disk.
b. Write to disk the starting microprogram address. (NOTE: This must be in the same form as the BR1/PCR address described previously.)
c. Write to disk the starting disk address.
d. Write to disk a DMA command.

After these four operations are performed the clock to the UP will be inhibited and a DMA transfer from the disk to micromemory will take place. When the transfer is complete the clock will be enabled and execution will continue at the address in the MPCR. Since the entry point of the new program is in general not the same address as the exit of the calling program, the following sequence of instruction is necessary for the transition.

```
NEW PROG -1 = AMPCR          % START ADDR INTO AMPCR
DW1; JUMP                    % WRITE DMA; LOAD STARTING
                               ADDR INTO MPCR
```

BIBLIOGRAPHY

Barrera, R., The Interpreter, Wright-Patterson Air Force Base, Ohio: Air Force Avionics Laboratory, May 1972.

Bartee, Thomas C., et al, Theory and Design of Digital Machines, New York, New York: McGraw-Hill Book Company, 1962.

Bell, C. Gordon, and Allen Newell, Computer Structures: Readings and Examples, New York: McGraw-Hill Book Company, 1971.

Bingham, H. W., et al, Microprogramming Manual for Interpreter Based Systems, Paoli, Pennsylvania: Burroughs Corporation, November 1970.

Chu, Yaohan, Computer Organization and Microprogramming, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1972.

Davis, R. L., The Interpreter, Paoli, Pennsylvania: Burroughs Corporation, February 1970.

Davis, R. L., et al, Aerospace Multiprocessor, Wright-Patterson AFB, Ohio: Air Force Avionics Laboratory, June 1973.

Donovan, John J., Systems Programming, New York, New York: McGraw-Hill Book Company, 1972.

Godfrey, R.D., A Fortran Microprogram Translator, Thesis Wright-Patterson AFB, Ohio: Air Force Institute of Technology, June 1972.

Hill, Frederick J., and Gerald R. Petterson, Introduction to Switching Theory and Logical Design, New York: John Wiley & Sons, Inc., 1968.

Husson, S., Microprogramming Principles and Practices, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1970.

Kohavi, Zvi, Switching and Finite Automata Theory, New York, New York: McGraw-Hill Book Company, 1970.

Stone, Harold S., Introduction to Computer Organization and Data Structures, New York, New York: McGraw-Hill Book Company 1972.

☆ U. S. GOVERNMENT PRINTING OFFICE: 1976 — 657-630/1043