CALIFORNIA STATE UNIVERSITY, NORTHRIDGE

NOVA CPU IMPLEMENTATION WITH 2901 BIT SLICE

A project report submitted in partial satisfaction of the requirements for the degree of Master of Science in

Engineering

by

Larry Wayne Abbott

January, 1980

The Project Report of Larry Wayne Abbott is approved:

C. V. Metzler

R. Pettit

R. A. Davidson

California State University, Northridge

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ABSTRACT

NOVA CPU IMPLEMENTATION WITH 2901 BIT SLICE

by

Larry Wayne Abbott

Master of Science in Engineering

There are several methods which can be used in the design of a digital computer. Each of these approaches has its advantages and its disadvantages. To learn the trade-offs that apply to the bit slice and microprogram methods, a partial build up of a NOVA CPU was done. In the build up, special attention was focused on the sequencing and control of the CPU. The Project Report presents the outcome of the hardware build up and, in particular, it addresses the issues involved in microcode sequencing and decoding. Two methods of sequencing and decoding are presented in detail. One method relies on firmware to do all the sequencing and mode decoding, such as address modes. The other method relies on firmware and the Mapping PROM to do the sequencing and mode decoding. This project Report investigates the implications of both methods on speed and memory requirements for the CPU. Finally, this Project Report presents technology trends, and investigates the potential use of bit slice technology in future systems.

1.0 CHOICE OF DESIGN APPROACH

The goal of this paper is to investigate the speed, firmware, and organizational requirements for a high performance minicomputer design. To fulfill this goal a computer central processing unit was constructed within the constraints of the time and money available to me.

Several design criteria had to be considered, balancing time and money available against firmware and hardware goals. The results of this trade-off are as follows:

1. The bit slice approach was chosen. This approach gives ease of interfacing the various elements of the computer. Bit slice fabrication technology is also capable of providing the speed necessary for a high performance minicomputer.
2. The instruction set chosen was an emulation of the Data General NOVA 1200 set. It is relatively easy to implement, offers adequate power, and has a large expanding software base. In addition, at least two software compatible microprocessors exist, the Fairchild 9440 and the Data General MN601.
3. Only representative instructions would be microprogrammed because of the large amount of time required to microprogram an instruction.

4. Certain sections of the CPU would not be completely built, and other sections would not be built at all because all the information wanted could be learned without a full implementation. For example, only one of the four Register and Arithmetic Logic Units (RALU) is used because only one is needed to verify proper RALU operation under the control of the computer control unit. In addition, the RALU is an expensive element, and any reduction in the number of units used reduces the cost greatly.
5. The sequencer is the heart of the Computer Control Unit (CCU), and the CCU is the heart of the CPU; therefore the sequencer and the other parts of the CCU (such as the microstore and the pipeline register) must be fully implemented and checked out.

Several approaches may be taken when designing a digital computer. These design approaches center around the technique used in the CCU to control various phases of the computer operation. The main techniques used in the control of the computer are ring counters, random logic, and microprogramming. In this paper the computer was designed around a microprogrammed CCU. In conjunction with the technique of microprogramming, a pipelined architecture was adopted to increase the computer speed.

1.1 HARDWARE ADVANTAGES AND DISADVANTAGES

The choice of the hardware and the architecture can make the difference between a successful and a disastrous design. Since a computer built up from SSI, MSI, and LSI components is much more expensive to build in terms of both time and hardware costs than the ubiquitous LSI microprocessor, it is imperative that such a computer have appreciably higher performance and flexibility than the LSI microprocessor. A typical computer built with bit slice techniques would require between fifty and one hundred integrated circuits just for its CPU. The cost of components for such a bit slice CPU starts at five hundred dollars, as opposed to ten dollars for the LSI microprocessor. It becomes obvious that high hardware costs for a bit slice computer are a definite disadvantage and that there must be performance gains to offset this disadvantage if the bit slice approach is to be used. This of course assumes that performance is needed in the first place.

Can the bit slice approach provide the necessary performance? One aspect of performance is the speed of the technology being used. As can be seen from figure 1.1 a bit slice computer using a bipolar technology such as Schottky, low powered Schottky, or ECL would provide the kind of speed that is necessary.

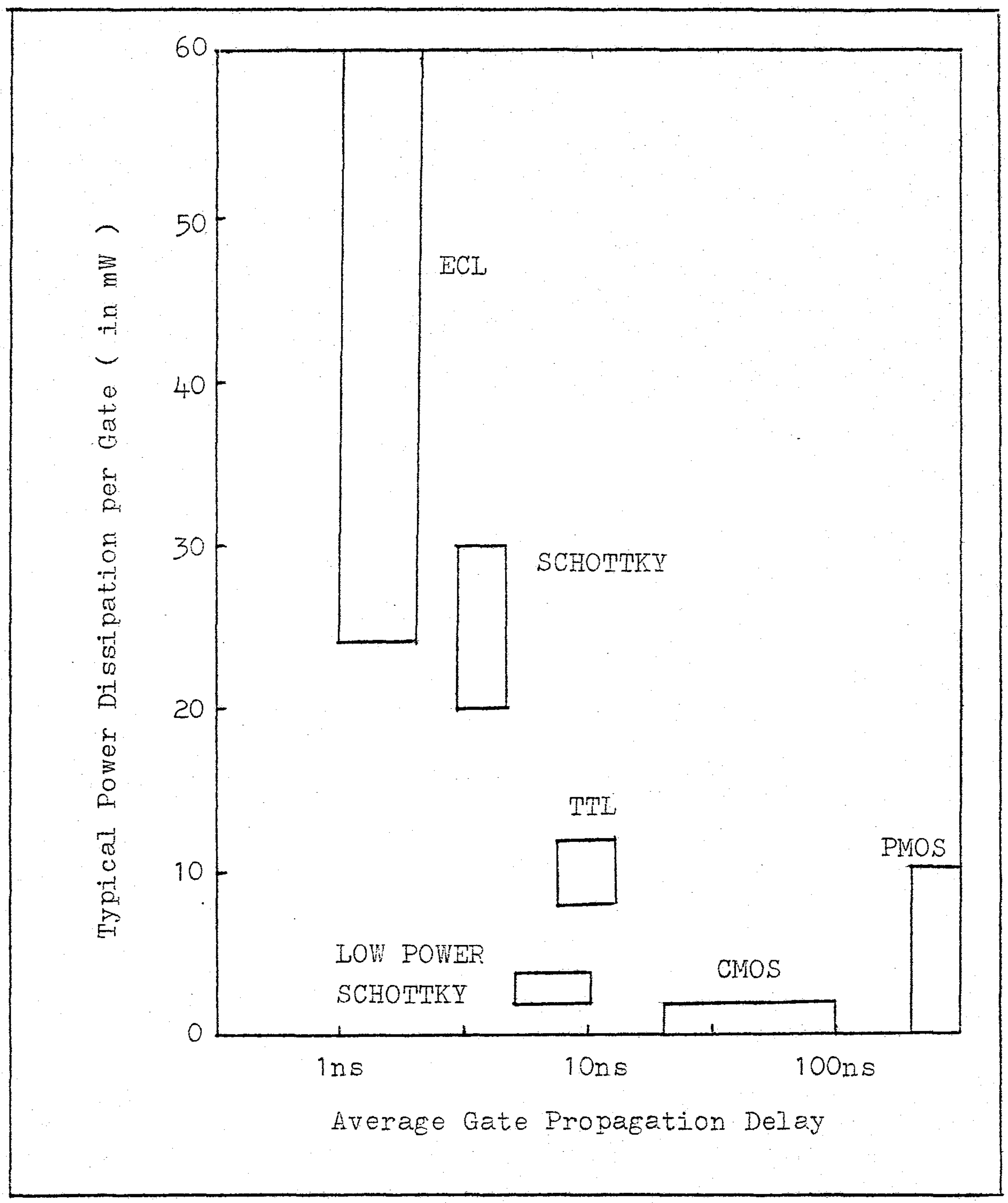


FIGURE 1.1 Comparison of gate delay verus power consumption for various technologies. Compiled from data books of AMD, Fairchild, and Motorola.

Picking the right integrated circuit technology is obviously no insurance that high performance will be achieved. The architecture of the computer is also of great importance. A well designed architecture will simultaneously perform as many computer operations as possible. This concurrency is achieved by using a pipelined microprogram architecture. In this type of architecture, a wide microprogram word, usually from 40 to 60 bits wide, is sent to an equally wide pipeline register. This technique allows one microinstruction, the one in the pipeline register, to be executed while another microinstruction is fetched from the microprogram memory (microstore).

Additional performance is gained from the width of the microinstruction. A wide microinstruction can command many actions at the same time, increasing the apparent speed of the computer.

A microprogram approach provides another advantage. Flexibility is a major strength of microprogramming. If it is necessary to add or change instructions, the microprogram can be easily changed. Most instruction sets have many instructions in common, so it may not be necessary to change all the microcode. It may be possible to simply change the addresses in the Mapping PROM for many of the instructions. So microprogramming makes the design extremely flexible.

Since the bit slice design has the form of an iterative array that can be expanded by adding more cells, the bit slice approach allows easy expansion of the address bus and the data bus while allowing the rest of the CCU to remain substantially the same. The expansion ability gives the bit slice approach flexibility through modularity.

The advantages and disadvantages of the concepts introduced for the bit slice approach to computer design are summarized in table 1.1.

TABLE 1.1

BIT SLICE COMPARISONS	
ADVANTAGES	DISADVANTAGES
SPEED: Bipolar technology and pipelining FLEXIBILITY: Modularity and micro-programming TIME: Reduced hard-ware design due to LSI	COST: Time to microprogram

1.2 FIRMWARE ADVANTAGES AND DISADVANTAGES

The major advantage of firmware is the flexibility gained from the microprogramming technique. The disadvantage is the large amount of time it takes to write microcode. The Advanced Micro Devices literature (AMD, 1977) describing the System 29 microprogramming development system gives the following time and cost for microprogramming. For manual microprogramming "one word of microcode per day is allowed on U.S. Government contracts. Three to five words of microcode per day appears to be a reasonable standard on commercial projects ....". This means that a 1000 word microprogram would take one man-year to accomplish, and even using the System 29 development system, it would take half a man-year to develop 1000 words of microcode. It is clear that the cost of microprogramming is a disadvantage. It is also evident why only representative instructions were microcoded for this project. In essence, the ease of hardware design comes at the expense of higher firmware cost.

2.0 SELECTION OF THE PROPER BIT SLICE FAMILY

After the decision has been made to design the computer using the combined techniques of bit slicing, pipelining, and microprogramming, there is the problem of selecting which of the bit slice families to use. One important criterion in selecting a device for a design is availability. This involves more than finding out whether or not the device is in stock. Availability involves consideration of such questions as whether the family is available from more than one distributor, and whether it is available at a competitive price with good delivery time. Without the purchasing power of a company, availability takes on a new dimension. Distributors are not eager to deal with an individual, especially in the small quantities required for a one of a kind graduate project. This latter consideration made the only practical choice of a bit slice the 2900 series. This choice, however, is a good one even under the normal commercial meaning of avialability, as shown below.

The available bit slices are shown in table 2.1. From the table one could pick out the reasons that certain bit slice families were not chosen. The following sections are presented however, to make it clear why certain families were not chosen for this project. This is not to say that these families are not well designed; actually,

TABLE 2.1

BIT SLICE FAMILIES

Company	Series	Process technology	ALU part number	ALU word size (bits)	Number of ALU instructions	Can ALU do BCD arithmetic	Maximum ALU clock rate (MHz)	General-purpose registers in ALU	ALU package size (DIP pins)	Microprogram sequencer number	Number of address bits	Maximum sequencer clock rate (MHz)	Number of sequencer commands	Sequencer stack size	Sequence package size (DIP pins)	Are parts TTL compatible	Voltages required (V)	Prototyping system available	Development software available	Specialized support circuits available	Comments
Advanced Micro Devices	2900	STTL	2901A	4	16	No	10	16	40	2909/11	4	10	12	4×4	28/20	Yes	5	Yes	Yes	Yes	Has widest number of second sources
		STTL	2903	4	25	No	10	16	48												ALU has nine more instructions than 2901, including multiply and divide.
Fairchild	Macrologic	STTL/ CMOS	9405/34705	4	64	No	10	8	24	9406	4	10	4	16×4	24	Yes	5	Yes	Yes	Yes	CMOS version (34705) operates at 2 MHz
	100k 8-bit	ECL	ADIU	8	27	Yes	20	1	*	*	*	*	*	*	*	*	−4.5, −2	1978	1978	1978	Only 8-bit slice
Intel	3000	STTL	3002	2	40	No	10	11	28	3001	9	10+	11	0	40	Yes	5	Yes	Yes	Yes	Only 2 bit ALU available
Monolithic Memories	5700/6700	STTL	57/6701	4	32	No	5	16	40	6710	9	10+	8	0	40	Yes	5	Yes	?	No	Has double-addressing capability
Motorola	10800	ECL	10800	4	105+	Yes	20	0	48	10801	4	20	16	4×4	48	No	−2, −5.2	Yes	Yes	Yes	Fastest bit slice available
National Semiconductor	IMP-4	PMOS	00A/520	4	8	No	5.714	20	24	4A/521	4	5.714	100+	in ALU	24	Yes	+5, −12,	Yes	Yes	No	Need external register file
	IMP-8	PMOS	08A/520	4	8	No	5.714	20	24	8A/521	8	5.714	100+	in ALU	24	Yes	+5, −12	Yes	Yes	No	Uses IMP-4 ALUs with big ROM
	IMP-16	PMOS	00A/520	4	8	No	5.714	20	24	16A/521	16	5.714	100+	in ALU	24	Yes	+5, −12	Yes	Yes	No	Two development systems available
Texas Instruments	SBP-0400A	I^2L	SBP 0400	4	512	No	5	10	40	74S482	4	20	64	4×4	20	Yes	Current	Yes	No	No	Has pipeline register
	SBP-0401A	I^2L	SBP 0401	4	512	No	5	10	40	74S482	4	20	64	4×4	20	Yes	Current	Yes	No	No	Does not have pipeline register
	74S481	STTL	74S481	4	24,725	No	10	0	48	74S482	4	20	64	4×4	20	Yes	5	Yes	No	Yes	Very flexible instruction set

Source: Electronic Design, 1977, page 60 some of the families are better suited than the 2900 for some applications.

2.1 THE IMP BIT SLICE

Without going into too much detail, the decision not to use the IMP was based on the technology and not on the functional design of the IMP chips. First, the IMP is implemented in PMOS which means it is slow, too slow for this application. Additionally, it uses multiple power supplies, requires TTL level shifters to be TTL compatible, and is not second sourced. There the IMP was not thought to be suitable for this application.

2.2 THE MOTOROLA 10800 FAMILY

The 10800, which uses ECL technology, is another family that was not chosen because of the technology. In this case the family is fast enough; in fact, it is too fast. The speed is accompanied by high power consumption, small (800 millivolt) logic swings, and by noise created by the fast switching speeds. Further, the 10800 is not TTL compatible because of the 800 millivolt logic swing. As a result of all these disavantages the 10800 family was not chosen. ECL is the type of technology that is more appropriate for high performance mainframe computers.

2.3 THE TEXAS INSTRUMENTS FAMILIES

The two families considered from Texas Instruments were implemented with Schottky and integrated injection logic (I^2L). The SN74S481 was the Schottky implementation, and the SBP0400 and SBP0401 were the I^2L implementation. Neither of the Texas Instruments families were chosen, however, each family was rejected for different reasons. In both cases the software support is practically non-existent, and, as has been pointed out earlier, microcoding is time consuming and needs to be done on a micro-programming development system for commercial applications.

The SBP0400 and SBP0401 were just too slow to be used. In fact the shortest microinstruction time was 350 ns and the maximum clock frequency was 3.3 MHz. Single chip 16 bit microprocessors can do as well.

The SN74S481 is an extremely fast (67ns) and versatile integrated circuit, but it does not fit into the architecture of the computer being designed. If the SN74S481 were used in this architecture its system speed would be much less than the 67ns the instruction time indicates. This paradox comes about because of the NOVA architecture. A NOVA uses four accumulators in the CPU for working registers; the SN74S481, on the other hand, uses 16 working registers in the main memory. To obtain four accumulators, the SN74S481 must locate them in main memory, and unless one is willing to accept the higher price and complexity of high speed cache memory one must settle for the more realistic 300 ns memory speed. This means that the cycle time of the computer designed with the SN74S481 is based on 300 ns cycle times and not on the 67 ns of the basic SN74S481. With cycle times approaching 367 ns the SN74S481 would appear no better than the SBP0400. The SN74S481 was not chosen because of this reason.

2.4 THE INTEL 3000

The Intel 3000 series has two problems associated with it. First, the slice is only two bits wide, which means that it would require twice as many chips as a four bit slice to accomplish the same job. Secondly, the sequencer (3001) addresses only 512 words of microprogram memory; however what is really difficult to live with is the fact that the sequencer can not go from any location in the microprogram store to any arbitrary location. The addressing scheme divides the microprogram store into rows and columns. The sequencer can only jump to locations in the row or column of the originating microinstruction. This was felt to be an unnecessary restriction, and, as a result, the Intel 3000 was not chosen.

2.5 FAIRCHILD MACROLOGIC 9405

The Fairchild Macrologic family comes in two versions, Schottky and CMOS. The CMOS version is too slow for this project; however, the Schottky version is quite good. The Schottky version, the 9405, is not quite as complex as the 2901; for instance, it does not offer the two port RAM so the RALU can not write and read the RAM at the same time or read two RAM locations at the same time. As a consequence, the 9405 comes in a smaller 24 pin package and costs less than the 2901 ($12.00 versus $14.70 in 100 quantity). The complexity of the 2901 allows more to be done in a microinstruction; however, it does not have an edge in chip count over the 9405. The 9405 does not seem to have as many support chips as the 2901, and in this area some applications may give the edge in chip count to the 2901. On the other hand, both groups of support chips seem to be well thought out, so a determination would have to wait for a preliminary design with both families.

On a technical basis the tradeoffs between the 9400 series and the 2900 series would make the choice a difficult one; in fact, for this application Fairchild has implemented a NOVA 3 emulation with the 9400 series. Even on a commercial availability basis the 9400 is acceptable. While it is true that the 2900 has more second sources than the 9400, the fact remains the 9400 is second sourced.

Therefore, the only reason for making the choice of the 2900 was the lack of availability of the 9400 through the low volume distributors that an individual must deal with.

2.6 MONOLITHIC MEMORIES 6700

The Monolithic Memories 6700 appears to be the forerunner of the 2900 series. As consequence, there is nothing the 6700 does that the 2900 cannot do better or faster. Even the pinouts are similar so it makes little sense to pick the 6700 series.

2.7 ADVANCED MICRO DEVICES 2900 SERIES

There were many obvious reasons for selecting the 2900 series such as availability (the 2900 has many sources, including AMD, Fairchild, Monolithic Memories, Motorola, National, Raytheon, Sescosem, and Signetics), single power supply, TTL compatibility, and Schottky speeds. However, these are the simple and obvious advantages. The important advantages are less obvious and more complicated.

The sequencer provides several of these advantages. With the 2909 or 2911 sequencers, any address can be reached from any other address in the microstore. The next address can also be reached via a four word microinstruction stack. The 2911 is discussed in greater detail in the section on hardware implementation, along with the other components of the 2900 chip set.

Another strong point of the 2900 series is the two port RAM in the 2901 RALU. With a two port RAM several RAM operations can occur during one microcycle. For instance, the contents of register A14 can be added to the contents of register B3 and loaded back into B3 with a left shift, all in one microcycle. The resulting throughput of the machine is much greater than its clock rate would indicate.

The large selection of support chips, such as the AM2930 Program Control Unit (PCU) makes the 2900 an especially powerful set from a total system point of view.

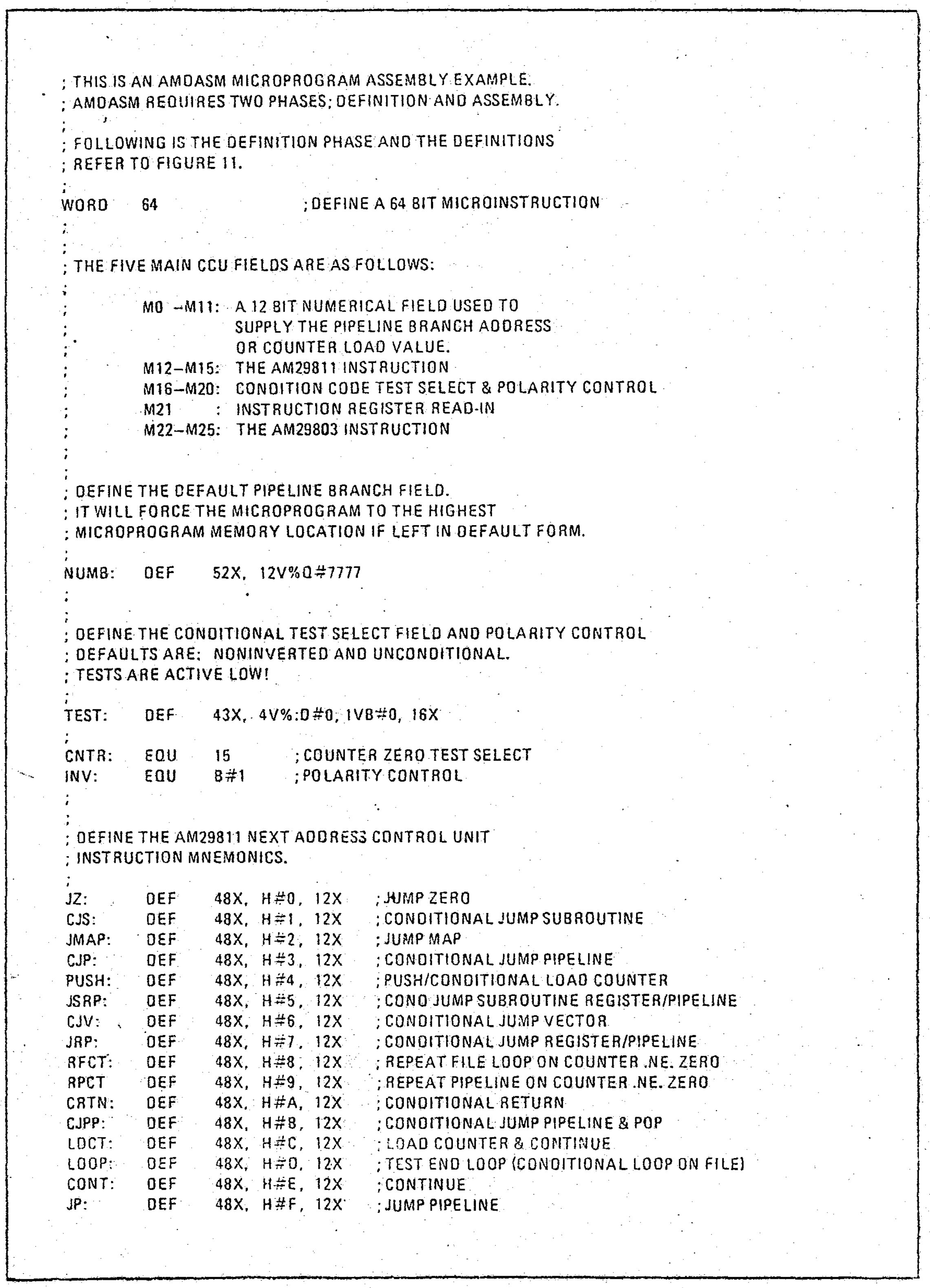
All the preceding elements are more throughly discussed in the section on hardware implementation. However, an area that is not discussed in detail elsewhere in the paper but is of upmost importance is the microprogram development system available for the 2900 series. It is not the only system available (see table 2.1 on bit slice families), but it is coupled with what is perhaps the best of all the bit slices.

Several versions of the development software are available. The first version is the AMDASM microcode assembler, which is available on national time sharing. Later developments are the System 29 and its implementation via floppy disk on the Intel 8080 development system. Since the System 29 runs under the control of an Am9080, its software is compatible with other hardware systems that use the 8080 or its derivatives.

The Advanced Micro Devices Microprogramming Handbook (AMD,1976a) contains the example shown as figure 2.1 of the use of the AMDASM microcode assembler with AMD's CCU design (figure 11 in the Microprogramming Handbook). Additional information can be obtained from Advanced Micro Devices or Raytheon in handbooks describing AMDASM or RAYASM in more detail.

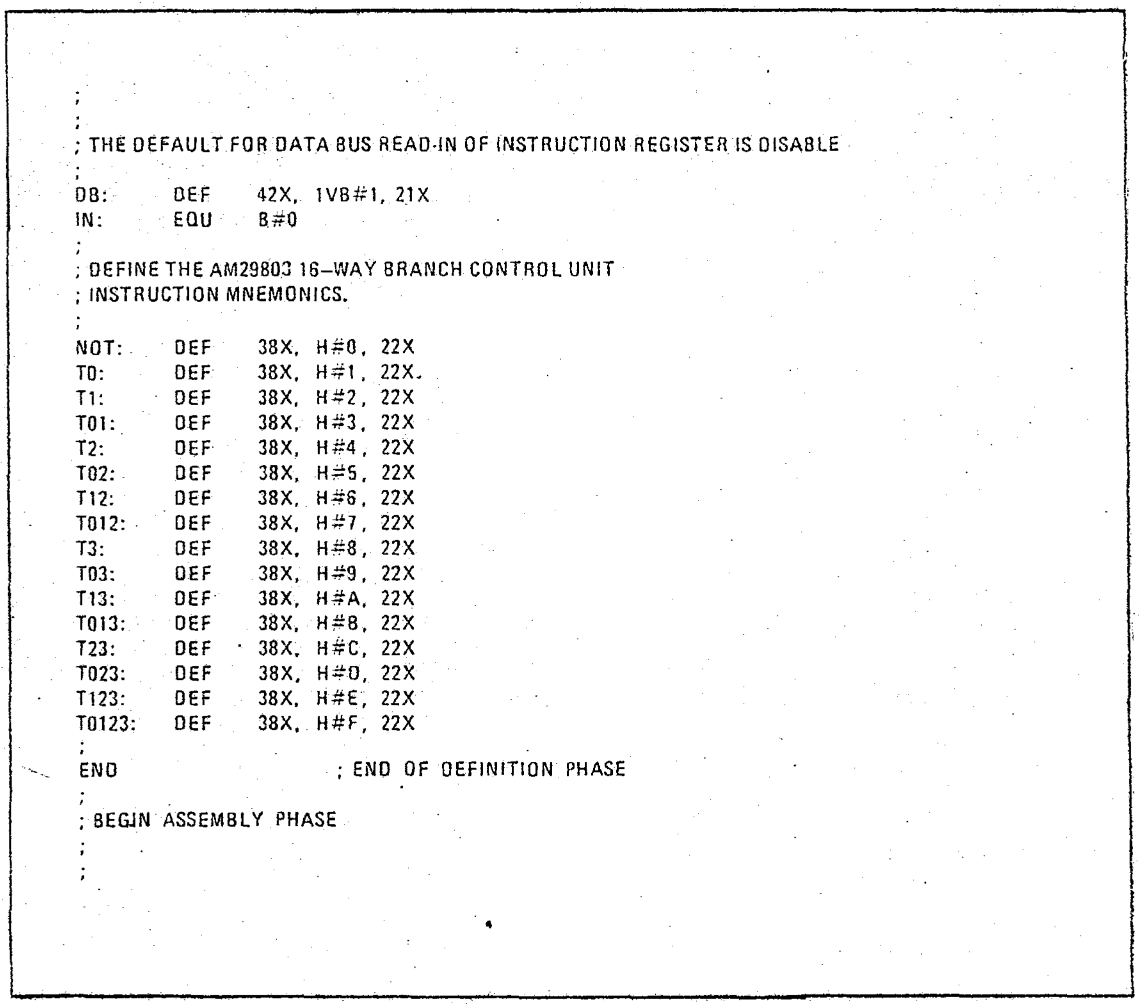
Figure 2.1 contains several microprogramming examples done with AMDASM. The examples assume AMD's design for a CCU (AMD,1976a). It should be noted that there is a great deal of similarity between AMD's CCU design and the CCU designed for this paper.

AMD's CCU uses 26 of the bits in the 64 bit wide microinstruction word. Table 2.2 describes the 26 bits and their functions by dividing the microinstruction into five fields.



```
; THIS IS AN AMDASM MICROPROGRAM ASSEMBLY EXAMPLE.
; AMDASM REQUIRES TWO PHASES; DEFINITION AND ASSEMBLY.
;
; FOLLOWING IS THE DEFINITION PHASE AND THE DEFINITIONS
; REFER TO FIGURE 11.
;
WORD     64                    ;DEFINE A 64 BIT MICROINSTRUCTION
;
;
; THE FIVE MAIN CCU FIELDS ARE AS FOLLOWS:
;
;           M0 –M11:  A 12 BIT NUMERICAL FIELD USED TO
;                     SUPPLY THE PIPELINE BRANCH ADDRESS
;                     OR COUNTER LOAD VALUE.
;           M12–M15:  THE AM29811 INSTRUCTION
;           M16–M20:  CONDITION CODE TEST SELECT & POLARITY CONTROL
;           M21    :  INSTRUCTION REGISTER READ-IN
;           M22–M25:  THE AM29803 INSTRUCTION
;
;
; DEFINE THE DEFAULT PIPELINE BRANCH FIELD.
; IT WILL FORCE THE MICROPROGRAM TO THE HIGHEST
; MICROPROGRAM MEMORY LOCATION IF LEFT IN DEFAULT FORM.
;
NUMB:    DEF     52X, 12V%Q#7777
;
;
; DEFINE THE CONDITIONAL TEST SELECT FIELD AND POLARITY CONTROL
; DEFAULTS ARE: NONINVERTED AND UNCONDITIONAL.
; TESTS ARE ACTIVE LOW!
;
TEST:    DEF     43X, 4V%:D#0, 1VB#0, 16X
;
CNTR:    EQU     15        ;COUNTER ZERO TEST SELECT
INV:     EQU     B#1       ;POLARITY CONTROL
;
;
; DEFINE THE AM29811 NEXT ADDRESS CONTROL UNIT
; INSTRUCTION MNEMONICS.
;
JZ:      DEF     48X, H#0, 12X    ;JUMP ZERO
CJS:     DEF     48X, H#1, 12X    ;CONDITIONAL JUMP SUBROUTINE
JMAP:    DEF     48X, H#2, 12X    ;JUMP MAP
CJP:     DEF     48X, H#3, 12X    ;CONDITIONAL JUMP PIPELINE
PUSH:    DEF     48X, H#4, 12X    ;PUSH/CONDITIONAL LOAD COUNTER
JSRP:    DEF     48X, H#5, 12X    ;COND JUMP SUBROUTINE REGISTER/PIPELINE
CJV:     DEF     48X, H#6, 12X    ;CONDITIONAL JUMP VECTOR
JRP:     DEF     48X, H#7, 12X    ;CONDITIONAL JUMP REGISTER/PIPELINE
RFCT:    DEF     48X, H#8, 12X    ;REPEAT FILE LOOP ON COUNTER .NE. ZERO
RPCT     DEF     48X, H#9, 12X    ;REPEAT PIPELINE ON COUNTER .NE. ZERO
CRTN:    DEF     48X, H#A, 12X    ;CONDITIONAL RETURN
CJPP:    DEF     48X, H#B, 12X    ;CONDITIONAL JUMP PIPELINE & POP
LDCT:    DEF     48X, H#C, 12X    ;LOAD COUNTER & CONTINUE
LOOP:    DEF     48X, H#D, 12X    ;TEST END LOOP (CONDITIONAL LOOP ON FILE)
CONT:    DEF     48X, H#E, 12X    ;CONTINUE
JP:      DEF     48X, H#F, 12X    ;JUMP PIPELINE
```

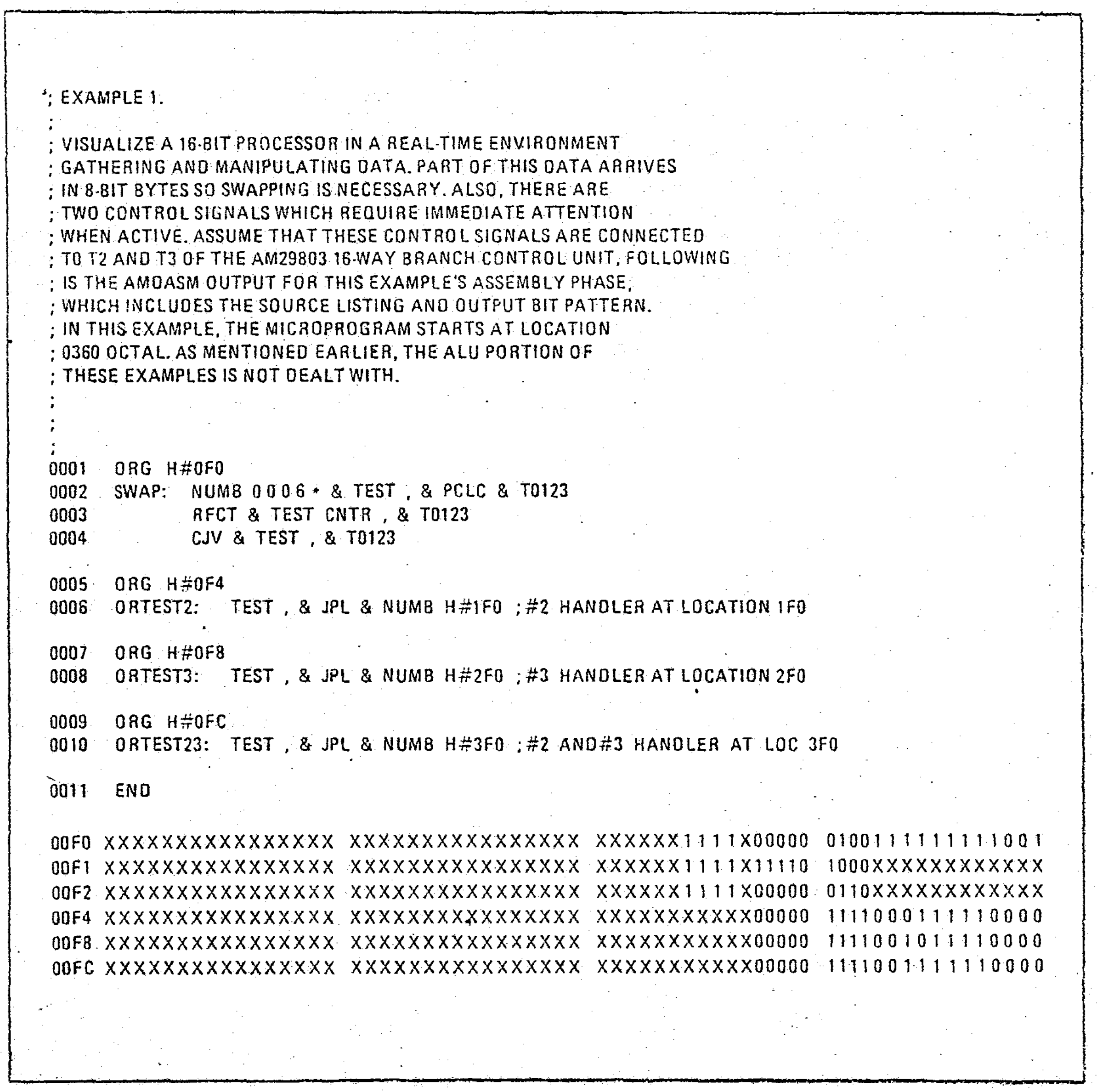
FIGURE 2.1 An example of AMDASM microcode assembly
From: AMD, 1976a page 1-16



```
;
;
; THE DEFAULT FOR DATA BUS READ-IN OF INSTRUCTION REGISTER IS DISABLE
;
DB:       DEF     42X,  1VB#1, 21X
IN:       EQU     B#0
;
; DEFINE THE AM29803 16-WAY BRANCH CONTROL UNIT
; INSTRUCTION MNEMONICS.
;
NOT:      DEF     38X,  H#0,  22X
T0:       DEF     38X,  H#1,  22X
T1:       DEF     38X,  H#2,  22X
T01:      DEF     38X,  H#3,  22X
T2:       DEF     38X,  H#4,  22X
T02:      DEF     38X,  H#5,  22X
T12:      DEF     38X,  H#6,  22X
T012:     DEF     38X,  H#7,  22X
T3:       DEF     38X,  H#8,  22X
T03:      DEF     38X,  H#9,  22X
T13:      DEF     38X,  H#A,  22X
T013:     DEF     38X,  H#B,  22X
T23:      DEF     38X,  H#C,  22X
T023:     DEF     38X,  H#D,  22X
T123:     DEF     38X,  H#E,  22X
T0123:    DEF     38X,  H#F,  22X
;
END                       ; END OF DEFINITION PHASE
;
; BEGIN ASSEMBLY PHASE
;
;
```

FIGURE 2.1 continued

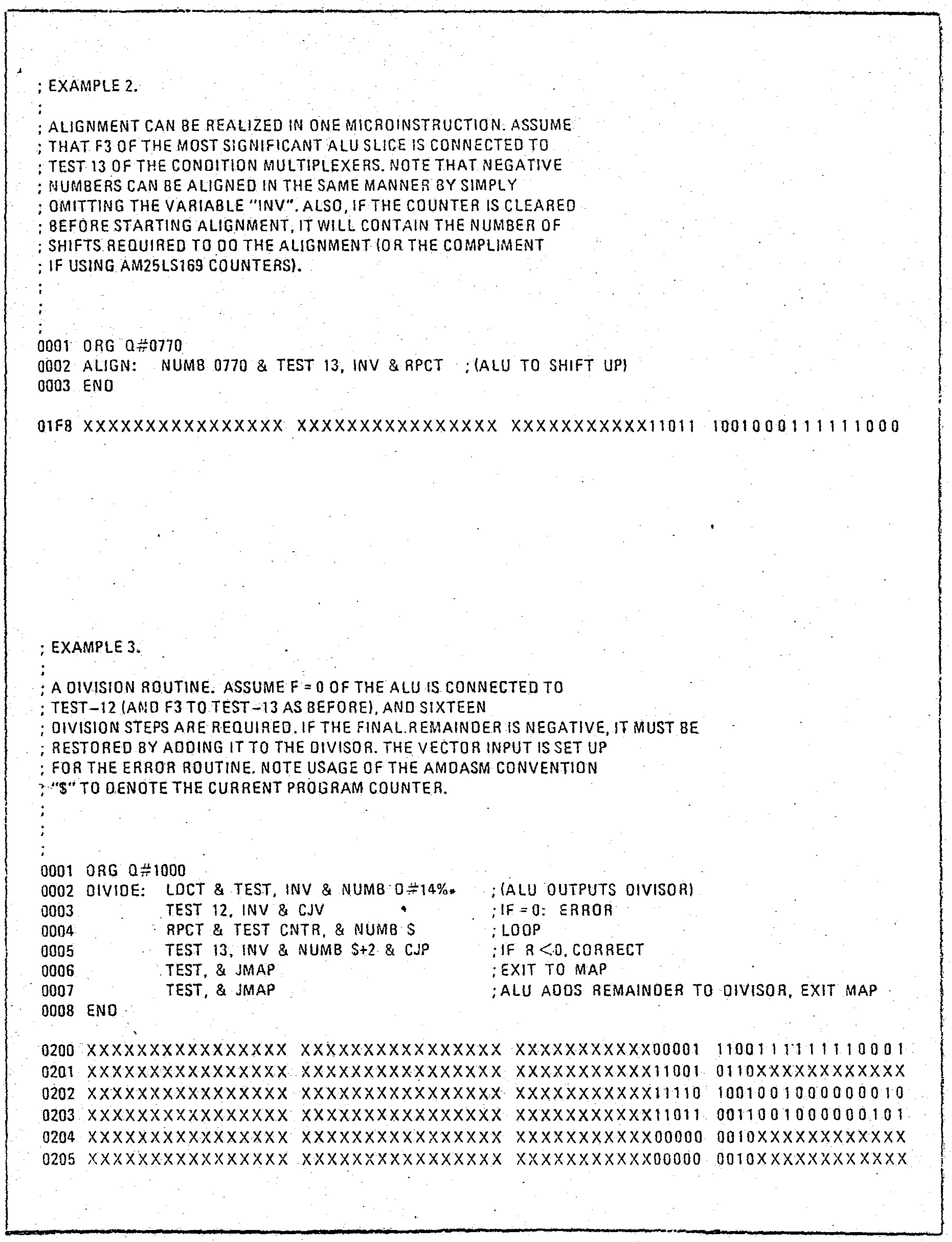
From: AMD, 1976a page 1-17



```
; EXAMPLE 1.
;
; VISUALIZE A 16-BIT PROCESSOR IN A REAL-TIME ENVIRONMENT
; GATHERING AND MANIPULATING DATA. PART OF THIS DATA ARRIVES
; IN 8-BIT BYTES SO SWAPPING IS NECESSARY. ALSO, THERE ARE
; TWO CONTROL SIGNALS WHICH REQUIRE IMMEDIATE ATTENTION
; WHEN ACTIVE. ASSUME THAT THESE CONTROL SIGNALS ARE CONNECTED
; TO T2 AND T3 OF THE AM29803 16-WAY BRANCH CONTROL UNIT. FOLLOWING
; IS THE AMDASM OUTPUT FOR THIS EXAMPLE'S ASSEMBLY PHASE,
; WHICH INCLUDES THE SOURCE LISTING AND OUTPUT BIT PATTERN.
; IN THIS EXAMPLE, THE MICROPROGRAM STARTS AT LOCATION
; 0360 OCTAL. AS MENTIONED EARLIER, THE ALU PORTION OF
; THESE EXAMPLES IS NOT DEALT WITH.
;
;
;
0001   ORG  H#0F0
0002   SWAP:   NUMB 0 0 0 6 * & TEST , & PCLC & T0123
0003           RFCT & TEST CNTR , & T0123
0004           CJV & TEST , & T0123

0005   ORG  H#0F4
0006   ORTEST2:    TEST , & JPL & NUMB H#1F0 ;#2 HANDLER AT LOCATION 1F0

0007   ORG  H#0F8
0008   ORTEST3:    TEST , & JPL & NUMB H#2F0 ;#3 HANDLER AT LOCATION 2F0

0009   ORG  H#0FC
0010   ORTEST23:  TEST , & JPL & NUMB H#3F0 ;#2 AND#3 HANDLER AT LOC 3F0

0011   END

00F0 XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXX1111X00000  0100111111111001
00F1 XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXX1111X11110  1000XXXXXXXXXXXX
00F2 XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXX1111X00000  0110XXXXXXXXXXXX
00F4 XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXX00000  1111000111110000
00F8 XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXX00000  1111001011110000
00FC XXXXXXXXXXXXXXXX  XXXXXXXXXXXXXXXX  XXXXXXXXXXX00000  1111001111110000
```

FIGURE 2.1 continued

From: AMD, 1976a page 1-18



```
; EXAMPLE 2.
;
; ALIGNMENT CAN BE REALIZED IN ONE MICROINSTRUCTION. ASSUME
; THAT F3 OF THE MOST SIGNIFICANT ALU SLICE IS CONNECTED TO
; TEST 13 OF THE CONDITION MULTIPLEXERS. NOTE THAT NEGATIVE
; NUMBERS CAN BE ALIGNED IN THE SAME MANNER BY SIMPLY
; OMITTING THE VARIABLE "INV". ALSO, IF THE COUNTER IS CLEARED
; BEFORE STARTING ALIGNMENT, IT WILL CONTAIN THE NUMBER OF
; SHIFTS REQUIRED TO DO THE ALIGNMENT (OR THE COMPLIMENT
; IF USING AM25LS169 COUNTERS).
;
;
;
0001  ORG Q#0770
0002  ALIGN:    NUMB 0770 & TEST 13, INV & RPCT   ;(ALU TO SHIFT UP)
0003  END

01F8 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11011 1001000111111000
```

```
; EXAMPLE 3.
;
; A DIVISION ROUTINE. ASSUME F = 0 OF THE ALU IS CONNECTED TO
; TEST-12 (AND F3 TO TEST-13 AS BEFORE), AND SIXTEEN
; DIVISION STEPS ARE REQUIRED. IF THE FINAL REMAINDER IS NEGATIVE, IT MUST BE
; RESTORED BY ADDING IT TO THE DIVISOR. THE VECTOR INPUT IS SET UP
; FOR THE ERROR ROUTINE. NOTE USAGE OF THE AMDASM CONVENTION
; "$" TO DENOTE THE CURRENT PROGRAM COUNTER.
;
;
;
0001  ORG Q#1000
0002  DIVIDE:  LDCT & TEST, INV & NUMB D#14%.     ;(ALU OUTPUTS DIVISOR)
0003           TEST 12, INV & CJV                 ;IF = 0:  ERROR
0004           RPCT & TEST CNTR, & NUMB $         ;LOOP
0005           TEST 13, INV & NUMB $+2 & CJP      ;IF R <0, CORRECT
0006           TEST, & JMAP                       ;EXIT TO MAP
0007           TEST, & JMAP                       ;ALU ADDS REMAINDER TO DIVISOR, EXIT MAP
0008  END

0200 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00001 1100111111110001
0201 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11001 0110XXXXXXXXXXXX
0202 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11110 1001001000000010
0203 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX11011 0011001000000101
0204 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 0010XXXXXXXXXXXX
0205 XXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXX XXXXXXXXXXX00000 0010XXXXXXXXXXXX
```

FIGURE 2.1 continued

From: AMD, 1976a page 1-19

TABLE 2.2

CCU ASSEMBLY DEFINITIONS

22–25	21	16–20	12–15	0–11	BIT NO.
Am29803 Instructions	Inst. Register	Test Select** and Polarity	Am29811 Instructions	Numerical Field*	Field Description
T0	IN	The Test	JZ	Any 4	Parameters
T1		Number	CJS	Digit (12	To Be
T01		(1–14) in	JMAP	Bit) Octal	Used
T2		Decimal,	CJP	Number	
T02		and:	PUSH		
T12		CNTR	JSRP		
T3		for Test	CJV		
T03		Select.	JRP		
T13		(Uncondi-	RFCT		
T013		tional by	RPCT		
T23		default) INV	CRTN		
T023		for Test	CJPP		
T123		Polarity	LDCT		
T012		(noninverted	LOOP		
T0123		by default)	CONT		
NOT			JP		

Source: AMD, 1976a, page 1-15

3.0 BASIC MICROPROGRAMMED PIPELINE ARCHITECTURE

The computer design in this paper incorporates the techniques of microprogramming and pipelining. A basic functional block diagram of this type of system is shown in figure 3.1. The functional blocks perform various phases of the computer's operation and are listed below along with a brief description of the function of each. A more detailed presentation of each element is given in the section on the hardware implementation. This section is a simple overview to familiarize the reader with the total architecture.

3.1 INSTRUCTION REGISTER

The instruction is clocked into the instruction register from the data bus when the pipeline register sends the proper command. The instruction is held in the instruction register (IR) until the pipeline register commands another IR load. From the instruction register the instruction is routed to the Mapping PROM, RALU, PCU, and the input and output (I/O) control.

3.2 MAPPING PROM

The Mapping PROM contains the address to the starting

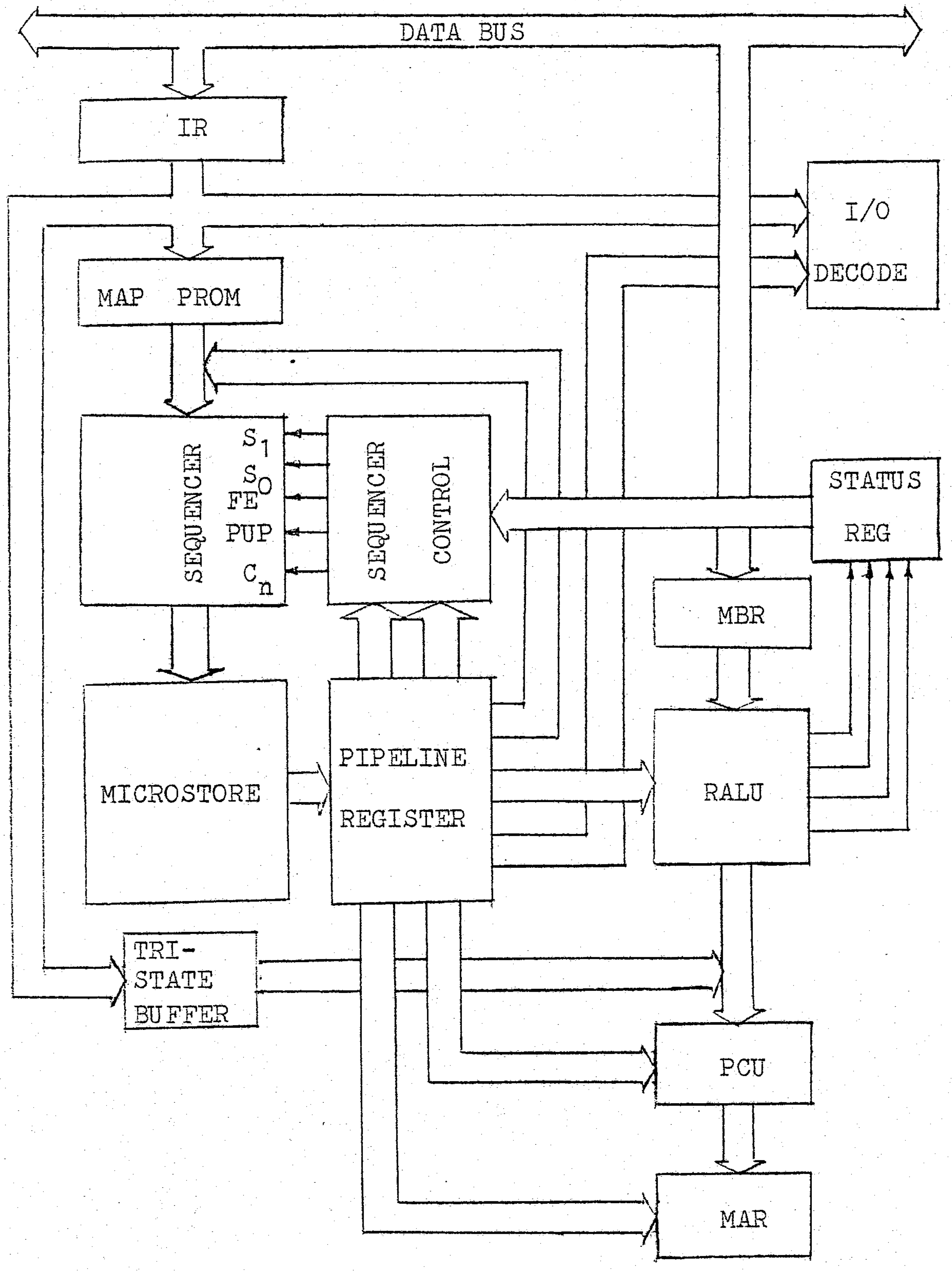


FIGURE 3.1 Functional block diagram of microprogram type computer.

points of each instruction's microcode in the microstore. By routing the instruction from the instruction register to the Mapping PROM, the correct starting address for the instruction's block of microcode is generated.

3.3 SEQUENCER

The sequencer selects the proper source for the next microinstruction address. The next address may come from the Mapping PROM, the pipeline register, the sequencer's stack, or the sequencer's R register. The source for the next instruction depends upon the situation, as will be seen later in the examples of microcode.

3.4 MICROPROGRAM

The output of the sequencer is sent to the microprogram memory and the output of the microprogram memory is clocked into the pipeline register on the next clock pulse. Once in the pipeline register, the instruction is executed by sending commands to the RALU, PCU, MAR, MBR, sequencer, and other computer elements. The microprogram sends the sequencer the code for the next address source and may also send the branch address if the situation is called for. This process allows the previous pipeline word to fetch the next microcode while the present code is being executed. This parallel operation allows the computer to run twice as fast as would be possible if the system was processing the microcode serially. Figure 3.2 illustrates how this overlapping operates.

EFFECTIVE DURATION OF EACH ADD DX INSTRUCTION = 3 MICRO CYCLES

ACTION									
FETCH INSTRUCTION	A			B			C		
DECODE INSTRUCTION			A			B			C
FETCH BASE ADDRESS			A			B			C
FORM EFFECTIVE ADDRESS				A			B		
FETCH OPERAND					A			B	
ADD OPERAND AND SAVE RESULT						A			B
MICRO CYCLES	1	2	3	4	5	6	7	8	9

ACTUAL DURATION OF EACH ADD DX INSTRUCTION = 6

FIGURE 3.2 Example of an instruction using pipelining and the effective increase in throughput. The instruction is a direct-indexed addition.

From: Muething, 1976

3.5 PIPELINE REGISTER

The pipeline register is loaded from the microprogram and is usually between 40 and 60 bits wide. These bits are formed into many different fields which control the different elements of the computer. For instance, the pipeline register has fields to control the next microinstruction address sources, the function of the ALU, the operation of the I/O unit, the program addressing, and all the associated registers.

3.6 RALU

The 2901 contains 16 two port registers, a Q register, an 8 function ALU, output multiplexer, and shifting capability (the more complex 2903 also has provisions to do two's complement and floating point operations).

3.7 PROGRAM CONTROL UNIT (PCU)

The PCU is a powerful integrated circuit that contains a stack, incrementer, and other necessary elements to do most forms of addressing, including direct, indirect, indexed, and relative.

4.0 SELECTION OF THE INSTRUCTION SET

Deciding on which instruction set to use in the computer design was the most agonizing part of the design. The goal was an instruction set that was both easy to implement and had a large existing software base. The choice soon narrowed down to either the PDP-11 or NOVA 1200 instruction set.

The PDP-11 instruction set (figure 4.1) has one of the largest bases of existing software. In addition, Digital Equipment Corporation and Western Digital produce a PDP-11 software compatible microprocessor. The availability of the microprocessors would allow the development of software on a relatively large and fast system, such as the Am2900, and allow total software transfer to a small dedicated system later if desired. However, the PDP-11 instruction set is particularly difficult to implement because of its poorly structured op code field. This op code field difficultly would have required expensive Programmable Logic Arrays (PLA) to implement the instruction set. Since the PLA's are not reprogrammable, this would have been cost prohibitive on a one of a kind system.

As can be seen from figure 4.2, the NOVA instruction set is simpler and more clearly structured than the PDP-11 instruction set. There are separate fields for the ALU

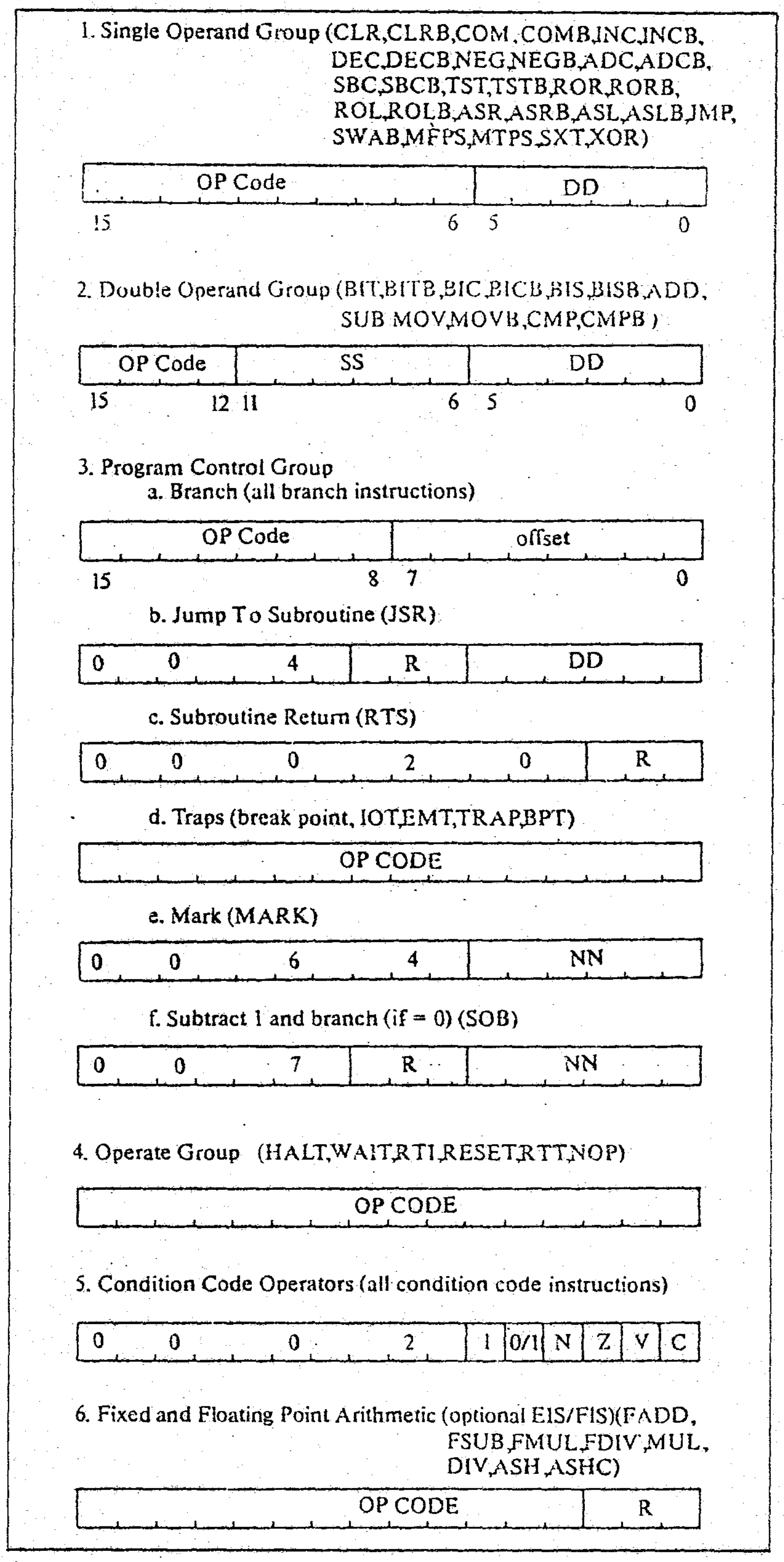
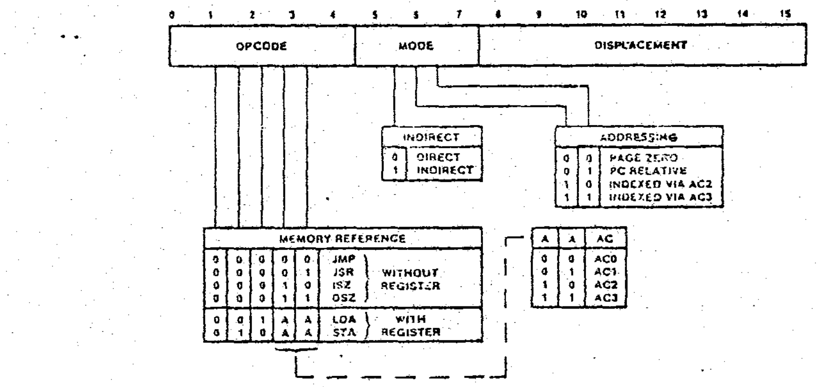
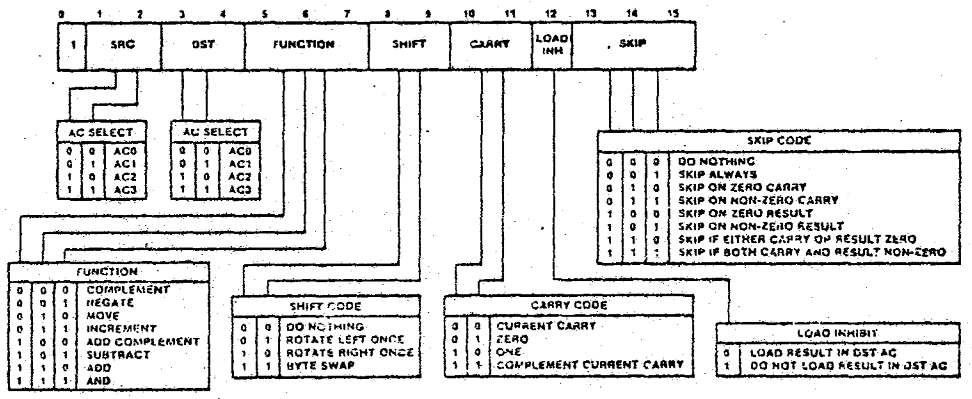

1. Single Operand Group (CLR,CLRB,COM,COMB,INC,INCB, DEC,DECB,NEG,NEGB,ADC,ADCB, SBC,SBCB,TST,TSTB,ROR,RORB, ROL,ROLB,ASR,ASRB,ASL,ASLB,JMP, SWAB,MFPS,MTPS,SXT,XOR)
OP Code
DD
15
6
5
0
2. Double Operand Group (BIT,BITB,BIC,BICB,BIS,BISB,ADD, SUB MOV,MOVB,CMP,CMPB)
OP Code
SS
DD
15
12
11
6
5
0
3. Program Control Group
a. Branch (all branch instructions)
OP Code
offset
15
8
7
0
b. Jump To Subroutine (JSR)
0 0 4
R
DD
c. Subroutine Return (RTS)
0 0 0 2 0
R
d. Traps (break point, IOT,EMT,TRAP,BPT)
OP CODE
e. Mark (MARK)
0 0 6 4
NN
f. Subtract 1 and branch (if = 0) (SOB)
0 0 7
R
NN
4. Operate Group (HALT,WAIT,RTI,RESET,RTT,NOP)
OP CODE
5. Condition Code Operators (all condition code instructions)
0 0 0 2
1
0/1
N
Z
V
C
6. Fixed and Floating Point Arithmetic (optional EIS/FIS)(FADD, FSUB,FMUL,FDIV,MUL, DIV,ASH,ASHC)
OP CODE
R

FIGURE 4.1 PDP-11 instruction set
From: DEC, page 33

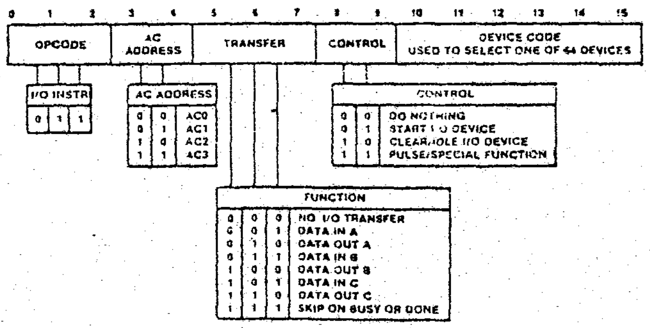
9440 INSTRUCTIONS



Memory Reference instructions without register are used for branching (JMP, JSR) without involving accumulators. These instructions are also used for modifying memory (ISZ, DSZ). Memory Reference instructions with register are used to move 16-bit words between the memory and the accumulators.



Arithmetic/Logic instructions perform arithmetic (ADD, ADC, INC, NEG, SUB) or Boolean (AND, COM, MOV) operations on the contents of two registers. The result of each operation together with the Carry bit can be rotated and tested for skip conditions as part of the same arithmetic/logic instruction; loading in the destination register is optional.



Input/Output instructions move data between the 9440 accumulators and three buffers in the peripheral device interface. These instructions also perform control functions in the I/O device and test the status flags in both the peripheral circuitry and the central processor.

FIGURE 4.2 Fairchild 9440 emulation of the NOVA 1200 instruction set

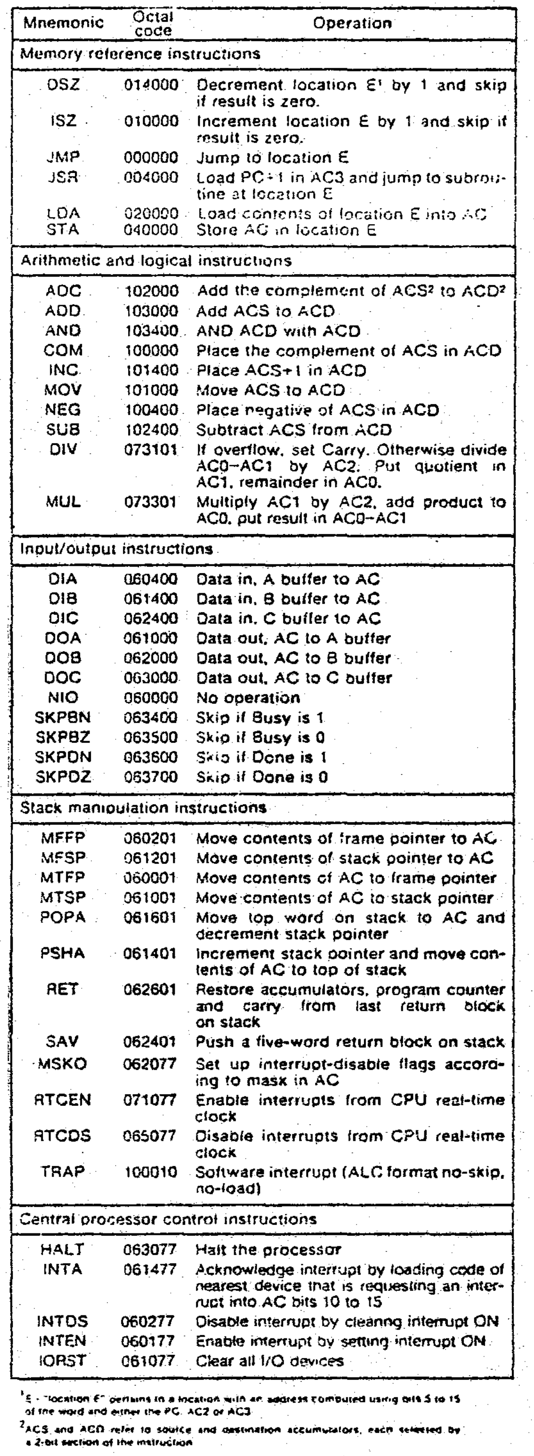
From: Wilnai, 1977, page 6

function, shift, and carry. The op code field does not have a variety of lengths as the PDP-11 does. This makes the implementation of the Mapping PROM easy since EPROM's can be used. There is the added plus of a large and expanding software base for the NOVA instruction set, and, as in the case of the PDP-11, there are microprocessors available that are compatible with the NOVA instruction set. One microprocessor is from Data General (the MN601) and the other is from Fairchild (the 9940).

It should be noted that the NOVA instruction set is not a particularly demanding set for the Am2900. For instance, the 2900 can easily provide information on comparisons such as A greater than B, or A less than B; the NOVA instruction set can not use this information and must take several steps to arrive at the same decision.

The NOVA instruction set has access to the four accumulators which are present in the NOVA architecture, however, since the Am2901 has sixteen registers, twelve of which the NOVA cannot use, the NOVA instruction set cannot take full advantage of the Am2901's capability.

When all the factors are considered, the NOVA instruction set is an acceptable choice for this project because it has adequate power and a straight forward structure for its instruction set. Figure 4.2 and figure 4.3 show the structure of the instruction set and the function of each instruction.



Mnemonic	Octal code	Operation
Memory reference instructions		
OSZ	014000	Decrement location E[1] by 1 and skip if result is zero.
ISZ	010000	Increment location E by 1 and skip if result is zero.
JMP	000000	Jump to location E
JSR	004000	Load PC+1 in AC3 and jump to subroutine at location E
LDA	020000	Load contents of location E into AC
STA	040000	Store AC in location E
Arithmetic and logical instructions		
ADC	102000	Add the complement of ACS[2] to ACD[2]
ADD	103000	Add ACS to ACD
AND	103400	AND ACD with ACD
COM	100000	Place the complement of ACS in ACD
INC	101400	Place ACS+1 in ACD
MOV	101000	Move ACS to ACD
NEG	100400	Place negative of ACS in ACD
SUB	102400	Subtract ACS from ACD
DIV	073101	If overflow, set Carry. Otherwise divide AC0–AC1 by AC2. Put quotient in AC1, remainder in AC0.
MUL	073301	Multiply AC1 by AC2, add product to AC0, put result in AC0–AC1
Input/output instructions		
DIA	060400	Data in, A buffer to AC
DIB	061400	Data in, B buffer to AC
DIC	062400	Data in, C buffer to AC
DOA	061000	Data out, AC to A buffer
DOB	062000	Data out, AC to B buffer
DOC	063000	Data out, AC to C buffer
NIO	060000	No operation
SKPBN	063400	Skip if Busy is 1
SKPBZ	063500	Skip if Busy is 0
SKPDN	063600	Skip if Done is 1
SKPDZ	063700	Skip if Done is 0
Stack manipulation instructions		
MFFP	060201	Move contents of frame pointer to AC
MFSP	061201	Move contents of stack pointer to AC
MTFP	060001	Move contents of AC to frame pointer
MTSP	061001	Move contents of AC to stack pointer
POPA	061601	Move top word on stack to AC and decrement stack pointer
PSHA	061401	Increment stack pointer and move contents of AC to top of stack
RET	062601	Restore accumulators, program counter and carry from last return block on stack
SAV	062401	Push a five-word return block on stack
MSKO	062077	Set up interrupt-disable flags according to mask in AC
RTCEN	071077	Enable interrupts from CPU real-time clock
RTCDS	065077	Disable interrupts from CPU real-time clock
TRAP	100010	Software interrupt (ALC format no-skip, no-load)
Central processor control instructions		
HALT	063077	Halt the processor
INTA	061477	Acknowledge interrupt by loading code of nearest device that is requesting an interrupt into AC bits 10 to 15
INTDS	060277	Disable interrupt by clearing interrupt ON
INTEN	060177	Enable interrupt by setting interrupt ON
IORST	061077	Clear all I/O devices

[1] E - "location E" pertains to a location with an address computed using bits 5 to 15 of the word and either the PC, AC2 or AC3

[2] ACS and ACD refer to source and destination accumulators, each selected by a 2-bit section of the instruction

FIGURE 4.3 microNOVA instruction set

From: Falkoff, 1977a

5.0 STATE TRANSITION DIAGRAM

The state transition diagram depicts, in graphical form, the sequence of events that the computer can go through in each cycle. The state transition diagram, shown as figure 5.1, is presented in the form typical of other state diagrams and should be self explanatory.

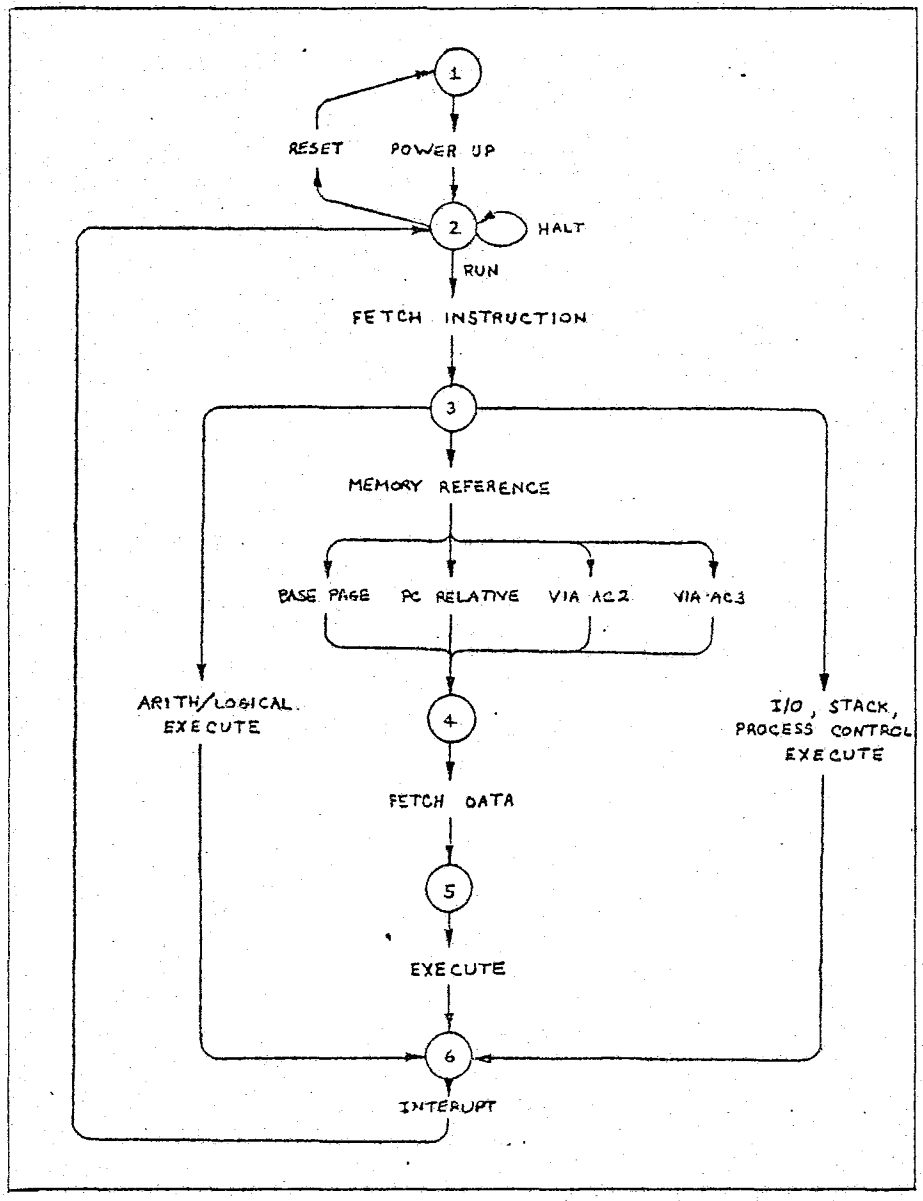


FIGURE 5.1 CPU state transition diagram

6.0 <u>HARDWARE IMPLEMENTATION</u>

Figure 6.0 shows the system design of the 2900 16 bit minicomputer. Figure 6.0 refers to the detailed drawings (figure 6.1 through 6.12) of the computer functions. The following sections describe the hardware implementation of these functions in detail.

6.1 IR AND MAPPING PROM

The computer starts its cycle by loading the instruction register (figure 6.1 in the system design). The instruction register latches the data and sends it on to the Mapping PROM. In a final design the Mapping PROM would consist of high speed bipolar PROM, however, for the development system relatively slow ultra violet erasable EPROM was used because it provided more flexibility and was more cost effective than throwing away a set of PROMs each time the microcode was changed. This necessarily slowed down the speed of the computer since the EPROMs used (the 2708) have access times of 450 ns as compared to 50 ns which is typical of microstore speeds.

The Mapping PROM sends the proper starting address to the sequencer(figure 6.2 in the system design) where one of several options will be performed depending on the contents of the pipeline register. For instance, a micro-

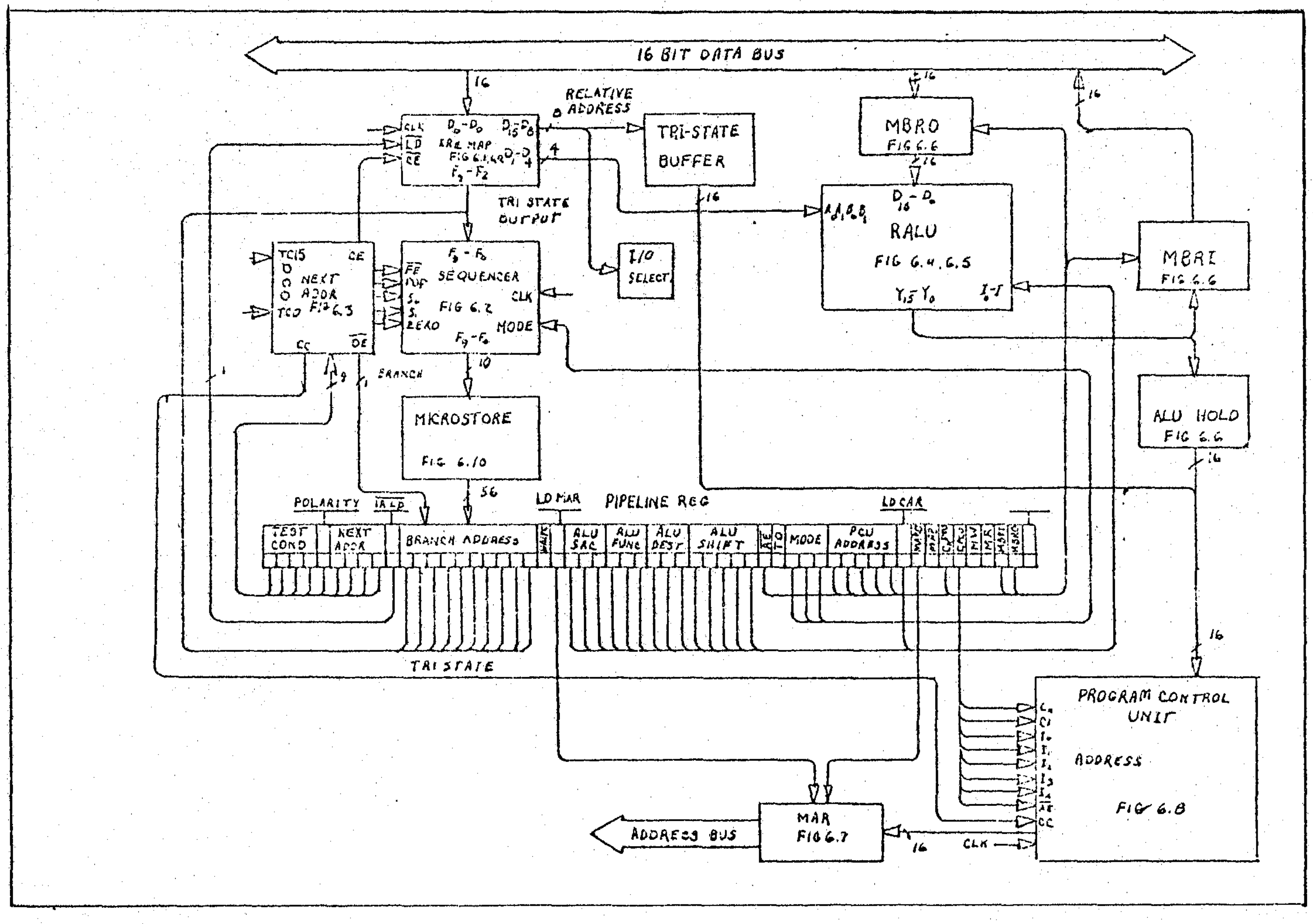


FIGURE 6.0 System architecture for 16 bit computer CPU

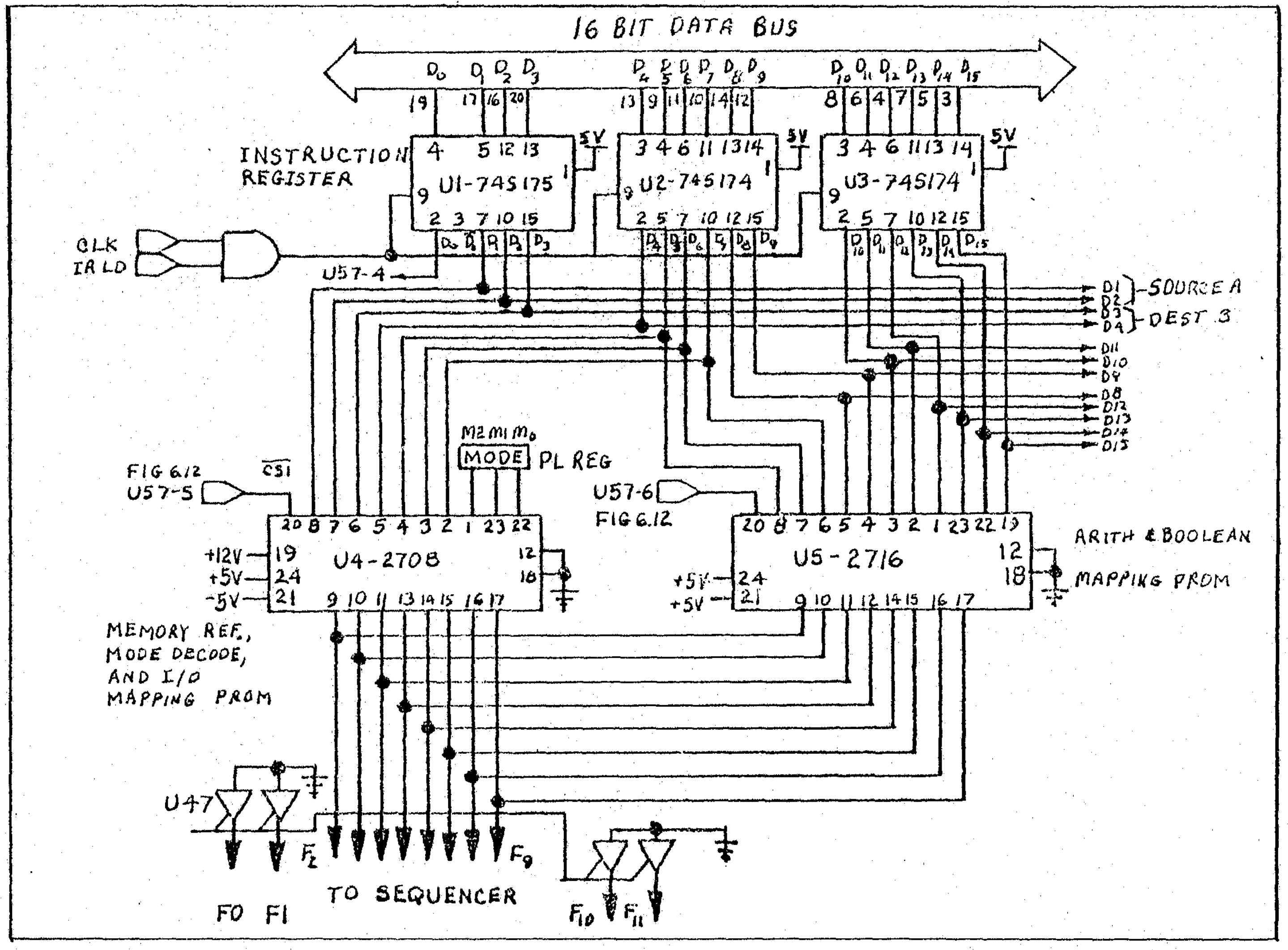


FIGURE 6.1 Instruction register and Mapping PROM

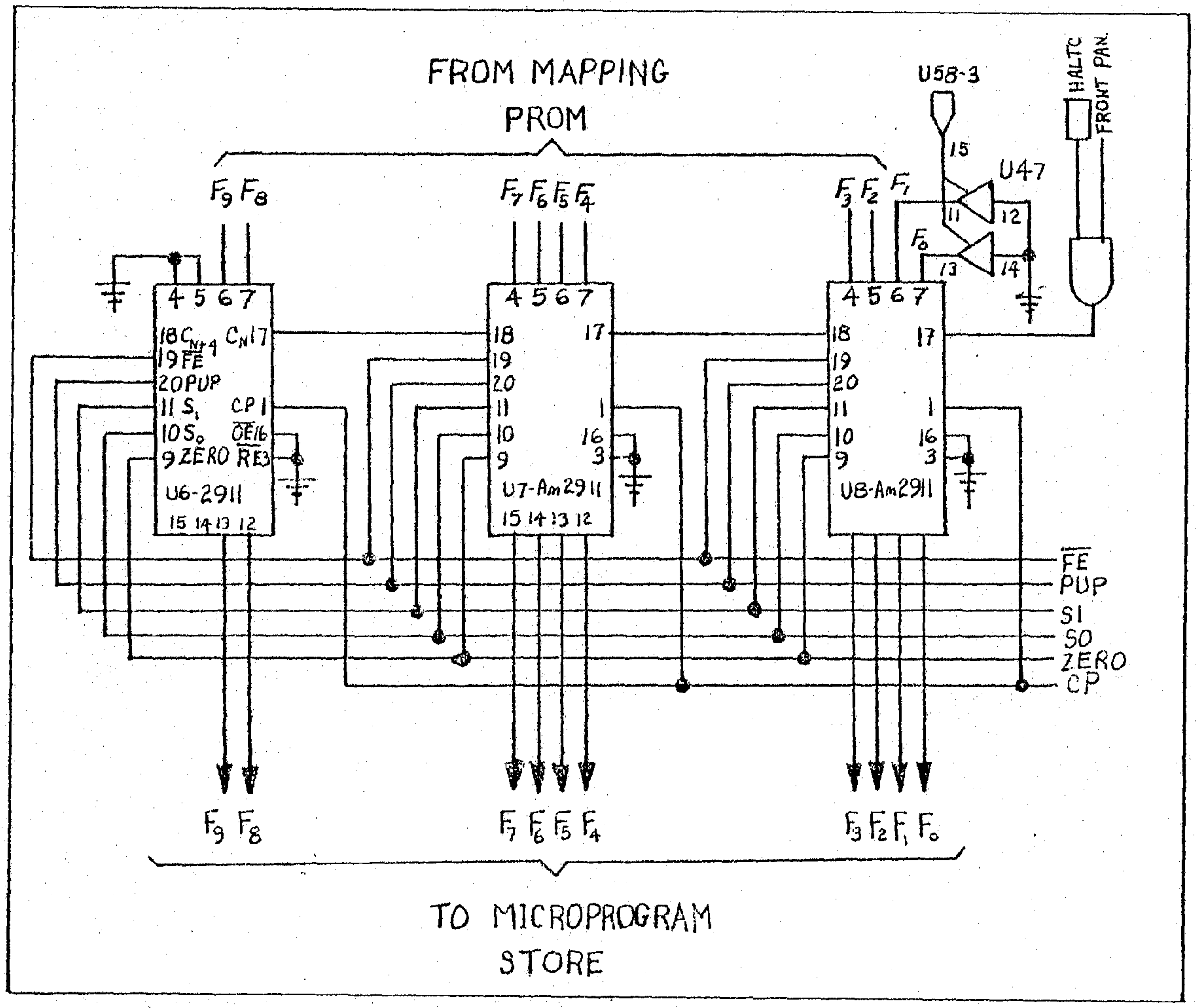


FIGURE 6.2 Sequencer

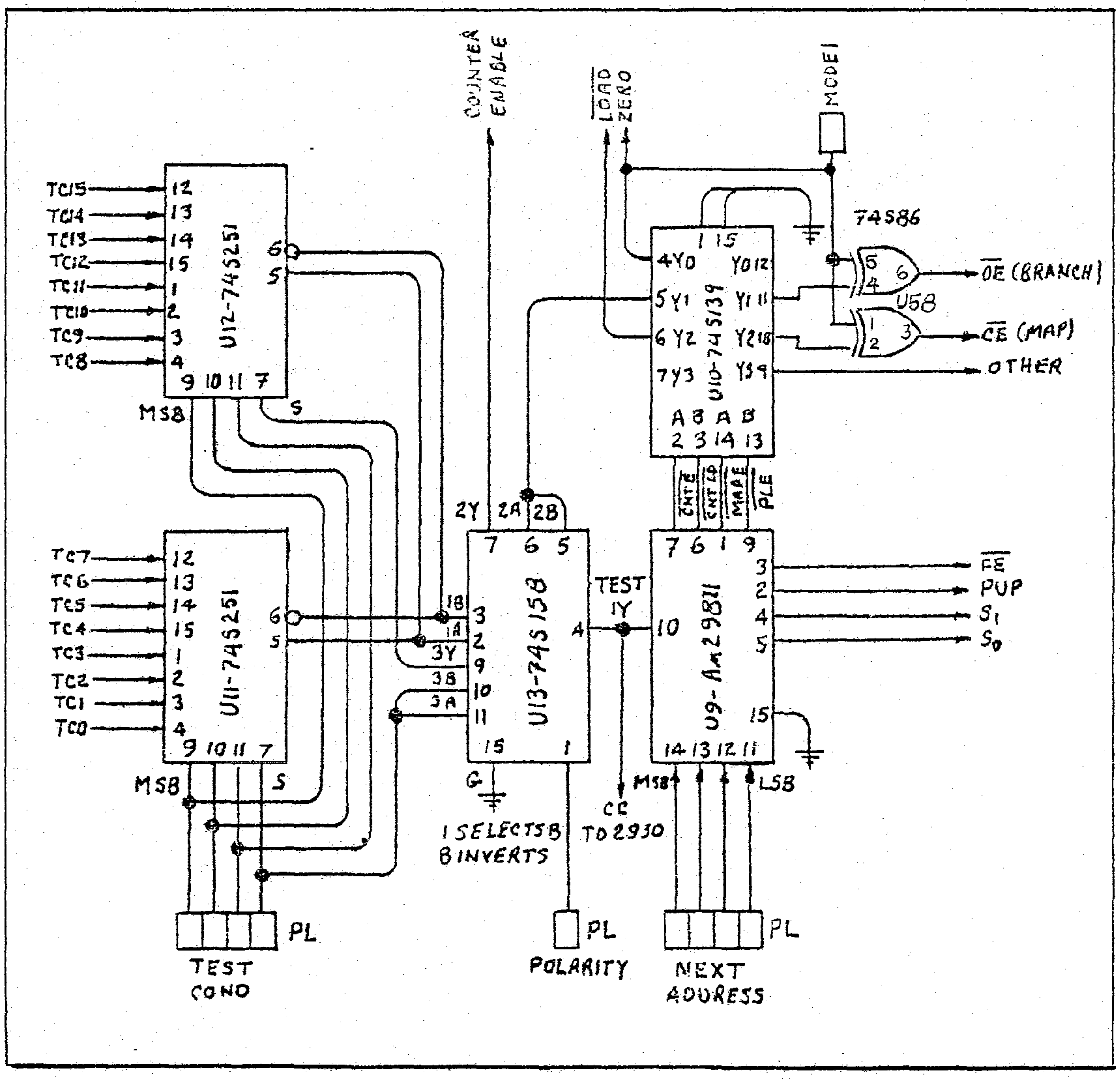


FIGURE 6.3 Next address control

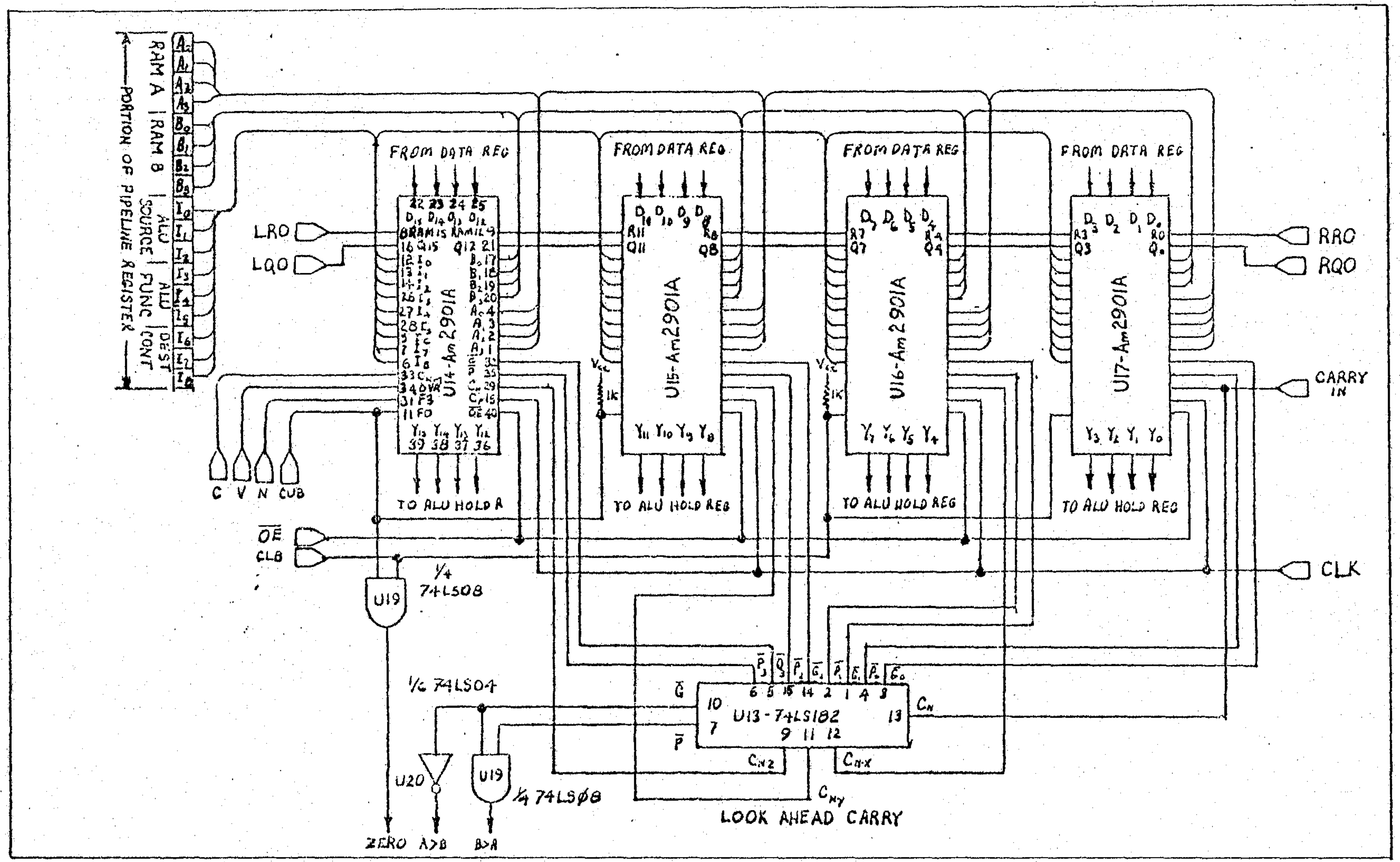


FIGURE 6.4 2901 RALU array

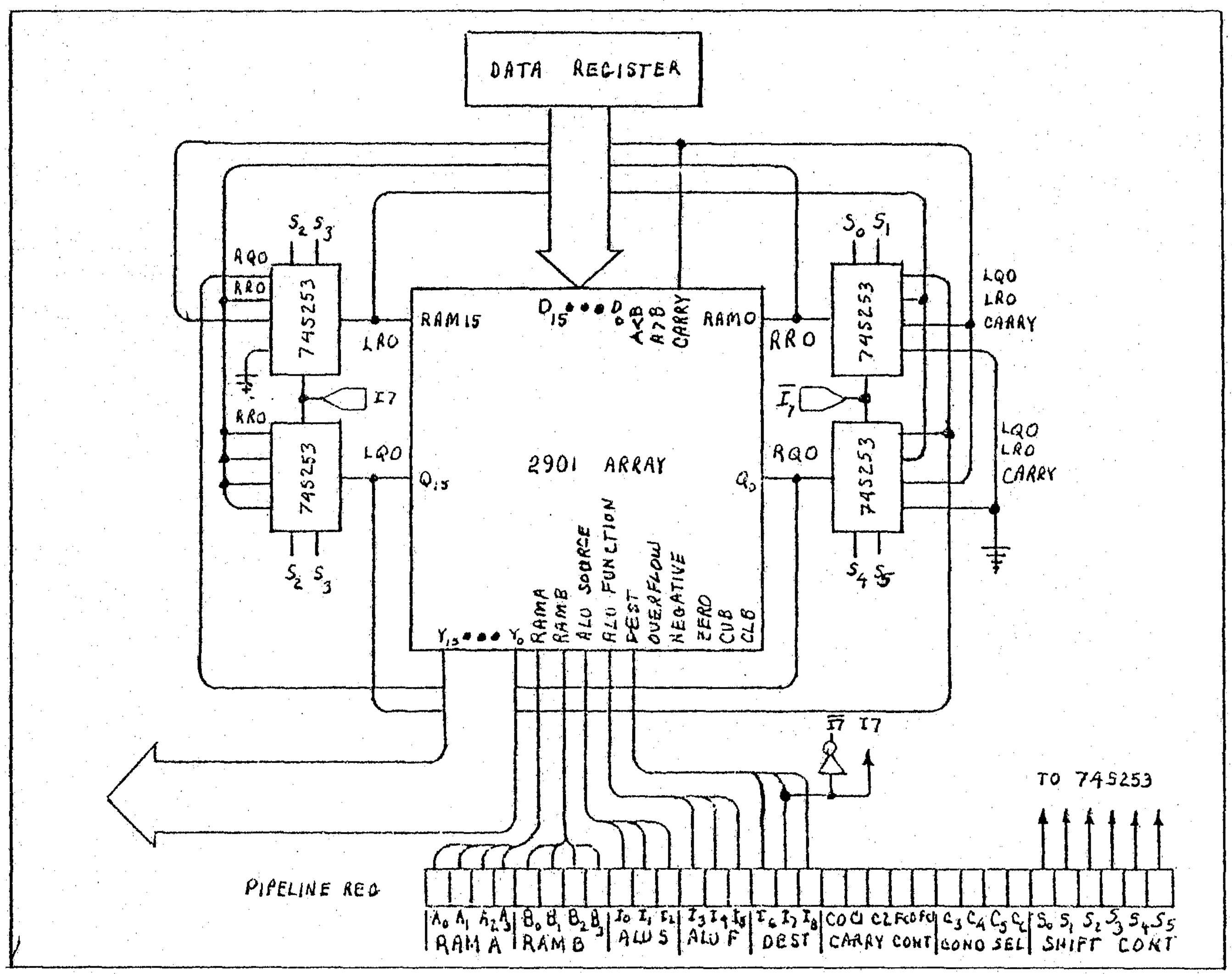


FIGURE 6.5 Arithmetic and logic unit

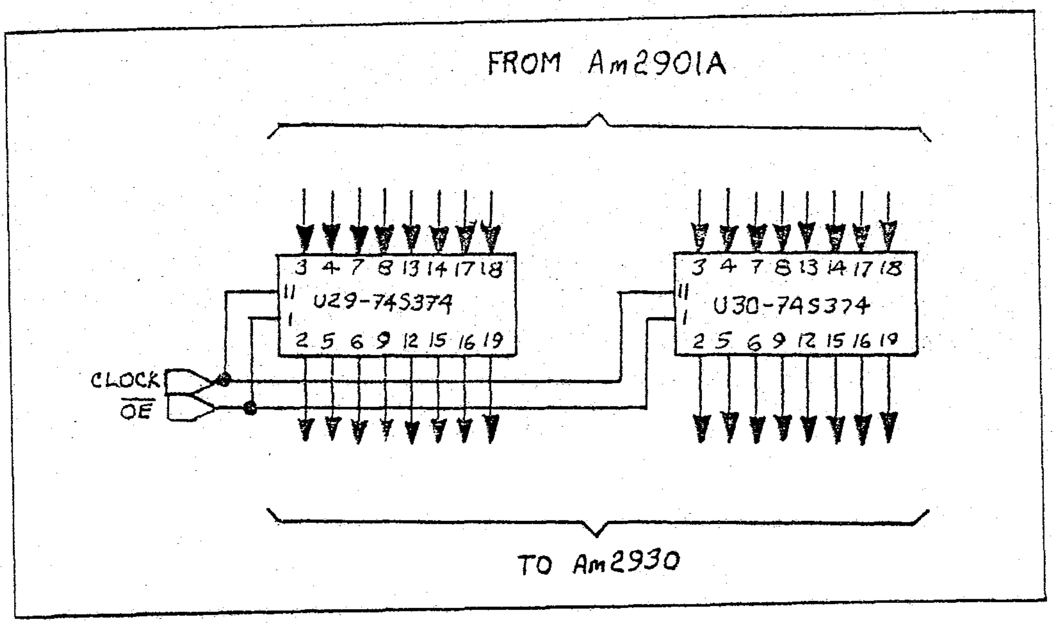


FIGURE 6.6 ALU holding register

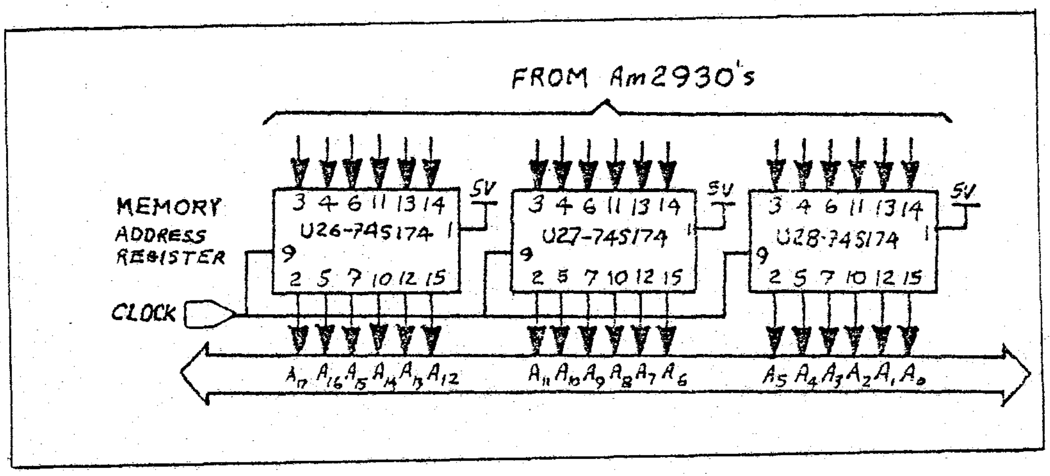


FIGURE 6.7 Memory Address Register

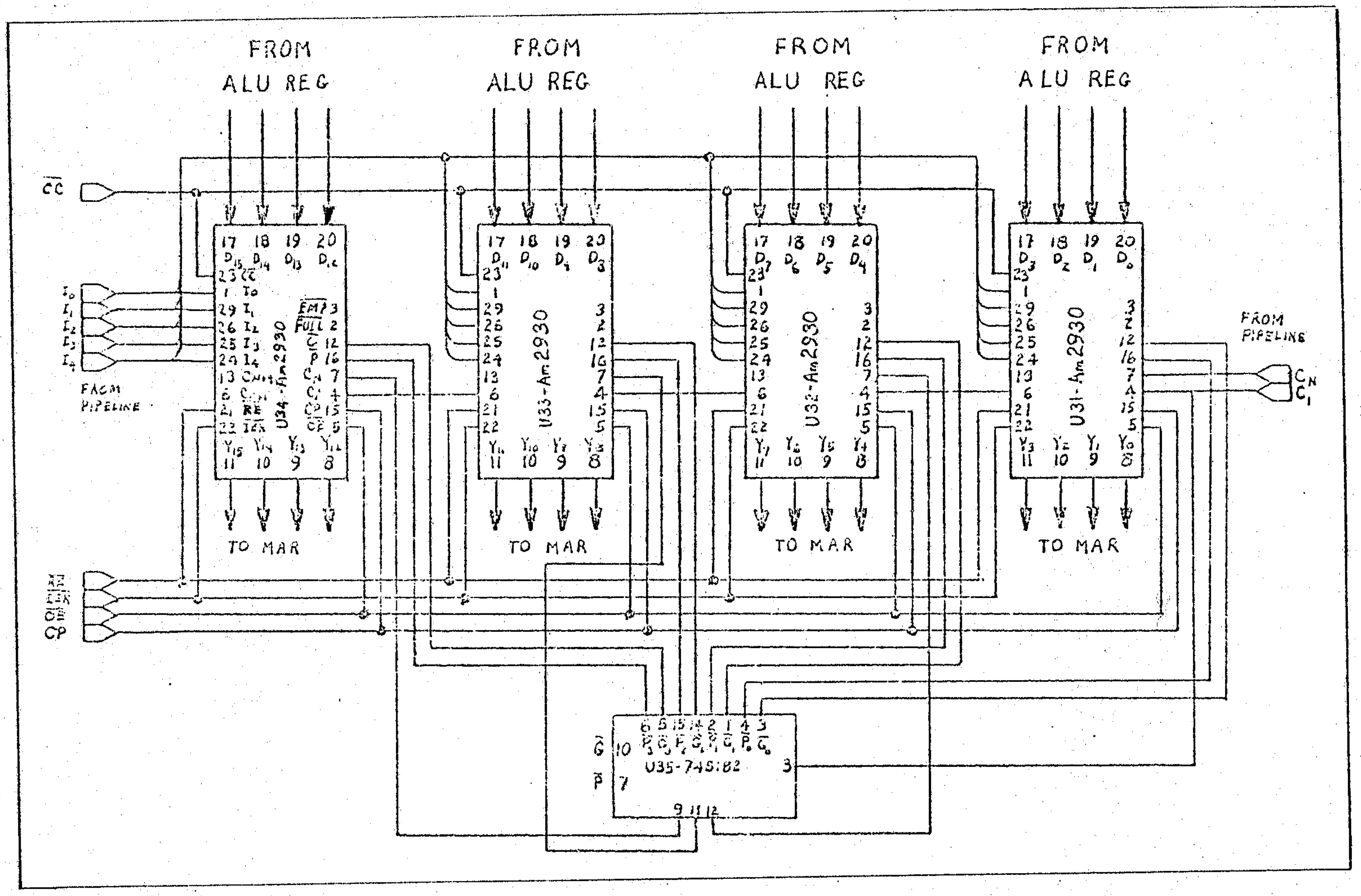


FIGURE 6.8 Program Control Unit

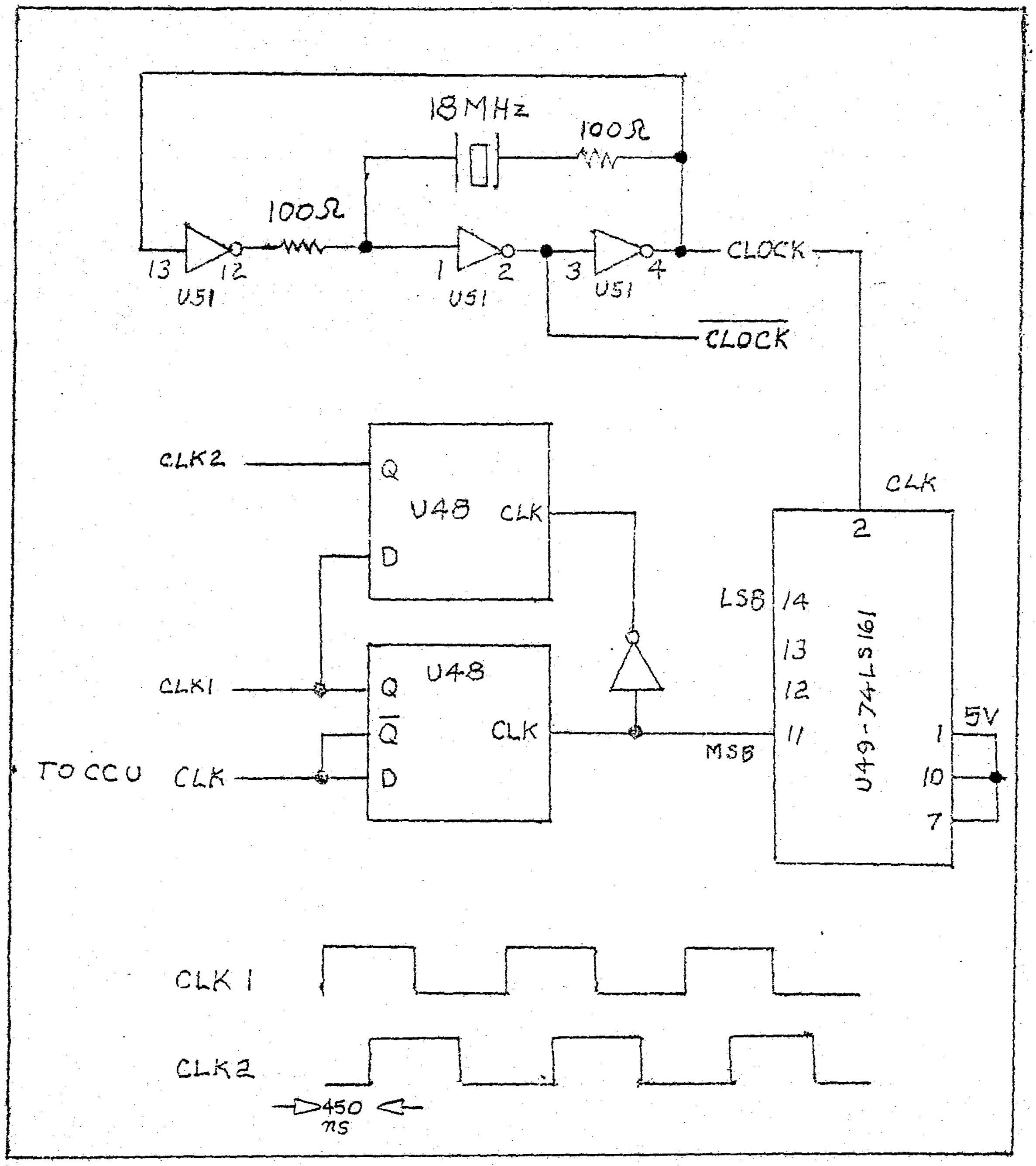


FIGURE 6.9 Clock generator

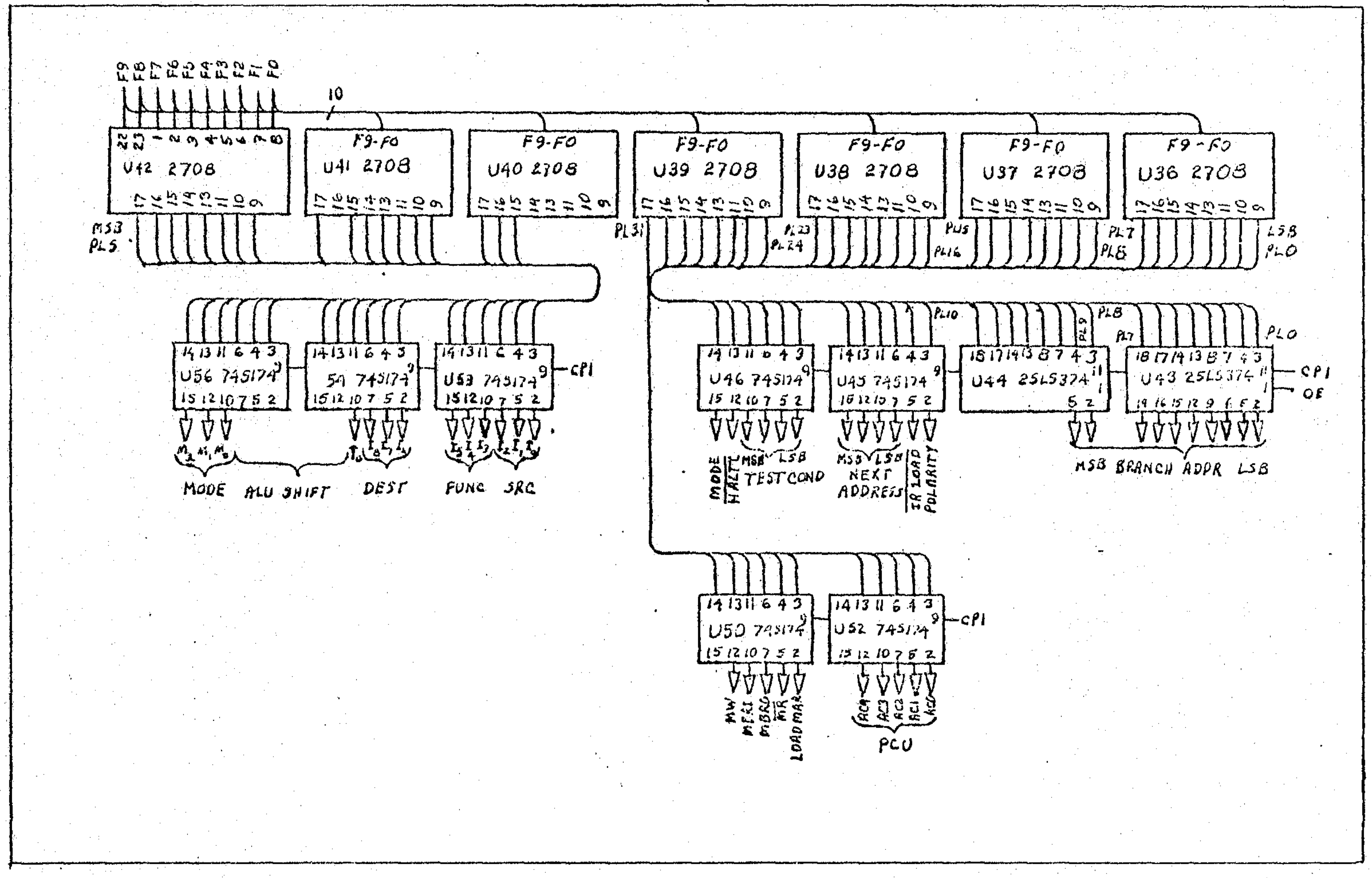


FIGURE 6.10 Microstore and Pipeline Register

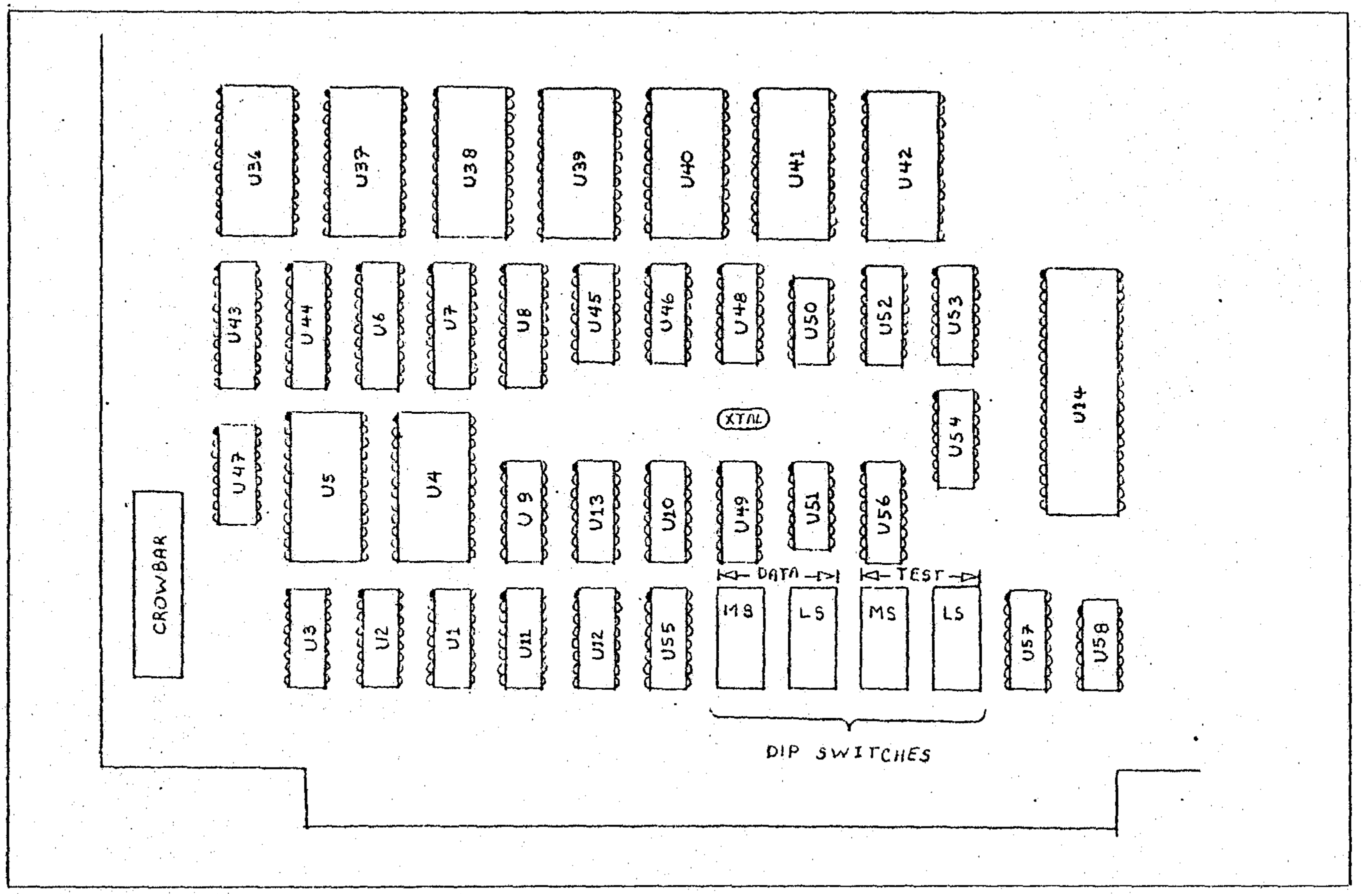


FIGURE 6.11a Physical layout of CCU

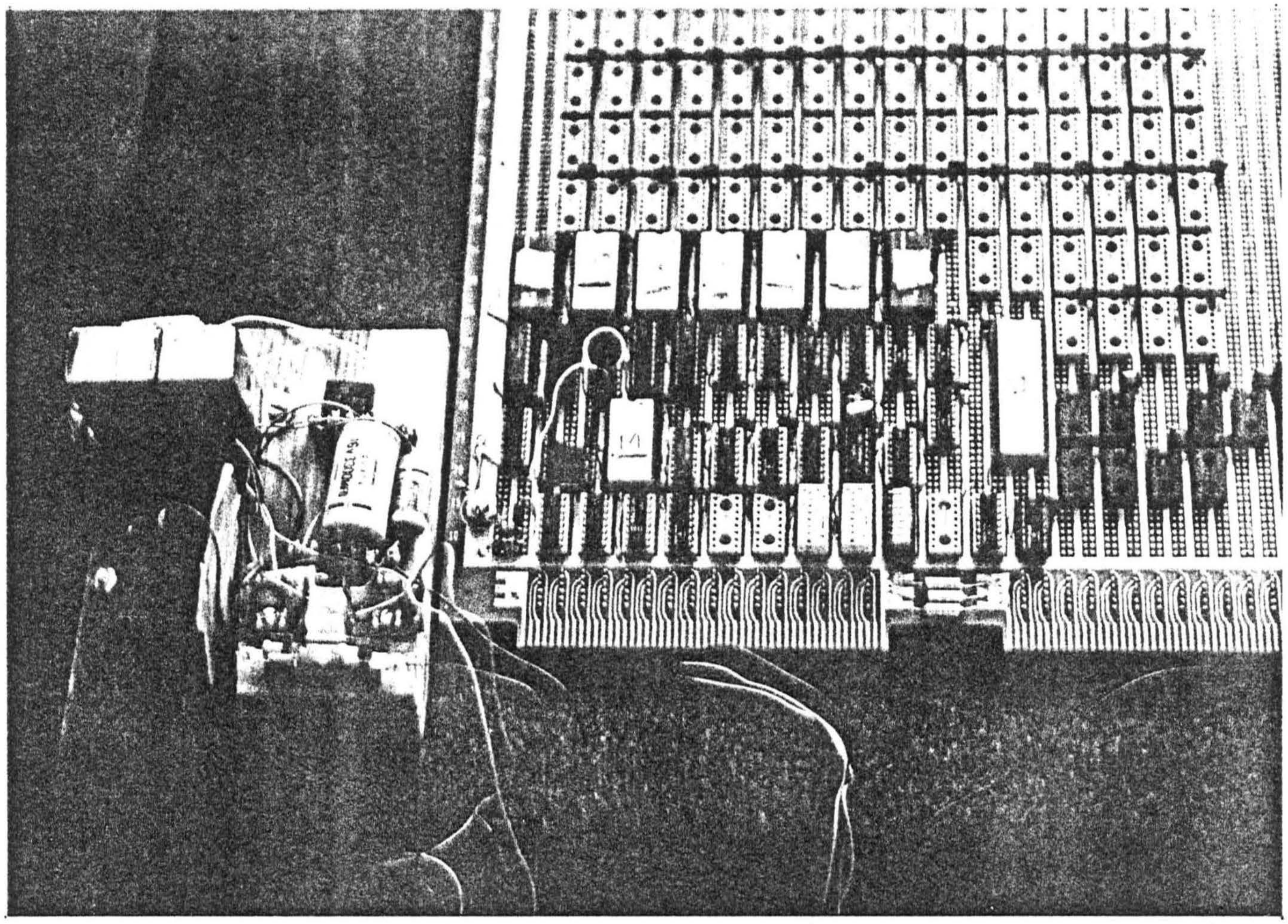


FIGURE 6.11b Photograph of the physical layout of the CCU

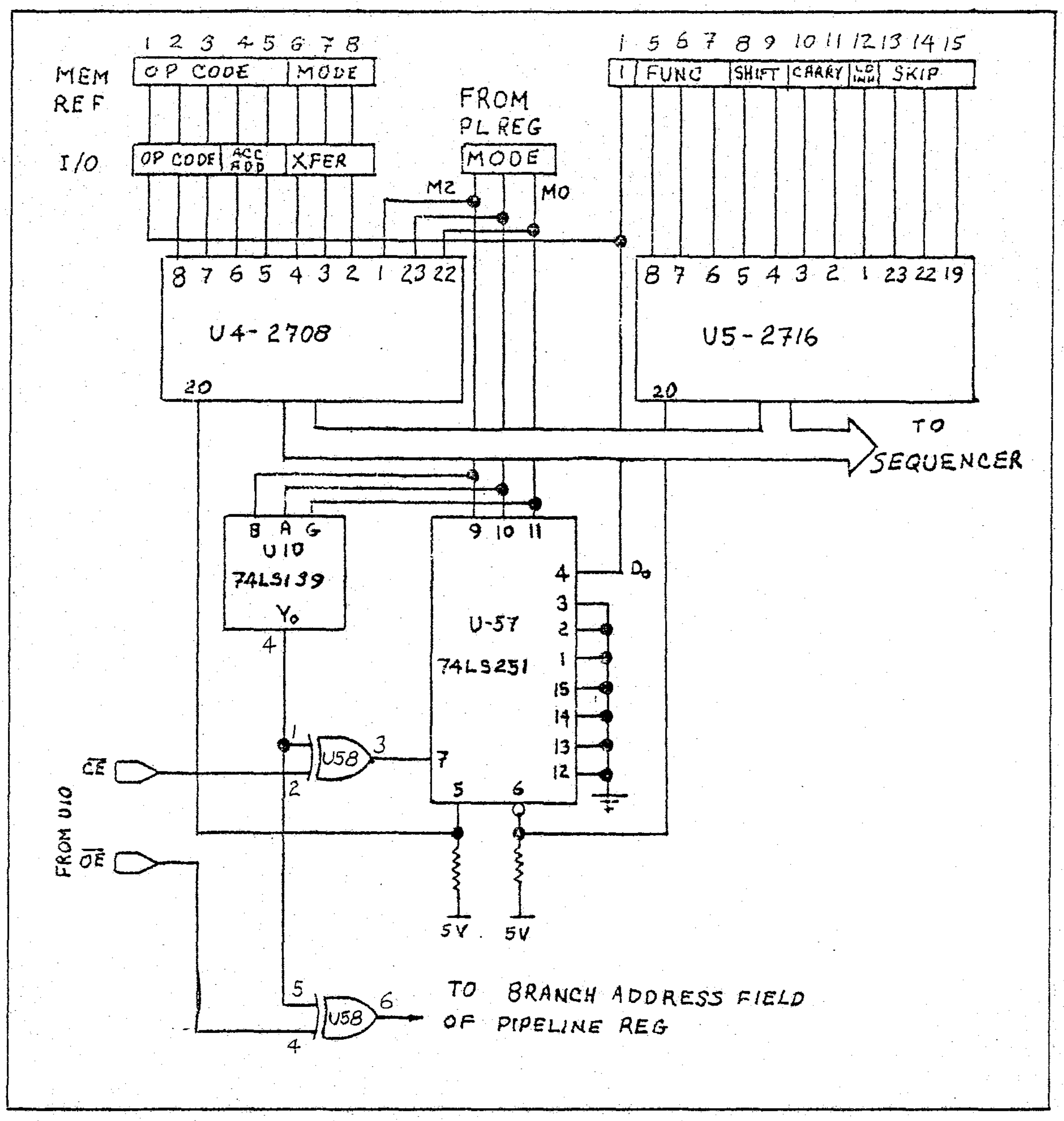


FIGURE 6.12 MODE implementation circuit

subroutine may be called or returned from, or a branch to a location selected by the pipeline register may be executed, or a branch to the location addressed by the mapping PROM may be executed. The appropriate action depends upon the next address and mode fields of the pipeline register. The next address field controls the operation of the Am29811 sequencer controller, which will be discussed in detail latter.

What is important about the Am29811 now is the fact that it does not allow for a jump to a subroutine given by the address from the mapping PROM. There are many variations on the basic NOVA 1200 instructions including direct, indirect, relative, and base page addressing combined with the shifts and skips that are a part of the ADD instruction. To decode these instructions into unique microcode routines requires either a large amount of memory, or a large amount of time, or a clever compromise to keep the amount of mapping PROM and microstore down while not slowing the computer down too much.

One of the options, the most memory intensive and the fastest, is to microcode each instruction permutation and map each variation into its unique block of microcode. This approach requires a much larger mapping PROM and microstore, but the computer does not waste time decoding instruction permutations with firmware.

Another method is to determine each permutation by jumping into microcode subroutines to determine the proper mode and returning when the determination is complete. This method is slow because of the time required to jump to a microcode subroutine, do a firmware determination of mode, and to return from the subroutine.

A more practical approach is to use the Mapping PROM as a decoder. To do this, the Mapping PROM must be available as a mapping function during normal modes, but must be converted into an instruction decoder during the appropriate modes. In this way, the computer goes to the proper instruction microcode without the use of firmware subroutines. Since there are several mode types, such as addressing, shifting, and skipping, and since the Am29811 does not have a MAP subroutine instruction, logic has to be added to accomplish calls to the subroutine pointed to by the Mapping PROM.

The way to implement the MAP subroutine instruction with the least amount of hardware would be to combine the next address and mode fields into one field and replace the Am29811 with a PROM large enough to decode the combined fields. However, the need for the mode field was not realized until after the hardware was constructed. Besides, I did not have access to the necessary PROM programmer. The next best approach was to construct the logic needed by using a 74LS251 tri-state eight input multiplexer as an universal logic module in combination with the unused half of a 74LS139 one of four decoder. During the addressing, shifting, and skipping modes the logic inverts the pipeline and mapping PROM enable lines so the jump to a pipeline subroutine is converted into a jump to the MAP subroutine. The circuit is shown as figure 6.12.

6.2 SEQUENCER

The sequencer is the heart of the computer's micro-programmed architecture. The sequencer controls the execution of the microprogram with its own instruction set (see table 6.1), but the sequencer is, in turn, controlled by the sequencer controller, a 29811.

The sequencer controller is part of the next address circuit (see figure 6.3). The next address circuit performs several functions. First, the next address circuit can test up to 16 test inputs and send the results to the 29811 sequencer controller which uses the information to decide the source of the next sequencer address. Secondly, in addition to the test input, the next address source for the sequencer is determined by the pipeline register commands to the sequencer controller. These pipeline register commands are from the sequencer controller's own instruction set (see tables 6.2, 6.3, 6.4).

TABLE 6.1

SEQUENCER COMMANDS

Address Selection

OCTAL	S_1	S_0	SOURCE FOR Y OUTPUTS	SYMBOL
0	L	L	Microprogram Counter	µPC
1	L	H	Register	REG
2	H	L	Push-Pop stack	STK0
3	H	H	Direct inputs	D_i

Output Control

OR_i	$\overline{ZERO}$	$\overline{OE}$	Y_i
X	X	H	Z
X	L	L	L
H	H	L	H
L	H	L	Source selected by S_0 S_1

Z = High Impedance

Synchronous Stack Control

$\overline{FE}$	PUP	PUSH-POP STACK CHANGE
H	X	No change
L	H	Increment stack pointer, then push current PC onto STK0
L	L	Pop stack (decrement stack pointer)

H = High
L = Low
X = Don't Care

CYCLE	S_1, S_0, $\overline{FE}$, PUP	µPC	REG	STK0	STK1	STK2	STK3	Y_{OUT}	COMMENT	PRINCIPLE USE
N N+1	0 0 0 0 –	J J+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	J –	Pop Stack	End Loop
N N+1	0 0 0 1 –	J J+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	J –	Push µPC	Set-up Loop
N N+1	0 0 1 X –	J J+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	J –	Continue	Continue
N N+1	0 1 0 0 –	J K+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	K –	Pop Stack; Use AR for Address	End Loop
N N+1	0 1 0 1 –	J K+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	K –	Push µPC; Jump to Address in AR	JSR AR
N N+1	0 1 1 X –	J K+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	K –	Jump to Address in AR	JMP AR
N N+1	1 0 0 0 –	J Ra+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	Ra –	Jump to Address in STK0; Pop Stack	RTS
N N+1	1 0 0 1 –	J Ra+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	Ra –	Jump to Address in STK0; Push µPC	
N N+1	1 0 1 X –	J Ra+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	Ra –	Jump to Address in STK0	Stack Ref (Loop)
N N+1	1 1 0 0 –	J D+1	K K	Ra Rb	Rb Rc	Rc Rd	Rd Ra	D –	Pop Stack; Jump to Address on D	End Loop
N N+1	1 1 0 1 –	J D+1	K K	Ra J	Rb Ra	Rc Rb	Rd Rc	D –	Jump to Address on D; Push µPC	JSR D
N N+1	1 1 1 X –	J D+1	K K	Ra Ra	Rb Rb	Rc Rc	Rd Rd	D –	Jump to Address on D	JMP D

X = Don't care, 0 = LOW, 1 = HIGH, Assume C_n = HIGH
Note: STK0 is the location addressed by the stack pointer.

Source: AMD, 1976a, page 2-6

TABLE 6.2

Am29811 INSTRUCTION SET

MNEMONIC	I_3 I_2 I_1 I_0	INSTRUCTION
JZ	L L L L	Jump to Address Zero
CJS	L L L H	Conditional Jump-to-Subroutine with Jump Address in Pipeline Register.
JMAP	L L H L	Jump to Address at Mapping PROM Output.
CJP	L L H H	Conditional Jump to Address in Pipeline Register
PUSH	L H L L	Push Stack and Conditionally Load Counter
JSRP	L H L H	Jump-to-Subroutine with Starting Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
CJV	L H H L	Conditional Jump to Vector Address.
JRP	L H H H	Jump to Address Conditionally Selected from Am2911 R-Register or Pipeline Register.
RFCT	H L L L	Repeat Loop if Counter is not Equal to Zero.
RPCT	H L L H	Repeat Pipeline Address if Counter is not Equal to Zero.
CRTN	H L H L	Conditional Return-from-Subroutine.
CJPP	H L H H	Conditional Jump to Pipeline Address and Pop Stack.
LDCT	H H L L	Load Counter and Continue.
LOOP	H H L H	Test End of Loop.
CONT	H H H L	Continue to Next Address.
JP	H H H H	Jump to Pipeline Register Address.

Source: AMD, 1976a, page 2-19

TABLE 6.3

Am29811 FUNCTION TABLE

MNEMONIC	INPUTS			OUTPUTS				
	INSTRUCTION I_3 I_2 I_1 I_0	FUNCTION	TEST INPUT	NEXT ADDR SOURCE	FILE	COUNTER	MAP-E	PL-E
JZ	L L L L	JUMP ZERO	X	D	HOLD	LL*	H	L
CJS	L L L H	COND JSB PL	L	PC	HOLD	HOLD	H	L
			H	D	PUSH	HOLD	H	L
JMAP	L L H L	JUMP MAP	X	D	HOLD	HOLD	L	H
CJP	L L H H	COND JUMP PL	L	PC	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
PUSH	L H L L	PUSH/COND LD CNTR	L	PC	PUSH	HOLD	H	L
			H	PC	PUSH	LOAD	H	L
JSRP	L H L H	COND JSB R/PL	L	R	PUSH	HOLD	H	L
			H	D	PUSH	HOLD	H	L
CJV	L H H L	COND JUMP VECTOR	L	PC	HOLD	HOLD	H	H
			H	D	HOLD	HOLD	H	H
JRP	L H H H	COND JUMP R/PL	L	R	HOLD	HOLD	H	L
			H	D	HOLD	HOLD	H	L
RFCT	H L L L	REPEAT LOOP, CNTR ≠ 0	L	F	HOLD	DEC	H	L
			H	PC	POP	HOLD	H	L
RPCT	H L L H	REPEAT PL, CNTR ≠ 0	L	D	HOLD	DEC	H	L
			H	PC	HOLD	HOLD	H	L
CRTN	H L H L	COND RTN	L	PC	HOLD	HOLD	H	L
			H	F	POP	HOLD	H	L
CJPP	H L H H	COND JUMP PL & POP	L	PC	HOLD	HOLD	H	L
			H	D	POP	HOLD	H	L
LDCT	H H L L	LOAD CNTR & CONTINUE	X	PC	HOLD	LOAD	H	L
LOOP	H H L H	TEST END LOOP	L	F	HOLD	HOLD	H	L
			H	PC	POP	HOLD	H	L
CONT	H H H L	CONTINUE	X	PC	HOLD	HOLD	H	L
JP	H H H H	JUMP PL	X	D	HOLD	HOLD	H	L

L = LOW
H = HIGH
X = Don't Care

DEC = Decrement
*LL = Special Case

Source: AMD, 1976a, page 2-20

TABLE 6.4

Am29811 TRUTH TABLE

MNEMONIC	FUNCTION	INPUTS					OUTPUTS							
							NEXT ADDR SOURCE		FILE		COUNTER			
		I_3	I_2	I_1	I_0	TEST	S_1	S_0	$\overline{FE}$	PUP	$\overline{LOAD}$	$\overline{EN}$	$\overline{MAP\ E}$	$\overline{PL\ E}$
	PIN NO.	14	13	12	11	10	4	5	3	2	6	7	1	9
JZ	JUMP ZERO	L	L	L	L	L	H	H	H	H	L	L	H	L
		L	L	L	L	H	H	H	H	H	L	L	H	L
CJS	COND JSB PL	L	L	L	H	L	L	L	H	H	H	H	H	L
		L	L	L	H	H	H	H	L	H	H	H	H	L
JMAP	JUMP MAP	L	L	H	L	L	H	H	H	H	H	H	L	H
		L	L	H	L	H	H	H	H	H	H	H	L	H
CJP	COND JUMP PL	L	L	H	H	L	L	L	H	H	H	H	H	L
		L	L	H	H	H	H	H	H	H	H	H	H	L
PUSH	PUSH/COND LD CNTR	L	H	L	L	L	L	L	L	H	H	H	H	L
		L	H	L	L	H	L	L	L	H	L	H	H	L
JSRP	COND JSB R/PL	L	H	L	H	L	L	H	L	H	H	H	H	L
		L	H	L	H	H	H	H	L	H	H	H	H	L
CJV	COND JUMP VECTOR	L	H	H	L	L	L	L	H	H	H	H	H	H
		L	H	H	L	H	H	H	H	H	H	H	H	H
JRP	COND JUMP R/PL	L	H	H	H	L	L	H	H	H	H	H	H	L
		L	H	H	H	H	H	H	H	H	H	H	H	L
RFCT	REPEAT LOOP, CTR ≠ 0	H	L	L	L	L	H	L	H	L	H	L	H	L
		H	L	L	L	H	L	L	L	L	H	H	H	L
RPCT	REPEAT PL, CTR ≠ 0	H	L	L	H	L	H	H	H	H	H	L	H	L
		H	L	L	H	H	L	L	H	H	H	H	H	L
CRTN	COND RTN	H	L	H	L	L	L	L	H	L	H	H	H	L
		H	L	H	L	H	H	L	L	L	H	H	H	L
CJPP	COND JUMP PL & POP	H	L	H	H	L	L	L	H	L	H	H	H	L
		H	L	H	H	H	H	H	L	L	H	H	H	L
LDCT	LD CNTR & CONTINUE	H	H	L	L	L	L	L	H	H	L	H	H	L
		H	H	L	L	H	L	L	H	H	L	H	H	L
LOOP	TEST END LOOP	H	H	L	H	L	H	L	H	L	H	H	H	L
		H	H	L	H	H	L	L	L	L	H	H	H	L
CONT	CONTINUE	H	H	H	L	L	L	L	H	H	H	H	H	L
		H	H	H	L	H	L	L	H	H	H	H	H	L
JP	JUMP PL	H	H	H	H	L	H	H	H	H	H	H	H	L
		H	H	H	H	H	H	H	H	H	H	H	H	L

L = LOW
H = HIGH

Source: AMD, 1976a, page 2-20

The result is rather like pulling yourself up by your boot straps. The sequencer is the heart of the micro-programmed architecture, but it is controlled by the next address circuit. The next address circuit is controlled by the pipeline register, which is controlled by the microstore. Finally, the microstore is controlled by the sequencer and the circle is complete.

6.3 MICROPROGRAM MEMORY AND THE PIPELINE REGISTER

Once the proper address is chosen for the microprogram memory (the source of the address may be the mapping PROM, the pipeline register, or the sequencer), the address is transmitted to the microprogram memory's address lines. For the development system the microprogram memory will be ultra violet erasable PROM and will require 450 ns before the data appears at the output of the microprogram memory.

When the data appears at the output of the micro-program it is loaded into the pipeline register. The function of the pipeline register can be seen as a latch into which the microprogram word is stored. While the microprogram word is stored in this latch two actions take place. First the current microprogram word, the one in the pipeline register, is executed. At the same time, the sequencer is instructed to fetch the next microprogram word. By the time the current microprogram is finished executing, the next microinstruction will be waiting at the input to the pipeline register.

As can be seen from the computer design (figure 6.1), the microprogram word is rather long. The microprogram word is composed of the following major fields:

1. Test Condition Field - This field contains the code that selects one of the 16 test inputs.
2. Polarity Field - This field determines the polarity of the test input chosen.
3. IR Field - This field is used to load the instruction register.
4. Microprogram Branch Field - This field provides the next address for a microprogram branch.
5. ALU Source Field - This field selects two data sources for the ALU function from among the 16 RALU registers, Q register, data inputs, and zero.
6. ALU Destination Field - This field selects one of the RALU's 16 general purpose registers for the destination of the results of the current ALU operation.
7. ALU Function Field - This field selects one of the eight ALU functions to operate on the source data.
8. SALU Field - This field enables the ALU outputs.
9. Shift Field - This field determines how the shift operation will be performed in the ALU.
10. ALU Carry Field - This field determines how the

carry will be used in the ALU.

11. PCU Carry Field - This field determines how the carry will be used in the program control unit.
12. PCU Address Field - This field determines the function of the PCU.
13. $\overline{\text{AE}}$ Field - This field enables the output of the PCU to the MAR.
14. MODE Field - The mode field determines the mode of the Mapping PROM. It specifies whether the Mapping PROM is to be used as a mapping function or as one of the decoding functions.

6.4 RALU

The output from the various fields of the pipeline register drives the functional blocks of the computer. One of the blocks is the 2901 RALU which performs all the arithmetrical and logical operations required by the computer. The RALU contain 16 general purpose registers (the registers are implemented with dual port RAM), a Q register (for temporary storage such as required in multiplication), an eight function ALU, and data routing circuits. The interaction of the various RALU components is controlled by the contents of the ALU source, destination, and function fields of the pipeline register. Table 6.5 contains the various RALU functions that are

TABLE 6.5

ALU OPERATION

MICRO CODE				ALU SOURCE OPERANDS	
I_2	I_1	I_0	Octal Code	R	S
L	L	L	0	A	Q
L	L	H	1	A	B
L	H	L	2	0	Q
L	H	H	3	0	B
H	L	L	4	0	A
H	L	H	5	D	A
H	H	L	6	D	Q
H	H	H	7	D	0

ALU Source Operand Control.

MICRO CODE				ALU Function	Symbol
I_5	I_4	I_3	Octal Code		
L	L	L	0	R Plus S	$R + S$
L	L	H	1	S Minus R	$S - R$
L	H	L	2	R Minus S	$R - S$
L	H	H	3	R OR S	$R \vee S$
H	L	L	4	R AND S	$R \wedge S$
H	L	H	5	$\bar{R}$ AND S	$\bar{R} \wedge S$
H	H	L	6	R EX-OR S	$R \veebar S$
H	H	H	7	R EX-NOR S	$\overline{R \veebar S}$

ALU Function Control.

MICRO CODE				RAM FUNCTION		Q-REG. FUNCTION		Y OUTPUT	RAM SHIFTER		Q SHIFTER	
I_8	I_7	I_6	Octal Code	Shift	Load	Shift	Load		RAM_0	RAM_3	Q_0	Q_3
L	L	L	0	X	NONE	NONE	$F \rightarrow Q$	F	X	X	X	X
L	L	H	1	X	NONE	X	NONE	F	X	X	X	X
L	H	L	2	NONE	$F \rightarrow B$	X	NONE	A	X	X	X	X
L	H	H	3	NONE	$F \rightarrow B$	X	NONE	F	X	X	X	X
H	L	L	4	DOWN	$F/2 \rightarrow B$	DOWN	$Q/2 \rightarrow Q$	F	F_0	IN_3	Q_0	IN_3
H	L	H	5	DOWN	$F/2 \rightarrow B$	X	NONE	F	F_0	IN_3	Q_0	X
H	H	L	6	UP	$2F \rightarrow B$	UP	$2Q \rightarrow Q$	F	IN_0	F_3	IN_0	Q_3
H	H	H	7	UP	$2F \rightarrow B$	X	NONE	F	IN_0	F_3	X	Q_3

X = Don't care. Electrically, the shift pin is a TTL input internally connected to a three-state output which is in the high-impedance state.
B = Register Addressed by B Inputs.
Up is toward MSB, Down is toward LSB.

ALU Destination Control.

OCTAL I_{543}	I_{210} OCTAL	0	1	2	3	4	5	6	7
	ALU Source / ALU Function	A, Q	A, B	0, Q	0, B	0, A	D, A	D, Q	D, 0
0	C_n = L R Plus S	$A+Q$	$A+B$	Q	B	A	$D+A$	$D+Q$	D
	C_n = H	$A+Q+1$	$A+B+1$	$Q+1$	$B+1$	$A+1$	$D+A+1$	$D+Q+1$	$D+1$
1	C_n = L S Minus R	$Q-A-1$	$B-A-1$	$Q-1$	$B-1$	$A-1$	$A-D-1$	$Q-D-1$	$-D-1$
	C_n = H	$Q-A$	$B-A$	Q	B	A	$A-D$	$Q-D$	$-D$
2	C_n = L R Minus S	$A-Q-1$	$A-B-1$	$-Q-1$	$-B-1$	$-A-1$	$D-A-1$	$D-Q-1$	$D-1$
	C_n = H	$A-Q$	$A-B$	$-Q$	$-B$	$-A$	$D-A$	$D-Q$	D
3	R OR S	$A \vee Q$	$A \vee B$	Q	B	A	$D \vee A$	$D \vee Q$	D
4	R AND S	$A \wedge Q$	$A \wedge B$	0	0	0	$D \wedge A$	$D \wedge Q$	0
5	$\bar{R}$ AND S	$\bar{A} \wedge Q$	$\bar{A} \wedge B$	Q	B	A	$\bar{D} \wedge A$	$\bar{D} \wedge Q$	0
6	R EX-OR S	$A \veebar Q$	$A \veebar B$	Q	B	A	$D \veebar A$	$D \veebar Q$	D
7	R EX-NOR S	$\overline{A \veebar Q}$	$\overline{A \veebar B}$	$\bar{Q}$	$\bar{B}$	$\bar{A}$	$\overline{D \veebar A}$	$\overline{D \veebar Q}$	$\bar{D}$

+ = Plus; − = Minus; $\vee$ = OR; $\wedge$ = AND; $\veebar$ = EX-OR

Source Operand and ALU Function Matrix.

Source: AMD, 1976, page 8

commanded by the pipeline register.

6.5 PCU

The Program Control Unit (PCU) is the final block to be discussed. The PCU is shown in figure 6.8. The PCU is implemented using a special integrated circuit (Am2930) designed for controlling the program addresses. Like so many other functional blocks in the 2900 computer, it too has a very powerful instruction set (see table 6.6). The PCU allows the program counter to be incremented by only one control bit. Other addressing modes, such as relative, indexed, and base page, can be handled almost as easily. In addition, the PCU provides for easy implementation of such operations as jump to subroutine, return from subroutine, and the associated stack manipulations.

TABLE 6.6

PCU INSTRUCTION SET

Instruction Number	I_4	I_3	I_2	I_1	I_0	$\overline{CC}$	$\overline{IEN}$	Instruction	Y_0-Y_3	Next State (after CP ⨍) (Note 3)				
											R			
										PC	$\overline{RE}$ = L	$\overline{RE}$ = H	RAM	SP
	X	X	X	X	X	X	H	Instruction Disable	Note 1	–	D	–	–	–
0	L	L	L	L	L	X	L	RESET	"0"	"0"+C_i	D	–	–	Reset
1	L	L	L	L	H	X	L	FETCH PC	PC	PC+C_i	D	–	–	–
2	L	L	L	H	L	X	L	FETCH R	R	PC+C_i	D	–	–	–
3	L	L	L	H	H	X	L	FETCH D	D	PC+C_i	D	–	–	–
4	L	L	H	L	L	X	L	FETCH R+D	R+D+C_n	PC+C_i	D	–	–	–
5	L	L	H	L	H	X	L	FETCH PC+D	PC+D+C_n	PC+C_i	D	–	–	–
6	L	L	H	H	L	X	L	FETCH PC+R	PC+R+C_n	PC+C_i	D	–	–	–
7	L	L	H	H	H	X	L	FETCH S+D	S+D+C_n	PC+C_i	D	–	–	–
8	L	H	L	L	L	X	L	FETCH PC → R	PC	PC+C_i	PC	PC	–	–
9	L	H	L	L	H	X	L	FETCH R+D → R	R+D+C_n	PC+C_i	R+D+C_n	R+D+C_n	–	–
10	L	H	L	H	L	X	L	LOAD R	PC	PC+C_i	D	D	–	–
11	L	H	L	H	H	X	L	PUSH PC	PC	PC+C_i	D	–	PC → Loc SP+1	SP+1
12	L	H	H	L	L	X	L	PUSH D	PC	PC+C_i	D	–	D → Loc SP+1	SP+1
13	L	H	H	L	H	X	L	POP S	S	PC+C_i	D	–	–	SP–1
14	L	H	H	H	L	X	L	POP PC	PC	PC+C_i	D	–	–	SP–1
15	L	H	H	H	H	X	L	HOLD	PC	–	D	–	–	
16-31	H	X	X	X	X	H	L	FAIL COND'L TEST (FETCH PC)	PC	PC+C_i	D	–	–	–
16	H	L	L	L	L	L	L	JUMP R	R	R+C_i	D	–	–	–
17	H	L	L	L	H	L	L	JUMP D	D	D+C_i	D	–	–	–
18	H	L	L	H	L	L	L	JUMP "0"	"0"	"0"+C_i	D	–	–	–
19	H	L	L	H	H	L	L	JUMP R+D	R+D+C_n	R+D+C_n+C_i	D	–	–	–
20	H	L	H	L	L	L	L	JUMP PC+D	PC+D+C_n	PC+D+C_n+C_i	D	–	–	–
21	H	L	H	L	H	L	L	JUMP PC+R	PC+R+C_n	PC+R+C_n+C_i	D	–	–	–
22	H	L	H	H	L	L	L	JSB R	R	R+C_i	D	–	PC → Loc SP+1	SP+1
23	H	L	H	H	H	L	L	JSB D	D	D+C_i	D	–	PC → Loc SP+1	SP+1
24	H	H	L	L	L	L	L	JSB "0"	"0"	"0"+C_i	D	–	PC → Loc SP+1	SP+1
25	H	H	L	L	H	L	L	JSB R+D	R+D+C_n	R+D+C_n+C_i	D	–	PC → Loc SP+1	SP+1
26	H	H	L	H	L	L	L	JSB PC+D	PC+D+C_n	PC+D+C_n+C_i	D	–	PC → Loc SP+1	SP+1
27	H	H	L	H	H	L	L	JSB PC+R	PC+R+C_n	PC+R+C_n+C_i	D	–	PC → Loc SP+1	SP+1
28	H	H	H	L	L	L	L	RETURN S	S	S+C_i	D	–	–	SP–1
29	H	H	H	L	H	L	L	RETURN S+D	S+D+C_n	S+D+C_n+C_i	D	–	–	SP–1
30	H	H	H	H	L	L	L	HOLD	PC	–	D	–	–	–
31	H	H	H	H	H	L	L	SUSPEND	Z (Note 2)	–	D	–	–	–

PC – Program Counter SP – Stack Pointer
R – Auxiliary Register D – Direct Inputs

Notes: 1. When $\overline{IEN}$ is HIGH, the Y_0-Y_3 outputs contain the same data as when $\overline{IEN}$ is LOW, as determined by I_0-I_4 and $\overline{CC}$.
2. Z = High impedance state (outputs "OFF").
3. – = No change

Source: AMD, 1977a

7.0 SYSTEM TIMING

As menitioned earlier, this computer was designed with high performance as a goal. However, for economy the microprogramming was done in ultraviolet erasible EPROM. This approach saves money because a new set of PROMs is not required each time the microcode is changed, but the computer runs rather slow as a result of the 450 ns access time of the 2708 EPROM. In order to present a clear picture of the system timing, two timing diagrams are given. The timing diagrams for the experimental system (figure 7.1) and for the potentially high performance system (figure 7.2) are presented along with a brief discussion of each diagram.

7.1 TIMING FOR THE EXPERIMENTAL SYSTEM

In order to establish the timing characteristics of the computer, the chain of events that take the longest time to complete must be identified. This chain of events is the critical timing path for the computer. By definition no computer operation can take longer to complete than the critical path. The critical path is the limiting factor for the speed of the computer. In other words, the time of one computer cycle can be no longer than the time required for the critical path.

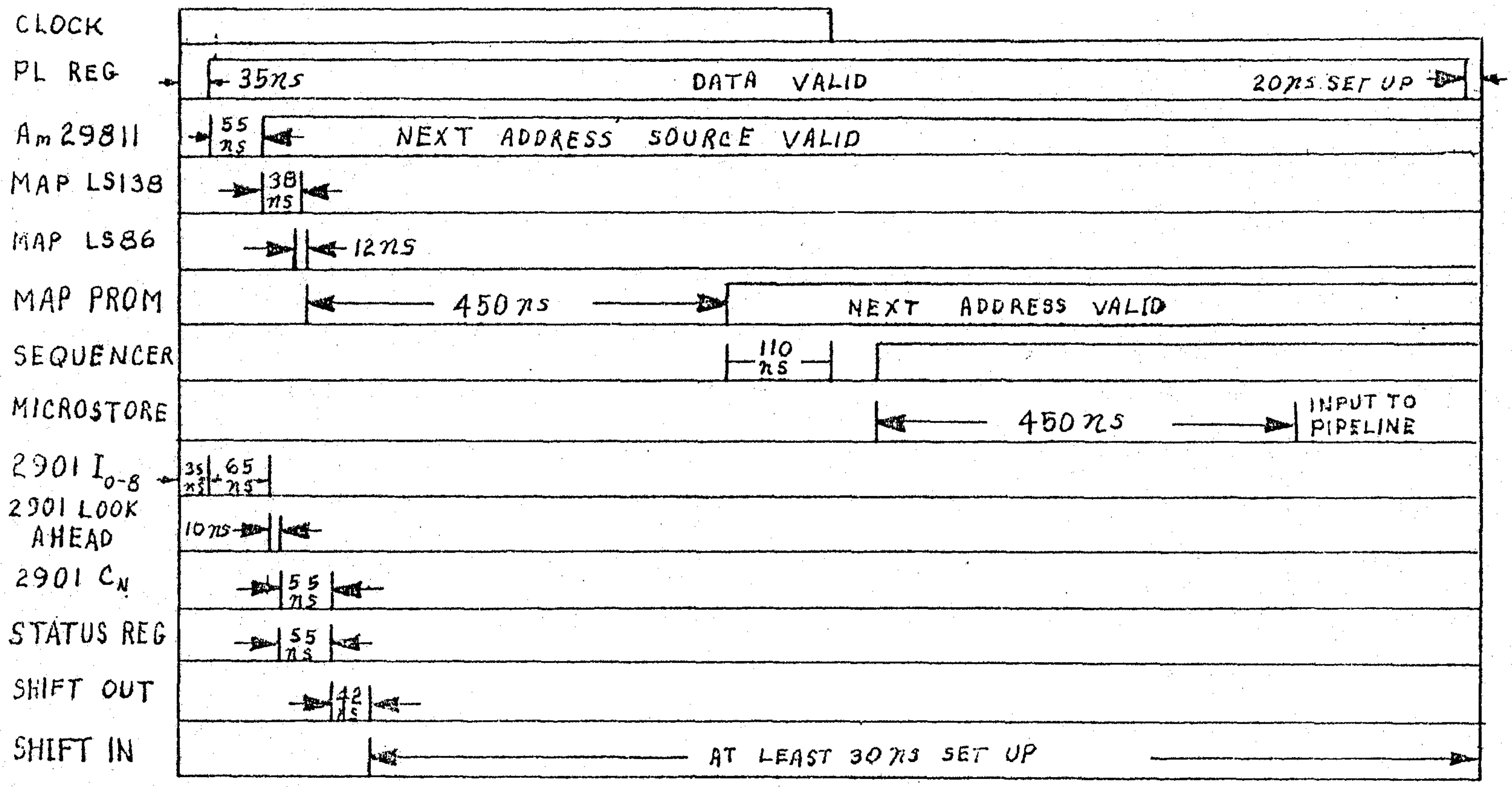


FIGURE 7.1 Timing diagram for the experimental system

The computer cycle begins on the leading edge of the clock. On the leading edge the microinstruction is clocked into the pipeline register. As can be seen from figure 7.1 (all times are shown as worst case times), the use of the 2708 EPROM for the microstore and again for the Mapping PROM in the decoder mode causes large delays. Figure 7.1 is set up for a 1.4 microsecond cycle, but the time could have been shortened by 260 nanoseconds.

7.2 HIGH PERFORMANCE SYSTEM

Since the objective is to design a high performance minicomputer, the experience gained from the experimental system needs to be examined to gain a reasonable expectation of the potential performance of the 2900 system. In the experimental version, 2708 EPROM's were used, and since these devices were so slow, low power Schottky registers and counters were used to save cost and power. However, for the full performance version envisioned, 50 ns PROM or ROM and full speed Schottky would be required. This is the premise of the timing presented in figure 7.2. Figure 7.2 uses the same time scale as figure 7.1 to give a better illustration of the performance increases gained by these changes.

The basic cycle time of the high performance computer is 270 ns. With the overhead for checking for halts and

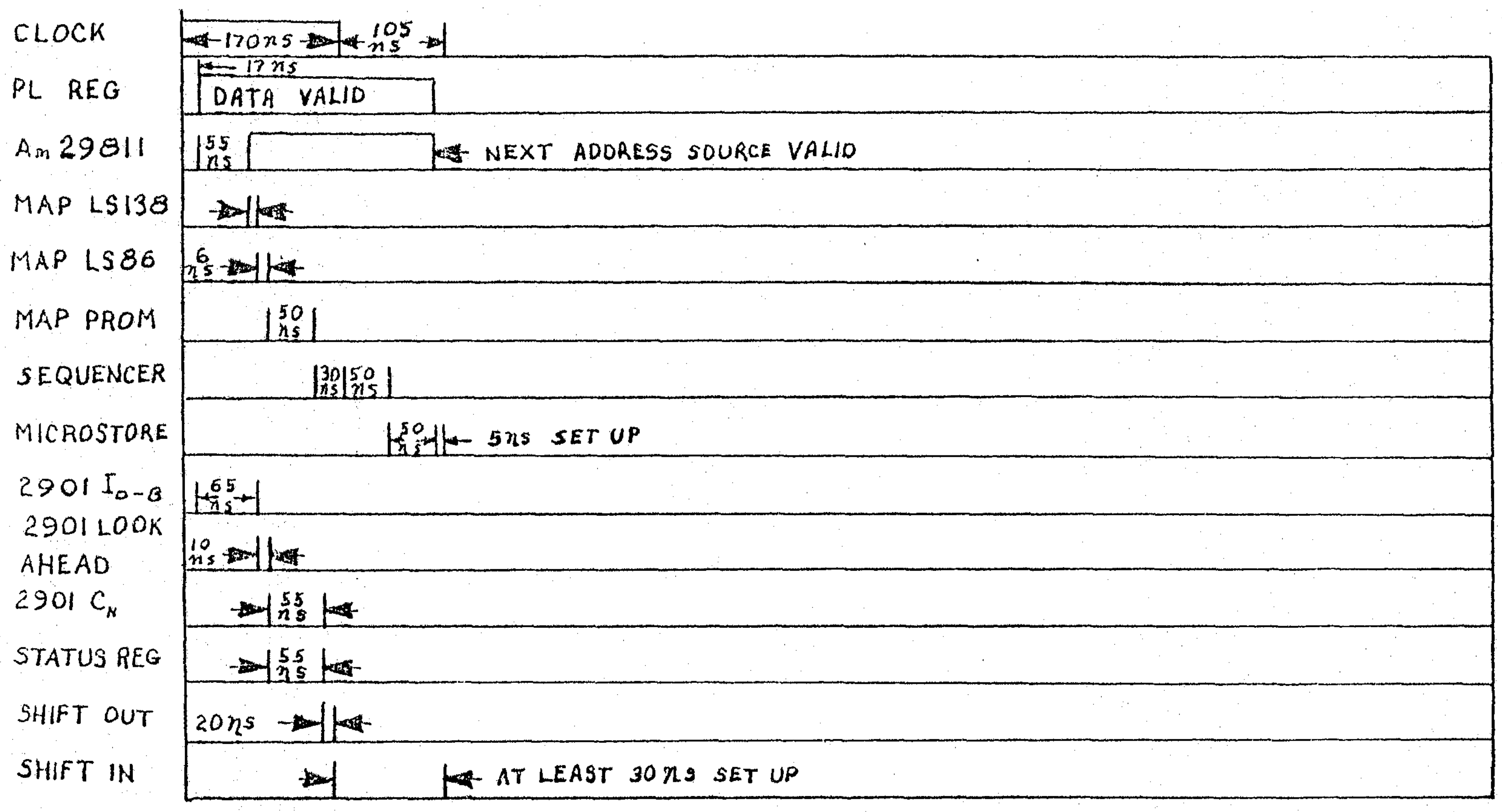


FIGURE 7.2 Timing diagram for the full performance system

interrupts, an ADD without a shift or skip can be done in 1.08 microseconds. While this is representative of modern minicomputer performance, it is not as impressive as was hoped for when the design started. The fault lies in the decoding scheme used for indirect addressing, relative addressing, mode 1, and the various other modes of operation that are decoded. As mentioned earlier, the decoding scheme drastically cuts the amount of Mapping PROM and microstore PROM needed, but what are the costs of this scheme? When compared to the 1.5 microsecond ADD time of Fairchild's 9440 single chip NOVA microprocessor, it is not clear that the improvement gain by using a micro-programmed bit slice technology is worth the cost in time and money.

Two approaches can be taken to increase speed. One approach would allow for extra time in the cycle only when the Mapping PROM is called as the next address source. The other approach would use extra Mapping PROMs to provide a 12 bit address that could directly address 4096 words of microstore. In the latter approach, each variation of an instruction would be addressed directly. This would eliminate the need to decode each instruction, but would add to the microprogramming task. As a result, the ADD time would drop to about 880 ns, and with newer and faster RALUs and sequencers, cycle time less than 800 ns could be achieved. This figure is more in the range of performance that was first envisioned.

7.3 COMPARISON OF THE 2900 SYSTEM TO EXISTING SYSTEMS

In table 7.1 comparisons are made between some of the existing variations of NOVA type machines. In this comparison the BLAZE, which is Fairchild's 9400 bit slice (see section 2.5) emulation of the NOVA 3, and the NOVA 3 are in the same class as the 2900 emulation of NOVA. The SPARK, FLAME, and NOVA 1200 are marginally slower than the 2900 emulation. The conclusion is that the design of the 2900 system is faster than present single chip NOVA implementations, and is in fact just as fast as other bipolar implementations.

TABLE 7.1

NOVA COMPUTER COMPARISONS

INSTRUCTION	SPEED IN MICROSECONDS			
	BLAZE-16 (a)	NOVA 3 (b)	9440 SPARK-16 FLAME-16 (c)	NOVA 1200
Load Accumulator	1.0	1.1 - 2.0	2.41	2.55
Store Accumulator	1.0	1.1 - 2.0	3.66	2.55
ISZ, DSZ	1.4	1.6 - 2.4	3.66	3.15
Jump	0.6	0.7 - 1.0	1.25	1.35
Jump to Subroutine	1.2	1.0 - 1.2	1.25	1.35
Add	1.0	0.7 - 1.0	1.25	1.35
Subtract	1.0	0.7 - 1.0	1.25	1.35
And	1.0	0.7 - 1.0	1.25	1.35
Move	1.0	0.7 - 1.0	1.25	1.35
+ Skip	0	0.3	1.25	1.35
I/O Input	1.6	2.0 - 2.2	2.08	2.55
I/O Output	1.6	2.0 - 2.2	2.08	3.15

(a) Oscillator frequency -- 10 MHz, memory Read cycle — 400 ns.
(b) Minimum for semiconductor memory, maximum for 16K core.
(c) Oscillator frequency — 12 MHz, Memory Busy < 120 ns.

Source: Suri, 1977, page 10

8.0 MICROPROGRAM

Because of the amount of time required to write microcode only a few of the microinstructions were microcoded. The microcode, timing diagrams, and comments for these instructions are included in the following figures. The microcode is also presented as it appears in the microstore memory.

The following are some of the constraints adopted for writing microcode:

1. The code must be as short as possible.
2. The code must generate the memory address and the memory read signal as far in advance as possible. This practice allows slower memories to be used.
3. The microcode must do as much in parallel as possible. This will speed up the throughput.

The application of the first constraint can best be illustrated by comparison of the ADD instructions. The original ADD instruction (figure 8.1) was quite slow because of the repeated use of jumps to subroutines to accomplish the various modes of the ADD instruction. For instance, the ADD instruction jumps to a shift subroutine in T1 and then jumps to a skip subroutine in T2. Each of these jumps requires at least two steps: the jump and the return. As a result, eight micro-cycles are required whereas the final ADD instruction (figure 8.2) actually uses only four microcycles. Instead of jumping to and from a subroutine, the microcode jumps to the shift subroutine then jumps directly to the skip subroutine under control of the Mapping PROM. Finally the microcode jumps directly to the fetch subroutine under control of the pipeline branch address.

The second constraint is shown clearly in the timing diagram for the FETCH instruction (figure 8.7). Here one can see the $\overline{\text{MR}}$ and LD MAR lines go low during T1. This generates the memory read signal and the memory address during T1. However, the memory is not needed until the IR LOAD in T3 which allows the memory two microcycles for set up. At the 270 ns cycle time, even cheap 500 ns memory can be used. If the faster, 200 ns cycle time, CCU suggested in section 7.2 is used, memory as slow as 400 ns can be used.

The third constraint is shown in the ADD instruction. For instance, in the ADD with no shift and no skip, cycle T0 contains a fetch and increment of the program counter, a read from RAM ports A and B with a store back into port B, and a mode 2 jump to the shift subroutine. That is a lot of work to do in one microinstruction.

ADD (original)	TIME: 2.16 microseconds min.	LOCATION 024_{16}
CYCLE	MICROCODE	COMMENT
T0	B←A+B, Fetch PC_1, LD CARRY	Add A plus B and load the sum into B as addressed by the IR. Output the present PC and increment by 1. Load the carry from the add into the carry bit.
T1	MODE 2 ← 1, CJS ONET MAP	Jump to the MODE 2 subroutine given by the jump address in the Mapping PROM to get the proper shift mode.
T2	MODE 3 ← 1, CJS ONET MAP	Jump to the MODE 3 subroutine given by the jump address in the Mapping PROM to get the proper skip mode.
T3	MAR ← Y, $\overline{MR}$, JP FETCH(002)	Load the PC into the MAR and send a memory read signal. Jump to T2 in the FETCH subroutine.
T4	see FETCH	
T5		

FIGURE 8.1 RTL and comments for original ADD microcode

ADD with NO SHIFT and NO SKIP	TIME: 1.08 microseconds	LOCATION 028_{16}
CYCLE	MICROCODE	COMMENT
T0	MODE 2 $\leftarrow$ 1, B $\leftarrow$ A+B Fetch PC_1 , CJS ONET MAP	Add A plus B and load the sum into B as addressed by the IR. Output the present PC and increment by 1. Jump to the MODE 2 subroutine given by the jump address in the Mapping PROM to get the proper shift mode.
T1	MODE 3 $\leftarrow$ 1, CJS ONET MAP	Jump to the MODE 3 subroutine given by the jump address in the MAPPING PROM to get the proper skip mode.
T2	JP FETCH(003), MAR $\leftarrow$ Y, $\overline{MR}$ HALTC $\leftarrow$ 1, IR $\leftarrow$ DATA BUS	Send out a memory read signal and load the PC into the MAR. Load the Data Bus into the IR and check for a HALT command.
T3	see FETCH	

FIGURE 8.2 RTL and comments for ADD with NO SHIFT and NO SKIP microcode

ADD with LEFT ROTATE and SKIP	TIME: 1.62 microseconds	LOCATION 028_{16}
CYCLE	MICROCODE	COMMENT
T0	MODE 2 ← 1, B ← A+B	Add A plus B and load the sum into B as addressed by the IR. Output the present PC and increment by 1. Jump to the MODE 2 subroutine given by the jump address in the Mapping PROM to get the proper shift mode.
T1	MODE 3 ← 1, B ← SHL B $B_0 \leftarrow B_{15}$, $\overline{MR}$, CJS ONET MAP	Jump to the MODE 3 subroutine given by the jump address in the Mapping PROM to get the proper skip mode. Shift RAM B left one and rotate the MSB to the LSB. Send out a memory read signal.
T2	JP FETCH(001), Fetch PC_1	Output the present PC and increment by 1 to accomplish a skip.
T3		
T4	see FETCH	
T5		

FIGURE 8.3 RTL and comments for ADD with LEFT ROTATE and SKIP microcode

LDA	TIME: 2.16 microseconds min.	LOCATION $01C_{16}$
CYCLE	MICROCODE	COMMENT
T0	MODE 1 $\leftarrow$ 1, CJS ONET MAP	Jump to the MODE 1 subroutine given by the jump address in the Mapping PROM to get the proper addressing mode.
T1	$\overline{\text{MR}}$, MAR $\leftarrow$ Y	Send out a memory read signal and load the effective address into the MAR.
T2	$\overline{\text{MR}}$, MBRO $\leftarrow$ DATA BUS	Send out a memory read signal and load the DATA BUS into the MBRO.
T3	ACC $\leftarrow$ MBRO, JP FETCH(000)	The MBRO is loaded into the accumulator addressed by the IR. Jump to FETCH.
T4		
T5	see FETCH	
T6		
T7		

FIGURE 8.4 RTL and comments for LOAD ACCUMULATOR microcode

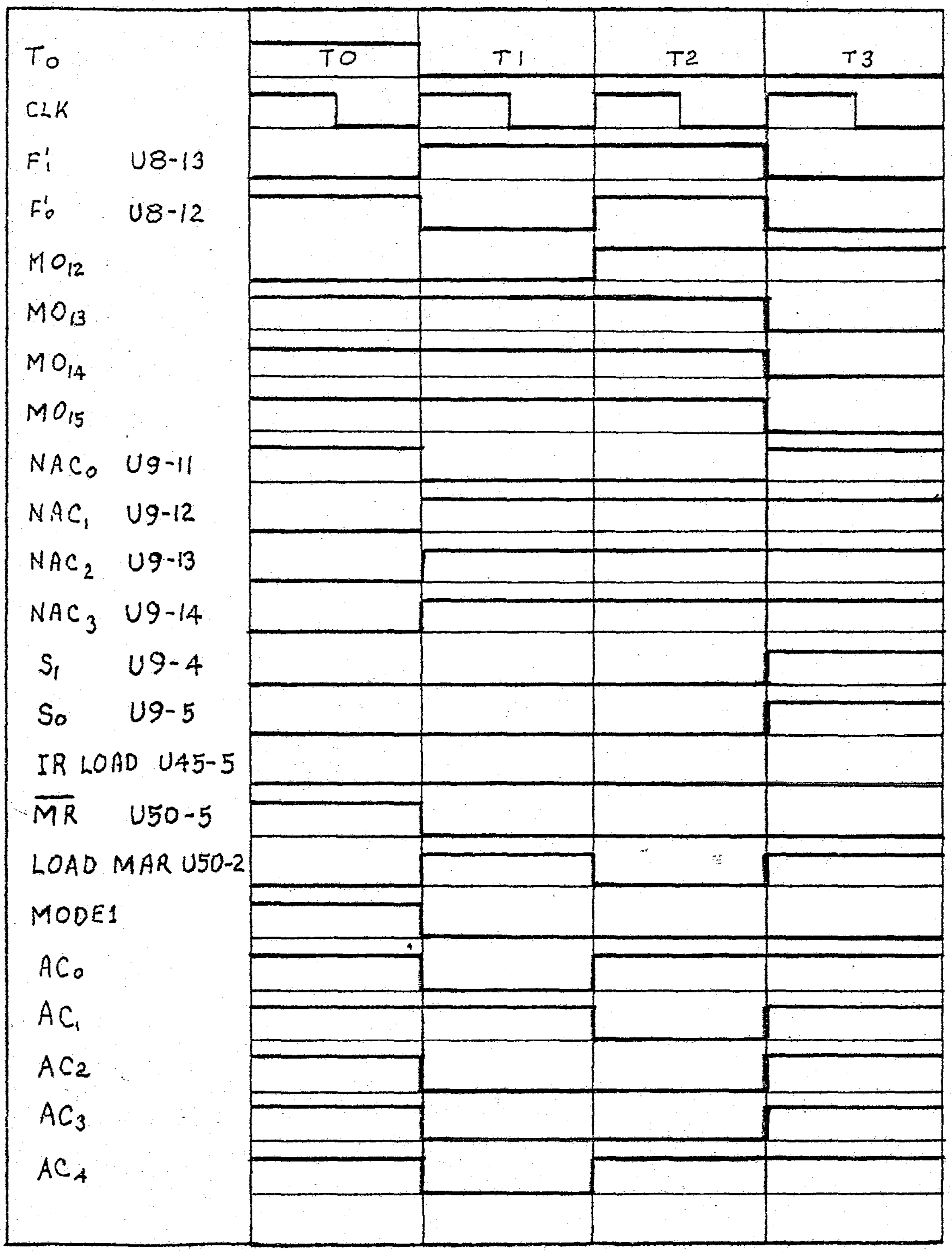


FIGURE 8.5 Timing for LDA

FETCH	TIME: 1.08 microseconds	LOCATION 000_{16}
CYCLE	MICROCODE	COMMENT
T0	Fetch PC_1	Output the present PC and increment by 1.
T1	MAR $\leftarrow$ Y, $\overline{MR}$	Load PC into MAR and send out a memory read signal.
T2	HALTC $\leftarrow$ 1, $\overline{MR}$ IR $\leftarrow$ DATA BUS	Checks for a HALT command, sends out a memory read signal, and loads the IR from the DATA BUS.
T3	JSRP INT MAP INTS	If there is an interrupt (INT) then jump to the interrupt subroutine (INTS) given by the Pipeline Register, else jump to the next instruction's microcode which is given by the jump address in the Mapping PROM.

FIGURE 8.6 RTL and comments for the FETCH microcode

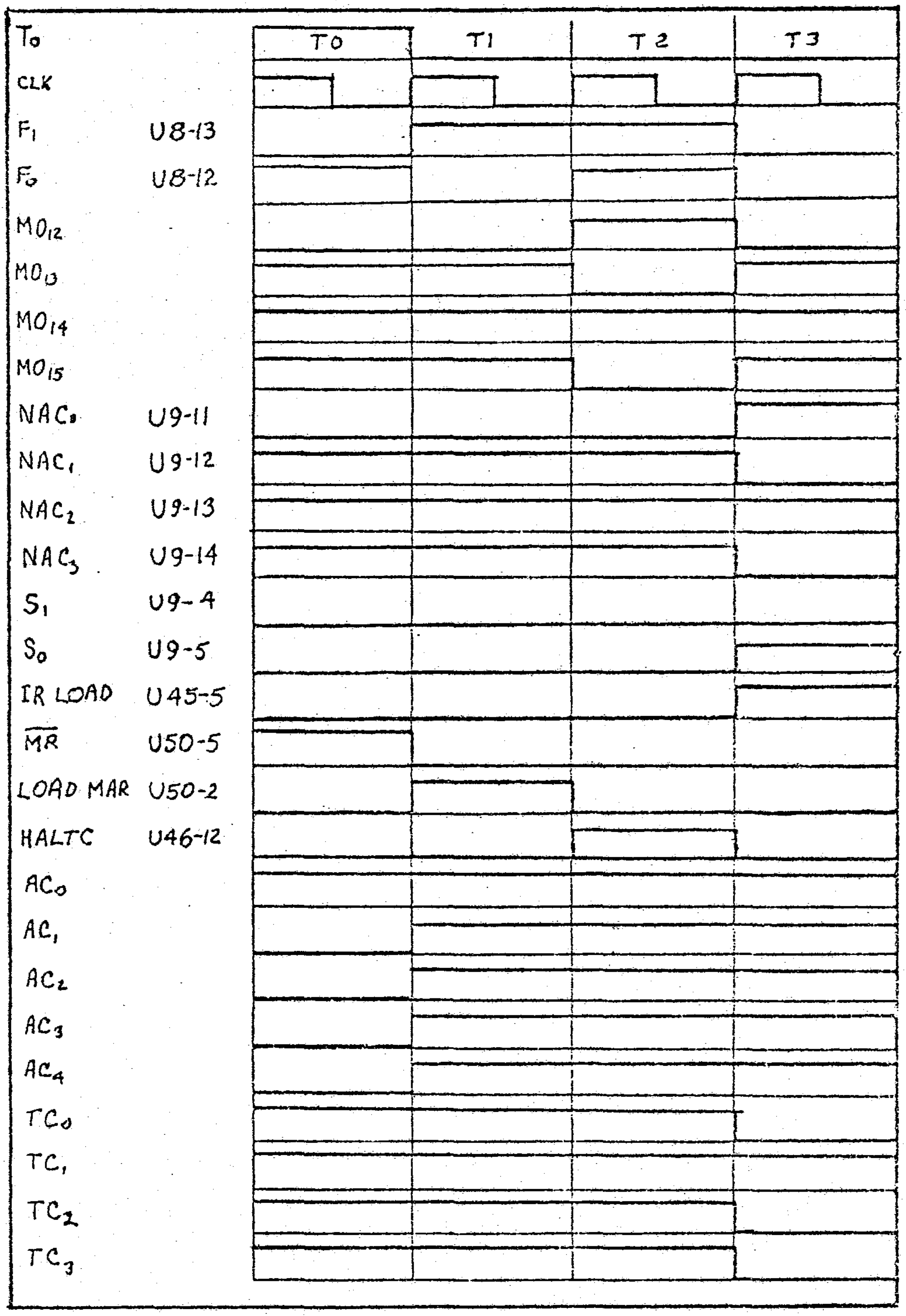


FIGURE 8.7 Timing for FETCH

BRANCH ADDR	0	1	1	1		1	0			1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	0				
	1	1	1	1	D	1	1			1	1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				
	2	1	1	1	D	1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	3	1	1	1	A	1	0			1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0				
	4	1	1	1		1	0			1	0	1	1	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	5	1	1	1	T	1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	6	1	1	1	N	1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	7	1	1	1	I	1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	8	1	1	1		1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
	9	1	1	1		1	0			1	0	0	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1	1	0				
POLARITY	10	0	0	0	1	0	0			0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
IR LOAD	11	0	0	0	1	0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
NEXT ADDR	12	0	0	0	1	1	1			1	1	1	1	1	0	0	0	1	0	0	1	1	0	0	1	1	0	0	1				
	13	1	1	1	0	0	1			0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				
	14	1	1	1	1	0	1			0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				
	15	1	1	1	0	0	1			0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				
TEST COND	16	1	1	1	0	1	1			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				
	17	1	1	1	1	0	1			0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1				
	18	1	1	1	0	0	1			0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				
	19	1	1	1	0	0	1			0	1	0	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1	1				
HALTC	20	0	0	1	0	0	0			0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0				
	21																																
LOAD MAR	22	0	1	0	0	0	1			0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	1				
MB	23	1	0	0	0	1	0			1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	0	0	0				
MBRO	24	0	0	0	0	0	0			1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	1	0				
MBRI	25	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0				
MW	26	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	1				
PCU ADDR	27	1	1	1	1	1	0			0	0	1	1	1	0	1	1	1	1	1	1	1	0	1	1	1	0	1	1				
	28	0	1	1	1	1	0			1	0	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	0	1				
	29	0	1	1	1	1	0			1	0	1	0	1	0	1	1	1	1	0	1	1	0	1	1	1	0	0	1				
	30	0	1	1	1	1	0			1	0	1	0	1	0	1	1	1	1	0	1	1	0	1	1	1	0	0	1				
	31	0	1	1	1	1	1			1	1	1	0	1	0	1	1	1	1	0	1	1	0	1	1	1	0	1	1				
C_i PCU	32	1	0	0	0	0	1			0	0	0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0				
C_n PCU	33	0	0	0	0	0	0			0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
$\overline{RE}$	34	1	1	1	1	1	1			1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1				
MARO	35	0	1	1	1	0	1			0	1	1	0	0	1	1	1	0	0	0	0	0	1	1	1	0	1	1	0				
MAB	36	0	0	0	0	0	0			1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0				
ALU FUNCTION	37											1																	1				
	38											0																	1				
	39											0																	0				
ALU SOURCE	40											0																	1				
	41											1																	1				
	42											0																	1				
ALU DEST	43																												0				
	44																												1				
	45																												0				
LD CAR	46																																
TO	47	1	0	0	0	1	0			1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0				
ALU SHIFT	48																																
	49																																
	50																																
	51																																
	52																																
MODE	53	0	0	1	0	1	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
	54	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	55	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
MNEMONIC		FETCH	OR	NIO		JMP				ISR		DSZ		ISZ								DSZ				LDA				STA			
PROM ADDR		000	001	002	003	004	005	006	007	008	009	00A	00B	00C	00D	00E	00F	010	011	012	013	014	015	016	017	018	019	01A	01B	01C	01D	01E	01F

FIGURE 8.8a Microcode

PROM ADDR	MNEMONIC	MODE			ALU SHIFT					TO	LD CAR	ALU DEST			ALU SOURCE			ALU FUNCTION			MAB	MARO	$\overline{RE}$	CN PCU	CI PSU	PCU ADDR					MW	MBRI	MBRO	$\overline{MR}$	$\overline{LOAD MAR}$		HALTC	TEST COND				NEXT ADDR				IR LOAD	POLARITY	BRANCH ADDR									
		55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
020	ADC																																																								
021																																																									
022																																																									
023																																																									
024	ADD	0	0	0						1	1	0	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0		0	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1
025		0	1	0						0											0	1	1	0	0	1	1	1	1	1	0	0	0	1	0		0	0	0	0	1	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1
026		0	1	1						0											0	1	1	0	0	1	1	1	1	1	0	0	0	1	0		0	0	0	0	1	0	0	0	1	0	0	1	1	1	1	1	1	1	1	1	1
027										0											0	1	1	0	0	1	1	1	1	1	0	0	0	0	1		0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	0
028	ADD	0	1	0						1	1	0	1	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0		0	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
029																																																									
02A																																																									
02B																																																									
02C	NO SHIFT	0	1	1						0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	0	1	1	1	1	1	0	0	0	0	1		0	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
02D																																																									
02E																																																									
02F																																																									
030	LEFT ROT	0	1	1						0	0	1	1	1	0	1	1	0	1	1	0	0	1	0	0	1	1	1	1	1	0	0	0	0	1		0	0	0	0	1	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1
031																																																									
032																																																									
033																																																									
034	NO SKIP	0	0	0						0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	0			1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	1	1
035																																																									
036																																																									
037																																																									
038	ALL SKIP	0	0	0						0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	1	0	0	0	0	1	0	0	0	1	0		0	1	1	1	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	1
039																																																									
03A																																																									
03B																																																									
03C	DOA	0	0	0						1	0	1	1	1	0	1	1	0	1	1			1	0	1	0	0	0	0	1	0	1	0	1	0	1	0	1	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1
03D		0	0	0						0	0	0	0	1	1	1	1	1	1	1	0	0	1	0	1	1	1	1	1	1	0	0	0	1	1	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	1
03E																																																									
03F																																																									

FIGURE 8.8b Microcode continued

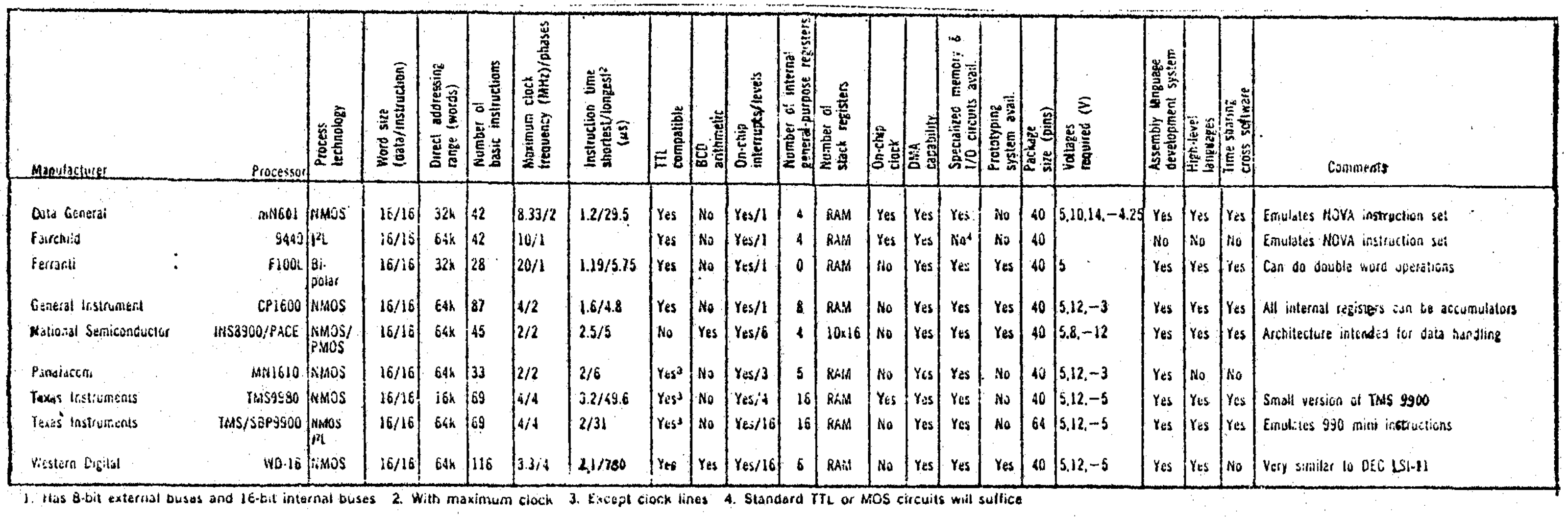
9.0 TECHNOLOGY TRENDS

The bit slice computer presented in this paper has an add time of 1.08 microseconds. By the use of more extensive hardware, the add time could be reduced to less than 800 ns. The question to be answered is as follows: given the present trend in technology, is it worthwhile to spend the time and money developing a bit slice machine?

If the present one chip 16 bit microprocessors are surveyed, as done in figure 9.1, the bit slice is 1.1 times faster than the fastest monolithic processor and 3 times faster than the slowest. At 800 ns, the bit slice is from 1.5 to 4 times faster than the monolithic microprocessors.

If there is to be an improvement in bit slice performance it must come in two areas: the sequencer, and the memory. The microstore and the Am29811 are PROM. If these devices were twice as fast, 128 ns would have been cut off the ADD instruction time. If the sequencer were twice as fast an additional 160 ns could be cut off. This would result in an ADD time of 512 ns, and this would be accomplished without the use of more advanced strategies such as multiple pipelining.

Are these speed increases reasonable? For the most part, these times represent the typical times for the same devices used in the 800 ns design (the design was on a worst case basis). Coupled with the new advance in



Manufacturer	Processor	Process technology	Word size (data/instruction)	Direct addressing range (words)	Number of basic instructions	Maximum clock frequency (MHz)/phases	Instruction time shortest/longest[2] (μs)	TTL compatible	BCD arithmetic	On-chip interrupts/levels	Number of internal general-purpose registers	Number of stack registers	On-chip clock	DMA capability	Specialized memory & I/O circuits avail.	Prototyping system avail.	Package size (pins)	Voltages required (V)	Assembly language development system	High-level languages	Time sharing cross software	Comments
Data General	mN601	NMOS	16/16	32k	42	8.33/2	1.2/29.5	Yes	No	Yes/1	4	RAM	Yes	Yes	Yes	No	40	5,10,14,−4.25	Yes	Yes	Yes	Emulates NOVA instruction set
Fairchild	9440	I²L	16/16	64k	42	10/1		Yes	No	Yes/1	4	RAM	Yes	Yes	No[4]	No	40		No	No	No	Emulates NOVA instruction set
Ferranti	F100L	Bi-polar	16/16	32k	28	20/1	1.19/5.75	Yes	No	Yes/1	0	RAM	No	Yes	Yes	Yes	40	5	Yes	Yes	Yes	Can do double word operations
General Instrument	CP1600	NMOS	16/16	64k	87	4/2	1.6/4.8	Yes	No	Yes/1	8	RAM	No	Yes	Yes	Yes	40	5,12,−3	Yes	Yes	Yes	All internal registers can be accumulators
National Semiconductor	INS8900/PACE	NMOS/PMOS	16/16	64k	45	2/2	2.5/5	No	Yes	Yes/6	4	10x16	No	Yes	Yes	Yes	40	5,8,−12	Yes	Yes	Yes	Architecture intended for data handling
Panafacom	MN1610	NMOS	16/16	64k	33	2/2	2/6	Yes[3]	No	Yes/3	5	RAM	No	Yes	Yes	No	40	5,12,−3	Yes	No	No	
Texas Instruments	TMS9980	NMOS	16/16	16k	69	4/4	3.2/49.6	Yes[3]	No	Yes/4	16	RAM	Yes	Yes	Yes	No	40	5,12,−5	Yes	Yes	Yes	Small version of TMS 9900
Texas Instruments	TMS/SBP9900	NMOS I²L	16/16	64k	69	4/4	2/31	Yes[3]	No	Yes/16	16	RAM	No	Yes	Yes	No	64	5,12,−5	Yes	Yes	Yes	Emulates 990 mini instructions
Western Digital	WD-16	NMOS	16/16	64k	116	3.3/4	2.1/780	Yes	Yes	Yes/16	6	RAM	No	Yes	Yes	Yes	40	5,12,−5	Yes	Yes	No	Very similar to DEC LSI-11

1. Has 8-bit external buses and 16-bit internal buses 2. With maximum clock 3. Except clock lines 4. Standard TTL or MOS circuits will suffice

FIGURE 9.1 A survey of 16 bit general purpose microprocessors

From: Electronic Design, 1977, page 56

tristate buffers for interfacing ECL and TTL, doubling the speed within one year should be no problem.

There is a limit to how fast a computer can operate, and the limit is established by the physical dimensions of the computer and the speed of light. Present bit slice technology requires 50 to 100 integrated circuits for the CCU. It will be very hard to package such a CCU in less than a square foot of area. In this dense configuration, the maximum distance would be about two feet. Since light travels approximately one foot every nanosecond, the time delay for a signal to be sent and its reply received would be 4 ns. This is equivalent to an extra gate level in the circuitry. On the other hand, since distances are measured in mils, the speed of light is not a practical consideration in the speed of single chip microprocessors.

It is apparent that the optical limits have or shortly will be reached in integrated circuit processing. This can be seen in figure 9.2. As figure 9.3 shows, there are two techniques being developed to take over where optical techniques leave off. The electron beam method promises a 100 fold density increase over the present density, and the x-ray approach offers a 1000 fold increase in density. At present the electron beam is the nearest to operational. From figure 9.3 the full impact of this technology can be seen. Texas Instruments projects a 32 bit microcomputer (not microprocessor) by 1983. This microcomputer would

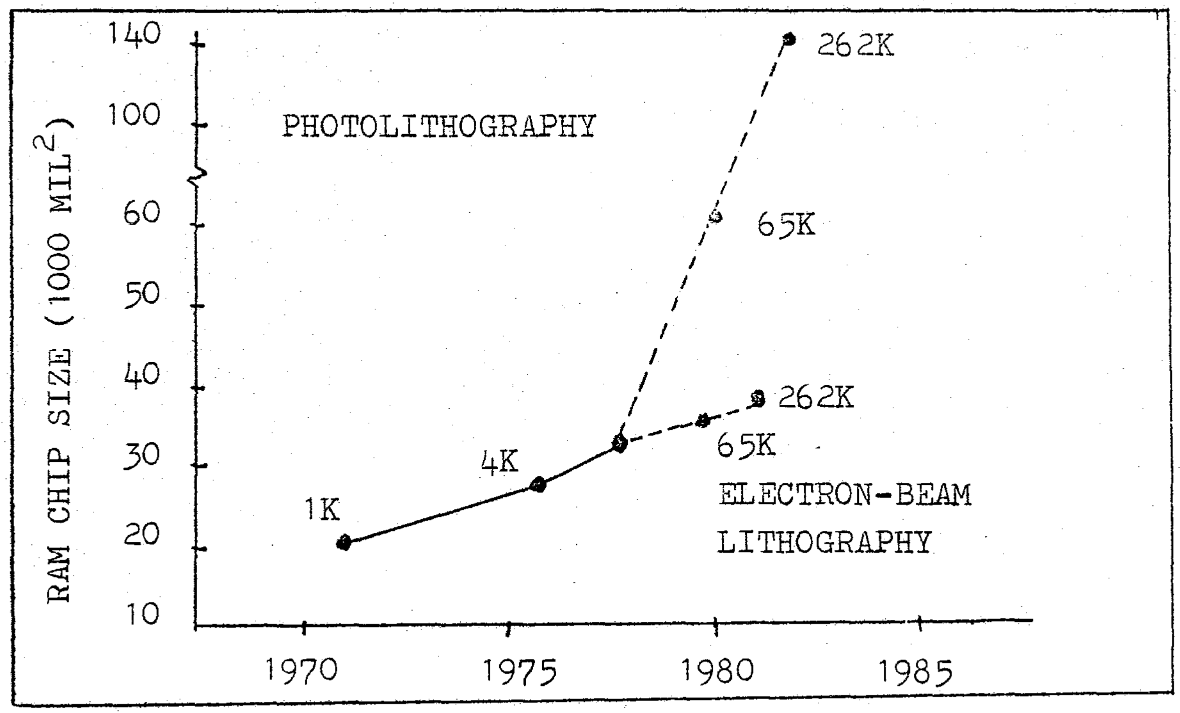


FIGURE 9.2 Future of electron beam technology

From: Altman, 1977

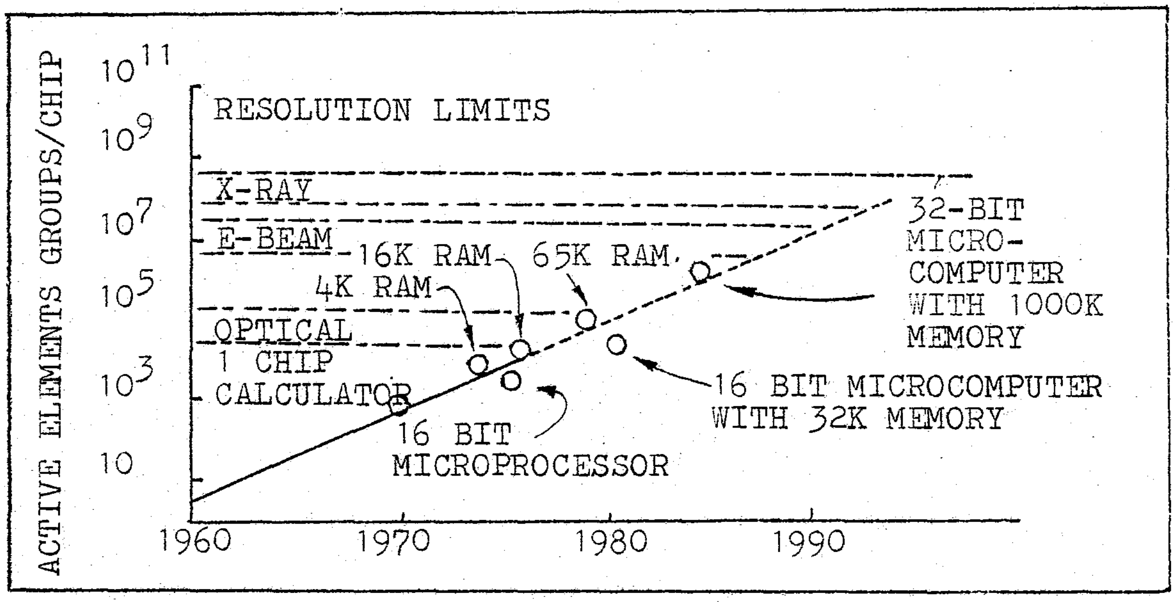


FIGURE 9.3 VLSI techniques

From: Altman, 1977a

have 32 K words of memory on the chip in addition to the CPU.

The 1983 microprocessor would require a 20 fold increase in density. This increase in density would decrease the capacitance of the integrated circuit. Since capacitance is the major speed killer in MOS circuitry, the decrease in capacitance by about 20 times would correspond to a 20 fold increase in speed. To be conservative and to take into account the extra carry time for a 32 bit machine, assume that the speed increase is only by a factor of 10. Since the fastest 16 bit MOS microprocessor available today has an instruction time range of 1.2 to 29.5 microseconds, the 1983 32 bit machine would do an ADD in 120 ns and a divide in 2.95 microseconds.

The design of complex circuits with electron beam technology will require even heavier dependence on computer aided design than present digital LSI designs. With the required advances in computer techniques and computer use for the 1980's, it is not hard to imagine a highly computerized and integrated design and manufacturing system for VLSI technology. When compared with the cost of designing a bit slice machine it may be cheaper to have an integrated circuit house design a custom microcomputer. That is unless a standard system can be used. After all, most applications that exist today can be accomplished with a computer capable of a 120 ns ADD.

While bit slice has a definite advantage now, the advantage will fade over the next decade unless the circuitry can be integrated into larger and faster slices. The problem is to get a faster technology into a smaller area. It is not an easy problem because most technologies require more power to go faster. As the size of the chip is reduced the power per unit area goes up. In the end, the monolithic microprocessor will probably win out, but until then the bit slice does offer some definite advantages.

10.0 SUMMARY

There are two broad areas of interest in this computer design. One area is cost, and the other area is performance. Cost includes the actual hardware cost, construction time, hardware design time, and firmware cost. In the area of performance we are concerned with the aspects of speed and flexibility.

The design of this system took several months of gathering and reading the material on bit slice technology and on the instruction sets for the PDP-11, NOVA, and PACE. Then came several design iterations as I tried to assimilate all the material. Finally, it took about six weeks to design the computer hardware.

In the construction phase, not counting the time it took to strip the wire wrap board, it took about two and a half weeks to wire wrap and document the board. An additional week was required to find the wiring mistakes and other problems with the wire wrap board.

As for the limited instruction set reduced to firmware, two weeks were required to write the original RTL programs, and two additional weeks were needed to write the binary code. Of these instructions only the FETCH, LDA, ADD (original), ADD without shifts or skips, and ADD with left rotate and skip always were checked out. The total time for the checkout was probably no more than two weeks, but the problem of programing the PROM's at work and checking the firmware at home added a great deal of time to the procedure.

In the performance area, the results were not as good as hoped for, although new and more innovative approaches could significantly reduce the execution time. It has become apparent that this is not a one man job. Rather the task should be attacked by a well coordinated design group. The complete design and construction of this computer could well take two man years. To avoid the problems of a project of this size, such as the demoralization that comes from chipping away at a large problem with no apparent progress, at least six people should be used. Two people should be used in the hardware area. One person should design the CCU and the other should design the memory and peripheral interface. Two people should be used to design the firmware and one person should design the monitors and assemblers so something can be done with the machine once the design phase is finished. Finally, one person is needed to bring the total design together into one cohesive effort. This person should be able to understand both the hardware and the software aspects of the computer so he can coordinate the two efforts towards the same goals and provide help in each area when it is needed.

REFERENCE LIST

Altman, 1977	Laurence Altman and Charles Cohen, "Gathering wave of Japanese technology", Electronics, June 9, 1977
Altman, 1977a	Laurence Altman, "Five technologies squeezing more performance from LSI chips", Electronics, August 18, 1977
AMD, 1976	Am2900 Bipolar Microprocessor Family, Advanced Micro Devices Corporation, Sunnyvale, California, June 1976
AMD, 1976a	Microprogramming Handbook, Advanced Micro Devices Corporation, Sunnyvale, California, November 1976
AMD, 1977	System 29, AM-PUB061, Advanced Micro Devices Corporation, Sunnyvale, California, 1977
AMD, 1977a	Am2930 (data sheet), Advanced Micro Devices Corporation, Sunnyvale, California, February 1977
DEC	LSI-11 microcomputer, Digital Equipment Corporation, digital components group, Marlborough, Massachusetts
Electronic Design, 1977	"Microprocessor Data Manual", Electronic Design, Vol. 25 No. 21, October 11, 1977
Falkoff, 1977a	Daniel Falkoff, Natalio Kerilenevich, and Philip Kreiker, "Exploit Existing NOVA software", Electronic Design, Vol. 25 No. 19, September 13, 1977
Muething, 1976	Gerald F. Muething Jr., "Designing the maximum performance into bit-slice minicomputers", Electronics, September 30, 1976
Suri, 1978	Ashok Suri and Dan Wilnai, "The Family Fire", Progress, Fairchild Corporation, Mountain View, California, Vol. 6 No. 2, March-April 1978
Wilnai, 1977	Dan Wilnai, "Mini-Computer CPU packed on One Chip", Progress, Fairchild Corporation, Mountain View, California, Vol. 5 No. 3, May-June 1977