MINICOMPUTER CONCEPTS

By

BENEDICTO CACHO

Bachelor of Science

Southeastern Oklahoma State University

Durant, Oklahoma

1973

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
MASTER OF SCIENCE
July, 1976

MINICOMPUTER CONCEPTS

Thesis Approved:

[illegible]

Thesis Adviser

[illegible]

James W. Maxwell

[illegible]

Dean of Graduate College

953275

PREFACE

This thesis presents a study of concepts used in the design of minicomputers currently on the market. The material is drawn from research on sixteen minicomputer systems.

I would like to thank my major adviser, Dr. Donald D. Fisher, for his advice, guidance, and encouragement, and other committee members, Dr. George E. Hedrick and Dr. James Van Doren, for their suggestions and assistance. Thanks are also due to my typist, Sherry Rodgers, for putting up with my illegible rough draft and the excessive number of figures, and to Dr. Bill Grimes and Dr. Doyle Bostic for prodding me on. Finally, I would like to thank members of my family for seeing me through it all.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

INTRODUCTION

Objective

In the past decade, the minicomputer industry was no doubt the fastest growing segment of the computer industry which to this day is the fastest growing industry in the world (30). Since the minicomputers first attracted the attention of end-users, much work has been done to optimize and expand the capabilities of these machines. Today they are powerful and versatile, and they cover a wide range of applications.

This paper presents a study of the concepts used in today's minicomputer designs. It is written from the viewpoint of a computer scientist with emphasis on logical organization. Details of circuit designs are not pursued. The basic elements of a minicomputer are described in Chapter II. Chapter III deals with the general system designs and some of the more important architecture found in today's minicomputers. Chapter IV focuses on what this author considers some of the more interesting systems being marketed. Chapter V contains a summary of the topics discussed, the future trends in the minicomputer industry, and the conclusions derived from this study. Appendix A contains a comparative chart of execution times of selected instructions for the minicomputers discussed in the text. Appendix B and Appendix C contain APL descriptions (8) of the effective address calculation schemes and the interrupt systems for the same machines.

The basis for this paper comes from the 1974 survey of minicomputers by Hobbs and McLaughlin (7). Other surveys reviewed include the one by Butler (2) in 1970 and Theis and Hobbs (28) in 1969. Thompson's work (29) on developing a general minicomputer simulation system is very much related to the subject of this paper and is recommended for complementary reading.

History of Minicomputers

In 1963, at a time when the computer industry was just beginning to stabilize from its fast paced revolution, the first of a new breed of computers was delivered by Digital Equipment Corporation of Maynard, Massachusetts (9) (30). It was physically much smaller than the typical computer of the time and it had just the very basic processing capabilities. Its primary function was to control discrete and continuous processes. New as it was, the "minicomputer" was undoubtedly unimpressive, especially because the general trend then was to centralize and concentrate computing power in larger and larger machines. So the first delivery did not attract much attention, but it did, however, set into motion a movement which was to be termed "the minicomputer revolution" (15) (30).

The original equipment manufacturers (OEM's) were at the outset the almost exclusive market for the minicomputers. They did the job of installing the little machines into large application systems primarily for process control. Before long, however, keen end users began noticing the price/performance benefits that the minicomputers offered--benefits that were unavailable in the medium and large scale computers. They began pointing to a wide variety of new applications suitable to the capabilities of the minicomputers. Such applications include:

- Instrumentation systems
- Automated test systems
- Data acquisition, monitor and control
- Communications control
- Transportation and distribution control
- Physical science monitoring, analysis, and control
- Medical science monitoring and analysis

By the end of 1965, an overwhelming interest in the new minicomputer industry was evident (28). In that year, over 1000 units were sold valued at approximately $25 million. The growth rate became phenomenal. In 1968 sales including peripheral equipment neared the $200 million mark. By 1972 over 35,000 units (54% of all computers installed in the country) had been sold and the industry had sustained a growth rate of over 30% per year (30). In 1975 alone world wide shipments were projected to reach $1.24 billion which includes 55,400 units (14).

What accounted for such an explosion? What made the minicomputers so attractive? It certainly was not their processing capabilities, not alone, anyway. Well then, what was the characteristic that caught the eyes of the then medium and large scale computer users? It was none other than the "price/performance benefits", the relatively minimal cost of owning and operating a minicomputer system. That characteristic coupled with the fact that improvements in technology generally brought down prices was the primary catalyst in the dynamic nature of the minicomputer industry. In March of 1969, a survey of minicomputers by Theis and Hobbs (28) included systems in the range of $50,000 or less. In October of 1970, a survey by J. L. Butler (2) included systems costing $25,000 or less. In July of 1974, a survey by Hobbs and McLaughlin (7) was limited to systems costing under $10,000. So the cost of minicomputers were low to begin with, and as technology continued to improve, those low prices continued to decrease.

What is the current state of the minicomputer industry, that is, what now are the general characteristics of the minicomputers, what are their roles in today's society, and who are the manufacturers?

The minicomputers of today have very impressive processing capabilities. They are much more powerful and much more versatile than those of the strictly monitor and process control days. A key factor in the hardware improvements is the increased use of integrated circuits (10). In effect, because of the microscopic nature of integrated circuits, it is now possible to implement some of the sophisticated processes of large scale computers within the mainframe of a minicomputer. Inherent in today's minicomputers are three such sophistications:

- use of general registers
- microprogramming
- interfacing through bus structures

As for the current roles of minicomputers, the applications mentioned earlier still make up close to 84% of the total number of applications. Of the remaining 16%, 9% consist of the more recent implementations in business functions, 4% in education, and 2% in the wide range of other applications (7) (10) (14). In the light of today's advances, predictions state that there will be more new applications as well as continued size and cost reductions and performance improvements.

As mentioned earlier, the nature of the minicomputer industry is dynamic. This characteristic is evident when the history of minicomputer manufacturers is examined. Hobbs and McLaughlin (7) cite some of the recent trends in their 1974 survey. In light of that article, it is safe to say that today there are well over 20 manufacturers. Table I shows Modern Data's (14) 1975 rating of the top nine.

TABLE I

1974 MINICOMPUTER SHIPMENTS

MANUFACTURERS	UNITS SHIPPED	DOLLAR VALUE	CUSTOMER SITES
Digital Equipment Corp	28%	33%	35%
Data General	20%	13%	16%
Hewlett Packard	11%	10%	12%
Texas Instruments	5%	5%	6%
General Automation	6%	5%	3%
Varian Data	2%	5%	4%
Digital Computer Controls	3%	5%	2%
Interdata	6%	3%	2%
Modular Computer Systems	2%	3%	3%

CHAPTER II

ELEMENTS OF MINICOMPUTER DESIGN

Introduction

Around 1830, an eccentric English mathematician named Charles Babbage, concerned with improving methods of computing mathematical tables, set forth the description of a machine which he called the Analytical Engine (6). In that description, the five most important features included the following:

1. It has an <u>input</u> medium, by means of which an unlimited number of operands and instructions may be entered.
2. It has a <u>memory</u>, where the operands and instructions may be stored and later retrieved.
3. It has a <u>calculating</u> unit capable of performing arithmetic and logical operations on the operands stored in the memory.
4. It has an <u>output</u> medium, by means of which the results of the calculations are returned to the user.
5. It has a <u>decision</u> capability, by means of which alternate courses of action may be taken depending on computed results.

Today, a computer system is described in terms of the five features listed above. Its basic organization is illustrated in Figure 1. One other feature added in 1947 by John Von Neuman of Princeton is the stored program concept in which the calculating unit does not recognize the difference between operands and instructions since both are stored in the same form, in the same memory.

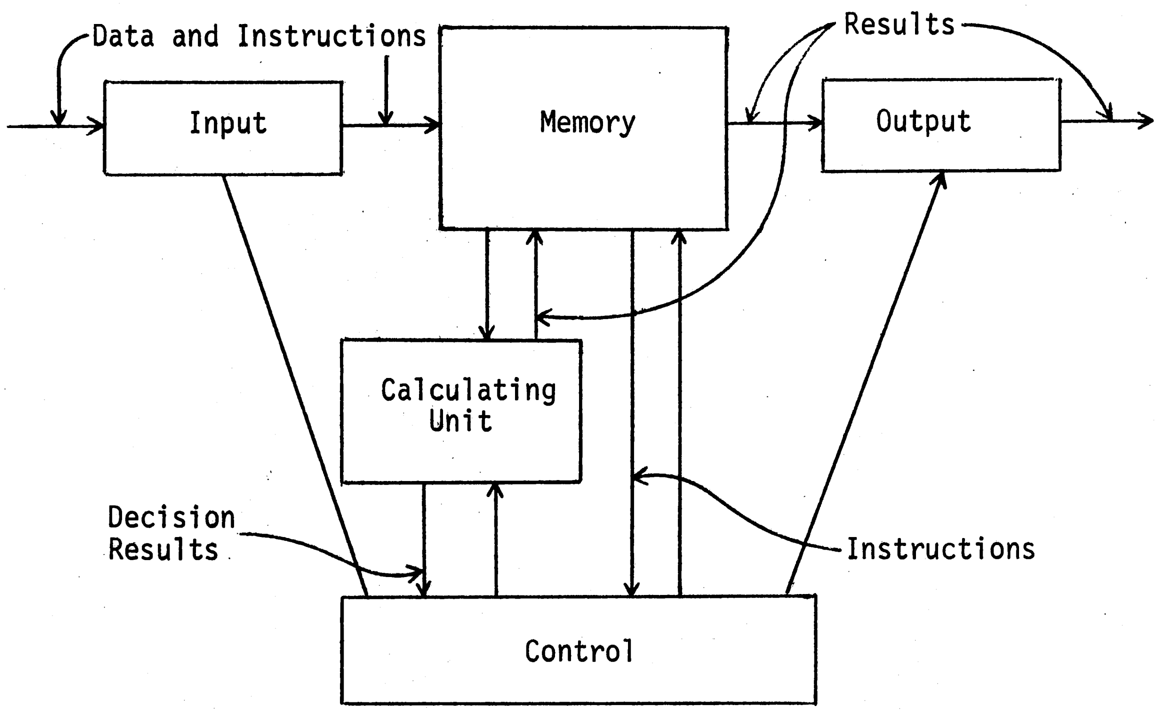


Figure 1. Basic Computer Organization

Minicomputers have the general features described above. When contrasted with medium and large scale computers, their distinguishing characteristics are:

- small physical size
- small word length
- small cost

Most minicomputers fit in a 19 x 11 x 21 inch mainframe. Their word lengths range from 8 to 24 bits with most systems using 16 bits. The cost of a basic system configuration including a processor, 4096 words of memory, and a teletype generally does not exceed $10,000 (10) (26). Although they have evolved into versatile units and have acquired some large scale computer features, it is at present inconceivable that a minicomputer system can actually replace a large scale computer system. There is still a big gap between the processing capabilities of the two classes of computers.

The description of the basic elements in a minicomputer system follow. The main topics correspond with the three major components:

- the processor
- the memory
- the input/output controllers

The Processor

The processor of a system is concerned with the major operations of a computer. It may be likened to the foreman in a group of workers who tells everyone what to do and when to do it. Thus all computer operations are initiated by the processor and when each operation is completed, the processor is notified.

Organization

The processor contains four basic elements:

- a set of registers
- an arithmetic/logic unit
- bus connections
- a control unit

Figure 2 shows one way these elements may be organized.

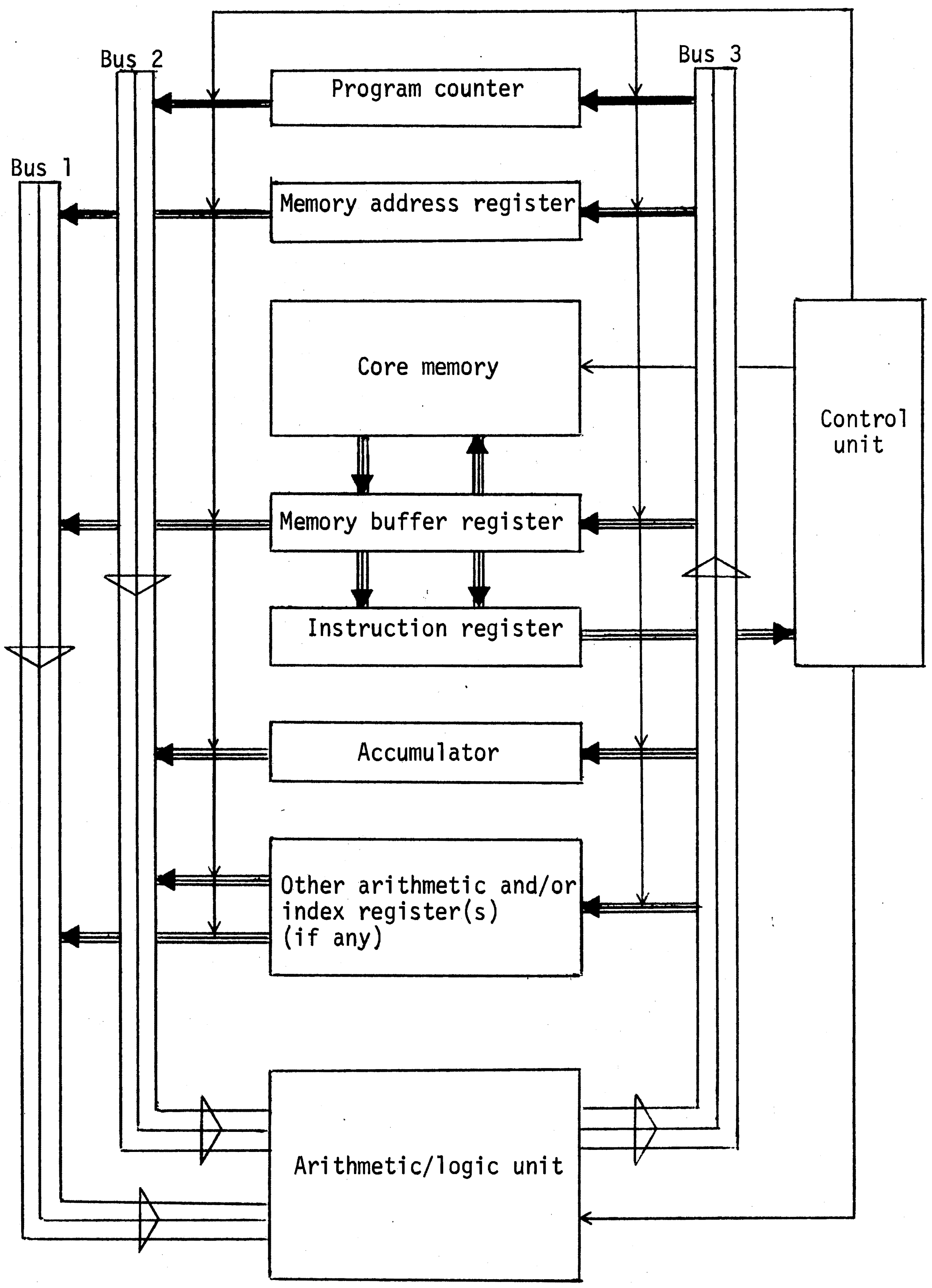


Figure 2. Processor Organization

The Register Set. Processor registers are fast memory units used by the processor. Physically each register consist of a set of "flip flops", memory devices each capable of storing one bit (binary digit) of information. The number of bits that a register can store depends on the word length of the system.

The functions of each register vary. In general some are used strictly by the processor while others are accessible to the user. Those used in today's minicomputers are described below (6) (9):

1. The program counter contains the address of the next instruction to be processed.
2. The instruction register contains the instruction currently being processed.
3. The memory address register contains the address of the memory location accessed or to be accessed.
4. The memory data register contains the operand or instruction to be stored into or just retrieved from the memory.
5. The status register or individual status indicators contain current status information about the processor.
6. The accumulator stores operands and results of arithmetic/logic operations.
7. The accumulator extension serves as an extended part of the accumulator for operations requiring more than the usual number of bits.
8. The index register is used in operand addressing.
9. The pointer register contains the address of an operand.
10. The stack pointer contains the address of "stacked" operands or results.
11. The general purpose register may serve any one of the above functions.

The Arithmetic/Logic Unit. The arithmetic/logic unit performs all calculations required by user programs. The unit consists of logic circuits capable of performing operations such as the following:

- add the contents of two registers
- logically "and" the contents of two registers
- complement the contents of a register
- shift or rotate the contents of a register
- increment or decrement the contents of a register

Bus Connections. Data paths between the arithmetic/logic unit and the registers are simplified by the use of data buses. In Figure 2 note the use of three data buses. Buses 1 and 2 are input buses from the registers to the arithmetic/logic unit. Bus 3 is an output bus from the arithmetic/logic unit to the registers.

Control Unit. The control unit coordinates all the actions of a computer by generating pulses to effect logical sequences. Control units may be hardwired or microprogrammed. In the hardwired version, the logic sequences are built into the logic circuits of the control unit. Thus the sequences are fixed and unalterable. The microprogrammed version consists of a microsequencer and a control memory which is separate from the main memory. The microsequencer is a control unit in itself but its operations are much more basic. It operates on microprograms stored in the control memory. The control sequences are thus defined by microprograms. By changing the contents of the control memory or by replacing the control memory with another control memory containing different microprograms, the control sequence is changed. The ability to be altered makes microprogrammed processors more adaptable to specific user needs. See Chapter III for a more detailed discussion of microprogramming.

Operations

The Basic Cycle. Processor operations involve a basic cycle of:

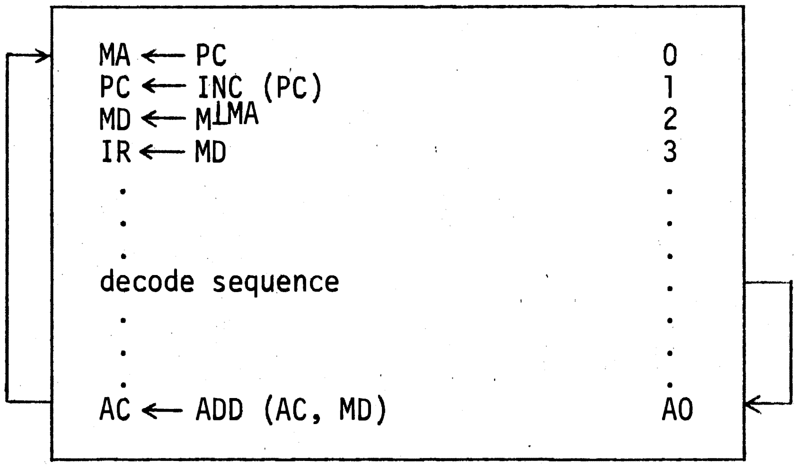
1. fetching an instruction from main memory
2. decoding the instruction
3. executing the instruction

Each of these operations are initiated and controlled by timed pulses generated by the control unit.

The first control pulse begins the instruction fetch by transferring the contents of the program counter (PC) into the memory address register (MA). Thus both registers contain the address of the next instruction to be processed. The next pulse gates the contents of PC through the arithmetic/logic unit (ALU) to be incremented and returned to the PC. The PC now contains the address of the next sequential instruction. The next control pulse is a read from memory. The contents of the location addressed by MA is transferred into the memory data register (MD). So MD contains the instruction which must be passed into the instruction register (IR) for the decoding operations. That transfer is effected by the next control pulse completing the instruction fetch cycle.

The decode stage of the cycle feeds the contents of IR into a set of decode logic circuits which performs a logic branch to the appropriate logic sequence. This logic sequence is associated with the machine instruction code in IR.

Suppose the machine instruction is an add operation. In the basic minicomputer one operand is assumed to be in the accumulator (AC). The second operand is taken from the memory location specified by the addressing portion of the instruction word. Once the second operand is fetched and placed into MD, control pulses gate the contents of AC and MD into the adder unit of the ALU where the sum is generated and then returned into AC. Thus the execute phase of the basic cycle is completed. The control sequence returns to the beginning where the next instruction is fetched, decoded, and executed. An APL (8) description of the process described above is shown in Figure 3a. The symbols used in the description are defined in Figure 3b.



```
MA ← PC                                 0
PC ← INC (PC)                           1
MD ← M⊥MA                               2
IR ← MD                                 3
 .                                      .
 .                                      .
 .                                      .
decode sequence                         .
 .                                      .
 .                                      .
 .                                      .
AC ← ADD (AC, MD)                      A0
```

(a)

Legend	
PC	program counter
MA	memory address register
M	memory
MD	memory data register
IR	instruction register
ADD	ALU add function
INC	ALU increment function

(b)

Figure 3. The Basic Processor Cycle

Interrupts. During the basic cycle, conditions requiring the attention of the processor may arise. An overflow in the result of a calculation, machine failure, an input/output device (initiated earlier) ready for the processor to activate a data transfer are examples of such conditions. The processor must be "interrupted" from its normal sequence of operation to take appropriate actions in returning the system to its normal state.

An interrupt is either internal or external. Internal interrupts are caused by various types of error conditions, such as arithmetic overflow, or invalid memory address. External interrupts are requests for attention from either the conventional I/O devices or external devices related to real time systems such as process control or lab experimentation.

Interrupts are monitored by the processor usually after the execute phase of the basic cycle. If an interrupt is required, the interrupting element must set an interrupt request indicator sometime during the current processing cycle. Upon recognition of the interrupt, the processor initiates an interrupt procedure by saving the "environment" of the program being interrupted. The environment of a program consists of the current contents of the registers and the status indicators. At the completion of the interrupt procedure, the environment of the interrupted program is restored and the processor continues with that program's execution.

Two general methods of processing interrupts are used in minicomputers. One of the methods uses one interrupt request line for all possible interrupts. When an interrupt is requested, the processor transfers to a general interrupt processing sequence where it must determine which element caused the interrupt. Once that is resolved, the processor transfers to an interrupt sequence that services the element that caused the interrupt.

The second method allows each interrupting element an interrupt ("vectored") address. Whenever an interrupt is granted by the processor, the next instruction executed is taken from the address associated with the interrupt. The instruction is often a branch to the appropriate service routine. This method of processing interrupts is faster and more efficient than the first method.

What happens when more than one interrupt occurs within a cycle? Some type of a priority system must be established. Internal interrupts are usually given higher priority over external interrupts. Among the external interrupts, real time devices with fast response requirements are usually given top priority. Then depending on physical location and speed, each I/O device is given its unique priority.

Priority schemes may be programmed or hardwired (built-in). One implementation of the hardwired version requires two interrupt lines for I/O processing (21). One line is used by the devices to request interrupts. The other is used by the processor to grant interrupt requests. The priority is determined by the order in which the grant signal is propagated through all the devices. The device connected closest to the processor on the grant line thus has the highest priority. If two devices request an interrupt simultaneously, the device with the higher priority receives the grant signal first thereby not allowing the grant signal to reach the second device.

Effective Address Calculations. Instructions involving an operand fetch from memory must deal with the minicomputer's inherent problem of the short word length (10). The problem lies in trying to code both the machine instruction and the operand address in one instruction word. In a system with 16 bit words, if 3 bits are used for the instruction code, then 13 bits are left for addressing the operand. With 13 bits, 8192 words (or bytes) is the maximum number of locations that are directly addressable. For some applications a minicomputer system with 8192 words of memory is sufficient. Yet, there are many more applications where 8K of memory is simply too small. And for some of those applications, 3 bits is often not enough to code all the necessary machine instructions.

To circumvent the problem, minicomputer designers devised a variety of addressing schemes. One scheme divides the memory into "pages". The size of a page depends on the number of bits used for the address part of an instruction word. For example, if the address is 8 bits long, then the page size is 2^8 or 256 words. If the memory size is say 16K words, then there are 64 pages of memory numbered from 0 to 63. With such a scheme, at least four modes of addressing are possible:

1. <u>Direct Page-0</u>. The effective address is taken to be the address specified in the instruction.

2. <u>Direct Current Page</u>. Enough high order bits of the program counter are concatenated with the address in the instruction to form the effective address.

3. <u>Relative to the Program Counter</u>. The address in the instruction is treated as a signed value and added to the current value of the program counter.

4. <u>Direct with a Page Register</u>. A separate register provides the high order bits in the calculation of the effective address.

The key point in the schemes described is the forming of a 16 bit effective address which allows access to 64K of memory.

There are many other addressing schemes used in minicomputers. Some of the more common ones are described below:

1. _Indirect Addressing_. The address specified in the instruction contains the effective address. Some systems have "multi-level" indirection where usually the most significant bit of an indirect address is tested. If it is set then the address points another indirect address, otherwise it points to the operand.

2. _Indexing_. The effective address is the sum of the address in the instruction and the contents of an index register.

3. _Extended Addressing_. The effective address is found in the location immediately following the instruction location.

4. _Immediate Addressing_. The operand is either in the instruction itself or in the word following the instruction.

In many systems, addressing schemes are combined. Terms such as _preindexing_ or _postindexing_ refer to the combination of index and indirect addressing. In preindexing, the indexing operation is performed, then the indirection is considered. It is vice versa for postindexing.

Machine Instructions. Machine instructions define the programmable operations of a computer. From one computer to another, the instruction sets usually differ according to their application. Generally, machine instructions are divided into three classes:

1. memory reference
2. register operate
3. input/output

These classes of instruction are discussed in the following paragraphs.

Memory reference instructions require some type of a memory access, either for fetching an operand or for transferring control. For minicomputers, instructions of this class usually include those listed in Figure 4a.

Register operate instructions deal mainly with the processor registers and the status indicators. There is no reference to the memory. The entire instruction word can thus be used to specify one or more register operations. Figure 4b lists the typical register operations.

Input/output (I/O) instructions deal with the transfer of data and device status information between the processor and the I/O devices. Three types of information, control, address, and data, may transferred. Control information are signals that initiates and/or terminates I/O operations. Address information refers to areas in the memory in which data is transferred in or out. The data, of course,

INSTRUCTION	ACTION
ADD	Add the contents of a register and a memory location, place the results in the register.
AND	Logically AND the contents of a register and a memory location, place the results in the register.
ISZ	Increment the contents of a memory location and skip the next instruction if the result is zero.
JUMP	Branch to a memory location and resume execution of the program.
JSUB	Store the address of the next instruction into a memory location and resume execution of the program at the location immediately following.
LOAD	Load a register with the contents of a memory location.
STORE	Store the contents of a register in a memory location.

(a) Memory Reference Instructions

Figure 4. Machine Instructions

INSTRUCTION	ACTION
CLEAR	Reset each bit in a register to zero.
COMP	Complement each bit in a register.
EXCH	Exchange the contents of two registers.
INC	Increment the contents of a register.
ROTATE	Rotate the contents of a register one bit left or right.
SET	Set each bit in a register to one.
SKIP	Skip the next instruction (conditional - the contents of a register is examined).
SHIFT	Shift the contents of a register one or more bits.

(b) Register Operate Instructions

INSTRUCTION	ACTION
DMAIN	Initialize a DMA input block operation.
DMAOUT	Initialize a DMA output block operation.
INBLK	Initialize a concurrent input block operation.
INPUT	Input a word from a device to a register or a memory location.
OUTBLK	Initialize a concurrent output block operation.
OUTPUT	Output a word from the memory or a register to a device.
SELECT	Transmit a specified function code to a device.
SENSE	Test the status of a device.

(c) Input/Output Instructions

Figure 4. (Continued)

is the information being transferred between registers or memory locations and the I/O devices. Typical I/O instructions are listed in Figure 4c.

The Memory

The memory of a computer performs the vital function of storing data, instruction sequences, and intermediate results of computations. Minicomputer memories generally range from 1024 to 32,768 words. Their speeds are in terms of cycle time, which is the time required to select and write data into a memory location. The cycle times vary from 250 to 2000 nanosecond (billionth of a second). The common practice is to manufacture memories in modules of 1024, 2048, 4096, or 8192 words. Thus users can start with the bare minimum and as needed for system expansions separate modules are purchased and installed (7) (10).

Two types of memories are most common: magnetic core, and semiconductor. Core memories are the slower of the two types but they have the distinct advantage of being non-volatile, which means their contents are not lost when the power supply is shut off. This characteristic coupled with the fact that core memories have, until recently, been generally cheaper, has made them the primary type of memory used today. However, with vast improvements in large scale integrations (LSI) drastically reducing their cost, semiconductor memories are now considered to be serious competition for the core memories (6) (10).

Minicomputers continue to improve. In memory design, special features such as those listed below are becoming more common.

1. Parity logic is used for error detection and correction.
2. Memory protect logic is used for maintaining the integrity of a system.

3. Scratchpad or cache memories are being used as fast (50 to 100 nsec) intermediate storage (14).

4. Memory modules are interleaved allowing overlapped memory access (15).

5. Memory banking techniques allow the use of up to 256K of memory (15).

Input/Output Elements

Computers must have a way of communicating with their users. This is done through the input/output elements which include peripheral devices such as card readers, line printers, tape drives, and teletype Keyboard/printers. The devices, however, cannot communicate directly with the processor. Interfacing elements must be provided to bridge the gap between the processor and the peripheral devices. These elements are called device controllers.

Device Controllers

Device controllers vary according to the type of peripheral devices they control. One type is used for serving devices that transfer data serially such as a teletype keyboard/printer or a cathode ray tube (CRT) keyboard/display. These are slow devices with transfer rates not exceeding 30 characters per second. Another type may be used to service a card reader with transfer rates up to 200 (80 column) cards per minute. Another type might service a line printer with a transfer rate of over 1000 (132 column) lines per minute. Then there are those that service high speed devices such as magnetic tape and disc drives. Controllers for these high speed devices often bypass the processor using the direct memory access (DMA) technique.

Generally a device controller is made up of two decoders. One decodes input from device selection lines. When an I/O operation is required the processor must send a device code through the device selection lines. All devices have access to these lines and each shall compare the signals with its unique device code. The device whose code matches those of the device selection lines then responds according to the function specified. The function is sent by the processor. It is decoded by a function decoder which activates the specified I/O operation - input/output of device status or input/ output of data.

I/O Operations

I/O operations involve some or all of the following steps (6):

1. Check to see if device is available.
2. When device becomes available, activate.
3. Transfer data.
4. Deactivate.

The first step can be accomplished in two ways. The first method involves a program loop where the status information of an unavailable device is checked and rechecked until the device becomes available. This method is very inefficient primarily because of the processor hold up. The second method uses the interrupt facility. The processor can request an I/O device to enter an interrupt request as soon as it becomes available. While waiting for the interrupt request, the processor is free to do some other computations.

The second step (activation) may not be necessary for some devices. Teletypewriters and CRT's are usually ready to go as soon as they become available. But for units such as a tape drive special activation processes must be performed.

The third step (transfer of data) can be accomplished using one of three methods:

- programmed
- buffered or concurrent
- direct memory access

Programmed data transfers make use of an interrupt procedure for each word transferred. This method is time consuming since every word transferred requires storage and restoration of the interrupted program. Buffered or concurrent data transfers usually involve a block of words and require extra hardware or microprogrammed logic. Once initiated by a special instruction, it interrupts the processes when it is ready for a transfer. It reads a buffer address and a word count from memory, determines the current address for the transfer, transfers the word in or out of memory, updates the word count, checks to see if the buffer is filled, and then returns to the interrupted program. In this method, the interrupted program need not be stored and restored, thus saving valuable processor time. The ultimate time saver, however, involves the use of the third method, direct memory access. In its implementation a separate processor is installed. The DMA processor consist of enough logic and registers to make data transfers in and out of the memory without having to go through the main processor. Like the buffered data transfer DMA transfers also usually involve a block of data words. Once initiated, the DMA processor "steals" a memory cycle from the processor each time it becomes ready to transfer a word. The main processor is not interrupted but "delayed" one memory cycle. When all I/O transfers are completed, the device is deactivated.

CHAPTER III

GENERAL SYSTEM DESIGNS

Considerations

The design of any computer system is influenced by the applications it is intended to cover. The systems of interest in this paper are the low cost general purpose minicomputers that are useable in the applications mentioned in Chapter I. Considerations involved in designing such systems include the following:

1. The system must be flexible. It must have the capability to assume a wide variety of configurations dictated specifically by the applications.
2. The system must be expandable. Structures for expanding the memory and the I/O capabilities must be implemented in the system design.
3. Designs involving the programmability of the system must be directed at achieving maximum effectiveness with minimum programming effort.
4. If possible, designs for a new system should also be directed at making the system compatible with earlier models. The software developed for the earlier models can thus be executable in the new system.

General Processor Designs

Processor operations generally involve information transfers to, from, and among the processor registers. The organization of the processor registers thus dictates the types of instructions that are to be included in a system's instruction set. In minicomputer systems, there are three general processor designs. The three designs are correlated to the type of programmable registers used. These three types are fixed purpose registers, multi-accumulators, and general purpose registers. Systems with fixed purpose registers make use of single operand address instructions. Those with multi-accumulators use double register operands. Systems with general purpose registers also use double operands, but the operands do not have to represent contents of registers.

Fixed Purpose Register Design

Minicomputers employing fixed purpose registers represent the basic, less sophisticated systems found in dedicated applications such as industrial process control, communications, and peripheral processing for larger computers. Systems belonging to this class of minicomputers include:

- Cincinnati Milacron CIP/2200
- Computer Automation ALPHA LSI-2
- Digital Equipment Corporation PDP 8/e
- Texas Instruments 980B
- Varian Data Machines VARIAN 520/i

Figure 5 is a simplified block diagram of the processor organization for this class of minicomputers. As shown, the programmable register set includes an accumulator, an accumulator extension, and an index register. Some systems, however, may not have all three registers. The PDP 8/e, for example, does not have an index register. The ALPHA LSI-2 does not have an accumulator extension. Then there are systems that have all three of those registers plus some others. The TI 980B has, in addition, a base register and a subroutine linkage register.

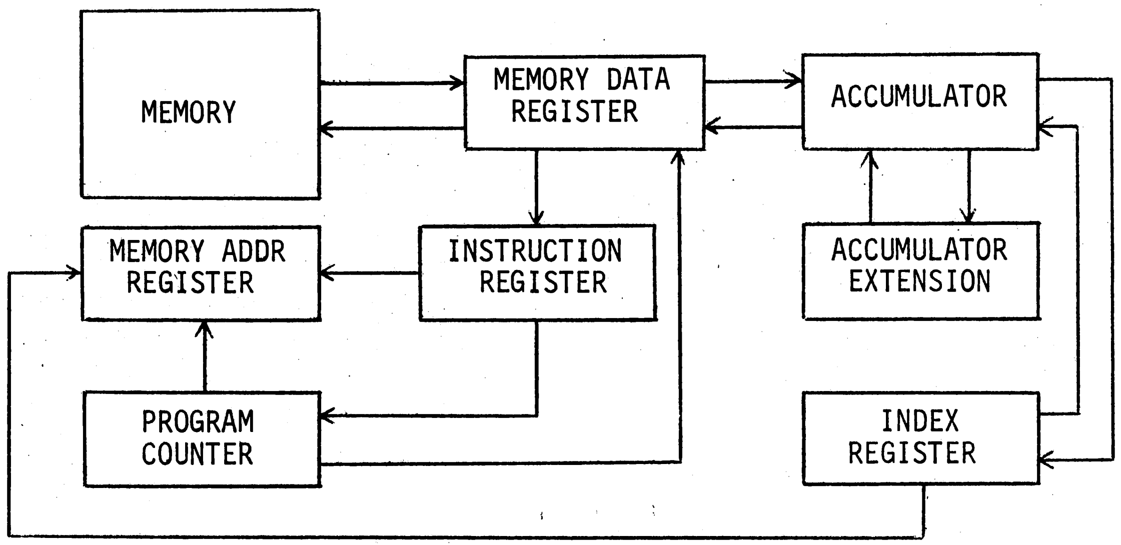


Figure 5. Processor Organization for Fixed Purpose Register Machines

Fixed purpose register systems are often referred to as single address machines. The implication comes from the use of one operand in each of the memory reference instructions. Figure 6 shows how such instructions are used in summing an array of values. The first instruction clears both the accumulator and the index register. The second instruction begins the summing loop. It instructs the system to add into the accumulator the contents of the memory location specified by the sum of the index register and the address value associated with ARRAY. The first time through the loop, the index register is zero, so the value added into the accumulator is 10. The next instruction increments the contents of the index register. Thus the next value to be added into the accumulator is taken from the address ARRAY + 1 which contains the value 25. The ISZ instruction increments the contents of COUNT from -4 to -3. Since COUNT is not zero the next instruction (JMP)

Mnemonic Code				Meaning
	OPR		CLA, DTX	Clear the accumulator and deposit to the index register.
LOOP	ADD	X	ARRAY	Add a memory value into the accumulator.
	OPR		INX	Increment the contents of the index register.
	ISZ		COUNT	Increment memory value and skip if the result is zero.
	JMP		LOOP	Branch back to process the next value.
	OPR		HALT	Terminate execution.
COUNT	DC		-4	Define constants.
ARRAY	DC		10	
	DC		25	
	DC		13	
	DC		75	
SUM	DS			Define storage.
	END			

Figure 6. Program for a Fixed Purpose Register Type System

returns control to the instruction labeled LOOP. The value of 25 is taken from memory and added into the accumulator to form the new accumulator contents of 35. The process of incrementing the index register and the negative counter is repeated. Since the counter contains a -2, the third value 13 is added into the accumulator making the sum 48. The program loops back for the last time to add the fourth value 75. When the ISZ instruction is executed this time the counter becomes zero, thus the JMP instruction is skipped. The STO instruction stores the contents of the accumulator into the memory location associated with SUM. The OPR HALT instruction terminates the execution of the program.

General Purpose Register Design

If the systems with the fixed purpose register design represent one end of the spectrum of minicomputers, then the other end is represented by the systems with the general purpose register design. Minicomputers in this class are geared for applications involving complex operations such as multi-tasking. For example, one of these systems may be used to automate industrial processes, monitoring and controlling multiple operations in real-time while simultaneously preparing and printing production reports for management. The following are a few of the minicomputers belonging to this class of computers:

- Digital Equipment Corporation PDP 11/40
- General Automation SPC-16
- Interdata Model 8/32
- Lockheed SUE
- Modular Computer Systems MØDCØMP II
- Raytheon Data Systems RDS-500
- Texas Instruments 960B

The processor organization of a system with a general purpose register design is shown in Figure 7. The programmable register set consists of eight general purpose registers numbered 0 to 7. Each of those registers can function as accumulators, accumulator extensions,

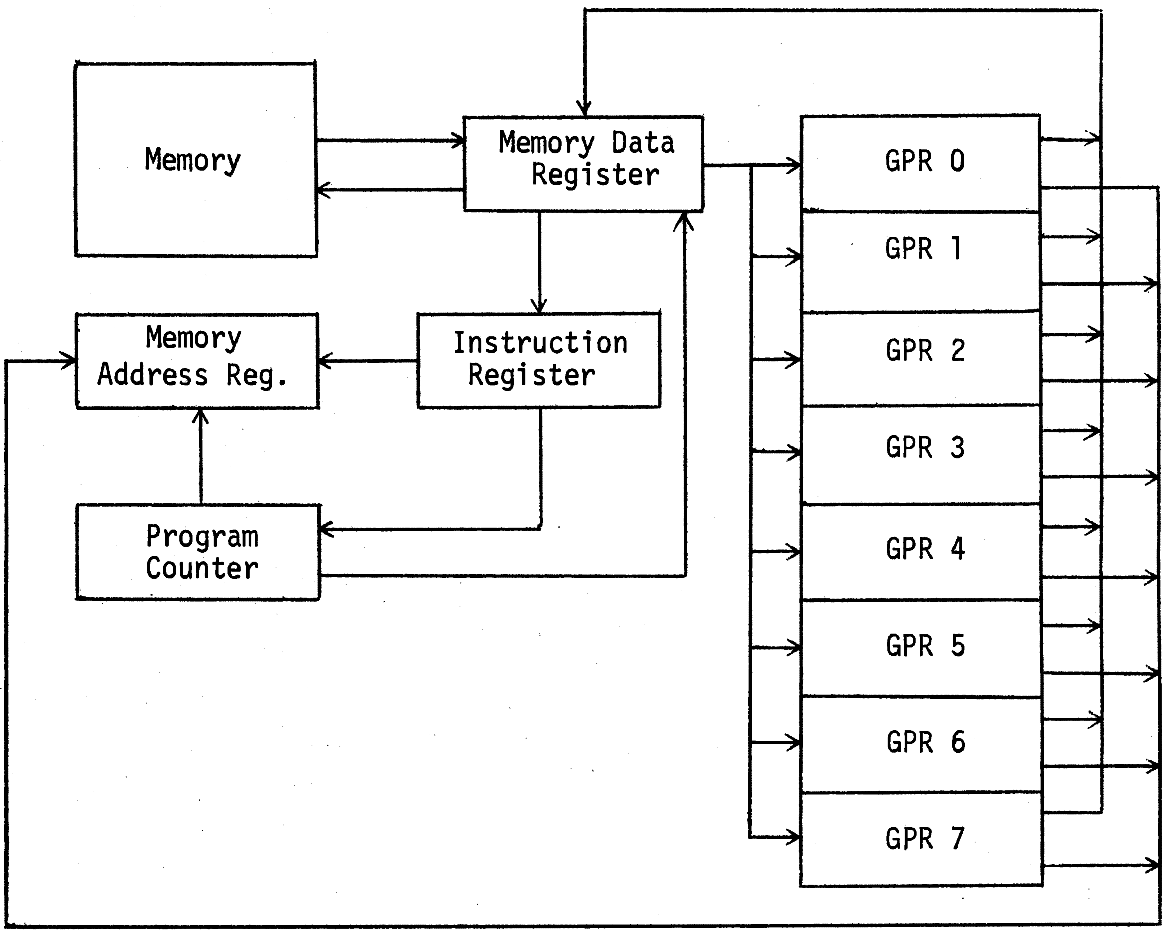


Figure 7. Processor Organization for General Purpose Register Machines

index registers, and operand pointers.

The programming example in Figure 8 illustrates how the general purpose registers are used. Like the program in Figure 6, it sums an array of values. Note the use of double operand instructions in which the operations are considered to be register-to-register or register-to-memory.

	Mnemonic Code		Meaning
	SR	2, 2	clear register 2
	LR	3, 2	clear register 3
LOOP	ADD	2, ARRAY(3)	add into register 2 the value in the location specified by the sum of register 3 and the address of ARRAY
	INC	3	increment register 3
	COM	3, COUNT	compare the contents of register 3 and location COUNT
	BLT	LOOP	branch to loop if register is less than the memory value
	STO	2, SUM	store the accumulated sum into the location SUM
	HALT		terminate execution

Figure 8. Programming a General Purpose Register Machine

Multi-accumulator Design

The multi-accumulator design is a combination of the previous two designs. Operations in systems with this design are generally centered around four accumulators, two of which can be used as index registers. Systems implementing this design include:

- Data General NOVA Computer
- Data General ECLIPSE Computer
- Digital Computer Controls D-116

The organization of a system with a multi-accumulator design is shown in Figure 9. Of the four programmable registers, the first two are used strictly as accumulators, the other two are used as accumulators or index registers.

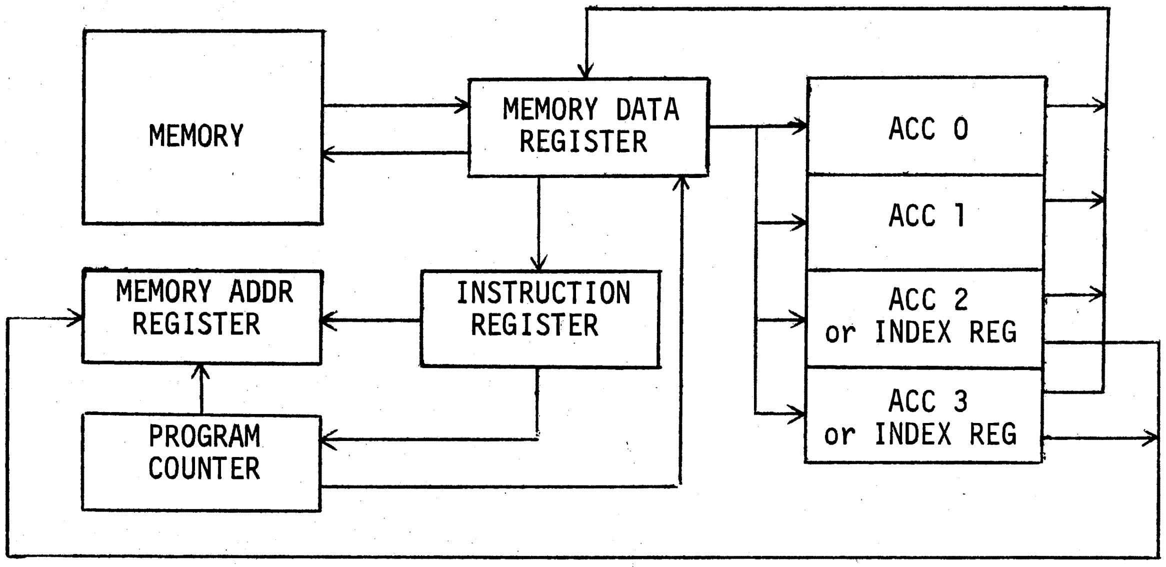


Figure 9. Processor Organization for Multi-accumulator Machines

The multi-accumulator design is an attempt at implementing a general purpose register design with the use of a minimum number of registers. In this design, there are only five memory reference instructions and these five do not include arithmetic/logic operations. The five instructions are:

- load
- store
- jump

- jump subroutine
- increment and skip if zero

All arithmetic/logic instructions are one cycle register-to-register instructions (they do not address memory). The extra instruction word bits can be used for other functions such as specifying a rotate of the resulting register and/or a conditional skip of the next instruction.

	Menmonic Code		Meaning
	SUB	0, 0	Clear accumulator 0.
	MOV	2, 0	Clear accumulator 2.
	LDA	3, COUNT	Load the negative count into ACC 3.
LOOP	LDA	1, ARRAY, 2	Load the value at the address specified by the sum of ACC 2 and address ARRAY into accumulator 1.
	ADD	0, 1	Add the contents of register 0 and register 1. Place the result in register 0.
	INC	2, 2	Increment index register 2.
	INC	3, 3, SZR	Increment negative counter in register 3. Skip next instruction if zero.
	JMP	LOOP	Branch to instruction labeled LOOP.
	STO	0, SUM	Store the contents of register 0 into memory location associated with SUM.
	HALT		Terminate execution.

Figure 10. Program For a Multi-accumulator System

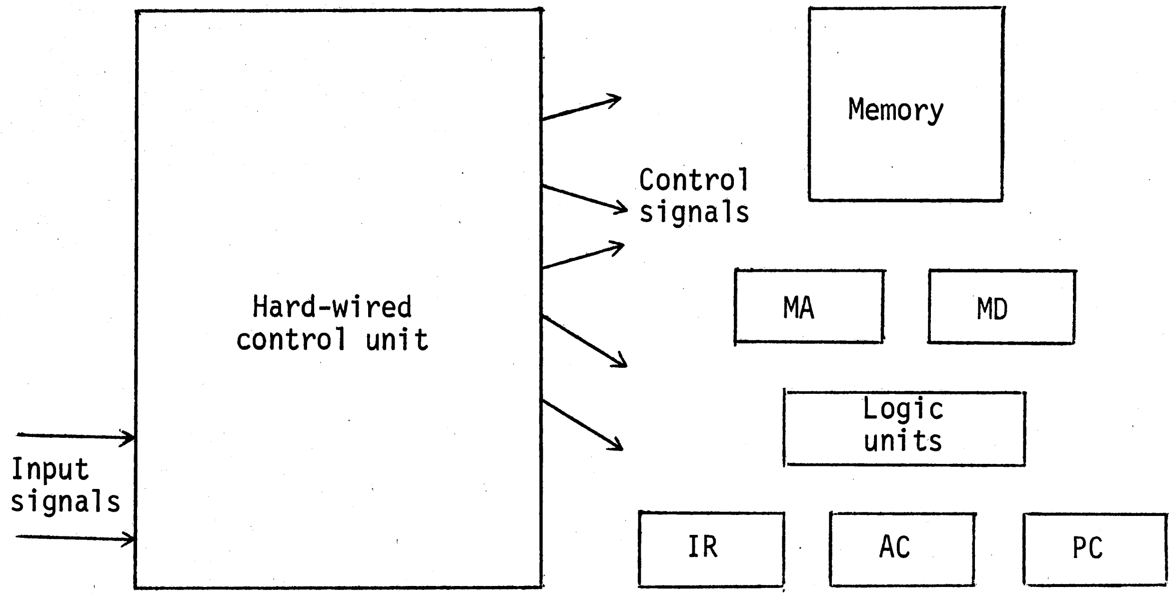
Figure 10 illustrates the use of some of those instructions. The algorithm for summing an array of values is used again for comparison. One might note that an extra load instruction had to be used since the ADD instruction does not reference memory.

Microprogramming

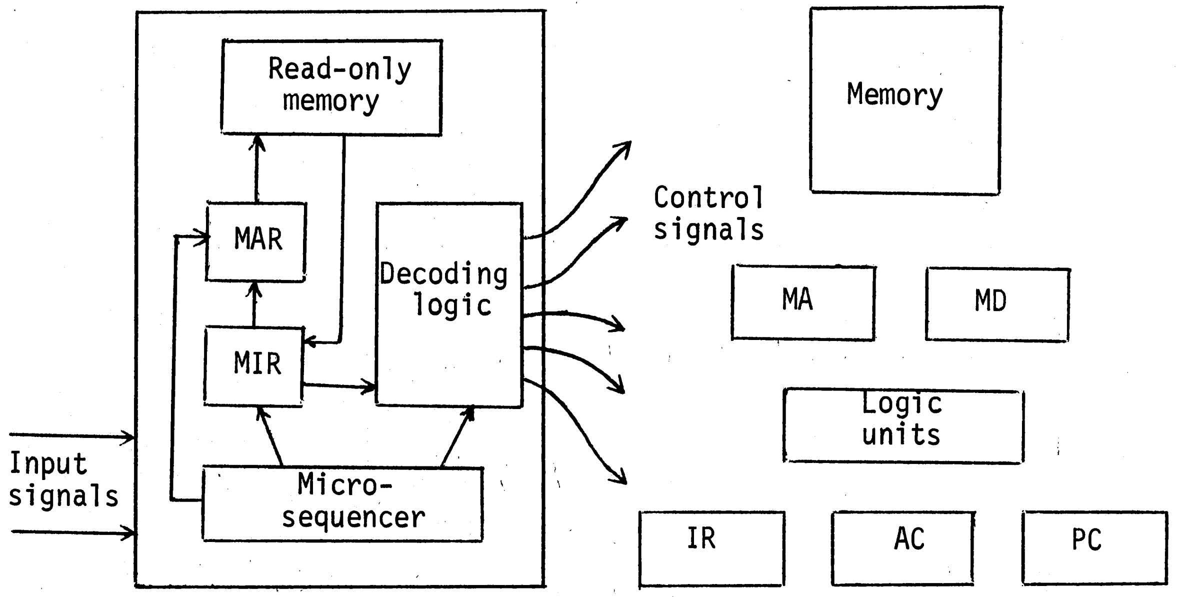
The concept of microprogramming as formulated by Professor M. V. Wilkes of Cambridge University (13) was incorporated into the design of minicomputers around 1970. The primary reasons were to give minicomputers added flexibility and to allow them to perform more sophisticated operations. The first large scale implementation of microprogramming was in the IBM 360 family of computers introduced in 1964 (6). One of the primary reasons for the implementation was to permit reasonably efficient emulation of earlier IBM computers for which the customer software had been developed. Needless to say, the microprogramming technique became a valuable marketing tool and it contributed greatly to the success of the new IBM computers.

The microprogramming concept is illustrated in Figure 11 along with the diagram for a non-microprogrammable machine. Note that the two architectures are identical except for the control unit. In effect, the same sequence of control pulses are generated by both versions. It is the means by which the control signals are generated that is different.

The microprogrammed control unit consists of a read only memory (ROM), a microaddress register (MAR), a microinstruction register (MIR), a microsequencer, and a network of decoding logic. The microsequencer acts as the controlling element in a microprogrammable control unit. The read only memory contains the microinstructions. The microprogram-



(a) Hard-Wired Control



(b) Microprogrammable Control

Figure 11. Two Types of Control Units

ming cycle begins by reading into MIR a microinstruction from a ROM word specified by the contents of MAR. From MIR the microinstruction is decoded to produce one of two actions--generate pulses for register transfers or modify the contents of MAR. The modification of MAR causes a microsequence branch. If a microsequence branch is not effected then the next microinstruction to be executed is read from the ROM word immediately following the ROM word of the current microinstruction (6).

An important feature in microprogramming is the ability to specify many different operations within a microinstruction word. As a matter fact, microinstruction word lengths are often longer than the word length of the main memory. This feature is useful in overlapping processor operations to save time.

One of the most important observations in microprogramming is the fact that the functions of the microprogrammable control unit are defined by the microprograms stored in the ROM. To meet different application requirements that dictates different control unit functions, one needs only to replace the microprograms. Thus microprogramming is advantageous in applications that require:

1. Implementation of large, sophisticated instruction set with a relatively simple processor.
2. Emulation of different computers with different designs for different applications.
3. Implementation of complex operations such as multiply/divide, floating point processing, and input/output.

Many of today's minicomputer systems employ a microprogrammed or microprogrammable control unit. In the Cincinnati Milacron CIP/2200, complex decimal number manipulation instructions including "edit and mark" and "translate and test" are implemented. Hewlett Packard 21MX and Data General ECLIPSE S/200 allows for customized instructions and subroutines through a writeable control store feature separate from the

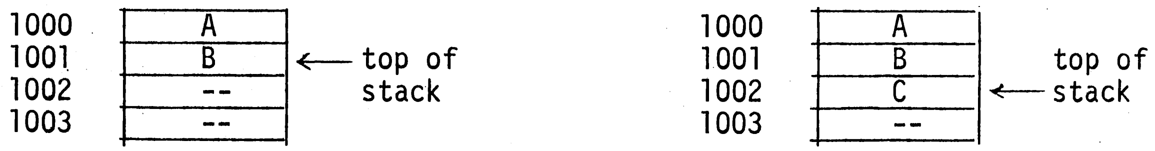
ROM of the control unit. One computer company, Microdata, manufactures only microprogrammable machines. The products include MICRO 800, MICRO 1600, and MICRO 3200. An interesting "firmware" (microprogram) development by Microdata is the MICRO 32/S. It is a MICRO 3200 processor microprogrammed to emulate a stack machine (11). The system is discussed in more detail in Chapter IV.

Stack Structures

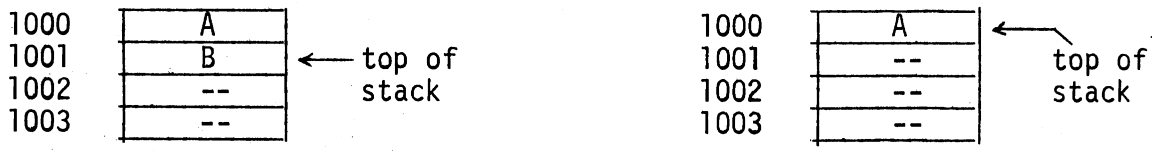
A useful special-purpose feature incorporated in many of the current minicomputers is a push-down storage unit, sometimes called an LIFO (last-in-first-out) list, or a stack. A stack is considered to be a list storage structure in which data items are inserted and deleted from one end only. Its use ranges from evaluation of arithmetic expressions to implementation of high level languages (11). The primary advantage in using a stack structure is its ability to allocate and deallocate storage locations dynamically.

A stack structure is analogous to a stack of cafeteria trays where the last tray placed on top of the stack is the first to be removed. Thus there are two major operations involved in a stack structure, "push" a data item onto the top of the stack and "pop" a data item from the top of the stack. These operations are illustrated in Figure 12. Error conditions "stack overflow" and "stack underflow" are illustrated in Figure 13.

Generally there are three levels of stack structure implementations in minicomputers. The first level of implementation involves automatic saving and restoring of environments in subroutine and interrupt processing. In the Cincinnati Milacron CIP/2200 it is called the control



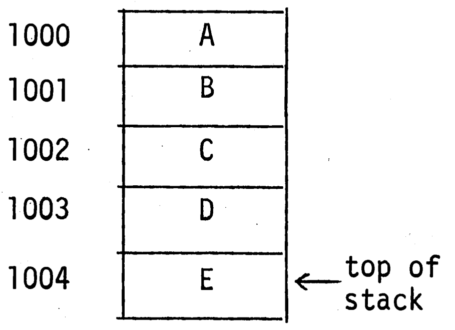
(a) Push Data C into Stack

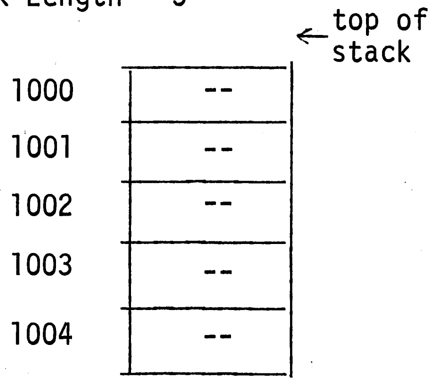


(b) Pop Data B. from Stack

Figure 12. Stack Operations

Stack Base = 1000 Stack Length = 5





(a) Overflow Condition (Push)

(b) Underflow Condition (Pop)

Figure 13. Stack Error Conditions

stack facility (3). The user has no direct access to the facility. The second level of implementation is user oriented. The user can define and manipulate his own stacks through special stack manipulating instructions. The facility is considered to be an added feature. Such a facility is implemented in the ALPHA LSI-2 (19) and ECLIPSE S/200 (22) computers. The third level of implementation involves a completely stack oriented system. In such a system the machine instructions are specifically designed to manipulate stacks. The Microdata 32/S (11) is an example of such a system.

Bus Structures

As microprogramming has contributed to the flexibility and usefulness of minicomputer systems, the use of bus structures has provided ease in the interfacing of a large number of peripherals, memory expansion and in some cases multiprocessor operations. Specifically the use of a universal bus has become very popular among minicomputer manufacturers. Just to name a few, the Computer Automation ALPHA LSI-2 has its MAXIBUS, the PDP 8/e has its OMNIBUS, the PDP 11/40 has its UNIBUS, Lockheed Electronics SUE system has its INFIBUS, and the Raytheon Data Systems RDS-500 has its SUPERBUS I and SUPERBUS II.

Figure 14 illustrates the relationship of a universal bus with its system components. In effect, the universal bus provides the communication link from one system component to another. Thus it is made up of communication lines with each line used for one of three types of signals -- address, data, or control.

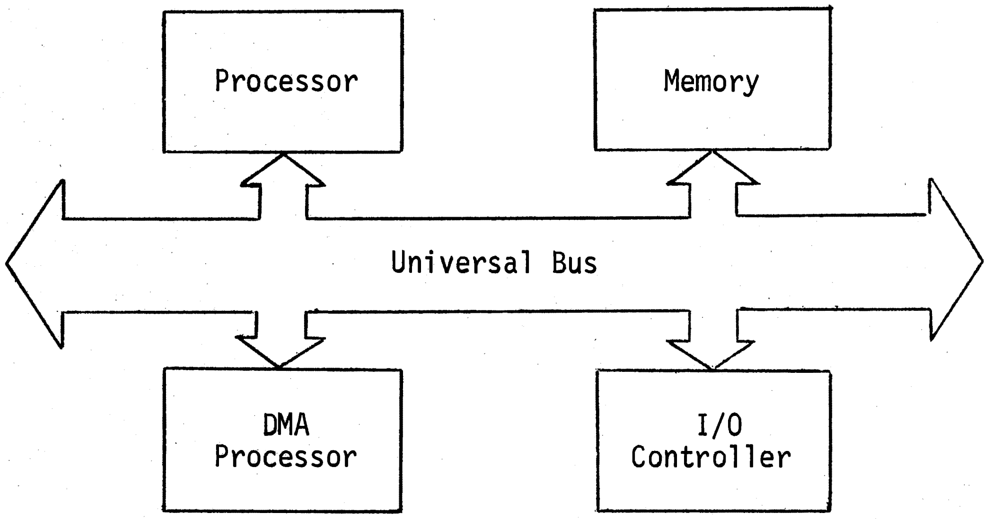


Figure 14. The Universal Bus

The address lines are used by the processor and DMA controllers. The processor uses them to send device and function codes. DMA controllers use them to address memory locations for I/O data transfers.

The data lines are shared by the processor, memory, and all I/O controllers. The processor uses them to read data from or write data into the memory. It also uses them for transferring data to and from the I/O controllers. The DMA controller uses them to read data from or write data into memory. All other I/O controllers use them to convey their unique interrupt addresses during interrupt processing.

The control lines are used by the processor to effect specific actions involving the memory and/or the I/O controllers. These lines can be subdivided into four categories -- I/O commands, utility signals, interrupt signals, and DMA signals. I/O command signals define the type of I/O operation (input, output, etc.) to be processed. Utility signals are used by the processor in resetting system status during a power-up procedure. The interrupt signals are associated with interrupt requests by the I/O devices and the interrupt processing that follows. The DMA signals are used for DMA interrupt priority signal propagation, DMA bus acquisition, and processor grant of DMA bus control.

Typical System Options

When an application requires more processing capabilities than what the standard equipment can offer, the user is usually made aware of the optional equipment. For the scientific applications where there is extensive use of mathematical computations, desirable options include the hardware multiply/divide and the floating point processor. These two greatly improve the speed of mathematical routines where the multiply/divide and floating point operations are normally done by slow software routines. Where power failure becomes a critical event, the power fail/restart option could be purchased to avoid disasters. This special option monitors the voltage levels in a system. When a voltage level drops below the normal operating level, an interrupt procedure saves the status of the current program in the non-volatile core memory. When the voltage level is restored, the restart procedure reloads the interrupted program. The normal processing operations are then reactivated at the point where the interruption occured. Where applications involve real world timing intervals, a real-time clock option is often necessary. Computer procedures may then monitor the clock and perform time-related operations. The options that have been discussed are the more typical options offered by today's minicomputers. Each system has its own set of options. Some may even offer some of the options described above as standard equipment.

CHAPTER IV

MINICOMPUTERS OF THE 70'S

Introduction

In this chapter, the system designs of ten of today's minicomputers are examined. Of the ten, four have fixed purpose registers, two have multi-accumulators, and three have general purpose registers. The last minicomputer has a specialized design implemented through microprogramming.

Fixed Purpose Register Machines

Digital Equipment Corporation PDP 8/e

The first PDP 8 model was introduced in 1964, a year after its predecessor, the PDP 5, hit the market (20). Through the years, the PDP 8 series has proven itself to be one of the most successful line of minicomputers. The primary reason for its success is the preservation of the original instruction set (10). The succeeding models were thus compatible with the earlier models allowing users to develop a massive amount of general-purpose and application software. It is no wonder that today in 1976, the PDP 8/e, with its seemingly obsolete 12-bit design, is still a very serious competitor for the overwhelming 16-bit systems because of the large and valuable software inventory.

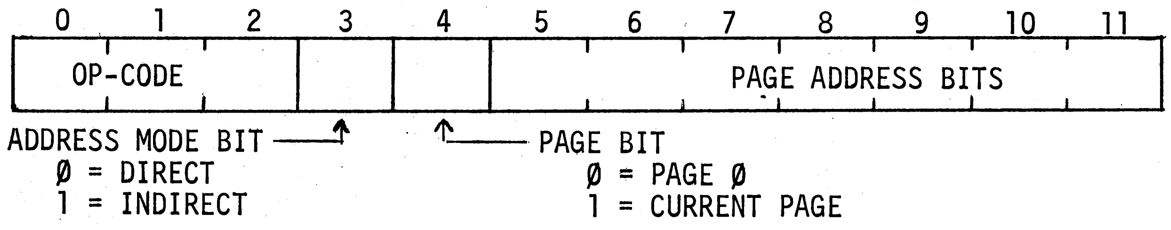
The functional characteristics of the PDP 8/e are listed in Table II. The instruction formats are shown in Figure 15. There are five two-cycle memory-reference instructions with one level of indirect addressing possible, and eight memory locations on page-0 serving as

TABLE II

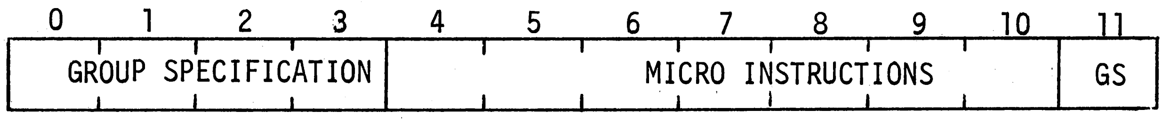
PDP 8/e FUNCTIONAL CHARACTERISTICS

Features	Characteristics
Processor	
Programmable Registers	1 accumulator 1 accumulator extension
Control Unit	hardwired
Instructions	34
Memory Reference	6
Register Operate	20
Interrupt	8
Addressing	
Direct	128 words
Current page	128 words
Indirect	4096 words
Interrupt System	polling (1 interrupt line)
Memory	
Word length	12 bits
Cycle time	1200 nsecs (core)
Capacity	
Minimum	4096 words
Maximum	32,768 words
Increment	2096 or 4096 words
Parity	option
Input/Output	
Maximum number of devices	60
Programmed	10 characters/sec
Direct memory access	833 K words/sec
Universal Bus	96 lines (bidirectional)

autoindex (automatic incrementing) registers. The processor has one accumulator and a temporary storage register whose contents can be transferred to and from, or exchanged with the accumulator by one-cycle instructions. Up to 512 I/O instructions are possible with the use of a single-level interrupt system. A DMA processor allows data transfer within one memory cycle or three memory cycles if the transfer is just one of a block transfer (10) (20).



(a) Memory Reference Instruction Format



Bits	Group 1	Group 2	Group 3
0-3	1110	1111	1111
4	Clear AC	Clear AC	Clear AC
5	Clear Link	Skip on AC 0	AC ext into AC
6	Complement AC	Skip on AC=0	------
7	Complement Link	Skip on Link 0	AC into AC ext
8	Rotate Right	Reverse Skip Logic	------
9	Rotate Left	Logical OR	------
10	Byte Swap	Halt	------
11	Increment AC	0	1

(b) Register Operate Instruction Format

Figure 15. PDP 8/e Instruction Formats

Cincinnati Milacron CIP/2200

The Cincinnati Milacron CIP/2200 is a general purpose, byte oriented minicomputer employing a microprogrammed control unit (3). It has an extensive instruction set including binary arithmetic, decimal arithmetic and character manipulation. The functional characteristics are listed in Table III. The instruction formats are shown in Figure 16.

The CIP/2200 has an 8-bit hardware data path and memory. The CPU registers, however, are 16 bits in length. The instruction set includes a complete set of 16 bit register-to-memory and register-to-register binary arithmetic instructions. In addition, variable length binary arithmetic on 8, 16, 24, or 32 bit operands are possible. Another group of instructions provides memory-to-memory decimal arithmetic and character string move and compare, code conversions, and decimal editing.

The CIP/2200 I/O structure consists of a microprogrammed serial I/O interface, a byte I/O facility, a microprogrammed facility for concurrent transfers, and up to two independent DMA processors. The serial I/O interface controls a teletype or other similar terminal devices. The byte I/O facility transmits 8-bit data between one of 32 peripheral devices and either a register or a memory location. The microprogrammed Direct Memory Channel (DMC) for concurrent transfers allows a maximum transfer rate of 86,000 bytes per second concurrently operating with program execution. The independent DMA processors compete with the CPU for access to main memory and have a maximum transfer rate of 909,000 bytes per second.

TABLE III

CIP/2200 FUNCTIONAL CHARACTERISTICS

Features	Characteristics
Processor	
Programmable Registers	1 accumulator 1 accumulator extension 1 index register
Control Unit	microprogrammed
Instructions	119
Arithmetic	14 (binary and decimal)
Memory moves	3
Register change	41
Shifts	12
String manipulation	6
Control transfers	19
Interrupt	13
I/O	8
Immediate	7
Addressing	
Direct	256 words
Indirect	32,768
Indexed	32,768
Extended	32,768
Immediate	1 - 4 bytes in instruction
Relative	128 behind - 127 ahead of program counter
Interrupt System	
Type	vectored with priority
Internal	6 lines
External	1 line (64 signals)
Memory	
Word length	16 bits
Cycle time	1.1 nsecs (core)
Capacity	
Minimum	8192 words
Maximum	32,768 words
Increment	8192
Parity	optional with 9 bit/byte memory
Protect	optional
Read Only Memory	1536 words used for teletype controls, bootstrap loader concurrent block I/O, instruction set extension

TABLE III (Continued)

Features	Characteristics
Input/Output	
Maximum number of devices	32
Maximum transfer rates	
Serial I/O	110 bits/sec
Byte I/O	10,000 bytes/sec
Concurrent block	86,000 bytes/sec
Direct memory access	910,000 bytes/sec

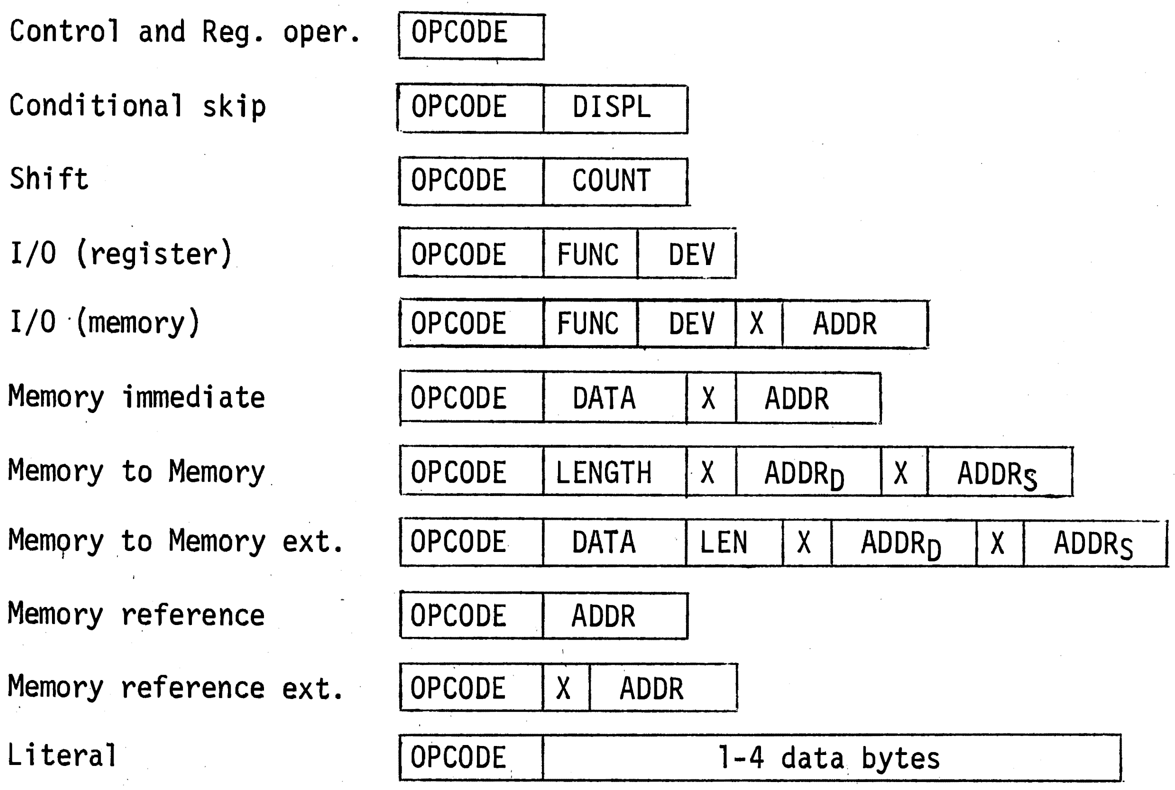


Figure 16. CIP/2200 Simplified Instruction Formats

The use of microprogramming in CIP/2200 has allowed instructions of considerable power and flexibility to be implemented. "Edit and mark" and "translate and test" are two such instructions. If more specialized instructions are needed, the writeable control store (WCS) feature of CIP/2200 may be used. The user may use special instructions provided by CIP/2200 to transfer to user written application microprograms residing in the WCS.

Variable Length Binary Arithmetic. Special variable word length instructions perform binary arithmetic on one, two, three, or four bytes of data. This is useful for character operations, single byte arithmetic, and extended precision arithmetic on 24 or 32-bit quantities.

Variable word length instructions use two operands, one in the accumulator (A) - accumulator extension (B) pair and the other in memory. For each operation a special word length indicator (WL) must be set to the desired length. Figure 17 shows which bytes of the register are

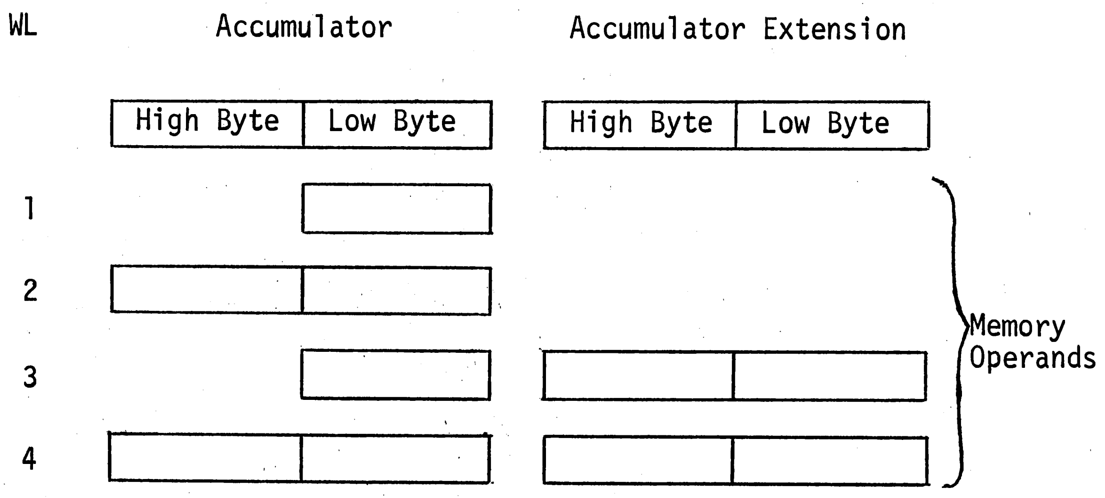


Figure 17. Variable Length Data Formats

involved for each word length. The variable length operations are described below:

1. set/reset the word length indicator
2. load/store variable word length data
3. add/subtract variable word length data
4. AND variable word length data
5. add/subtract word length to/from index register

Decimal Arithmetic. Decimal numbers are represented as strings of ANSCII decimal digit characters in varying lengths from 1 to 16 digits. Each digit is represented in memory as one byte. The first four bits contain digit zone, the last four contain the decimal digit value. The digit zone of the least significant digit contains the sign of the decimal number. A minus sign is an all zero digit zone pattern, the plus sign is a 1011 digit zone pattern. Examples are shown in Figure 18. Decimal operations include add, subtract, multiply and divide. The two operands reside in memory and the result replaces one of them.

Decimal Number	Machine Representation			
1234	10110001	10110010	10110011	10110100
-5678	10110101	10110110	10110111	00001000

Figure 18. Decimal Data Representation

The Control Stack Facility. The CIP/2200 uses a control stack to implement state switching where the saving and restoring of computer state information are required in operations such as interrupt processing and subroutine linkage. The state information consists of the contents of the accumulator, the accumulator extension, the index register, the program counter, and all status indicators. The stack mechanism is based on the "Last In First Out" (LIFO) technique. Each entry in the control stack consists of a complete set of state information. The most recently saved set is at the "top" of the stack, the oldest at the "bottom".

In normal useage, each subroutine saves the machine state immediately after being called. The information is restored when the subroutine executes a return to the calling program instruction, when there are more than one level of subroutine processing, the control stack has an entry for each of the subroutine calls. As the successive returns are executed, corresponding entries are "popped" from the top of the stack.

Computer Automation ALPHA LSI-2

The ALPHA LSI-2 computer is a package product of an integrated family of compatible components including two central processors, three kinds of memories, and a wide variety of device controllers (19). Through the implementation of a universal bus (the MAXIBUS), the user can mix memories of varying speeds, sizes, and technologies with either of the two processors (which differ in speed and performance) and the necessary I/O devices to obtain the best price/performance margin for his purposes. The ALPHA LSI-2 package includes the LSI-2 processor which is the faster of the two. The functional characteristics are listed in Table IV. Special features are discussed in the following sections.

TABLE IV

ALPHA LSI-2 FUNCTIONAL CHARACTERISTICS

Features	Characteristics
Processor	
Programmable Registers	1 accumulator 1 index register also used as the accumulator extension
Control Unit	hardwired
Instructions	188
Memory Reference	30 (standard hardware mult/div)
Immediate	10
Stack	15
Register Change	52
Shifts	16
Control	20
Interrupt	12
Input/Output	33
Addressing	
Direct	256 words
Relative	256 words foreward, 255 backward
Indexed	32K
Indirect	32K - multi-level
Indirect (Post Indexing)	32K
Immediate	1 byte in instructions and
Interrupt System	vectored with priority
Internal	2 lines or levels
External	3 levels - unlimited device support

TABLE IV (Continued)

Features	Characteristics
Memory	
Word length	16 bits
Cycle time	
Core (3 speeds)	980 nsec, 1200 nsec, 1600 nsec
Semiconductor	1200 nsec
Capacity	
Minimum	1024 words
Maximum	32,768 words (262,144 with memory banking)
Increment	1024 or 2096 words
Parity	optional
Interleaving	optional
Banking	optional
Input/Output	
Maximum number of devices	248
Maximum transfer rates	
Programmed	130,000 words/sec (via registers) 90,000 words/sec (direct to memory)
Concurrent block	80,000 words/sec
Direct memory access	1,020,000 words/sec (1,666,000 with interleaving)
Universal Bus	
Address lines	16 bidirectional
Data lines	16 bidirectional
Control lines	27 unidirectional

General Stack Processing. Fifteen stack instructions allow the use of any memory location as a stack pointer to maintain a last-in-first-out (LIFO) stack anywhere else in memory. Any number of routines can maintain any number of stacks with the possibility of using any number of separately maintained stack pointers that access the same physical stack. Furthermore, arithmetic, logic, and compare operations on data contained in stacks are also implemented separate from the conventional set of instructions. These facilities invite the use of sophisticated programming techniques.

Automatic Memory Scan. A "Scan Memory" instruction compares the contents of the accumulator with the contents of memory locations in a data buffer defined by sequentially indexed addresses. If a match is found, the scan is terminated and the next sequential instruction is executed. Initially, the index register contains the number of words to be scanned, it is decremented after each compare. Thus, the data buffer is scanned in descending order, beginning with the highest memory location and ending with the lowest. When a match is found, the index register contains the number of words remaining to be scanned. The remainder of the data buffer can be scanned simply by executing the scan instruction again. If a match is not found when the index register reaches zero, the scan is terminated and the next instruction is skipped. This feature becomes valuable in applications where limited serial search routines are prevalent.

Memory Interleaving. Memory interleaving allows memory modules to be paired so that even and odd addresses are assigned different memory modules. Since a relatively high percentage of memory accesses are normally sequential, this feature allows alternate memory accesses to address different memory modules thereby saving time because of the overlap in the alternate accesses. The asynchronous universal bus can thus support a much higher data transfer rate which effectively reduces the execution times in the ALPHA LSI-2.

Memory Banking. Memory Banking consists of an optional Memory Bank Controller that allows the programmer access to a maximum of 256K words of memory. Up to 32K can be enabled at any given time. The user can specify which memory modules are to be used by using special instructions that enable or disable the desired modules. In the example of Figure 19, there are four primary modules, two are 4K and the other two 8K. The computer can operate normally as a 24K computer using these modules. The two 4K modules (P00 odd and P00 even) are interleaved, the 8K's (P10 and P20) are not. There are seven alternate modules in the example. Each alternate module can be an alternate for only one primary module. For example, modules A11, A12, and A13 are the first, second and third alternates for the primary module P10. Under software control, the Memory Bank Controller can disable P10 and enable A11, A12, or A13. Thus a total of 32K words are available between addresses 8K and 16K, but only 8K of the 32K are available at any given time.

Texas Instruments 980B

The Texas Instruments 980B Computer is one of the more powerful general purpose computers in the fixed purpose register class of mini-computer. Its processing capabilities are enhanced by standard features that are often optional in other minicomputers (18). Such features include:

- hardware multiply/divide
- programmable memory protection
- power fail detection and automation restart

Other important features are listed in Table V.

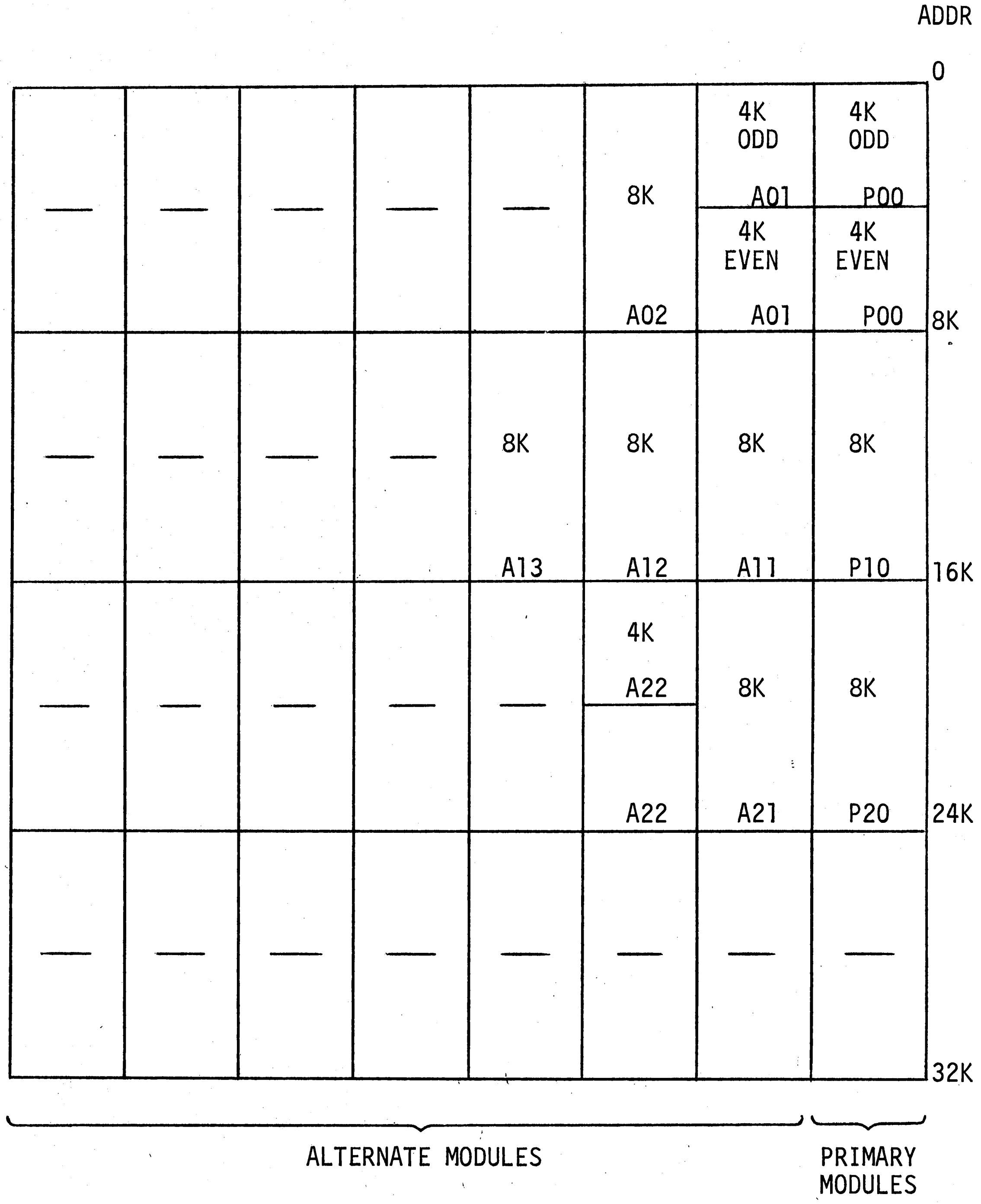


Figure 19. Memory Banking Example

MOS Semiconductor Memory. The main memory for the 980B consist of MOS Semiconductor elements (10). Each bit is stored in what amounts to a one-bit shift register whose output is fed back to the input through a clock gated MOS refresh amplifier. Thus every 33 microseconds each bit is regenerated. The cycle time for the memory used in TI 980B is 750 nanoseconds. Since semiconductor memories are volatile, an optional plug-in battery pack is available for sustaining the contents of the memory during power failure conditions. A 16K memory can be sustained for 20 hours.

Programmable Memory Protection. The 980B computer incorporates a memory protect feature that allows a system programmer to write system programs that prohibits a user program from:

- changing the memory protection boundary
- bringing the computer to an idle
- branching into or accessing data in protected memory
- changing the status register
- interfacing with system I/O operations.

Two status register bits (4 and 9) control the memory protect feature. When bit 4 is set, any attempt to access protected memory causes a system interrupt. The protected memory is defined by upper and lower limit registers in the memory controller. The two registers are loaded under program control with a standard I/O instruction. When bit 9 is set, program relocation operations are in effect. All addresses used in any memory access are modified by the contents of the lower limit register. Thus programs are relocated automatically with no changes in the programs required.

TABLE V

TI 980B FUNCTIONAL CHARACTERISTICS

Features	Characteristics
Processor	
Programmable Registers	1 accumulator 1 accumulator extension 1 index register 1 subroutine link register 1 base register
Control Unit	Microprogrammed
Instructions	98
Memory Reference	31
Register Operate	14
Shift	20
Skip	20
Bit Manipulation	8
Input/Output	5
Addressing	
Direct	256 words
Relative	128 backwards; 127 forward
Indexed	65K
Indirect	65K
Indirect (Post indexing)	65K
Extended	65K
Base-Displacement	65K
Base-Displacement Index	65K
Immediate	8, 16, or 32 bits
Interrupt System	Vectored with priority
Internal	1 line
External	3 lines
Memory	
Word Length	16 bits
Cycle Time (MOS Semiconductor)	750 nanoseconds
Capacity	
Minimum	8,192 words
Maximum	65,536
Memory Protect	Standard
Read Only Memory	256 units used for bootstrap loaders
Input/Output	
Maximum device controllers	256
Maximum transfer rates	
Programmed	130K words/sec
DMA	1,000K words/sec

Multi-accumulator Machines

Digital Computer Controls D-116

The Digital Computer Controls D-116 is a basic multi-accumulator machine that incorporates large scale integration (LSI) packaging concepts. The D-116 is offered in two versions. The D-116S version has a 1200 nanosecond cycle time. The D-116H version is faster with a 980 nanosecond cycle time. Both versions can be expanded up to 32K of memory. With the optional Memory Expansion and Protection Unit, maximum memory becomes 128K. Table VI shows the other important attributes of D-116.

The D-116 computer has three classes of instructions -- memory reference, arithmetic/logic, and input/output. As a multi-accumulator machine, one of the distinguishing characteristics is the separation of memory access and arithmetic/logic operations. The memory reference instructions use the two formats shown in Figure 20. There are only six memory reference instructions and none of them involve arithmetic/logic operations. The arithmetic/logic operations are restricted to interregister instructions. The format for these instructions are shown in Figure 21. Note that the arithmetic/logic instruction format allows for multiple functions. Thus an ADD instruction can also rotate right or left, or swap bytes of a register, test a result (or its carry bit) for a skip condition, and indicate whether the result should be retained (no load bit).

Data General ECLIPSE S/200

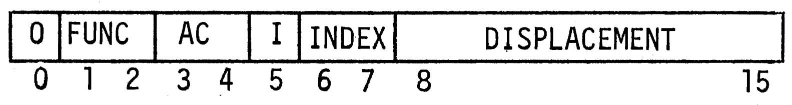
The Data General ECLIPSE S/200 is a multi-accumulator system with

TABLE VI

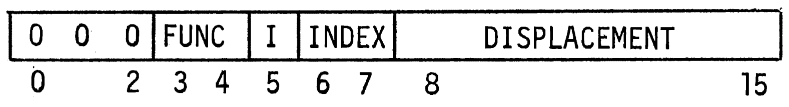
D-116 FUNCTIONAL CHARACTERISTICS

Components	Characteristics
Processor	
Programmable Registers	4 accumulators (2 can be used as index registers).
Control Unit	hardwired
Instructions	32
Arithmetic/Logic	8
Conditional Skips	7
Shifts	3
Memory Reference	6
Input/Output	8
Addressing	
Direct	256 words
Indirect	32,768
Indexed	32,768
Relative	256
Autoincrement Locations	8
Autodecrement Locations	8
Interrupt System	hardware party-line
Memory	
Word Length	16 bits
Cycle Time	960 nanosecond or 1200 nanosecond
Capacity	
Minimum	4096
Maximum	32,768 (128K with optional memory expansion and protect).
Increment	4K, 8K, or 16K words
Input/Output	
Maximum number of devices	62

an extensive range of features that makes it ideal for data communications, critical real-time process control, and high-speed computation applications. Among those features are Memory Error Checking and Correction and Memory Allocation and Protection providing high data and



(a) With Accumulator (I - Indirect)



(b) Without Accumulator

Figure 20. D-116 Memory Reference Instruction Formats

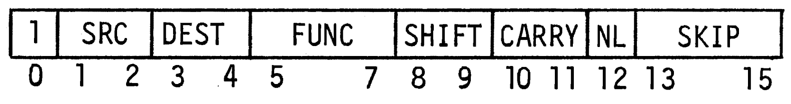


Figure 21. D-116 Arithmetic/Logic Instruction Format (NL - No Load)

system integrity. A hardware stack, hardware interrupt processing, Floating Point Processor, and memory interleaving are effective in maximizing system throughout. Semiconductor memories employing a "cache" system produce effective cycle times from 200 to 640 nanoseconds. Custom instructions can be developed through microprogramming with the writeable control store feature. Those features make the ECLIPSE S/200 one of the more sophisticated general purpose minicomputers today.

The functional characteristics of the ECLIPSE S/200 are listed in Table VII. Three of its features (Memory Error Checking and Correction, Writeable Control Store, and the Cache system) are described below.

TABLE VII

ECLIPSE S/200 FUNCTIONAL CHARACTERISTICS

Components	Characteristics
Processor	
Programmable Registers	4 accumulators (2 can be used as index registers) 4 64-bit floating point registers
Control Unit	microprogrammed
Instructions	193
Fixed Point Arithmetic	20
Logical Operations	9
Shifts	6
Bit/Byte Manipulation	10
Block Moves	2
Stack Manipulation	5
Control Transfers	15
Microprogramming	4
Memory Allocate and Protection	7
Floating Point Arithmetic	55
String and Decimal Manipulation	37
Input/Output	8
Interrupt	9
Special Instructions	6
Addressing	
Direct	256 words
Indirect	32,768 (multi-level)
Indexed	32,768
Extended	32,768
Relative	256
Autoincrement Locations	8
Autodecrement Locations	8
Interrupt System	
Type	Priority Polling
Levels	16
Memory	
Word length	16 bits
Cycle time (MOS)	700 nanoseconds
Capacity	
Minimum	8192 words
Maximum	32,768 (without MAP) 256K (with MAP)
Increments	8192

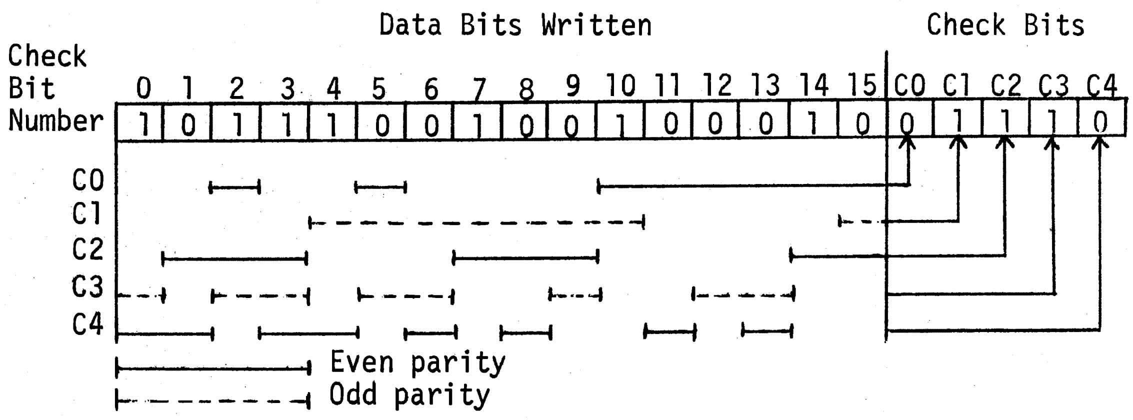
TABLE VII (Continued)

Components	Characteristics
Parity	single-bit error detect and correction, multiple-bit error detect
Writeable Control Store	256 56-bit words
Input/Output	
Maximum number of devices	59
DMA transfer rates	
Maximum input	1,250,000 words/sec
Maximum output	715,000 words/sec

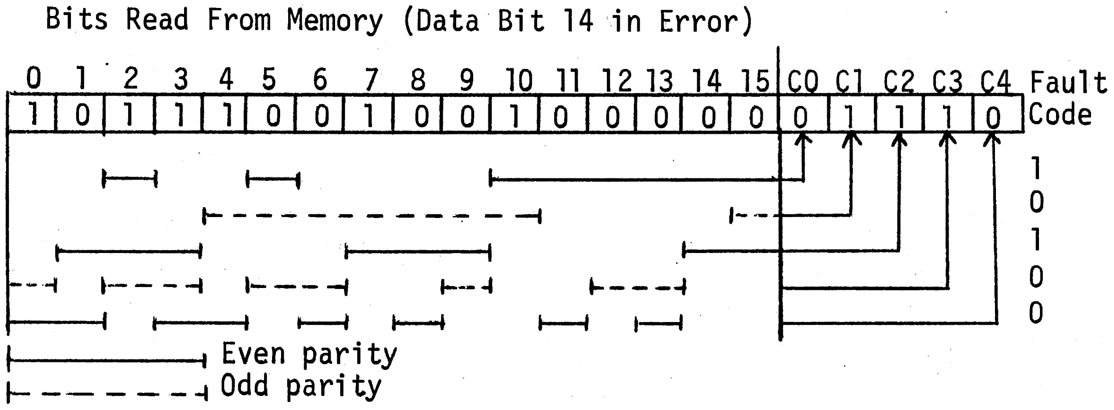
Memory Error Checking and Correction. The Error Checking and Correction (ERCC) feature of the ECLIPSE S/200 detects and corrects all single bit errors in the words of main memory. The feature also detects multiple-bit errors but does not correct them.

Figure 22 illustrates how ERCC works. When a word is written into memory a 5-bit check field is generated. Each check bit is set or cleared depending on its parity rule (odd or even) and the contents of selected fields in the word. For the example in Figure 22a the check bits generated are 01110.

Errors are detected during read-from-memory operations. When a word is read, a 5-bit fault code is generated. The check bits generated when the word was written are used in the calculation of the fault code. The Memory Fault Code Table (Table VIII) is then used to pinpoint the bit or bits in error. In Figure 22b, the fault code generated is 10100 indicating that bit 14 is error. Since it is a single bit error, the



(a) Word Written Into Memory


Bits Read From Memory (Data Bit 14 in Error)
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 C0 C1 C2 C3 C4
1 0 1 1 1 0 0 1 0 0 1 0 0 0 0 0 0 1 1 1 0
Fault Code
1
0
1
0
0
Even parity
Odd parity

(b) Word Read From Memory

Figure 22. Error Checking and Correction Example

error is corrected by complementing bit 14.

The Writeable Control Store. The Writeable Control Store (WCS) feature of the ECLIPSE S/200 allows the user to write and execute microprograms for customized applications (22). There are four instructions used in the process:

1. Specify Address
2. Load Microcode

TABLE VIII

MEMORY FAULT CODES

FAULTCODE	MEANING	FAULTCODE	MEANING
00000	No error	10000	Error in check bit 0
00001	Error in check bit 4	10001	Error in data bit 11
00010	Error in check bit 3	10010	Error in data bit 12
00011	Error in data bit 0	10011	Error in data bit 13
00100	Error in check bit 2	10100	Error in data bit 14
00101	Error in data bit 1	10101	All data bits and check bits in location are 0
00110	Multiple-bit error		
00111	Error in data bit 3	10110	Error in data bit 2
01000	Error in check bit 1	10111	Multiple-bit error
01001	Error in data bit 4	11000	Error in data bit 10
01010	All data bits and check bits in location are 1	11001	Multiple-bit error
		11010	Error in data bit 5
01011	Error in data bit 6	11011	Multiple-bit error
01100	Error in data bit 7	11100	Error in data bit 15
01101	Error in data bit 8	11101	Multiple-bit error
01110	Error in data bit 9	11110	Multiple-bit error
01111	Multiple-bit error	11111	Multiple-bit error

3. Load Decode Address

4. Enter WCS

The WCS itself contains 256 56-bit words with a 200 nanosecond cycle time. There is a WCS word register used in accessing specific WCS words.

The Specify Address instruction transfers the contents of a specified accumulator into the WCS word register. The format of the information in the accumulator depends on the instruction to be executed next. If it is a Load Microcode instruction then the format in Figure 23a applies. If a Load Decode Address instruction is executed then the

BIT NUMBER	CONTENTS
0-5	Unused
6-13	Address in WCS of the 56-bit microword that will be loaded by the following LOAD MICROCODE instruction.
14-15	Field of the 56-bit microword that will be loaded by the following LOAD MICROCODE instruction. If these bits are 00, the field is microcode bits 0-15. If these bits are 01, the field is microcode bits 16-31. If these bits are 10, the field is microcode bits 32-47. If these bits are 11, the field is microcode bits 48-55.

(a) Accumulator Format for Load Microcode

BIT NUMBER	CONTENTS
0-10	Unused
11-14	Entry number--from bits 6-9 of the corresponding XOP1 instruction.
15	Decode number. If this bit is 0, the following LOAD DECODE ADDRESS instruction will specify a decode 1 address. If this bit is 1, the following LOAD DECODE ADDRESS instruction will specify a decode 2 address.

(b) Accumulator Format for Load Decode Address

Figure 23. Specify Address Accumulator Formats

format in Figure 23b applies. The Load Microcode instruction transfers the contents of a specified accumulator into the field of the WCS word specified by the WCS word register according to the format in Figure 23a. The Load Decode Address instruction transfers bits 8-15 of a specified accumulator into the WCS word specified by the WCS word register according to the format in Figure 23b. Thus the first three instructions are used to write the microprograms. To execute them, the Enter WCS instruction must be executed. The feature allows for 16 possible entry points. The Enter WCS instructions must specify which entry point microcontrol transfers to.

Cache Memory. The Cache system used in the ECLIPSE S/200 computer consists of a 16-word, 200 nanosecond bipolar (10) memory that front-ends each 8K of 700 nanosecond MOS semiconductor memory modules. The Cache system combines the speed of fast, expensive bipolar memory with the economy of the slower MOS semiconductor memory. The system is fully automatic requiring no programming effort.

Figure 24 is a simplified diagram of the organization of a Cache system. When the processor requests a memory word, the content addressable memory (CAM) is used to find the requested word in the Cache. If the word is in the Cache, it is transferred to a processor register in 200 nanoseconds. If the word is not in the Cache, then CAM retrieves it from the MOS memory. The word goes to a processor register. At the same time the Cache is loaded with four sequential words (a block) including the word requested. If sequential program flow continues, the next word requested will be in the Cache, ready for a 200 nanosecond transfer.

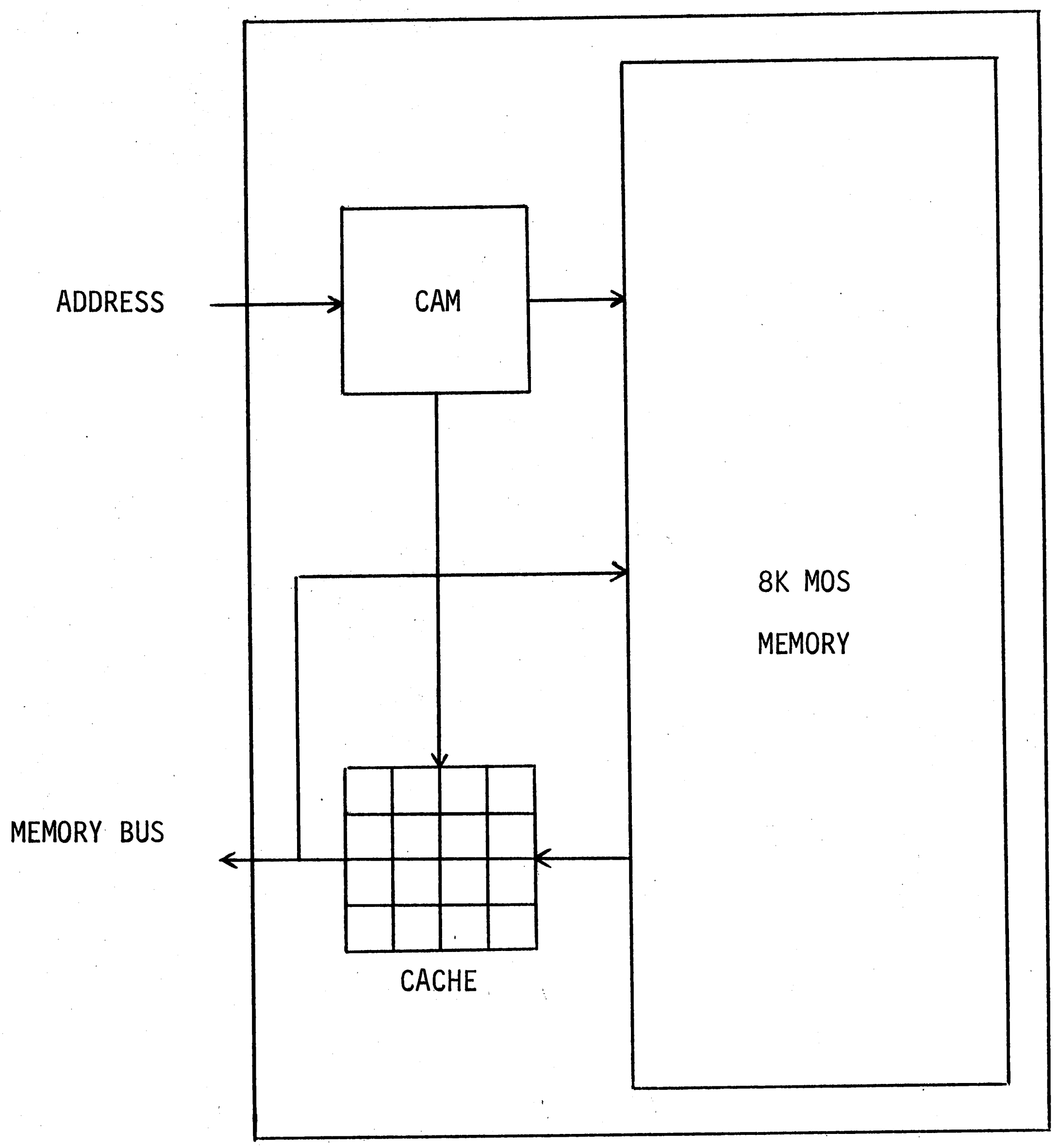


Figure 24. ECLIPSE S/200 Cache System

General Purpose Register Machines

Digital Equipment Corporation PDP 11/40

Introduced in the late 1960's by Digital Equipment Corporation, the PDP 11 system rivals the PDP 8 series in importance concerning minicomputer history. The architecture of a PDP 11 system promotes simplicity in designing complete system configurations involving a wide range of peripheral devices. The key feature in the architecture is the use of a universal bus (UNIBUS), through which all device to device communications are accomplished. The UNIBUS is also the key to the comprehensive, powerful, and straight forward instructions implemented.

The PDP 11/40 is one of the latest processors in the PDP 11 family. It is oriented towards a multi-programming environment such as time-sharing systems where many users at terminals are interacting concurrently, or real-time systems where many tasks may be active at the same time. The functional characteristics of the PDP 11/40 are listed in Table IX. The following sections are devoted to the structure of the UNIBUS, the instruction formats, and the hardware automatic priority interrupt system.

The UNIBUS. The UNIBUS of the PDP 11 systems is a single, common path that connects the central processor, memory, and all peripherals. Address, data, and control information are sent along 56 (mostly bi-directional) lines of the bus. The form of communication is the same for every device of the UNIBUS. The processor uses the same set of signals to communicate with memory as with peripheral devices. Each device, including memory locations, processor registers, and peripheral

TABLE IX

PDP 11/40 FUNCTIONAL CHARACTERISTICS

Components	Characteristics
Processor	
Programmable Registers	6 general purpose registers 1 system stack pointer 1 program counter
Control Unit	microprogrammed
Instructions	69
Arithmetic/Logic	10
Register change	19
Control transfers	22
Interrupt	11
Shifts and Rotates	7
Addressing	
Register Pointer	32,768 words (with autoincrement and autodecrement operations)
Indirect	32,768
Index	32,768
Immediate	operand word follows instruction
Extended	32,768
Relative	32,768
Interrupt System	
Type	vectored automatic priority
Traps (Internal)	9
External	5 lines (unlimited numbers of devices)
Memory	
Word length	16 bits
Cycle time	980 nanoseconds (core)
Capacity	
Minimum	8192 words
Maximum	32,768 (128K with memory management)
Increment	8K or 16K words
Parity	standard
Management and Protect	optional
Input/Output	
Maximum number of devices	unlimited
Maximum rates	
non-DMA	150K words/sec
DMA	2000K words/sec
Bus lines	56 bidirectional

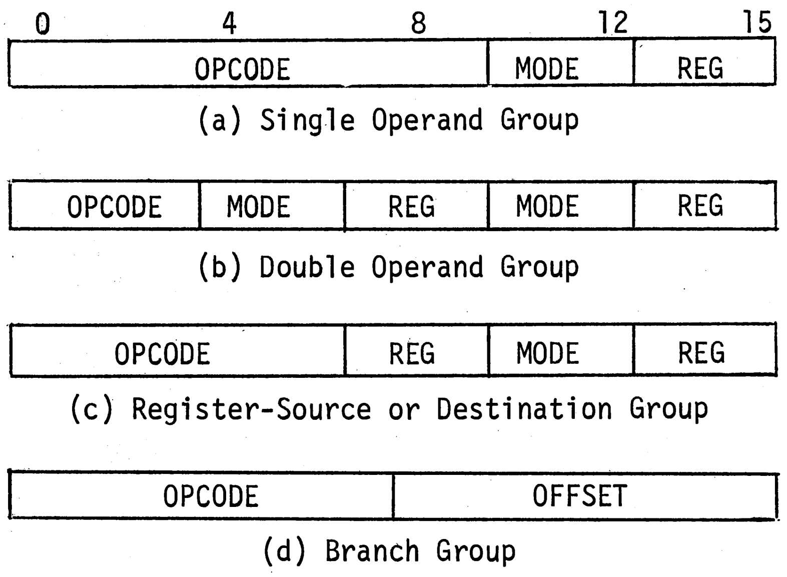
device registers, is assigned an address on the UNIBUS. Thus, peripheral device registers may be manipulated as flexibly as main memory by the central processor. All the instructions that can be applied to data in main memory can be applied equally well to data in peripheral device registers. This feature is especially powerful considering that data in main memory can be processed as though it were in a processor register.

Communication between two devices on the bus is in the form of a master-slave relationship. At any given time, there is one device that has control of the UNIBUS. The controlling device is called the "bus master". The device communicating with the bus master is called the "slave". A typical example of this relationship is the processor as the master device fetching an instruction from memory which is always the slave.

The master-slave relationship is interlocked or asynchronous. For each control signal issued by the master, there must be a response signal from the slave to complete the transfer. This does away with time constraints usually enforced on devices allowing operations at maximum possible speeds.

The Instruction Formats. Unlike conventional minicomputers which usually have three classes of instructions (memory reference, register operate, and input/output), all operations in the PDP 11/40 are accomplished with one set of instructions. This is due to the fact that processor registers, main memory locations, and peripheral device registers are processed in the same manner. Thus, the CPU can add data directly to a peripheral device register without bringing the device data into the memory or disturbing any of the general purpose registers.

In the same manner, data in main memory may be rotated as if it were in a processor register. The word formats for the PDP 11/40 instructions are shown in Figure 25. The addressing modes are listed in Table X.



(a) Single Operand Group

(b) Double Operand Group

(c) Register-Source or Destination Group

(d) Branch Group

Figure 25. PDP 11/40 Instruction Formats

The Automatic Priority Interrupt System. The PDP 11/40 has a multi-line, multi-level priority interrupt structure which is illustrated in Figure 26. Bus requests from external devices can be made on one of five request lines. Highest priority is assigned to non-processor requests (NPR). These are direct memory access transfers allowed by the processor between bus cycles of an instruction execution. Bus request 7 (BR7) has the next highest priority, BR4 has the lowest. Requests on lines BR4 through BR7 are honored between instruction executions. On each request line, higher priority is given to devices closer to the

TABLE X

PDP 11/40 ADDRESSING MODES

Mode Bits	General Register	Program Counter	Meaning
000	Register	Not used	Register contains operand
001	Register deferred	Not used	Register contains address of operand
010	Auto-increment	Immediate	Register contains address of operand then incremented by two
011	Auto-increment indirect	Extended	Register contains address of operand then incremented by two
100	Auto-decrement	Not used	Decrement register by two, result is address of operand
101	Auto-decrement indirect	Not used	Decrement register by two, result is address of operand address
110	Index	Relative	Operand address is the sum of register and contents of memory location following instruction word
111	Index deferred	Relative deferred	Operand address is indirect of address calculated by mode 6 scheme

UNIBUS. Thus, the priority system is two dimensional and provides each device with a unique priority.

Raytheon Data Systems RDS-500

The Raytheon Data Systems RDS-500 is a fast, general purpose register machine with unique characteristics. Its architecture allows for

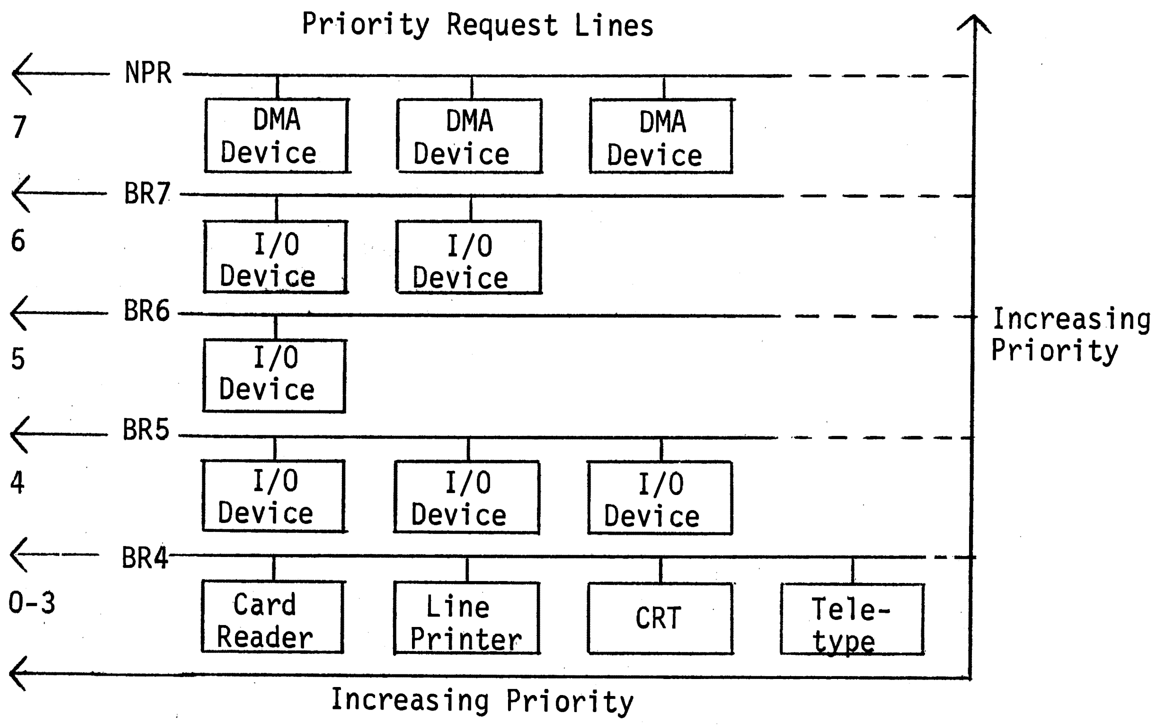


Figure 26. Automatic Priority Interrupt Structure

multiprogramming systems where real-time and batch operations can be processed simultaneously. Other applications include seismic data processing, production automation, communication systems, and business data processing. The RDS-500 functional characteristics are given in Table XI. The following sections describe its unique set of instructions and dual bus structure.

The Instruction Set. As mentioned in Chapter III, the type of instructions that a system uses usually depends on the organization of its programmable registers. In an RDS-500 system there are eight programmable registers -- one accumulator, one index register, and six general purpose registers. There is also a 6-bit extension register

TABLE XI

RDS-500 FUNCTIONAL CHARACTERISTICS

Components	Characteristics
Processor	
Programmable Registers	6 general purpose 1 accumulator 1 index
Control Unit	hardwired
Instructions	103
Arithmetic/Logic	26
Register Operate	10
Control	18
Interrupt and I/O	8
Shifts and Rotates	20
Conditional Skips	21
Addressing	
Direct	2048 words
Direct with extension	65,536
Index	65,536
Interrupt System	
Type	polling
Internal	4 levels
External	13 levels
Memory	
Type	Magnetic Core
Word length	16-bits
Cycle time	500 nanoseconds
Capacity	
Minimum	8192 words
Maximum	65,536
Increment	8192 or 16,384
Input/Output	
Maximum number of devices	
DMA	16
Non-DMA	16
Maximum DMA rate	2000K words/sec

used in effective address calculation.

In its basic instruction set, the RDS-500 has eleven classes of instructions described in the user's manual (23). Each class of instructions is associated with an instruction format. The association is illustrated in Figure 27. It is interesting to note that the following classes of instruction use only the accumulator and/or the index register:

- Memory Word Address
- Memory Byte Address
- 4-Bit Operand
- Shift
- No Operand
- Literal Byte Operand
- Input/Output
- Byte Page Specification

If there were no single register and two register classes of instructions the resulting instruction set would define a system using only fixed purpose registers -- the accumulator and the index register.

The Dual Bus Structure. Certainly a strong point in the RDS-500 design is its dual bus architecture illustrated in Figure 28. Superbus I connects the CPU, memory, programmed I/O, and eight DMA channels. Superbus II is attached to a second memory port and connects eight more DMA devices that can access memory simultaneously with the CPU. The optional Floating Point Processor is attached to Superbus I. The high speed devices are attached to a DMA multiplexer on either bus. The low-speed devices are attached to the programmed I/O channel that is

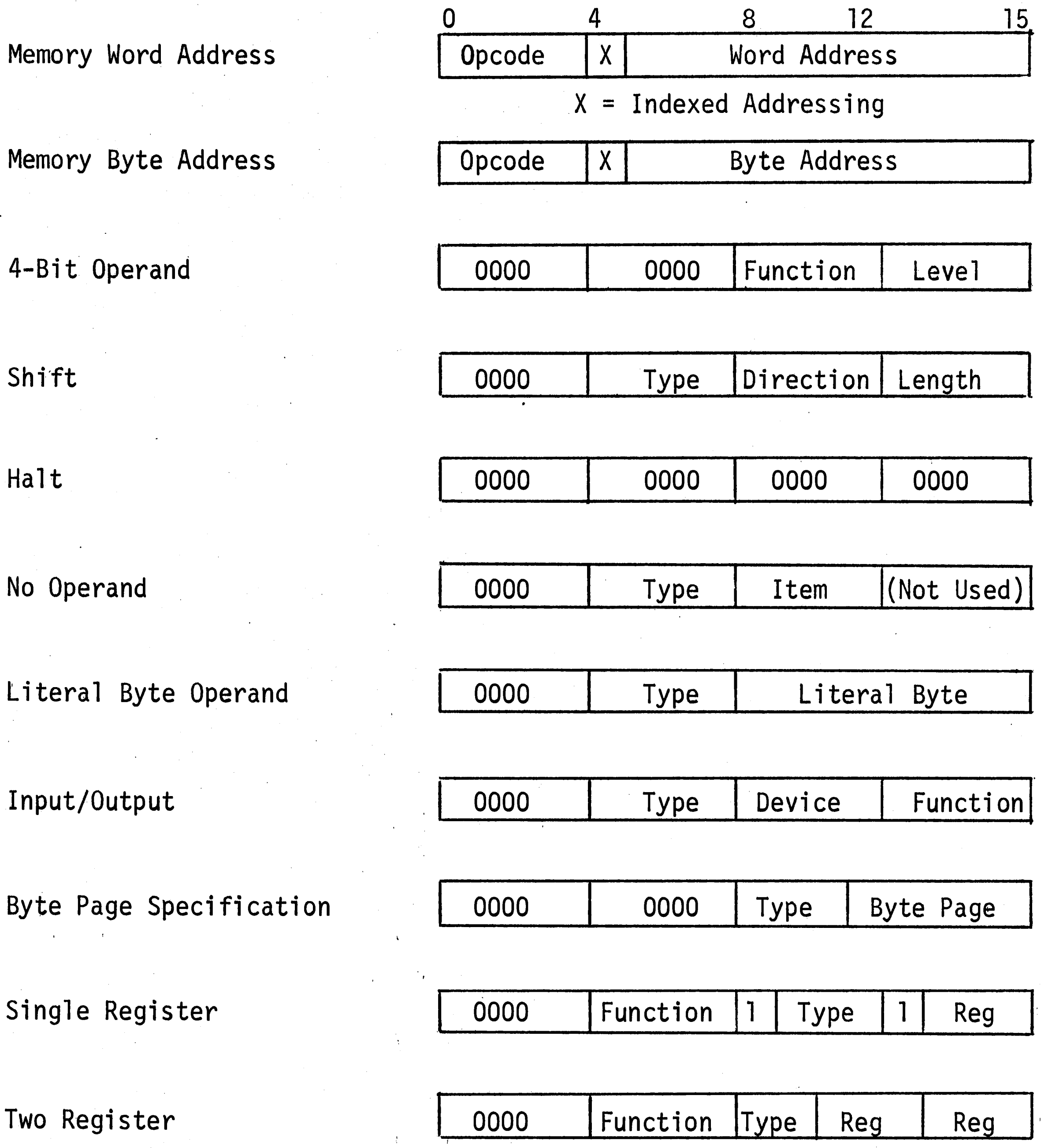

0 4 8 12 15
Memory Word Address: Opcode | X | Word Address
X = Indexed Addressing
Memory Byte Address: Opcode | X | Byte Address
4-Bit Operand: 0000 | 0000 | Function | Level
Shift: 0000 | Type | Direction | Length
Halt: 0000 | 0000 | 0000 | 0000
No Operand: 0000 | Type | Item | (Not Used)
Literal Byte Operand: 0000 | Type | Literal Byte
Input/Output: 0000 | Type | Device | Function
Byte Page Specification: 0000 | 0000 | Type | Byte Page
Single Register: 0000 | Function | 1 | Type | 1 | Reg
Two Register: 0000 | Function | Type | Reg | Reg

Figure 27. RDS-500 Instruction Formats

connected to Superbus I through the I/O and Interrupt Processor. With such a design the RDS-500 is capable of fast and powerful I/O operations.

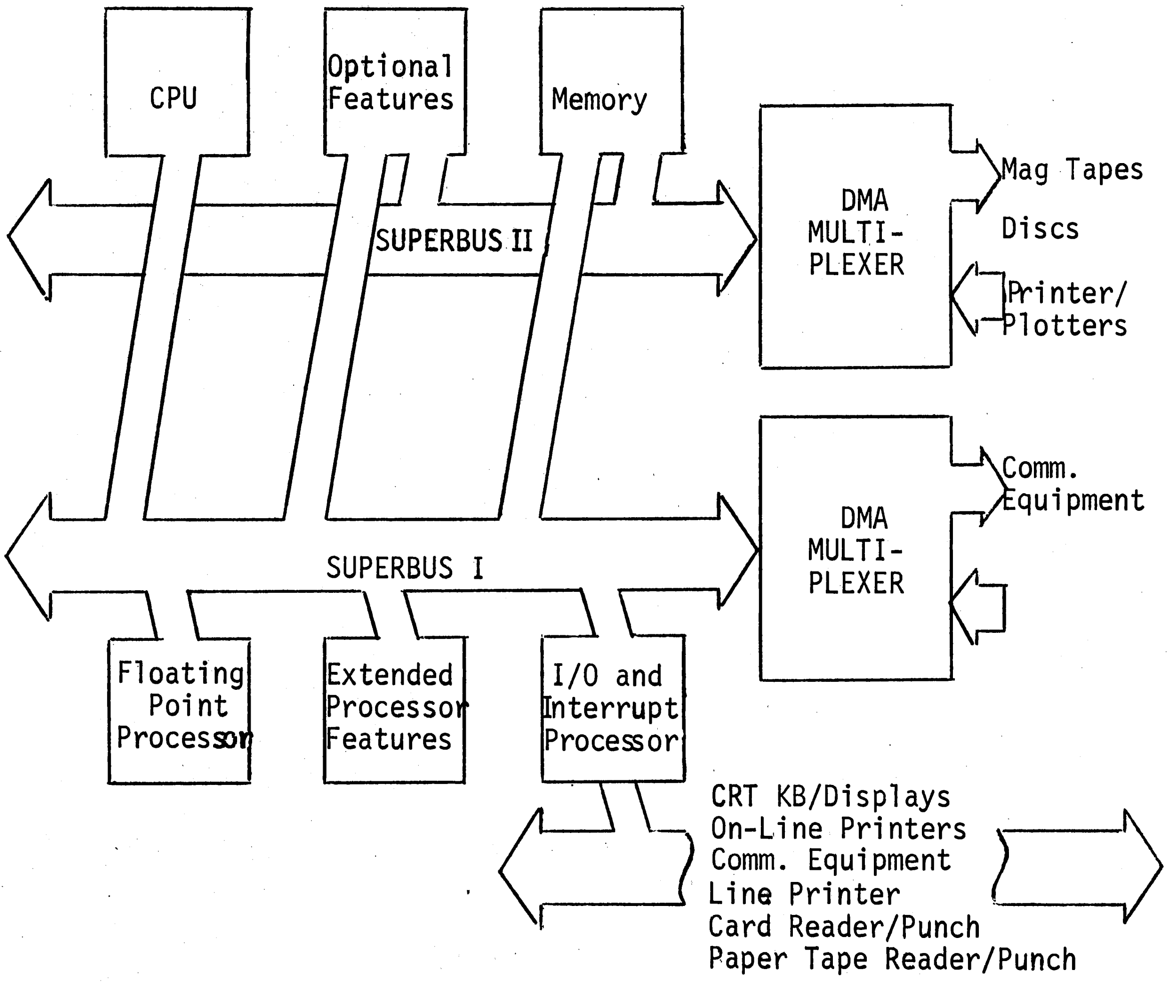


Figure 28. RDS-500 Dual Bus Architecture

Interdata Model 8/32

The Interdata Model 8/32 is a general purpose register machine marketed by Interdata as a high performance minicomputer (16). A look at its system block diagram (Figure 29) and functional characteristics (Table XII) reveals that the 8/32 is a system that can compete with medium and large-scale computers. The following sections describe the important features of the Interdata 8/32.

The 32-bit Architecture. The Interdata 8/32 provides a full 32-bit parallel structure. The memory word size, the general purpose registers, and the data paths are all 32-bits in length. Such an architecture is obviously superior to that of the typical 16-bit minicomputer. For example, the range of the integers used in a 16-bit machine is $\pm$ 32,768, in a 32-bit machine the range is $\pm$ 2,147,483,648. Another example involves the respective addressing capabilities. A 16-bit machine can address (directly, indirectly, or with indexing) at most 65,536 bytes of memory. In order to expand its addressing capabilities beyond 65,536 bytes, the 16-bit system must resort to some type (hardware, firmware, or softward) of a memory management package. The 32-bit system can address 1,048,576 bytes without the aid of a memory management package.

How does Interdata justify calling a 32-bit machine a minicomputer? Interdata's contentions are twofold. First is the fact that the processor and memory fit in a 19 x 14 x 28 inch mainframe. The typical minicomputer fits in a 19 x 11 x 21 inch mainframe (5) (26). The second of Interdata's contentions is that an Interdata 8/32 can be purchased at a competitive price. Whether or not this machine actually belongs in the minicomputer class is debatable. In either case the machine has very

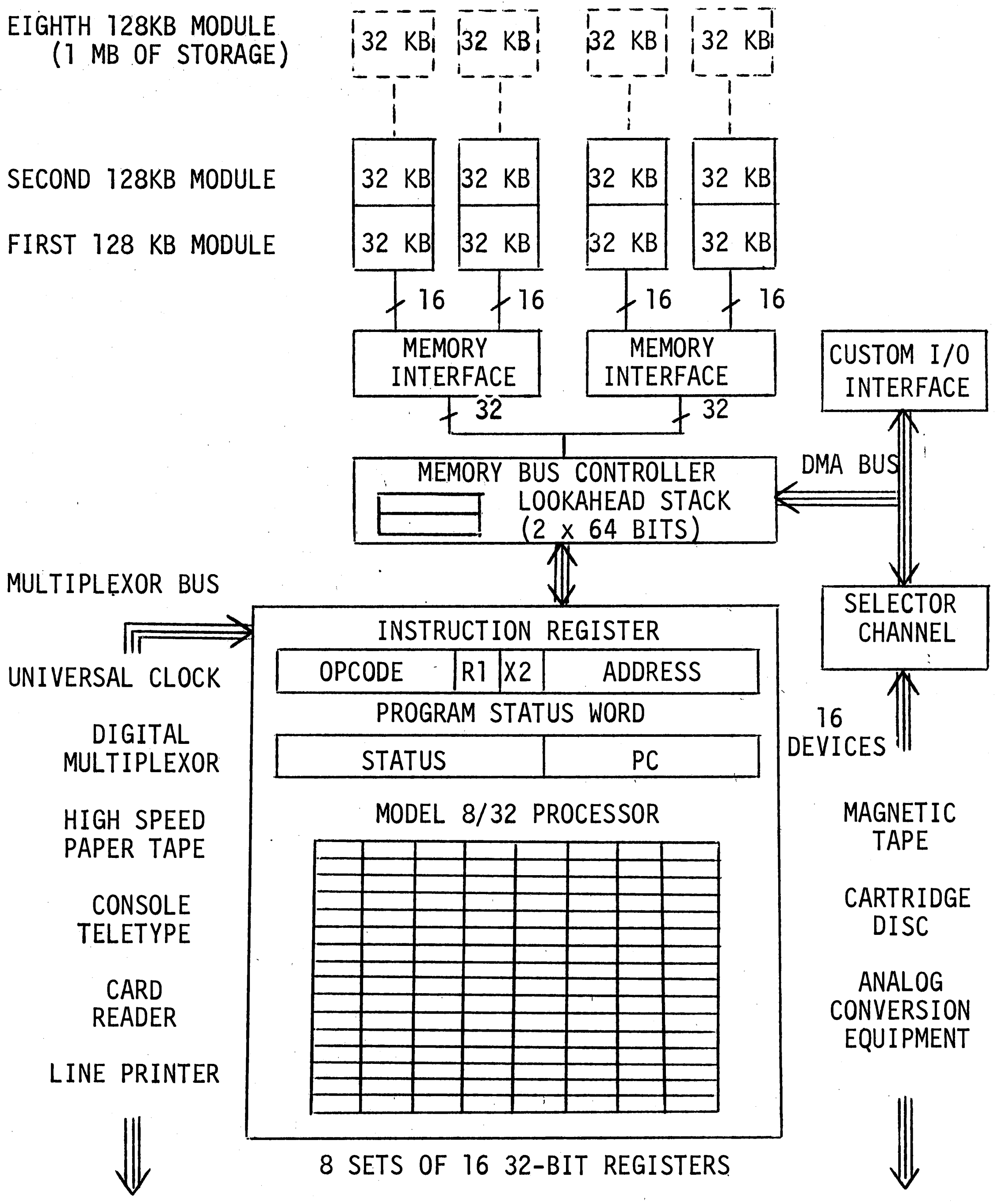


Figure 29. Model 8/32 Processor Block Diagram

TABLE XII

MODEL 8/32 FUNCTIONAL CHARACTERISTICS

Components	Characteristics
Processor	
Programmable Registers	2 or 8 sets of 16 32-bit general purpose registers 8 32-bit floating point registers 8 64-bit double precision floating point registers
Control Unit	microprogrammed (50 ns ROM memory)
Instructions	165
Load and Store	14
Fixed Point Arithmetic	27
Shifts	14
Floating Point	30
Status and Control	4
List Manipulation	4
Input/Output	18
Byte Manipulation	6
Branch on Condition	13
Communications	4
Bit Manipulation	5
Microprogramming	4
Addressing	
Direct	1,048,576 bytes
Relative	± 16,384 bytes
Indexing	1,048,576 bytes
Interrupt System	
Type	vectored priority
Levels	4
Memory	
Word Length	32 bits
Cycle Time	750 nanoseconds (CORE)
Memory Capacity	
Minimum	131,072 bytes
Maximum	1,048,576 bytes
Increment	131,072 bytes
Parity	optional 1 bit per 16 data bits

TABLE XII (Continued)

Components	Characteristics
Input/Output	
Maximum number of devices	1024
Number of DMA ports	7
Maximum transfer rates	
Programmed	166 bytes/sec
Block	387K bytes/sec
DMA	3.2M bytes/sec 6.0M bytes/sec (burst mode)

interesting features. These features are discussed below.

Multiple Register Sets. A Model 8/32 machine has at least two sets of 16 general purpose registers. Optionally the number of register sets may be expanded to eight. The 8/32 has a Program Status Word (PSW) that defines the state of the processor at any given time. Bits 24-27 of the PSW are used to designate the current register set. If only two sets are implemented then Bit 24 is used to select one of the two sets. If eight sets are implemented bits 25-27 are used to select a register set. Figure 30 illustrates the numbering of the register sets.

Basically the 8/32 is in one of three states when executing, the operating system state, and input/output state, or a user state. Multiple register sets simplify programming in switching from one state to another. In a system with one register set, changing from one state to another involves storing and restoring of the one register set. With multiple register sets, each state or levels within a state (see Figure

BITS 24-27 OF PSW	REGISTER SET NUMBER		I/O PRIORITY LEVEL
0000	0	EXECUTIVE SET	0
0001	1		1
0010	2	OPTIONAL	2
0011	3	REGISTER	3
0100	4	SETS	
0101	5		
0110	6		
	7		
	8		
	9		
	10	UNIMPLEMENTED	
	11	SETS	
	12		
	13		
	14		
1111	15	USER SET	

Figure 30. Register Set Numbering

30) may be assigned a specific register set. Furthermore, assigning two or three register stacks to the I/O system allows the "nesting" of device response on a hierarchical priority basis. State switching is thus rapid and straightforward.

High Speed Processing. One of the primary emphasis in the design of the Interdata Model 8/32 is speed. Two key features in the 8/32 design are the four-way interleaved memory and the lookahead stacks. With the four-way interleaved memory, a 750 nanosecond core memory can have an effective cycle time of 300 nanoseconds. The lookahead stacks (see Figure 29) act as a high speed dual memory buffer allowing the CPU and the memory to run largely in parallel. In systems where the CPU is executing varying length instructions of varying execution times, either the memory is waiting on the CPU or the CPU is waiting on the memory. With the lookahead stacks, the memory can anticipate the CPU's memory requests (since program execution is primarily sequential), fill these request in the dual 64-bit lookahead stacks, and go on to perform other memory functions such as DMA memory requests.

The Instruction Formats. The Interdata 8/32 has an instruction repertoire of 165 commands defining bit, byte, halfword and multi-word operations. The seven basic formats are shown in Figure 31. The abbreviations used have the following meanings:

OP	Operation code
R1	First operand register
R2	Second operand register
N	A four bit immediate value
X2	Second operand single index register
D2	Second operand displacement

FX2	Second operand first index register
SX2	Second operand second index register
A2	Second operand direct address
I2	Second operand immediate value

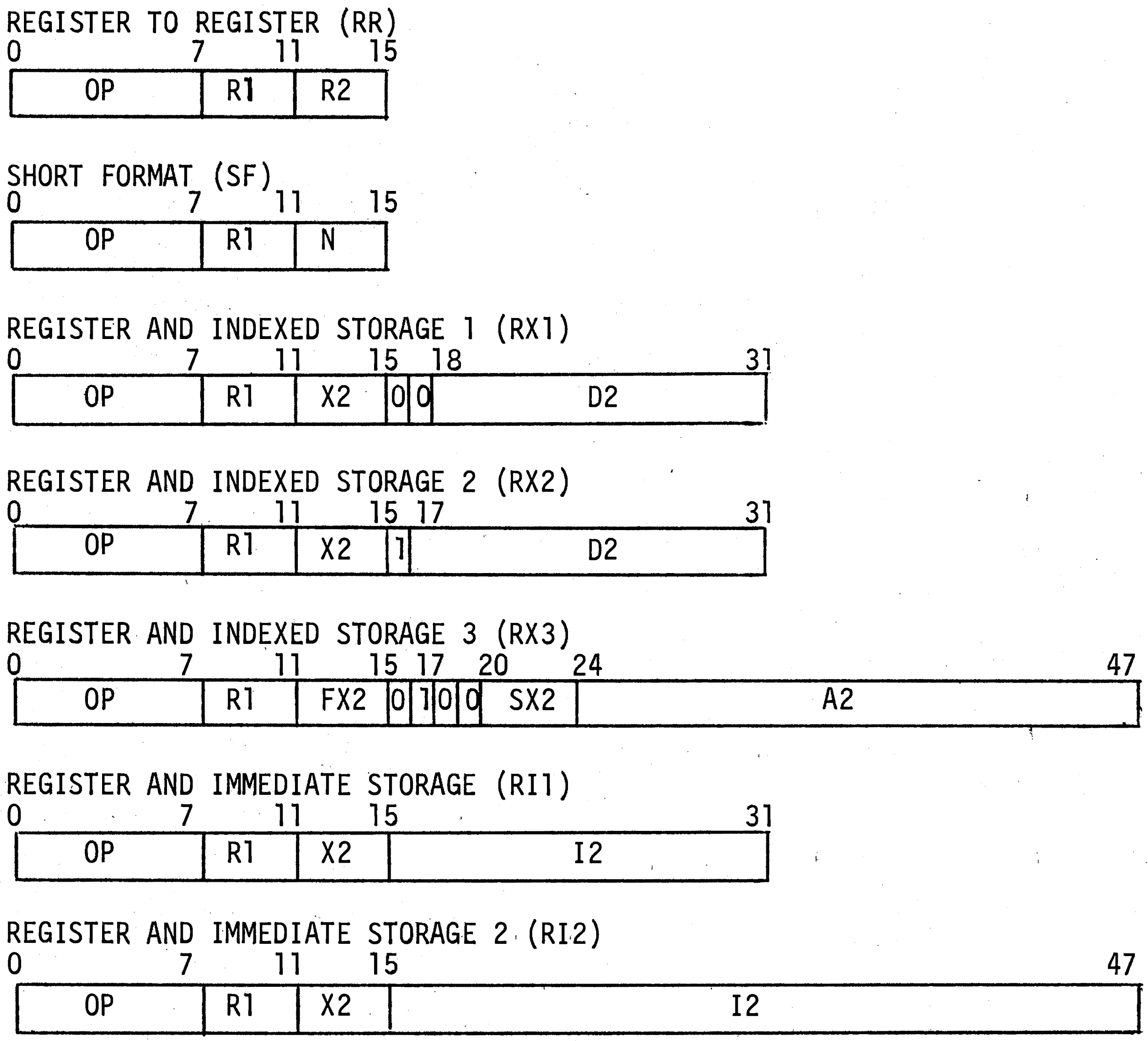

REGISTER TO REGISTER (RR)
0 7 11 15
OP R1 R2
SHORT FORMAT (SF)
0 7 11 15
OP R1 N
REGISTER AND INDEXED STORAGE 1 (RX1)
0 7 11 15 18 31
OP R1 X2 0 0 D2
REGISTER AND INDEXED STORAGE 2 (RX2)
0 7 11 15 17 31
OP R1 X2 1 D2
REGISTER AND INDEXED STORAGE 3 (RX3)
0 7 11 15 17 20 24 47
OP R1 FX2 0 1 0 0 SX2 A2
REGISTER AND IMMEDIATE STORAGE (RI1)
0 7 11 15 31
OP R1 X2 I2
REGISTER AND IMMEDIATE STORAGE 2 (RI2)
0 7 11 15 47
OP R1 X2 I2

Figure 31. Model 8/32 Instruction Formats

A Stack Machine--The Microdata 32/S

The Microdata 32/S is a push-down architecture computer implemented via firmware on the microprogrammable Microdata 3200 computer (11). Its architecture is designed in conjunction with the design of the Microdata Programming Language (MPL), a high-level language based upon the extensive and sophisticated language PL/I. In effect the MPL Compiler is used as a replacement for a 32/S assembler. Since the structures of MPL and 32/S are coordinated, the machine code produced by the MPL compiler is as efficient as the code which can be obtained with assembly language programming on a conventional architecture computer. The relationship of the 3200 microprocessor, the 32/S Computer, and the MPL machine are summarized in Figure 32.

SYSTEM	LOGICAL MACHINE	PROGRAMMING METHOD
3200	MICROPROGRAMMABLE MACHINE	MICRO INSTRUCTION (32 BITS)
32/S	3200 + 32/S FIRMWARE	MACRO INSTRUCTION (VARIABLE LENGTH)
MPL MACHINE	32/S + MPL COMPILER	MPL STATEMENTS

Figure 32. The 3200, 32/S, MPL Heirarchy

The 3200 Microprocessor

The 3200 Microprocessor is a 16-bit machine with 4K to 128K words of 300 nanosecond MOS semiconductor main memory, addressable to the 8-bit/byte level. It is microprogrammed with a bipolar 32-bit control memory which is expandable to 4K words, and which operates with 135 nanosecond cycle time. A common bus (Monobus) is implemented for connecting the microprocessor with all main memory modules and I/O device controllers. Like the UNIBUS or the PDP 11/40, the Monobus is asynchronous allowing memories and controllers of various speeds to be mixed and uniformly accessed with standard memory reference instructions. Input/output can be byte or word oriented under program control, or block oriented under computer control (concurrent I/O) or DMA control. Four external interrupt lines establish the relative priorities of groups of I/O device controllers. Relative priority among the controllers on each line is established by their positions along the Monobus. Each I/O device controller may be manually assigned a specific address and interrupt line. Thus for each I/O device address, a unique interrupt processing procedure and environment is specified.

The 32/S Architecture

The Microdata 32/S is a firmware implemented 16-bit, 350-450 nanosecond MOS memory cycle, push-down stack architecture computer. Figure 33 is a simplified block diagram of the 32/S system configuration. Its functional characteristics are given in Table XIII.

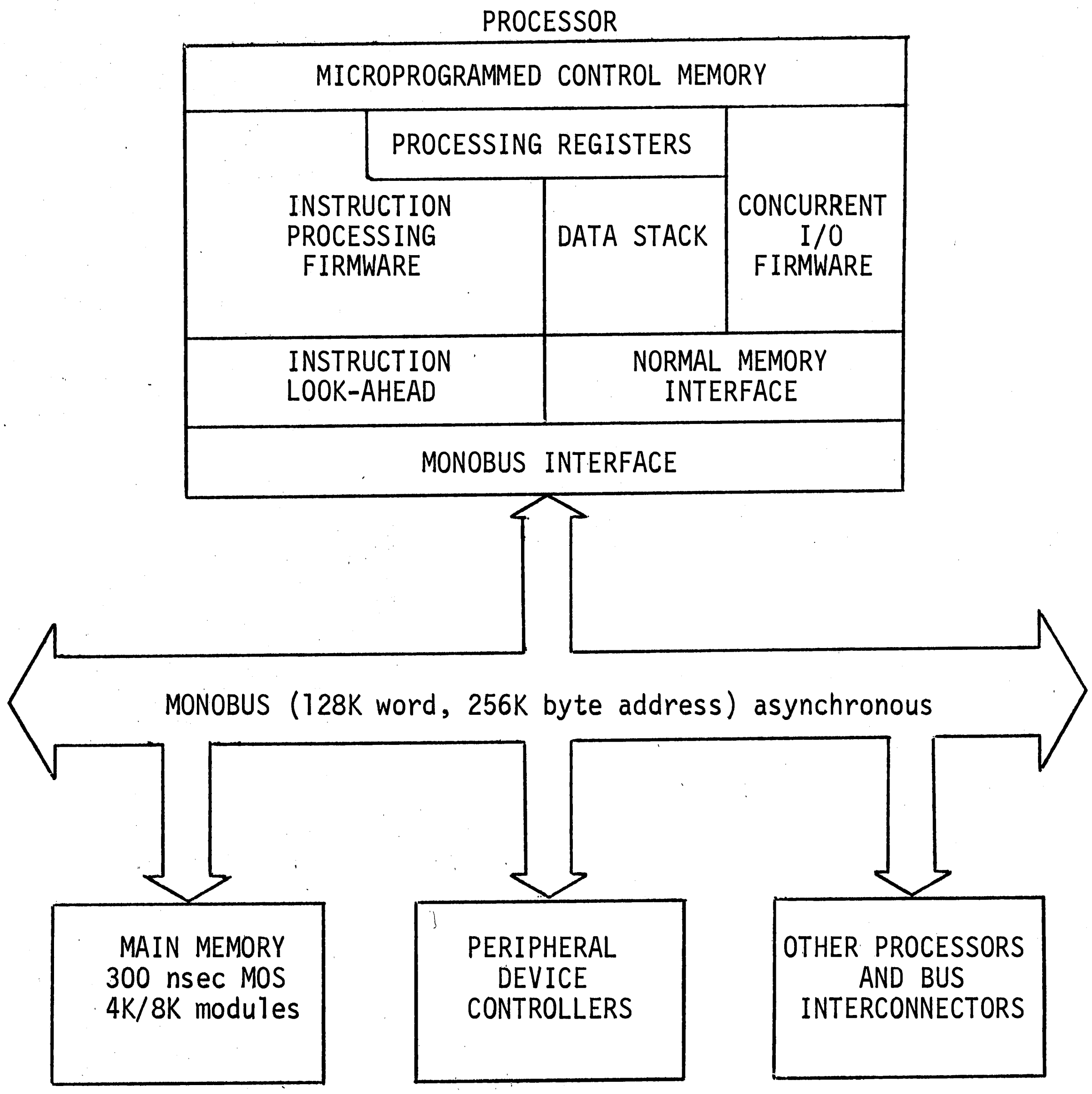

PROCESSOR
MICROPROGRAMMED CONTROL MEMORY
PROCESSING REGISTERS
INSTRUCTION PROCESSING FIRMWARE
DATA STACK
CONCURRENT I/O FIRMWARE
INSTRUCTION LOOK-AHEAD
NORMAL MEMORY INTERFACE
MONOBUS INTERFACE
MONOBUS (128K word, 256K byte address) asynchronous
MAIN MEMORY 300 nsec MOS 4K/8K modules
PERIPHERAL DEVICE CONTROLLERS
OTHER PROCESSORS AND BUS INTERCONNECTORS

Figure 33. 32/S System Configuration

TABLE XIII

Microdata 32/S Functional Characteristics

Components	Characteristics
Processor	
Programmable Registers	program base program pointer program length stack base environment pointer stack pointer stack length 5 stackhead registers
Control Unit	microprogrammed
Control Memory	
Word length	32 bits
Cycle time	135 nanoseconds
Maximum size	4096 words
Instructions	151
Memory reference	15
Stack operate	88
Branch	17
Control	22
Addressing modes	
Global direct	64K bytes
Local direct	256 bytes
Indexed	64K bytes
Indirect	64K bytes
Interrupt system	
Type	vectored priority
External	4 lines
Memory	
Type	MOS Semiconductor
Cycle time	300 nanoseconds
Word length	16-bits
Capacity	
Minimum	8K bytes
Maximum	256K bytes
Increments	8K or 16K bytes
Input/Output	
Maximum number of devices	1024
Maximum transfer rate	5M bytes/sec

The Monobus. The Monobus has an addressing range of 256K bytes. Modules on the Monobus include main memory, I/O device controllers, and control memory (see Figure 34). The 256K byte addressing range is divided into four 64K-byte banks. This division into banks results from the fact that all address arithmetic is performed on the least-significant bits of the 18-bit byte-level Monobus addresses. No carry from this 16-bit arithmetic is propagated into the most significant 2-bit field of the Monobus addresses. Therefore, addresses which should cross into the next bank when incremented or when increased by a displacement or index "wraps around" to the beginning of the same bank.

Main memory is provided in 16K byte modules. Each module has a 4-bit switch to select the 16-K range of Monobus addresses for that module. Modules are assigned sequential 16K ranges starting at Monobus address 0 to form a contiguous memory. The first 32 bytes are reserved for special purposes. In addition, up to 1024 bytes beginning at location 32 are reserved for the program library (PLIB), a table of pointers to program segments. The remainder of main memory is assigned (by the loader and/or software operating system) to program segments and data stacks.

The control memory interfaces with the CPU for control purposes via the control memory bus. In addition, however, control memory provided on an optional read-only memory board can be read through the Monobus, and control memory provided in optional writeable control memory modules can be read or written through the Monobus. Starting at location 224K on the Monobus, control memory may be addressed.

I/O device controllers are assigned 16 byte blocks of Monobus addresses, referred to as Device Register Blocks (DRB). A multi-channel

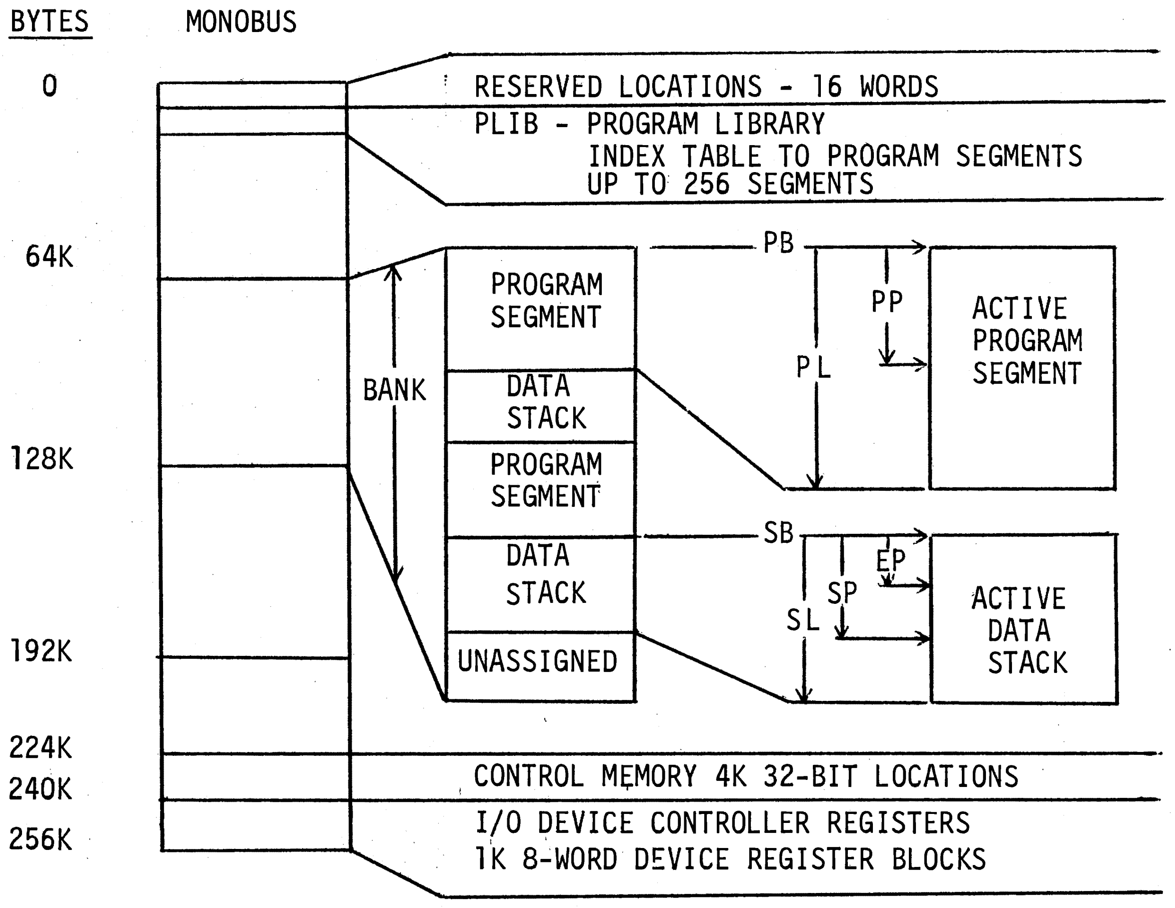


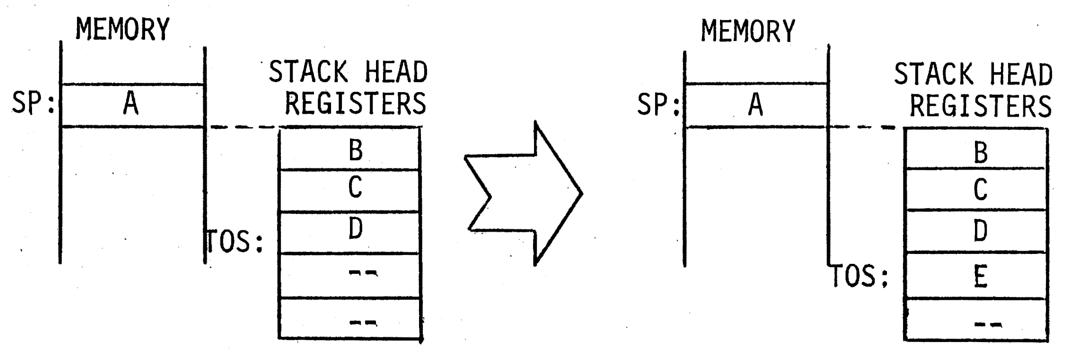
Figure 34. Monobus Organization

controller has a DRB for each device it controls. The Monobus addresses for the DRB's begin at location 240K. A switch is provided on each controller to select one of 1024 device numbers for each DRB associated with a controller.

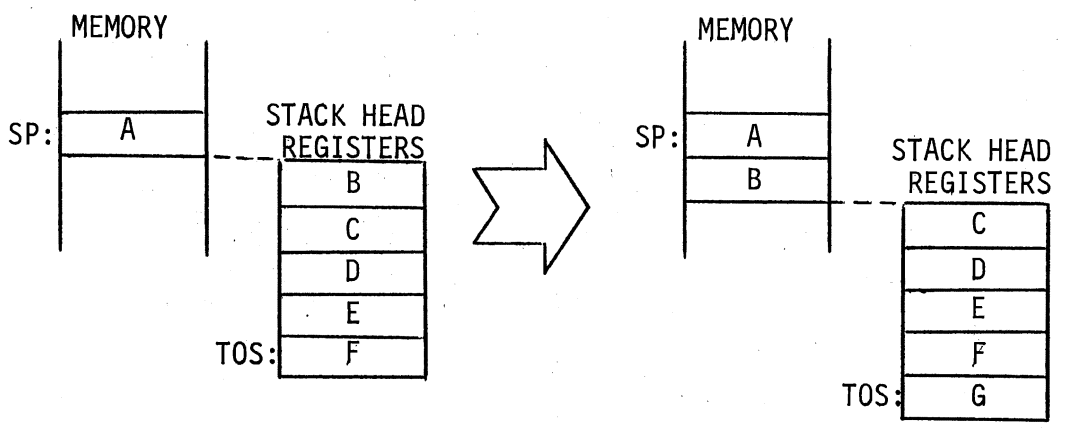
Active Program and Data Stack Registers. At any given point in the operation of the 32/S machine there is an active program segment and an data stack. A program segment contains the code generated for one or more MPL procedures, literal and constant data, and an indirect address table to entry points within the procedure code. Three registers define the active program segment: the program base (PB) specifies the base address of the segment; the program pointer (PP) specifies the address of the instruction to be executed; and the program length (PL) specifies the size of the segment. A data stack is an area in memory allocated for data of a user. Four registers define the active data stack: the stack base (SB) specifies the base address of the stack; the environmental pointer (EP) specifies the beginning location, relative to SB, of the current environment in the stack; the stack pointer (SP) specifies the location, relative to SB, of the top of the stack in main memory; and the stack length (SL) specifies the maximum size allocated to the stack. Figure 34 illustrates the use of these registers.

The Stack Head Registers. The data stack head registers consist of five high-speed registers. The number of active stack head registers is variable. Hardware logic maintains a record of which stack head registers are empty and which one (if any) is the current top of the stack, so that data can be pushed into the stack or popped from the stack without transferring data between registers. Most data stack operations as a result, actually can be performed within 135 nanosecond clock time.

The processor operates in such a way as to use the stack head registers to minimize accesses to main memory. When data is pushed into the data stack, it goes into a stack head register rather than into main memory as long as there are any empty stack head registers. If the stack head registers are filled, or become filled during a push operation, the deepest entry in the registers overflow into main memory. Both situations are illustrated in Figure 35. Note that the stack pointer (SP) always points to the highest stack location in main memory.



(a) With Empty Stack Head Registers

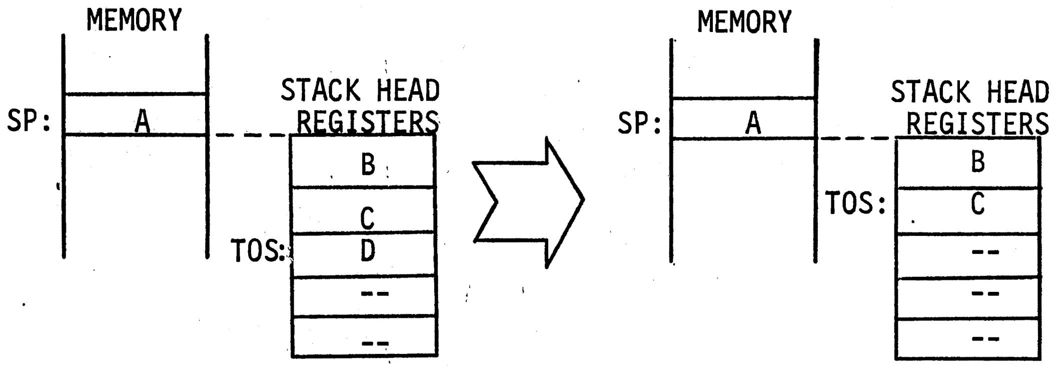


(b) All Stack Head Registers Filled Initially

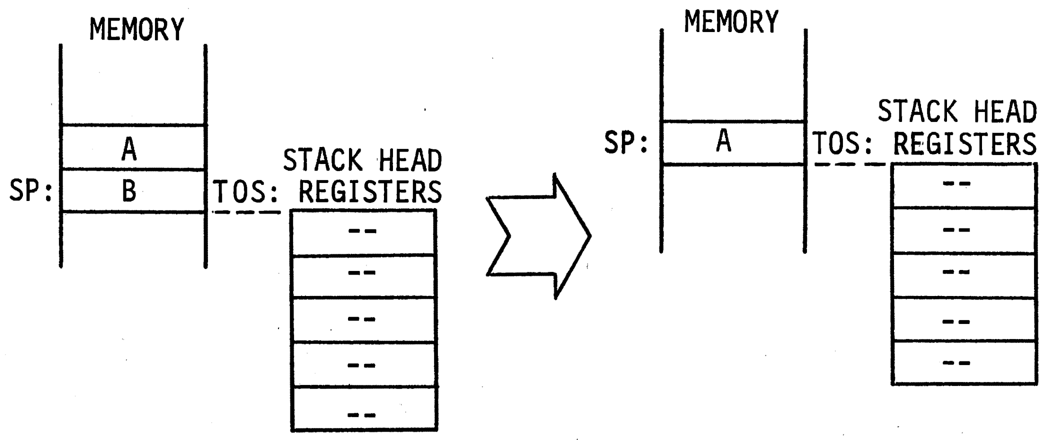
Figure 35. Push Stack Operation

During the operation of popping data from the stack, the processor pops data from the stack head registers without accessing main memory as long as data is available within the stack head registers. A pop operation occuring with the stack head empty simply causes the contents of the SP register to be decremented. No data is moved. The pop operation is illustrated in Figure 36.

Machine Instructions. The Microdata 32/S Computer Reference Manual (11) divides the 32/S non-input/output machine instructions into the following five categories:



(a) Filled Stack Head Registers Initially



(b) All Stack Head Registers Empty Initially

Figure 36. Pop Stack Operation

- Memory Reference
- Stack Operate
- Branch
- Control
- String

This section describes the basic characteristics of each of the five categories.

The memory reference instructions perform operations that load and store data of varying lengths, add and subtract a word to a data stack, and add a word to a memory location. Three formats, shown in Figure 37 are used to accomodate eight addressing modes for each of the 15 types of instructions. The addressing modes are listed in Table XIV. Definitions of the terms used in the effective address expressions are:

SB	Stack Base register value, 18 bit absolute address of base of data stack.
EP	Environmental pointer register value, 16 bit address of the base of the current Mark, relative to SB.
D8	8-bit address displacement field of instruction.
D16	16-bit address displacement field of instruction.
TOS(X)	16-bit index contained in TOX, the top level of the stack. This index specifies a number of data items, independent of data length; e.g., number of bytes, number of words, etc. It is converted to a byte-level index when the memory reference instruction is executed. For example, if an indexed doubleword instruction is executed, the index value is multiplied by 4.
TOS(D16)	16-bit address displacement contained in TOS, the top level of the stack.
TOS1(D16)	16-bit address displacement contained in TOS1, the second to top level of the stack.

TOS1(D18) 18-bit base address contained in TOS1, the second to top level of the stack. NOTE: Only the most significant 16 bits of the D18 displacement are stored in TOS1; the least significant 2 bits are assumed to be zeroes. The TOS1(D18) is multiplied by 4 (as shown for mode 7) when calculating the effective address.

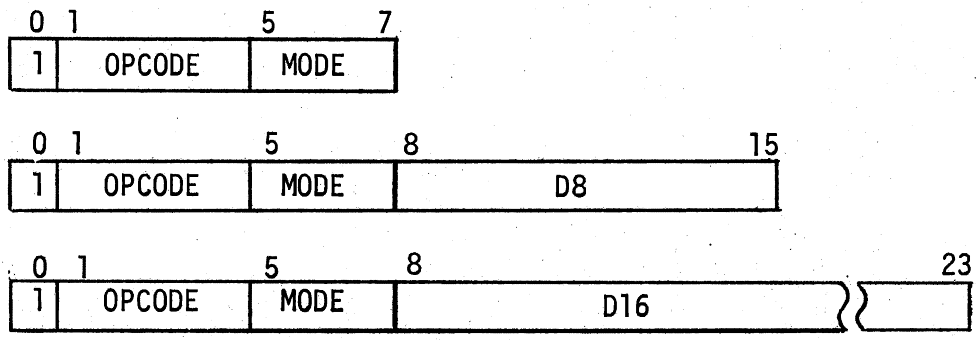


Figure 37. Memory Reference Instruction Formats

TABLE XIV

ADDRESSING MODES AND EFFECTIVE ADDRESSES

ADDRESSING MODE	EFFECTIVE ADDRESS	USE
0	SB + D16	GLOBAL DIRECT
1	SB + D16 + TOS (X)	GLOBAL DIRECT, INDEXED
2	SB + EP + D8	LOCAL DIRECT
3	SB + EP + D8 + TOX (X)	LOCAL DIRECT, INDEXED
4	SB + TOX (16)	INDIRECT THRU TOS
5	SB + TOX (X) + TOS 1 (D16)	INDIRECT THRU TOS, INDEXED

TABLE XIV (Continued)

ADDRESSING MODE	EFFECTIVE ADDRESS	USE
6	PB + D16 + TOS (X)	CONSTANT DIRECT, INDEXED
7	TOS (X) + 4 * TOS 1 (D18)	ABSOLUTE, INDEXED

The stack operate instructions operate on one or two operands in the top of the stack, or push a literal operand into the stack. The stack operate instructions may be categorized as follows:

- arithmetic, word operand
- arithmetic, double word operand
- logical
- comparison, arithmetic word and double word, logical, floating
- shift word and double word
- load literal
- stack modifications
- field description generation

The stack operate instruction formats are shown in Figure 38. The top two formats, which consist only of one or two-byte operation code, are used for all instructions except the load literal instructions. All operands for the instructions using these two formats are popped from the top of the stack during execution of the instruction. The result produced is pushed into the top of the stack. The last five formats in Figure 38 are used for the load literal instructions. The L field (one

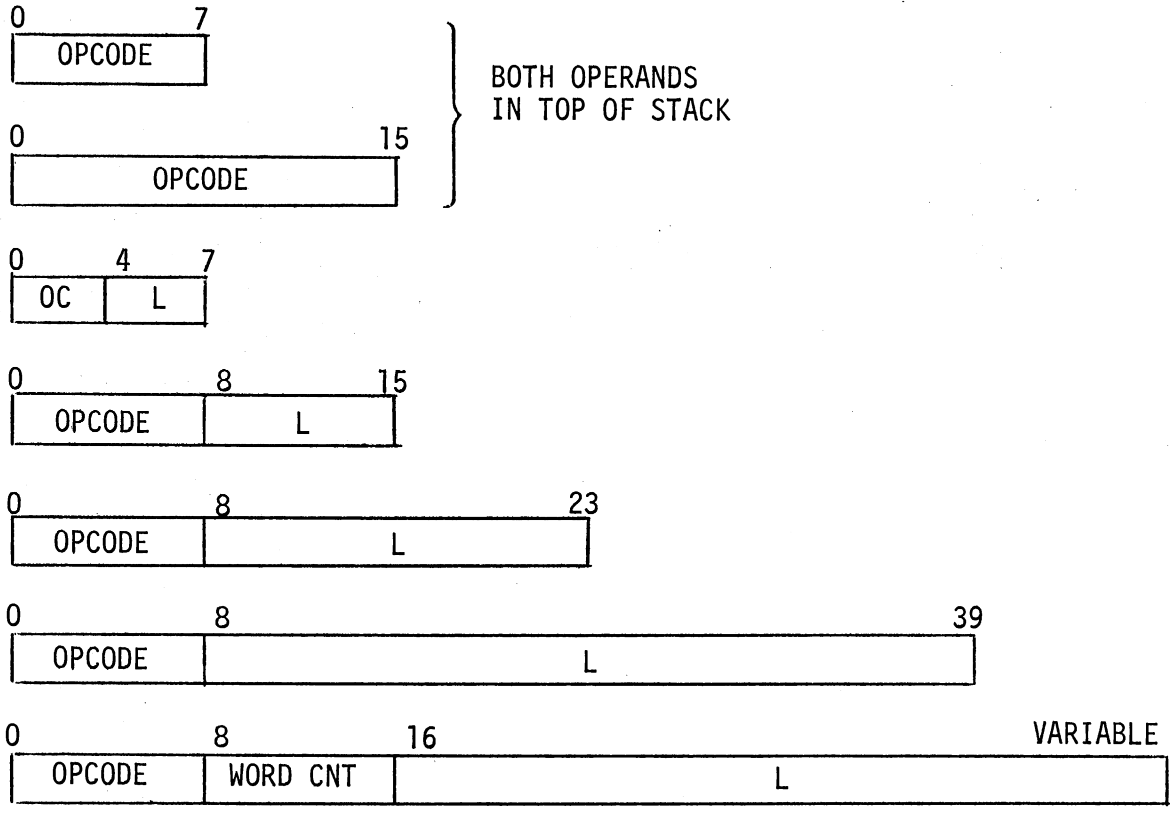


Figure 38. Stack Operate Instruction Formats

or more bytes) is the literal to be pushed into the stack.

Three categories of 17 branch instructions are shown in Figure 39. Branch backward instructions use a two byte format, with the rightmost byte being an eight bit displacement (D8). The effective address to be placed in the program pointer (PP) is computed by subtracting the displacement D8 from the current value of PP which is the address of the next instruction. Branch long instructions use a three byte format, with the rightmost two bytes being a 16-bit address (ADDR). Branch indirect via the top of the stack (TOS) register instructions use a one-byte format.

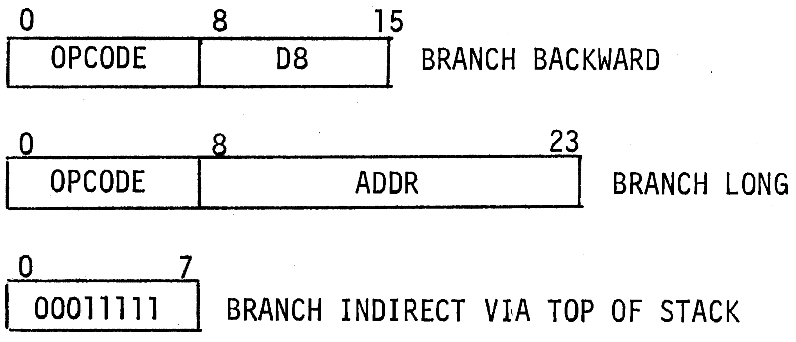


Figure 39. Branch Instruction Formats

Control instructions involve complex operations such as the following:

- Begin block entry and exit
- Procedure call and exit
- Interrupt exit
- Wait for interrupt
- Supervisor call
- Initiate microprogrammed procedure
- Pushing and popping procedure or block environments

For details on such instructions refer to the Microdata 32/S Computer Reference Manual (11).

The 32/S provides two groups of string instructions, move and compare. The pairs of string operands for these instructions are each defined by a two-word string descriptor illustrated in Figure 40. String move instructions move a source string into the string locations designated by the destination string descriptor. The move is complete whenever either string is decremented to zero. String compare instructions scan two strings from left to right the end of one or both strings, or until a difference (bytes do not match) is found. Comparisons are made on 8-bit bytes as positive integers. If the strings are equal byte by byte up to the end of one string, the shorter string is considered less than the other string. Strings are equal only if they have identical lengths and each character equals its corresponding character in the other string.

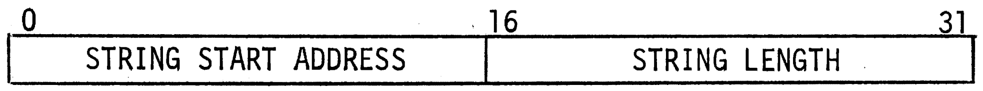


Figure 40. String Descriptor

CHAPTER V

SUMMARY

Since Digital Equipment Corporation first released the PDP 5 (predecessor of the PDP 8) in 1963, minicomputers have gained widespread popularity. They have become widely available at relatively low costs and they cover a wide range of applications. With the help of medium- and large-scale integration, sophisticated processing capabilities previously found only in larger computer systems have become common in minicomputer design. In this paper, features of ten currently marketed minicomputer systems have been discussed. Table XV summarizes the distinctive characteristics of each of the ten machines.

Trends in the minicomputer industry include continued size and cost reductions with improvements in performance. The emphasis, however, is shifting towards improving current applications and finding new ones. Cost reduction and hardware improvements is predicted to follow an evolutionary rather than a revolutionary trend (7).

On a one-for-one basis, large scale computer systems face no immediate threat of being replaced by minicomputer systesm. If the threat exists, however, it is in the form of distributed systems where the processing and information storage are resident within various operating components of an organization. For each component there is a complete minicomputer system. The application programs at each site are accessible to the minicomputers of the other sites. Thus each

TABLE XV

SUMMARY OF MINICOMPUTER CHARACTERISTICS

Minicomputer	Distinctive Characteristics
Digital Equipment Corporation PDP 8/e	fixed purpose registers 12-bit word length OMNIBUS multifunction register operate instructions
Cincinnati Milacron CIP/2200	fixed purpose registers microprogrammed decimal arithmetic control stack facility variable length binary arithmetic
Computer Automation ALPHA LSI-2	fixed purpose registers general stack processing automatic memory scan memory interleaving memory banking MAXIBUS structure
Texas Instruments 980B	fixed purpose registers standard hardware multiply/ divide standard programmable memory protect standard power fail/restart MOS semiconductor memory
Digital Computer Controls D-116	multi-accumulator multi-function arithmetic/ logic instructions overlapped instruction processing
Data General ECLIPSE S/200	multi-accumulator memory error checking and correction general stack processing memory interleaving memory cache system microprogramming

TABLE XV (Continued)

Minicomputer	Distinctive Characteristics
Digital Equipment Corporation PDP 11/40	general purpose registers UNIBUS structure generalized instruction set automatic priority interrupt system
Raytheon Data System RDS-500	general purpose registers dual bus structure
Interdata Model 8/32	general purpose registers 32-bit word length multiple sets of registers instruction lookahead memory interleaving
Microdata 32/S	push-down stack architecture MONOBUS structure high level language implementation (MPL)

site is generally not dependent on the operations of the minicomputers of the other sites. In a system centralized around a fast large-scale computer, remote job-entry terminals are used to allow computer processing at the different sites. Management of such systems requires teams of highly skilled experts providing the necessary coordination and control. Operations at each site are dependent on the operations of the main computer.

At the other end of the spectrum, do the microcomputers, equipped with microprocessors mounted on silicon chips, pose a threat to the minicomputer industry? Hobbs and McLaughlin (7) say they do not.

Microcomputers have merely relieved minicomputers of smaller scale, lower cost applications.

Each class of computers (microcomputers, minicomputers, and maxicomputers) in general pose no real threat to the other classes. It is much more feasible to think that in the development of networks of computers, computers of each class can serve separate specific purposes interacting with computers of the other two classes.

Two conclusions are drawn from this study: 1) Minicomputers can be classified according to the organization of the processor registers. Such a classification yields three general classes -- fixed purpose register machines, multi-accumulator machines, and general purpose register machines. 2) Minicomputers today are versatile and powerful because of the widespread implementation of features such as microprogramming, multiple general purpose registers, universal bus architecture, stack processing, and semiconductor memories. The following are some suggestions for future work in the area of minicomputers:

- A study of the organization, operation, and concepts used in a distributed system or any type of a network of minicomputers.
- A simulation of a minicomputer with a push-down stack architecture such as the Microdata 32/S.
- A similar study of the concepts used in the design of microcomputers (computer with processors mounted in silicon chips).
- A study of minicomputer interfacing techniques and a survey of minicomputer peripheral devices.

SELECTED BIBLIOGRAPHY

(1) 21MX Computer Series Reference Manual. Cupertino, Calif: Hewlett-Packard Company, 1974.

(2) Butler, J. L. "Comparative Criteria for Mini Computers." Instrumentation Technology, XVII (October, 1970), 67-82.

(3) CIP/2200 Computer Reference Manual. Lebanon, Ohio: Cincinnati Milicron, 1973.

(4) D-116 16-Bit LSI/MSI Computer Handbook. Fairfield, N. J.: Digital Computer Controls, 1972.

(5) Gruenberger, Fred and David Babcock. Computing With Mini Computers. Los Angeles: Melville Publishing Company, 1973.

(6) Hill, Frederick J. and Gerald R. Peterson. Digital Systems: Hardware Organization and Design. New York: John Wiley & Sons., 1973.

(7) Hobbs, L. C. and Richard A. McLaughlin. "Minicomputer Survey." Datamation, X (1974), 50-61.

(8) Iverson, Kenneth E. A Programming Language. New York: John Wiley, 1962.

(9) Kenney, Donald P. Minicomputers. New York: Amacon, 1973.

(10) Korn, Granino A. Minicomputers for Engineers and Scientists. New York: McGraw Hill Book Company, 1973.

(11) Microdata 32/S Computer Reference Manual. Irvine, Calif.: Microdata Corporation, 1975.

(12) Microprogramming 21MX Computers Operating and Reference Manual. Cupertino, Calif.: Hewlett-Packard Company, 1974.

(13) Microprogramming Handbook. Santa Ana, Calif.: Microdata Corporation, 1971.

(14) "1975 Minicomputer Market Survey." Hudson, Ma.: Modern Data Service, Inc., 1975.

(15) "The Mini-Computer's Quiet Revolution." EDP Analyzer, X (Dec., 1972), 1-13.

(16) Model 8/32 Processor User's Manual. Oceanport, N. J.: Interdata, Inc., 1975.

(17) Model 960 B Computer Systems Characteristics. Houston: Texas Instruments, Inc., 1974.

(18) Model 980 B Computer System Characteristics. Houston: Texas Instruments, Inc., 1974.

(19) Naked Mini LSI Series Computer Handbook. Irvine, Calif: Computer Automation, 1973.

(20) PDP 8E, PDP 8/M, And PDP 8/F Small Computer Handbook. Maynard, Massachusetts: Digital Equipment Corporation, 1973.

(21) Processor Handbook PDP 11/40. Maynard, Mass.: Digital Equipment Corporation, 1972.

(22) Programmer's Reference Manual ECLIPSE Line Computers. Southboro, Mass: Data General Corporation, 1974.

(23) Raytheon Data Systems RDS-500 Central Processor User's Manual. Norwood, Mass.: Raytheon Data Systems, 1975.

(24) SPC-16 Computer Family. Anaheim: General Automation, Inc., 1975.

(25) Sue Lockheed Electronics Computer Handbook. Los Angeles: Lockheed Electronics Company, Inc., 1973.

(26) Stein, Philip G. and Don R. Boyle. "How to Cope with Nickel and Diming in your Minicomputer." Mini Computer Trends and Applications. New York: Institute of Electrical and Electronics Engineers, Inc., 1973, 5-7.

(27) Systems Design Handbook. Ft. Lauderdale: Modular Computer Systems, Inc., 1975.

(28) Theis, D. J. and L. C. Hobbs. "Mini-Computers for Real-Time Applications." Datamation, XV (March, 1969), 39-61.

(29) Thompson, Glenn Ray. "A Microprogrammed Simulation System for General Purpose Register and Fixed Purpose Register Mini-computers." (Unpublished Master of Science thesis, Oklahoma State University, Stillwater, Oklahoma, 1976.)

(30) Vosatka, G. J. "The Minicomputer--Evolution or Revolution." Minicomputer Trends and Applications. New York: Institute of Electrical and Electronics Engineers, 1973, 1-4.

APPENDIX A

INSTRUCTION EXECUTION TIMES (IN MICROSECONDS)

MNEMONIC	MEANING
ADD	add the contents of a specified memory location to a register
AND	logically "and" the contents of a specified memory location to a register
CLR	clear a register
CMP	complement the contents of a register
DIV	divide the contents of a register by the contents of a specified memory location
INC	increment the contents of a register
ISZ	increment (and skip if zero) the contents of a specified memory location.
JMC	conditional transfer of control
JMP	unconditional transfer of control
JMS	transfer of control to a subroutine, store return address in a specified memory location
LDR	replace the contents of a register by the contents of a specified memory location
MUL	multiply the contents of a register by the contents of a specified memory location
SKP	conditional skip of next instruction
SLL	shift left logical one bit
STR	replace the contents of a specified memory location by the contents of a register
TRR	replace the contents of a register by the contents of another register

MINICOMPUTER	ADD	AND	CLR	CMP	DIV	INC	ISZ	JMC
PDP 8/e	2.60	2.60	1.20	1.20	7.40	1.20	2.60	---
CIP/2200	11.70	12.10	6.60	6.60	232.50	7.00	12.50	12.80
ALPHA LSI-2	8.55	8.55	5.40	5.40	128.40	5.40	12.10	7.00
TI 980B	1.75	1.75	1.00	1.00	7.75	1.00	2.75	---
D-116	---	---	1.35	1.35	---	1.35	4.50	---
ECLIPSE S/200	---	---	0.60	0.60	8.20	0.60	1.50	---
PDP 11/40	2.38	---	0.99	0.99	13.08	0.99	2.53	1.76
RDS-500	1.00	1.00	0.50	0.50	6.00	0.50	---	---
INTERDATA 8/32	1.25	1.25	1.25	1.25	5.80	1.25	---	1.95

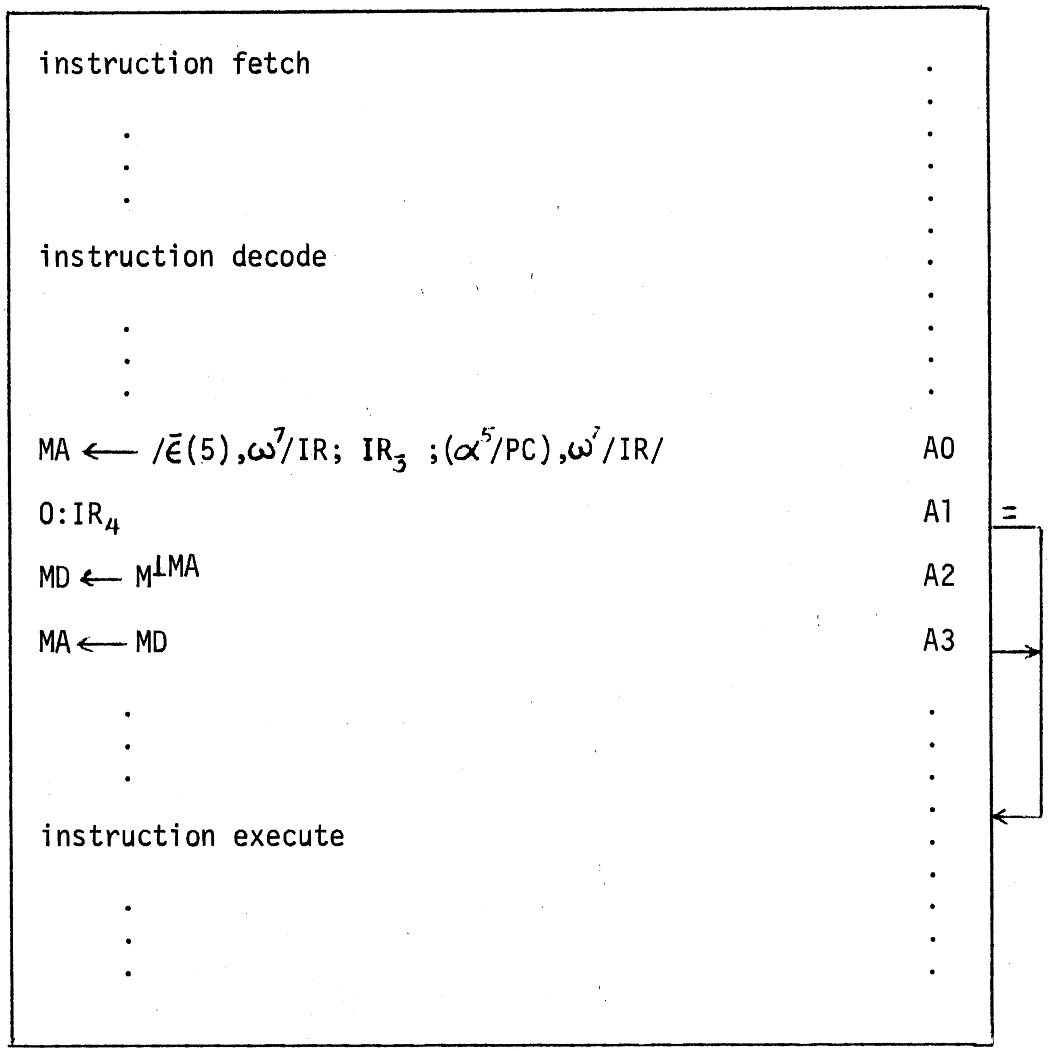
MINICOMPUTER	JMP	JMS	LDR	MUL	SKP	SLL	STR	TRR
PDP 8/e	1.20	2.60	---	7.40	1.20	1.20	2.60	1.20
CIP/2200	10.10	13.00	12.10	328.50	8.40	9.60	12.00	7.30
ALPHA LSI-2	6.80	10.10	7.00	136.40	---	5.40	6.90	5.40
TI 980B	1.25	1.50	1.75	6.25	1.00	0.75	2.00	1.00
D-116	1.35	1.35	2.55	---	2.55	---	2.55	1.35
ECLIPSE S/200	0.65	0.65	1.00	7.20	1.00	1.00	1.00	0.60
PDP 11/40	1.80	2.94	2.24	9.66	---	1.25	2.42	0.90
RDS-500	0.50	0.50	1.00	5.00	0.50	0.50	1.00	0.50
INTERDATA 8/32	1.95	2.19	1.25	4.50	---	0.70	2.00	0.80

APPENDIX B

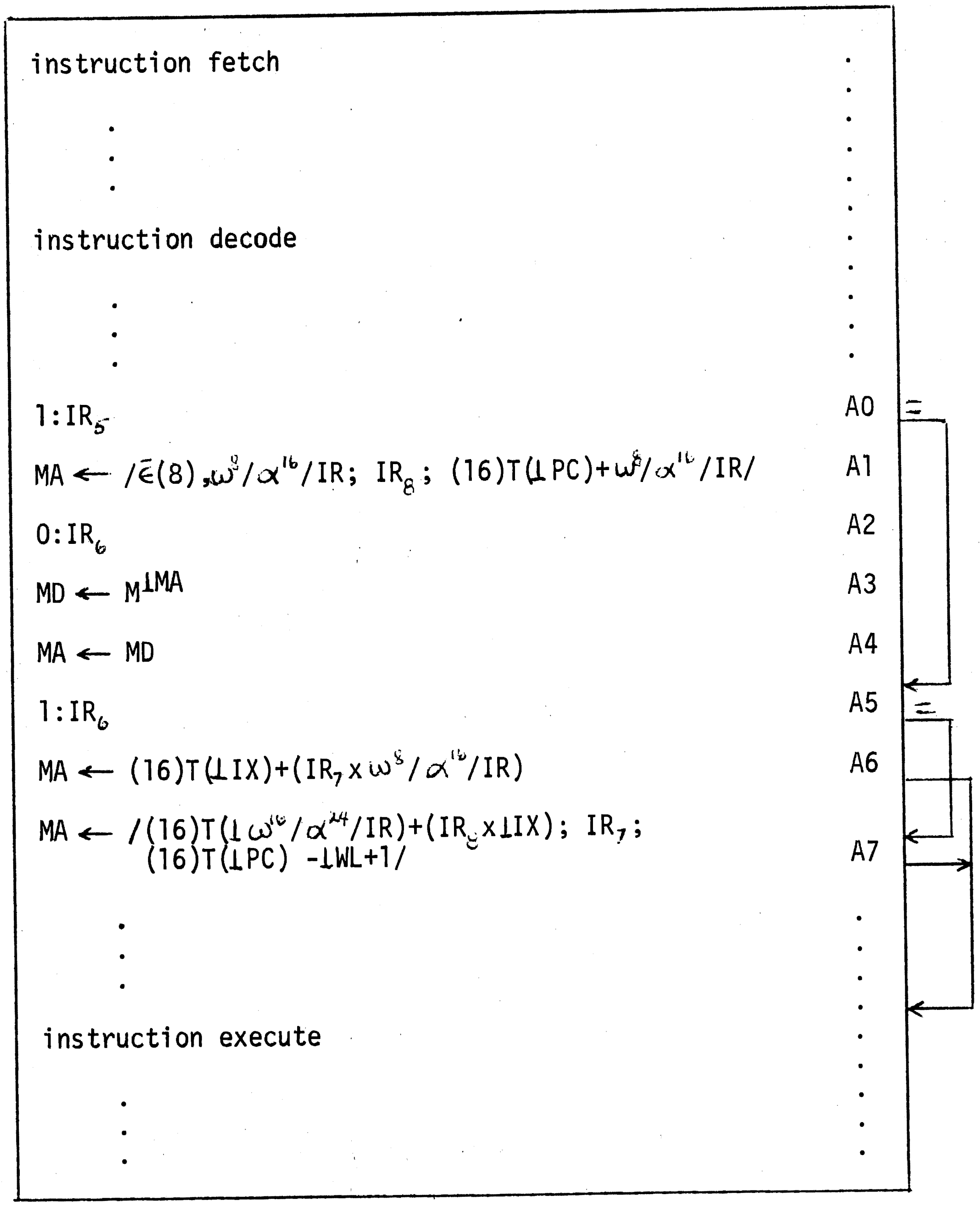
APL DESCRIPTION OF EFFECTIVE ADDRESS CALCULATING SCHEMES

SYMBOL	MEANING
$\bar{\epsilon}(n)$	A vector of n bits, all zeros
α^j	Prefix vector, used for isolating the leftmost j bits of a vector
ω^j	Suffix vector, used for isolating the rightmost j bits of a vector
DEC	Decrement logic subroutine
GR	General purpose registers
INC	Increment logic subroutine
IR	Instruction register
IX	Index register
M	Memory
MA	Memory address register
MD	Memory data register
PC	Program counter
PR	Page register
SR	Status register
TOS	Top of the stack pointer
TOS1	Next to the top of the stack pointer
WL	Instruction word length register

PDP 8/e


instruction fetch
instruction decode
$MA \longleftarrow /\bar{\epsilon}(5),\omega^7/IR;\ IR_3\ ;(\alpha^5/PC),\omega^7/IR/$
A0
$0:IR_4$
A1
=
$MD \leftarrow M^{\perp MA}$
A2
$MA \longleftarrow MD$
A3
instruction execute

CIP/2200



instruction fetch

⋮

instruction decode

⋮

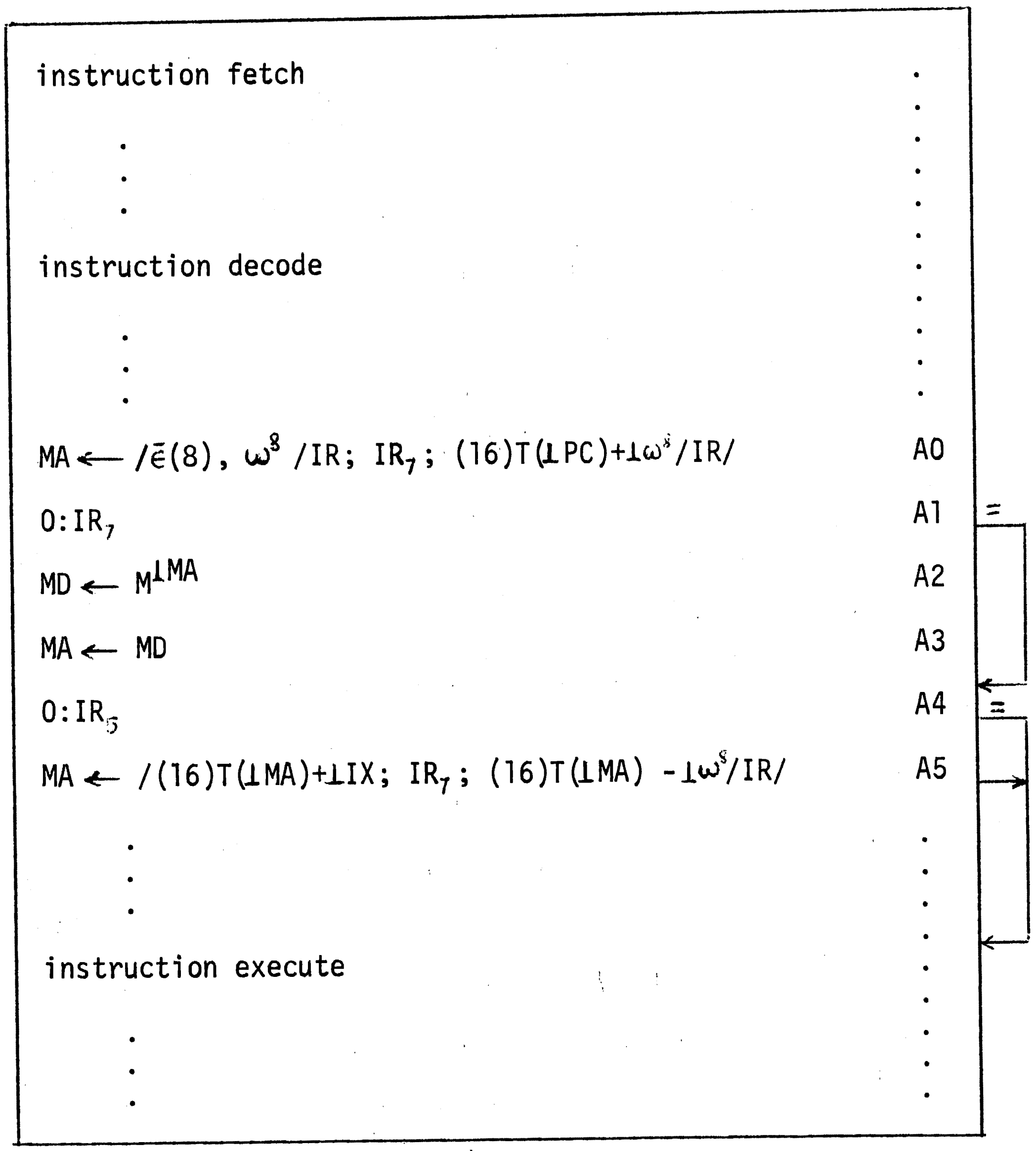
$1:IR_5$	A0
$MA \leftarrow /\bar{\epsilon}(8), \omega^8/\alpha^{16}/IR;\ IR_8;\ (16)T(\perp PC)+\omega^8/\alpha^{16}/IR/$	A1
$0:IR_6$	A2
$MD \leftarrow M^{\perp MA}$	A3
$MA \leftarrow MD$	A4
$1:IR_6$	A5
$MA \leftarrow (16)T(\perp IX)+(IR_7 \times \omega^8/\alpha^{16}/IR)$	A6
$MA \leftarrow /(16)T(\perp \omega^{16}/\alpha^{24}/IR)+(IR_8 \times \perp IX);\ IR_7;\ (16)T(\perp PC) - \perp WL+1/$	A7

⋮

instruction execute

⋮

ALPHA LSI-2



instruction fetch

⋮

instruction decode

⋮

$MA \leftarrow /\bar{\epsilon}(8), \omega^8/IR;\ IR_7;\ (16)T(\perp PC)+\perp\omega^8/IR/$ A0

$0:IR_7$ A1

$MD \leftarrow M^{\perp MA}$ A2

$MA \leftarrow MD$ A3

$0:IR_5$ A4

$MA \leftarrow /(16)T(\perp MA)+\perp IX;\ IR_7;\ (16)T(\perp MA) - \perp\omega^8/IR/$ A5

⋮

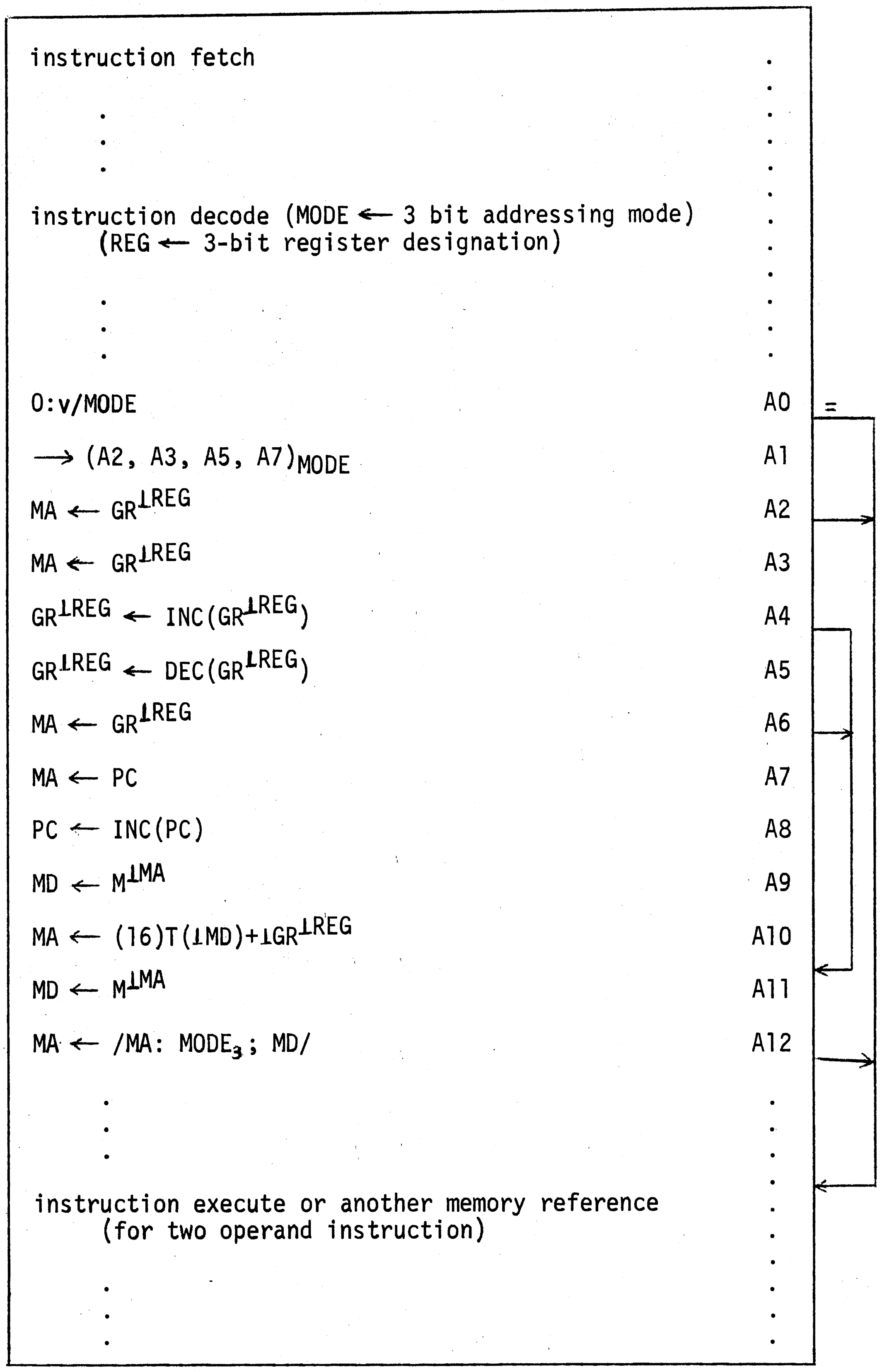
instruction execute

⋮

ECLIPSE S/200

```
instruction execute                                                  .
                                                                     .
      .                                                              .
      .                                                              .
      .                                                              .
                                                                     .
instruction decode                                                   .
                                                                     .
      .                                                              .
      .                                                              .
      .                                                              .

$1:IR_6$                                                             A0   =
$MA \leftarrow /\bar{\epsilon}(8),(\omega^8/IR);\ IR_7;\ (16)T(\perp PC)+\perp\omega^8/IR/$   A1
$MA \leftarrow (16)T(\perp\omega^8/IR)+\perp AC^{(2,3)}IR_7$        A2
$0:IR_5$                                                             A3   =
$MD \leftarrow M^{\perp MA}$                                         A4
$MA \leftarrow MD$                                                   A5
≠ $0:MD_0$                                                           A6
$MD \leftarrow M^{\perp MA}$                                         A7
$MD \leftarrow /MD;\ 15<(\perp MA)\wedge(\perp MA)<24;\ INC(MD)/$   A8
$MD \leftarrow /MD;\ 23<(\perp MA)\wedge(\perp MA)<32;\ DEC(MD)/$   A9
$M^{\perp MA} \leftarrow MD$                                         A10
      .                                                              .
      .                                                              .
instruction execute                                                  .
                                                                     .
      .                                                              .
      .                                                              .
      .                                                              .
```

PDP 11/40



instruction fetch

.
.
.

instruction decode (MODE ← 3 bit addressing mode)
(REG ← 3-bit register designation)

.
.
.

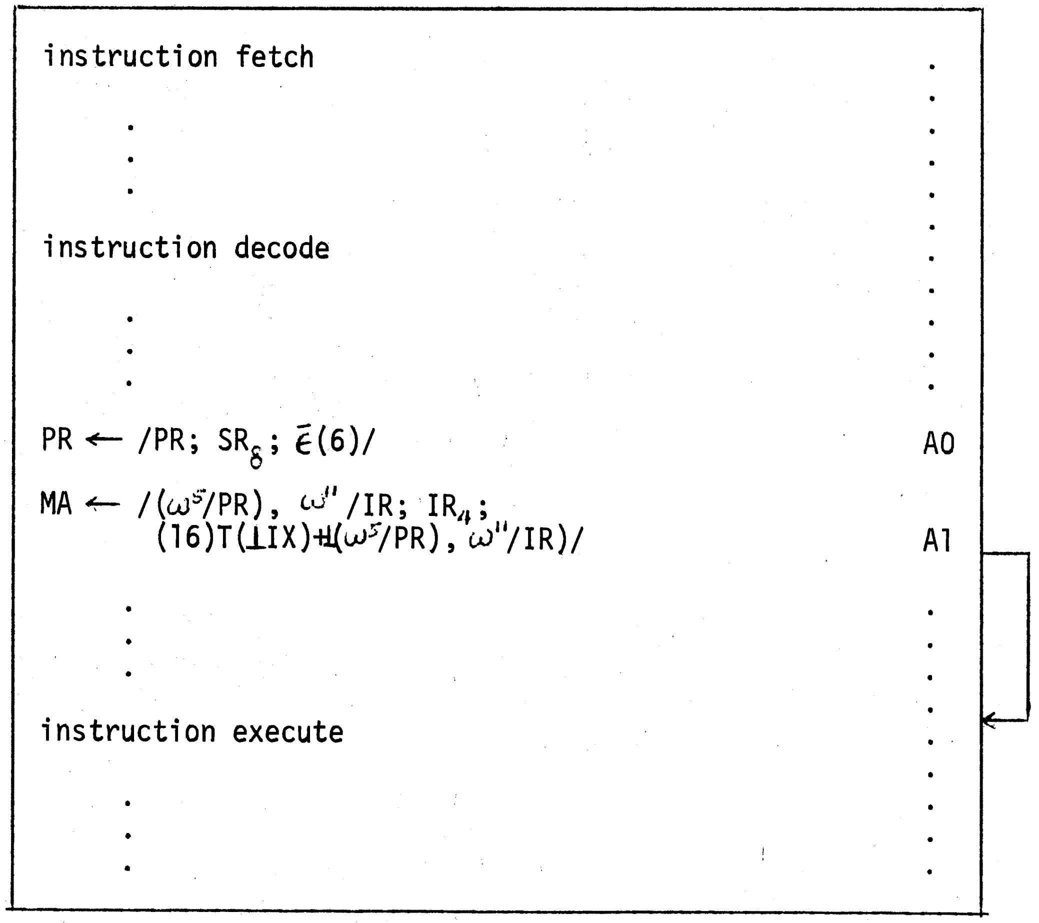
$0{:}v/MODE$	A0
$\rightarrow (A2,\ A3,\ A5,\ A7)_{MODE}$	A1
$MA \leftarrow GR^{\perp REG}$	A2
$MA \leftarrow GR^{\perp REG}$	A3
$GR^{\perp REG} \leftarrow INC(GR^{\perp REG})$	A4
$GR^{\perp REG} \leftarrow DEC(GR^{\perp REG})$	A5
$MA \leftarrow GR^{\perp REG}$	A6
$MA \leftarrow PC$	A7
$PC \leftarrow INC(PC)$	A8
$MD \leftarrow M^{\perp MA}$	A9
$MA \leftarrow (16)T(\perp MD)+\perp GR^{\perp REG}$	A10
$MD \leftarrow M^{\perp MA}$	A11
$MA \leftarrow /MA{:}\ MODE_3;\ MD/$	A12

.
.
.

instruction execute or another memory reference
(for two operand instruction)

.
.
.

RDS-500



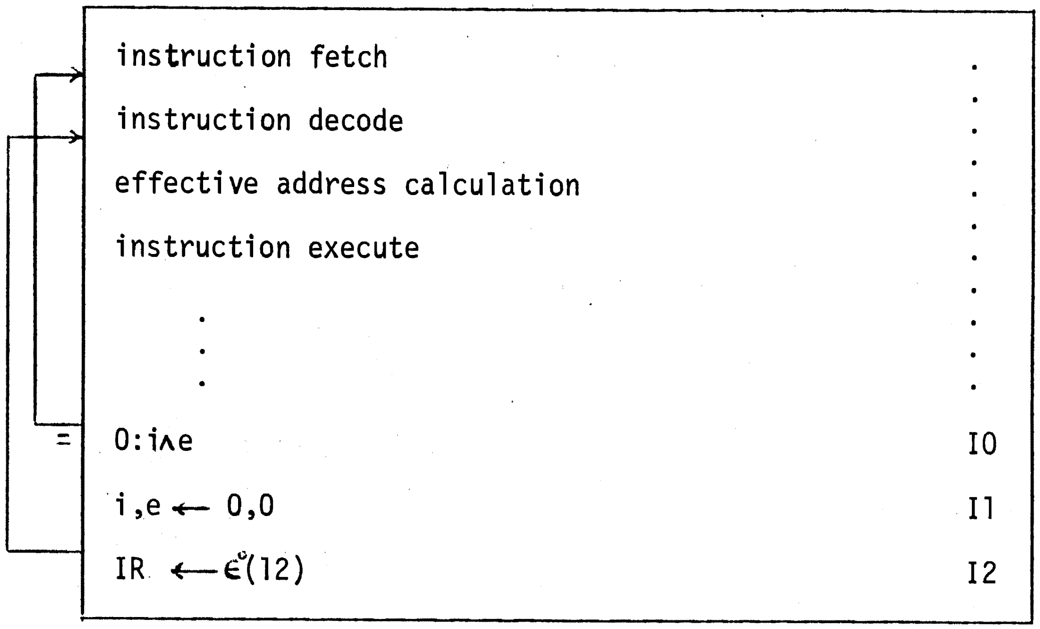
```
instruction fetch                                                 .
                                                                  .
      .                                                           .
      .                                                           .
      .                                                           .
                                                                  .
instruction decode                                                .
                                                                  .
      .                                                           .
      .                                                           .
      .                                                           .

$PR \leftarrow /PR;\ SR_8;\ \bar{\epsilon}(6)/$                          A0

$MA \leftarrow /(\omega^5/PR),\ \omega^{11}/IR;\ IR_4;$
        $(16)T(\perp IX)+\perp(\omega^5/PR),\ \omega^{11}/IR)/$          A1

      .                                                           .
      .                                                           .
      .                                                           .
                                                                  .
instruction execute                                               .
                                                                  .
      .                                                           .
      .                                                           .
      .                                                           .
```

APPENDIX C

APL DESCRIPTION OF INTERRUPT OPERATIONS

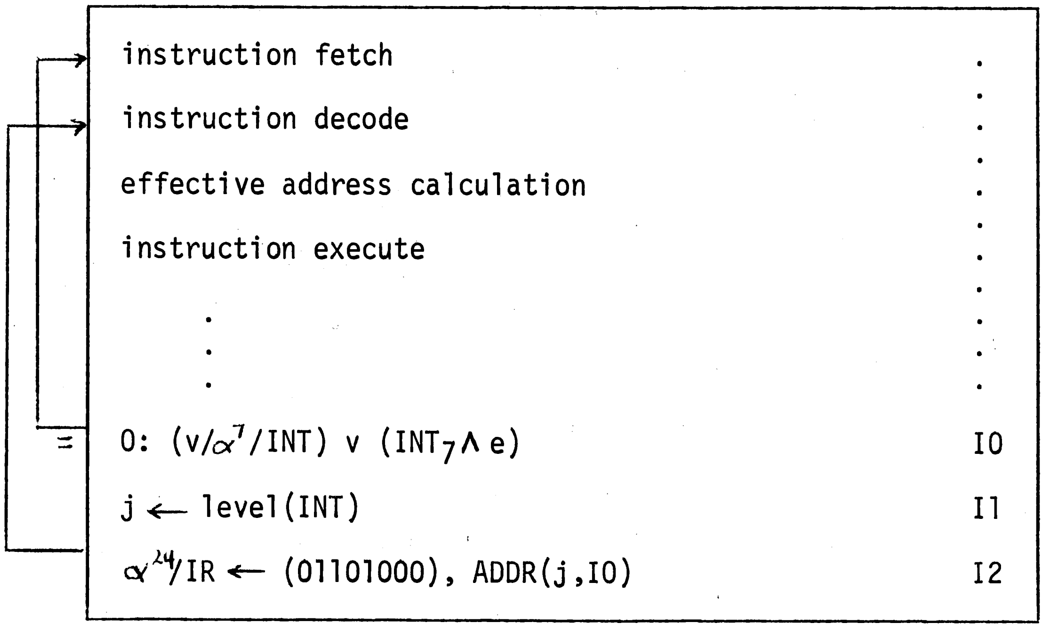
SYMBOL	MEANING
α^j	Prefix vector
$\epsilon^j(n)$	Unit vecotr, all zeros except at the jth position
$\epsilon(n)$	Full vector, all ones
ω^j	Suffix vector
e	Master enable/disable interrupt indicator
i	Single interrutp request line
level	Logic subroutine, resolves priority and returns level of interrupt
mask	Logic subroutine - resolves masking of external device interrupts
prev	Priority level of previous interrupt
ADDR	Logic subroutine - returns vector address of interrupt to be processed
E(n)	Vector of enable/disable indicators for n+1 devices
INC	Logic subroutine - increments a register
INT	Vector of interrupt request lines
IØ	IØ device address register
IR	Instruction register
M	Memory
MA	Memory address register
MD	Memory data register
MR	Mask register for interrupt processing
PC	Program counter
SP	Control stack pointer
SR	Status register or registers

PDP 8/e


instruction fetch
instruction decode
effective address calculation
instruction execute
=
0:i∧e
I0
i,e ← 0,0
I1
IR ← $\in^{\circ}(12)$
I2

CIP/2200

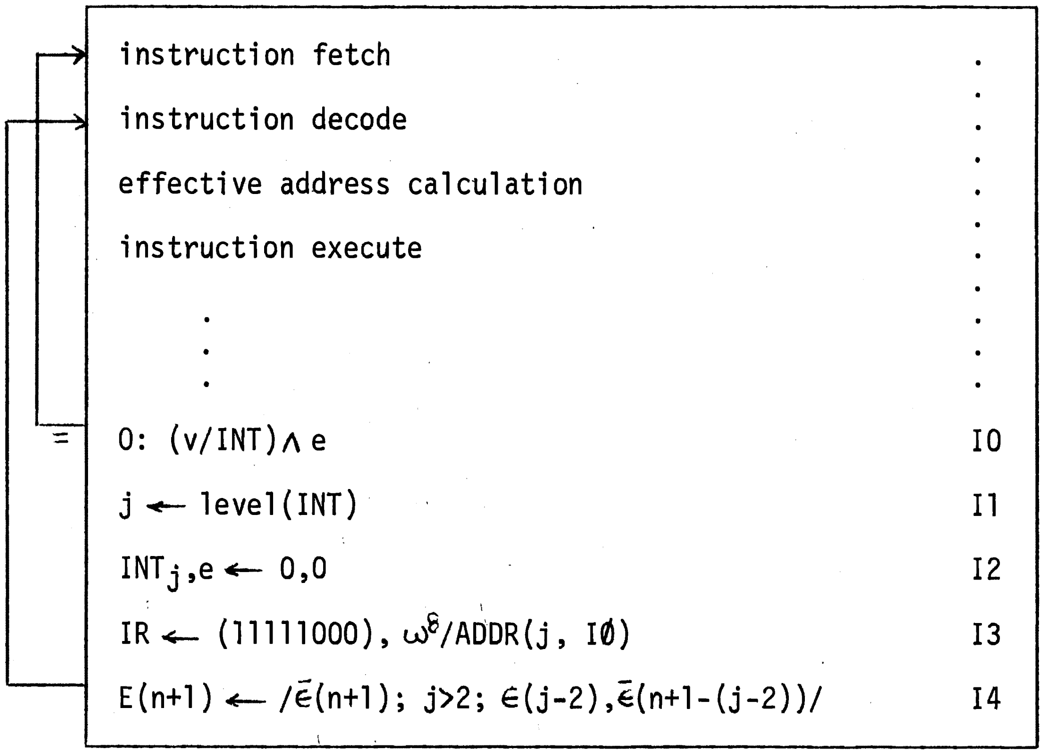
LINE	INTERRUPT TYPE
INT_0	Console
INT_1	DMA channel
INT_2	Interval timer
INT_3	Memory parity
INT_4	Control stack underflow/overflow
INT_5	Power fail
INT_6	Power restart
INT_7	Multiple I/O devices and other external signals



```
instruction fetch
instruction decode
effective address calculation
instruction execute
    .
    .
    .
= 0: $(\vee/\alpha^7/INT) \vee (INT_7 \wedge e)$          I0
j $\leftarrow$ level(INT)                                  I1
$\alpha^{24}/IR \leftarrow (01101000)$, ADDR(j,I0)          I2
```

ALPHA LSI-2

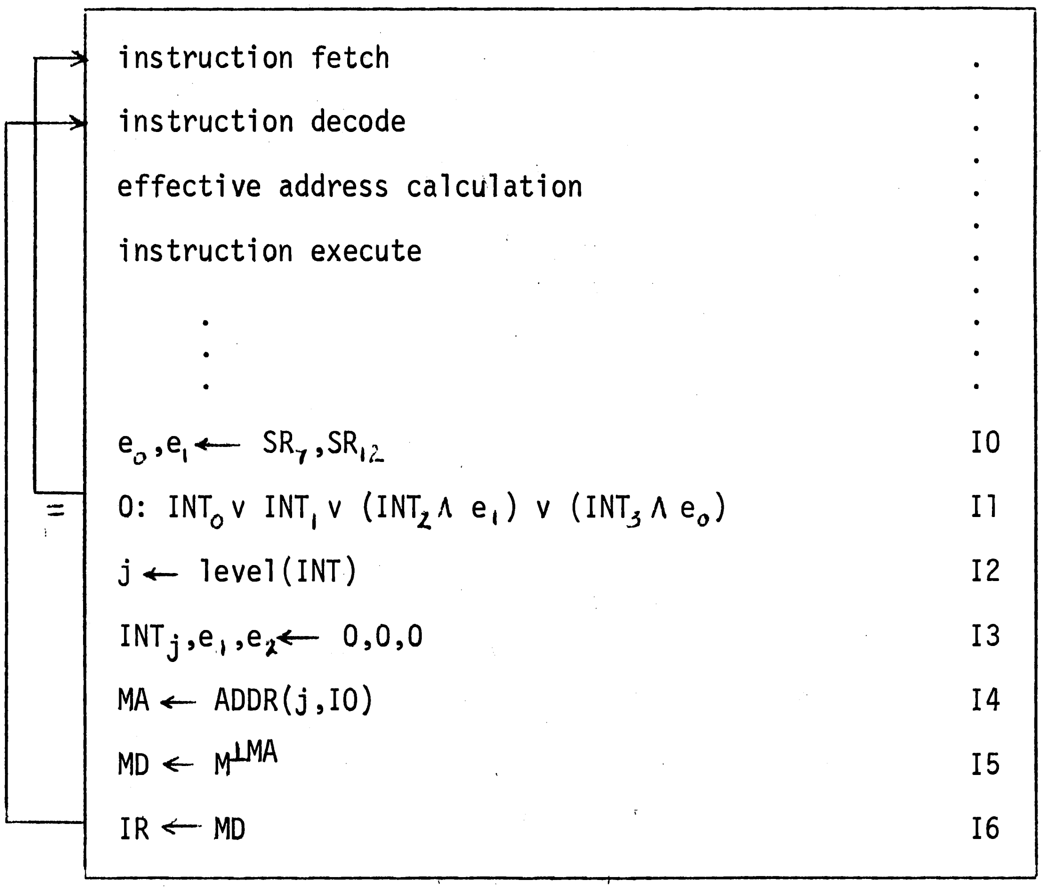
LINE	INTERRUPT TYPE
INT_0	Power fail
INT_1	Trap/console
INT_2	Single high speed I/O device
INT_3	Single high speed I/O device
INT_4	Multiple I/O devices and other external signals



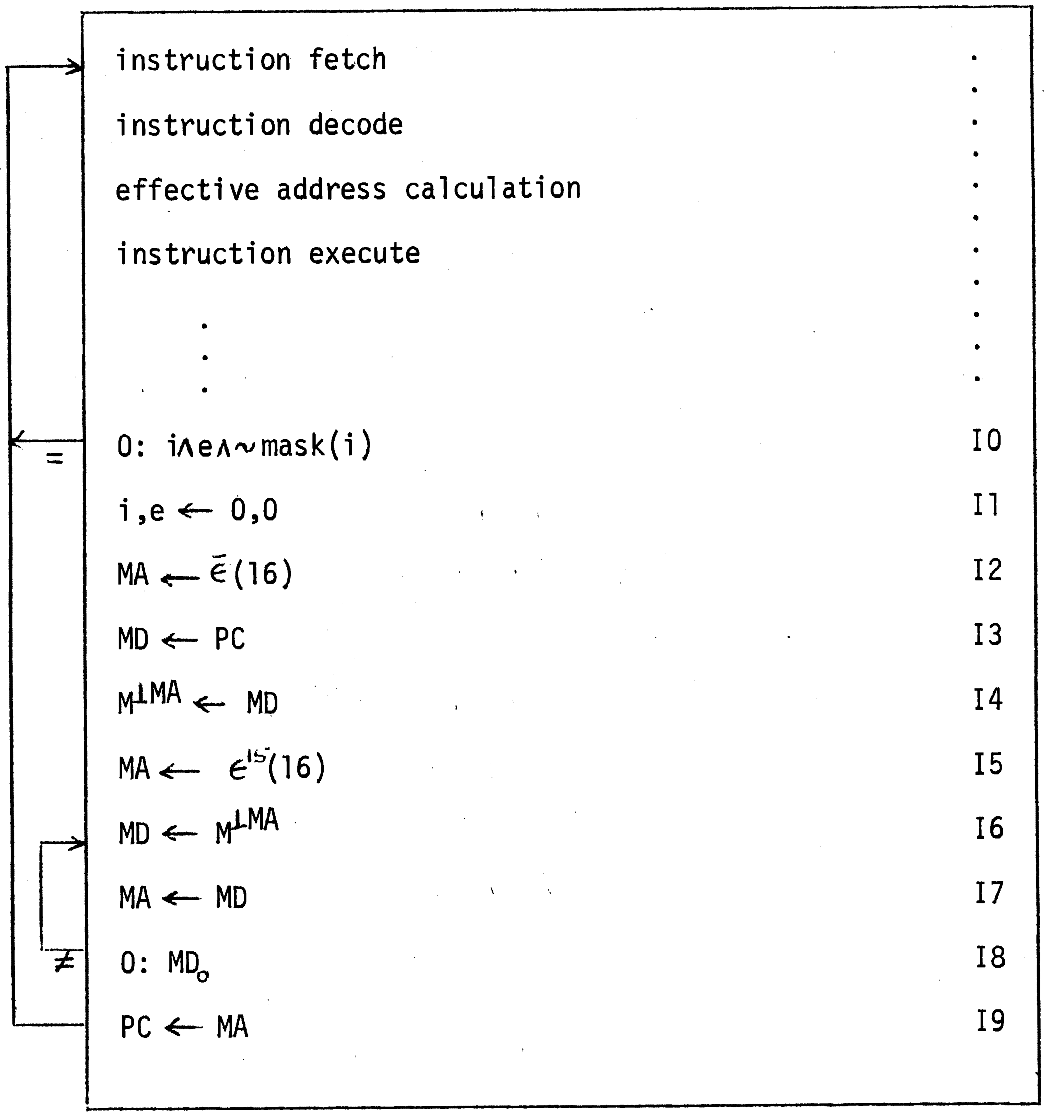
```
instruction fetch                                          .
instruction decode                                         .
effective address calculation                              .
instruction execute                                        .
      .
      .
      .
0: (v/INT)∧ e                                              I0
j ← level(INT)                                             I1
INT_j,e ← 0,0                                              I2
IR ← (11111000), ω^8/ADDR(j, IØ)                           I3
E(n+1) ← /ϵ̄(n+1); j>2; ϵ(j-2),ϵ̄(n+1-(j-2))/               I4
```

TI 980B

LINE	INTERRUPT TYPE
INT_0	Internal
INT_1	External signals (real time devices)
INT_2	DMA device
INT_3	Multiple I/O devices


instruction fetch
instruction decode
effective address calculation
instruction execute
$e_0, e_1 \leftarrow SR_7, SR_{12}$ I0
$=$
$0: INT_0 \vee INT_1 \vee (INT_2 \wedge e_1) \vee (INT_3 \wedge e_0)$ I1
$j \leftarrow \text{level}(INT)$ I2
$INT_j, e_1, e_2 \leftarrow 0,0,0$ I3
$MA \leftarrow ADDR(j, IO)$ I4
$MD \leftarrow M^{\perp MA}$ I5
$IR \leftarrow MD$ I6

ECLIPSE S/200


instruction fetch
instruction decode
effective address calculation
instruction execute
=
0: $i \wedge e \wedge \sim mask(i)$
I0
$i,e \leftarrow 0,0$
I1
$MA \leftarrow \bar{\epsilon}(16)$
I2
$MD \leftarrow PC$
I3
$M^{\perp MA} \leftarrow MD$
I4
$MA \leftarrow \epsilon^{15}(16)$
I5
$MD \leftarrow M^{\perp MA}$
I6
$MA \leftarrow MD$
I7
≠
0: MD_0
I8
$PC \leftarrow MA$
I9

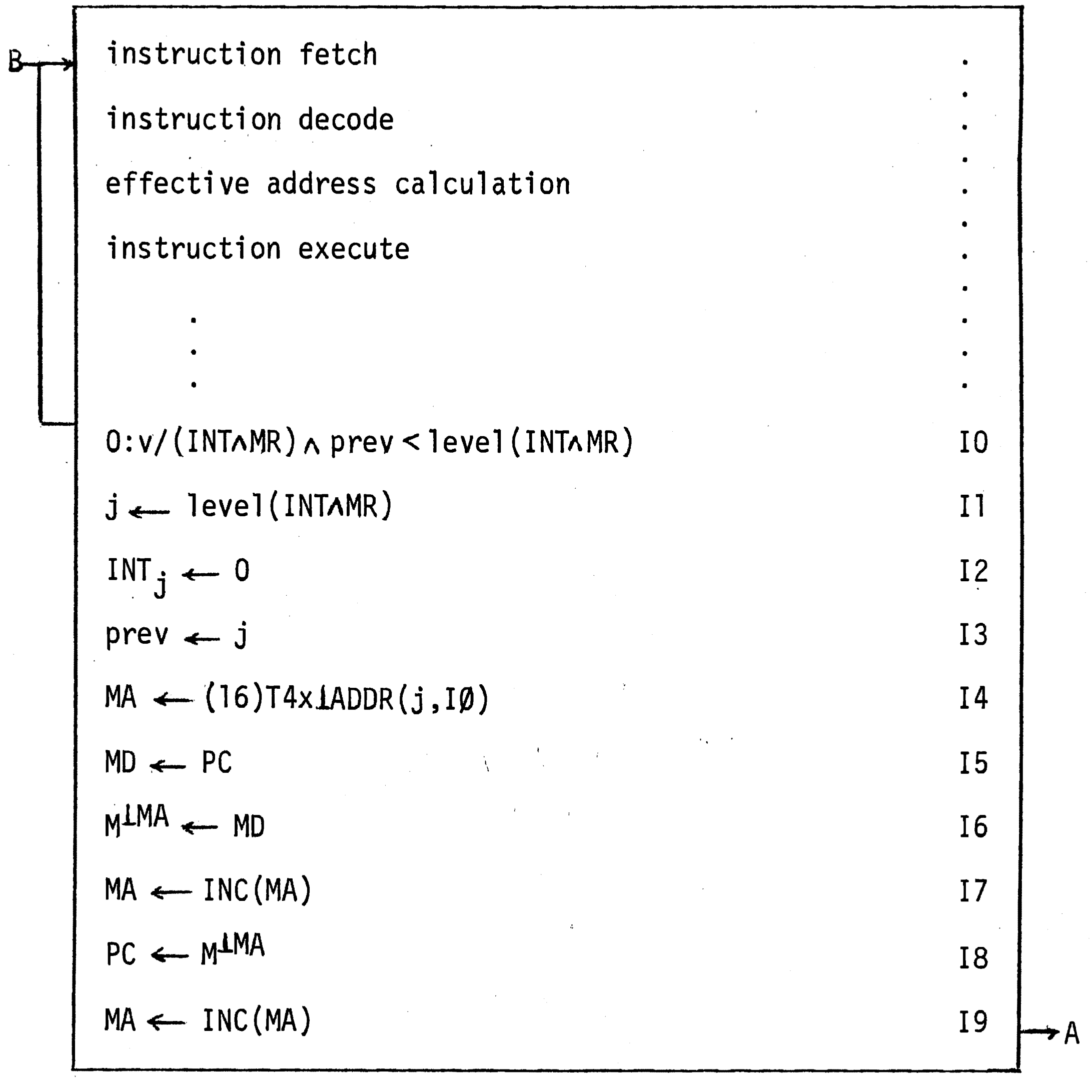
PDP 11/40

LINE	INTERRUPT TYPE
INT_0	Odd addressing
INT_1	Trap instruction
INT_2	Power fail/restart
INT_3	Reserved instruction
INT_4-INT_7	Multiple I/O device (separate levels)

```
instruction fetch                                            .
instruction decode                                           .
effective address calculation                                .
instruction execute                                          .
      .                                                      .
      .                                                      .
0: $(\vee/INT) \wedge (\perp\omega^3/\alpha^{11}/SR) < level(INT)$          I0
$j \leftarrow level(INT)$                                    I1
$TR^0, TR^1 \leftarrow SR, PC$                               I2
$MA \leftarrow ADDR(j, I0)$                                  I3
$SR \leftarrow M^{\perp MA}$                                 I4
$MA \leftarrow (16)\top(\perp ADDR(j, I0)) + 2$              I5
$PC \leftarrow M^{\perp MA}$                                 I6
$MA \leftarrow SP$                                           I7
$M^{\perp MA} \leftarrow TR^0$                               I8
$MA, SP \leftarrow INC(MA), INC(SP)$                         I9
$M^{MA} \leftarrow TR^1$                                     I10
$SP \leftarrow INC(SP)$                                      I11
```

RDS-500

LINE	INTERRUPT TYPE
INT	Power fail
INT	Memory protect
INT	Memory parity
INT -INT	IØ devices and console
INT	Paper tape reader/punch or teletype



```
B→ instruction fetch                                  .
   instruction decode                                 .
   effective address calculation                      .
   instruction execute                                .
        .
        .
        .
   0:∨/(INT∧MR)∧prev<level(INT∧MR)                   I0
   j ← level(INT∧MR)                                  I1
   INT_j ← 0                                          I2
   prev ← j                                           I3
   MA ← (16)T4x⊥ADDR(j,IØ)                            I4
   MD ← PC                                            I5
   M^⊥MA ← MD                                         I6
   MA ← INC(MA)                                       I7
   PC ← M^⊥MA                                         I8
   MA ← INC(MA)                                       I9 →A
```

RDS-500 (Continued)

A $\rightarrow$

$MD \leftarrow SR^{o}$ I10

$M^{\perp MA} \leftarrow MD$ I11

$MA \leftarrow INC(MA)$ I12

$M^{\perp MA} \leftarrow SR^{1}$ I13

$SR^{o}_{8} \leftarrow 1$ I14 $\rightarrow$ B

VITA 2

Benedicto Cacho

Candidate for the Degree of

Master of Science

Thesis: MINICOMPUTER CONCEPTS

Major Field: Computing and Information Sciences

Biographical:

Personal Data: Born in Banguio City, Philippines, January 13, 1950, the son of Mr. and Mrs. Modesto Cacho.

Education: Graduated from Atoka High School, Atoka, Oklahoma, in May, 1969; received Bachelor of Science degree from Southeastern Oklahoma State University, Durant, Oklahoma, in May, 1973, with a major in Mathematics and with a minor in Physics and Computer Science; completed requirements for the Master of Science degree at Oklahoma State University, Stillwater, Oklahoma, in July, 1976.

Professional Experience: Graduate Assistant, Oklahoma State University, Computing and Information Sciences Department, Stillwater, Oklahoma, August, 1973, to December, 1975.