A THREE-ADDRESS MINICOMPUTER FOR PROCESS CONTROL

By

HAMID NAJAFI

Bachelor of Science

Arya-Mehr University of Technology

Tehran, Iran

1975

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
May, 1977

A THREE-ADDRESS MINICOMPUTER FOR PROCESS CONTROL

Thesis Approved:

Paul A. McCollum
Thesis Adviser

Edward L. Shreve

Richard L. Cummins

Norman N. Durham
Dean of the Graduate College

PREFACE

This study is concerned with the design of an unconventional minicomputer utilizing three-address instructions. The primary objective is a comprehensive study of the problems which usually arise in process control. No specific process control is treated separately nor emphasized. However, the attention is given to the general problems which can be solved or at least eased by a digital minicomputer.

The author wishes to express his appreciation to his adviser, Professor Paul A. McCollum, for his guidance and assistance throughout this study. Appreciation is also expressed to the other committee members, Dr. Edward L. Shreve and Dr. Richard L. Cummins, for their invaluable assistance in the preparation of the final manuscript.

Special gratitude is expressed to my wife, Shahrzad, for her understanding and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

NOMENCLATURE

$Ad_1[IR]$	bit numbers 32 through 47 of IR
$Ad_2[IR]$	bit numbers 16 through 31 of IR
$ad_3[IR]$	bit numbers 0 through 15 of IR
A_i	accumulator number i, i = 1,2,3
CAR	control memory address register
DM	address of memory location of an Input/Output device
(DM)	contents of memory location DM in an Input/Output device memory
Imm	immediate data
IR	instruction register
J	conditional code register bit number $1 \leq J \leq 14$
LAB	label to an instruction
M	address of location in the main memory
(M)	contents of memory location M in the main memory
MIR	micro instruction register
OP[IR]	operation code (bit numbers 48 through 54 of IR)
PC	program counter

CHAPTER I

INTRODUCTION

1.1 Evolution of Process Control and Computers

The era of machines, initiated by the invention of the steam engine, introduced the concept of process control to human productivity and industry. James Watt, the "father" of the steam engine, invented the flyball governor in 1788 to control the speed of a steam engine. This principle of "process control" has withstood the many years of time and is still being used.

The more sophisticated controllers and related instruments, however, have been introduced largely since 1930. In 1934, Ivanoff [1] and Hazen [2] made the first efforts toward an analytical treatment of automatic control.

Many techniques have been developed and applied to the design and analysis of process control systems since the end of World War II. However, the gigantic leap forward in process control was made possible by the application of computers.

The computer era is considered by many to have started on August 7, 1944, the date Mark I was completed at Harvard University. Mark I, primarily an electromechanical computer, constructed mostly of switches and relays was followed by the ENIAC, the world's first electronic digital computer, which used vacuum tubes instead of relays. In the mid-50s a

few oil companies were experimenting with computer control of refineries and chemical plants. Phillips Petroleum Company in Bartlesville, Oklahoma was a leader in this field.

However, computers were not practically used for process control until the mid-60s, mainly because of their extremely high cost and probably because they were too difficult to use and also due to the tendency of manufacturers to stay with the rather established techniques of process control. In 1965 the PDP-8, costing $50,000, brought computers into the manufacturing plant's production line; and the minicomputer industry was born.

In 1969 the Intel Corporation built the first computer-like logic device, a very elementary computer with a very limited instruction set made up in one chip. And the so-called INTEL 8008 represented the first microcomputer brought into the world.

Today, microcomputers cost approximately $5 to $250 and can be used to control processes very economically, either as a whole or as a part of a minicomputer or a large computer system.

1.2 Variations in the Number of Addresses Per Instruction

The number of addresses in one instruction can range from zero (in a stack-oriented instruction) to any number, depending upon the way the computer is designed to operate. However, in most cases, four addresses is the practical limit to handle all the operations a computer might be requested to perform in a single instruction. These four addresses are often referred to as the following:

1. First operand's address.

2. Second operand's address.

3. Destination address.

4. Next instruction address (important to a serial memory machine).

In a zero-address computer, often called a stack-oriented computer, all the addresses are predetermined. For example, the first operand and the destination can simply be designed to be the contents of an accumulator. The second operand's address might be found in the top-most location of a stack, and finally the next instruction address may be the contents of a program counter. The KDF-9 [3] and B5500 [4] are examples of zero-address computers.

In a single-address computer, only the address of the second operand is specified by the instruction. The first operand and the destination usually refer to an accumulator. The next instruction address is given by a program counter. Most of the computers, minicomputers and microcomputers being used all over the world are single-address machines.

Two-address machines usually specify the address of the first and second operand in their instructions. The IBM 1620 [5] is an example of a two-address computer.

Usually, in a three-address computer, the address of the second operand and the destination address are the same and are given as one of the addresses in the instruction. The other two addresses are the first operand's address and the next instruction address. The Honeywell 8200 [6] is an example of a three-address computer.

A four-address computer specifies all of the four addresses in one instruction and can be considered as the fastest, most powerful, and

the most convenient to program compared to computers with a lesser number of addresses. Some of the advantages of a four-address computer can be summarized as:

1. There is no need for an unconditional jump instruction.

2. A single instruction can test a data and branch if it is zero, negative or positive.

3. By fetching one instruction per three operands in memory, the computer becomes significantly faster than the computers with less than four addresses.

The third advantage, and probably the most important one, decreases the number of times the programmer needs to read the operands from the memory.

Despite these advantages, computer manufacturers ceased building four-address computers principally because it called for memories with very long words which in turn raised the cost of the memory.

However, the advent of LSI technology is rapidly reducing the cost of highly dense logic devices, like a large memory system, and justifies, in many cases, paying for a memory which has long words to store multi-address instructions in order to speed the computer up and make the programming more convenient and efficient.

1.3 Why a Three-Address Computer?

A three-address computer can be considered to be a compromise. In most applications, four addresses for a single instruction is too much. For example, a three-address computer does not necessarily need to have unconditional jump instructions. It can use one of the three addresses as the next instruction address and branch to that address

unconditionally. However, since all the computers execute the instructions sequentially, mostly from successive cells, the next instruction address is, in most cases, the current instruction address plus one. In this regard, assigning one of the three addresses of the instruction to be always the next instruction address is not necessary. Similarly, in the case of branching if the contents of a memory location is zero, negative, or positive, the data which is to be tested can be considered to be the contents of an accumulator. Therefore, the three addresses in the instruction can serve as the target addresses.

The examples given above lead us to the idea of a "Floating-Address" computer.

In a floating-address computer each address is not solidly assigned to be a first operand address, a second operand address, a destination, or finally a next instruction address. In other words, all the addresses attain different meanings when associated with different instruction codes. For example:

ADD A,B,C

can be assigned to initiate a routine which stores (A)+(B) in location C and fetches the next instruction from the location addresses by the program counter. However:

ADI A,B,C

may store (A)+(B) into B and branch to location C unconditionally. Another instruction can be:

ADJ A,B,C

which stores the result of (A)+(Accumulator) into B and branches unconditionally to C. This aspect will lead the computer to have a very

powerful instruction set with significant convenience and flexibility of programming.

CHAPTER II

GENERAL DESCRIPTION AND HARDWARE LAYOUT

The three-address minicomputer for process control (TAMP) is a micro-programmed special-purpose machine. To satisfy the requirements of a process control computer, it has a comparatively high I/O capability and powerful macro instructions to make programming easier. In describing the computer operation and its features, several characteristics which are common to all general purpose computers are not discussed in detail and more emphasis is given to the particular aspects of a process control digital computer. The problems associated with logic design (signal gating, speed, memory cycle, instruction cycle, etc.) are not discussed and are left to a logic designer.

2.1 Bus Organization

A 55-bit bus performs the task of transferring data within the computer and between the computer and I/O devices. A 16-bit address bus transfers addresses. Figure 1 shows the pathways of both buses.

2.2 Description of Registers

2.2.1 Accumulators (A-Registers)

Three A-registers, each 16 bits wide, hold the result of arithmetic and logic operations performed by executing the instructions. Each

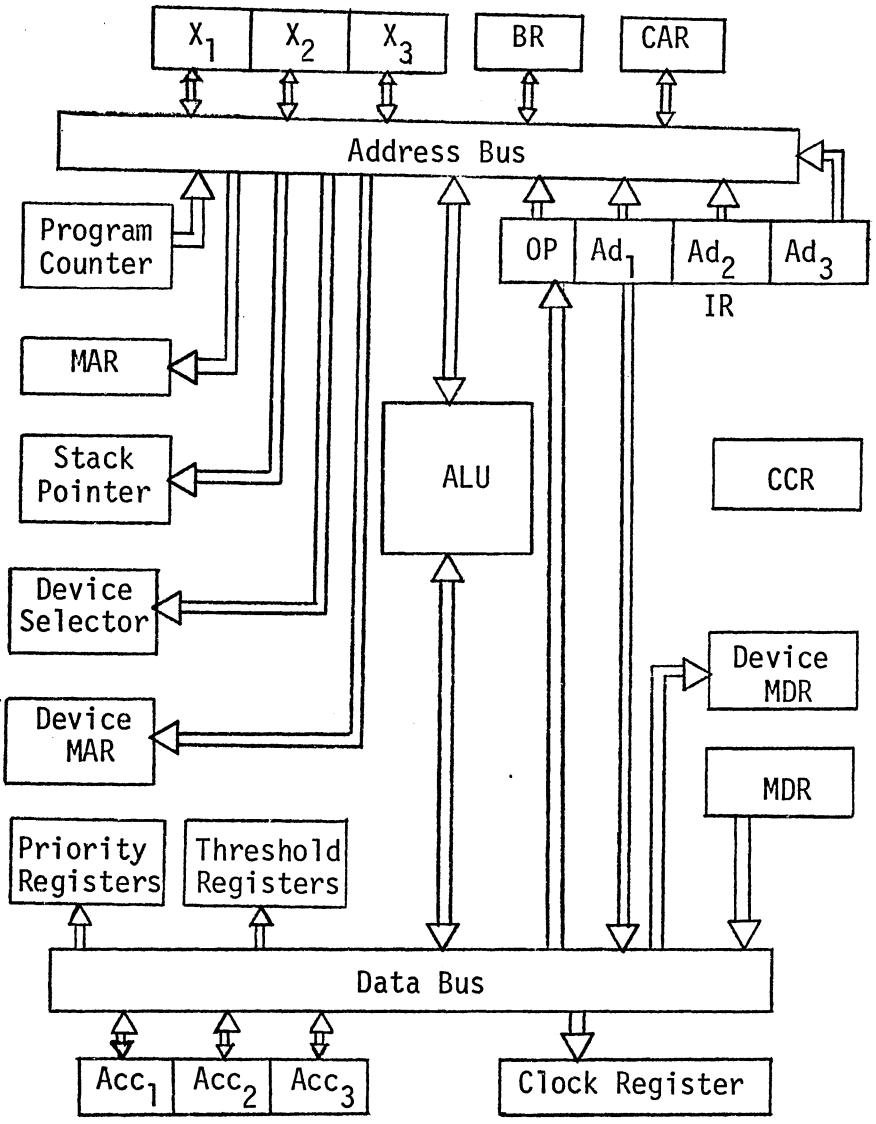


Figure 1. Register Layout

register can be addressed in assembly language by a numeric index given to the letter "A". For example, "A_2" indicates the second accumulator.

2.2.2 Instruction Register (IR)

A 58-bit instruction register holds the instructions fetched from memory.

2.2.3 Threshold Registers (TR)

Two 16-bit threshold registers hold the boundary values for a comparison instruction.

2.2.4 Clock Register (CR)

A 32-bit clock register which can be loaded by programmed instructions and transfer the control to the location specified by the Buffer Register (see below). CR is used to perform timing control instructions.

2.2.5 Buffer Register (BR)

This is a 16-bit register which holds some temporary data and the target address of a branch in certain instructions.

2.2.6 Priority Registers (PR)

Three 32-bit priority registers are used to determine the I/O devices priority pattern. They can be loaded immediately or from the contents of the memory locations. Each bit corresponds to one device; therefore, 96 different devices can be connected to the computer.

2.2.7 Program Counter (PC)

A 16-bit program counter holds the next instruction address.

2.2.8 Stack Pointer (SP)

A 16-bit stack pointer is used to allow push-down and pop-up stack operations on memory.

2.2.9 Index Registers (X-Registers)

. Three 16-bit index registers are used to establish an indexed addressing capability for the computer.

2.2.10 Condition Code Register (CCR)

CCR is a 14-bit register which has the following bit pattern:

Bit number one is set when the result of the last ALU operation is negative.

Bit number two is set when the result of the last ALU operation is zero.

Bit number three is set when an overflow is resulted by the last ALU operation.

Bit number four is set when the last ALU operation results in a carry.

Bits number five, six, and seven are used to indicate indexed addressing (refer to section 5.2).

Bits number eight, nine, and ten represent indirect addressing (refer to section 5.1).

Bit number eleven is set when the contents of the clock register are zero.

Bit number twelve is set when a Branch and Nest routing is specified (refer to section 4.2).

Bit number thirteen is the interrupt flag and is set when an interrupt request is serviced.

Bit number fourteen is set when an overflow or underflow occurs after a stack operation.

2.3 Memory Organization

The 64K memory system consists of two parts, a RAM and a ROM.

2.3.1 Random Access Memory (RAM)

The RAM consists of 32K of 58-bit words which hold temporary programs and data.

2.3.2 Read Only Memory (ROM)

The ROM consists of 32K of 58-bit words which hold the main program, microprograms and some data.

2.4 Instruction Format

Each instruction consists of 58 bits arranged in the format given below:

Bits number zero through 15 contain the third address.

Bits number 16 through 31 contain the second address.

Bits number 32 through 47 contain the first address.

Bits number 48 through 54 contain the instruction code.

Bit number 55 is set by the assembler when the third address is specified. It is cleared when the third address field is left blank.

Bit number 56 is set by the assembler when the second address is specified. It is cleared when the second address field is blank.

Bit number 57 is set by the assembler when the first address is specified. It is cleared when the first address field is blank.

CHAPTER III

THE INSTRUCTION SET

The set of instructions and the description of their operation is given in this chapter. Due to the significance and variety of input/output instructions, they are discussed in a separate section, while the rest of the instructions are given in another section.

3.1 Input/Output Instructions

TAMP, as a process control mini-computer, has a set of 22 I/O instructions described below. (For a better understanding of the descriptions refer to the nomenclature.)

Mnemonic	Operands	Description of Operation
OUT	M,SC,DM	Output (M) into DM of device #SC
INP	M,SC,DM	Input (DM) of device #SC into M
OUC	M,SC,DM	Output one's complement of (M) into DM of device #SC
IPC	M,SC,DM	Input complement of (DM) of device #SC into M
ONI	M,SC,LAB	Output (M) to the fast output device (for example, a seven-segment display latch) #SC without enabling interrupt system. Branch to LAB
INI	M,SC,LAB	Input to M from a fast input device #SC without enabling interrupt system. Branch to LAB
OUI	Imm,SC,DM	Output immediate to DM of device #SC

Mnemonic	Operands	Description of Operation
OUR	M,SC_1,SC_2	Output (M) to SC_1. If SC_1 is busy, output to SC_2 and skip the next instruction. If SC_2 is busy too, skip the next two instructions. The current memory location of each device is used to store output data
INR	M,SC_1,SC_2	Input from SC_1 to M. If SC_1 is busy, input from SC_2 and skip the next instruction. If SC_2 is busy too, skip the next two instructions. The current memory location of each device is used to input form
OUS	M,SC_1,SC_{1+k}	Output (M) to the current memory location of the device numbers SC_1 through SC_{1+k}
INS	M,SC_1,SC_{1+k}	Input to M through M+k from the current memory location of SC_1 through SC_{1+k}, respectively
OSI	Imm,SC_1,SC_{1+k}	Output immediate to the current memory locations of SC_1 through SC_{1+k}
OUB*	M,SC_1,SC_{1+k}	Output bit #1 through bit #1+k of (M) to set or clear the state of the device #SC_1 through device #SC_{1+k}, respectively
INB*	M,SC_1,SC_{1+k}	Input from the state of the device #SC_1 through SC_{1+k} to bit #1 through bit #1+k of the memory location M
OIB*	Imm,SC_1,SC_{1+k}	Output bit #1 through bit #1+k of the immediate data to SC_1 through SC_{1+k}, respectively
IWP	SC_1,SC_2,SC_3	Input from the current memory locations of SC_1, SC_2, and SC_3 into A_1 (accumulator #1), A_2, and A_3, respectively
OWP	SC_1,SC_2,SC_3	Output from A_1 (accumulator #1), A_2, and A_3 to the current memory locations of SC_1, SC_2, and SC_3, respectively
IWB	M,SC,LAB	Input from the current memory location of the device #SC to M and branch to LAB

Mnemonic	Operands	Description of Operation
OWB	M,SC,LAB	Output from (M) to the current memory location of device #SC and branch to LAB
ISR	M,SC,LAB	Input to M from the current memory location of device #SC using the service routine starting at the address given by the contents of LAB (refer to 4.2.1 and 4.2.2)
OSR	M,SC,LAB	Output from (M) to the current memory location of device #SC using the service routine starting at the address given by the contents of LAB (refer to 4.2.1 and 4.4.2)
FDB	SC_1,SC_2,M	Feedback instruction: Select device #SC_1 as an output device and device #SC_2 as an input device. Compare the data from SC_1 with the data from SC_2. If the error is greater than (M), change the data output to SC_1, check the error again and continue until the error becomes no greater than (M) This will let the programmer select any two measuring points of the process as the input and output points in a feedback loop To avoid instability or infinite loop, the programmer needs to load X_1 register with the maximum number of iterations allowed. The computer branches out of the loop when this number is reached

*In these instructions the data is transferred to or from a series of one bit devices such as relays, switches, etc.

3.2 Other Instructions

Mnemonic	Operands	Description of Operation
ADA	Imm,M_1,M_2	Imm+(M_1)→M_2
ADB	M_1,M_2,LAB	(M_1)+(M_2)→M_2, branch to LAB
ADD	M_1,M_2,M_3	(M_1)+(M_2)→M_3
SUA	Imm,M_1,M_2	Imm-(M_1)→M_2
SUB	M_1,M_2,LAB	(M_1)-(M_2)→M_2, branch to LAB
SUD	M_1,M_2,M_3	(M_1)-(M_2)→M_3
ANA	Imm,M_1,M_2	Logical AND of Imm and (M_1)→M_2
ANB	M_1,M_2,LAB	Logical AND of (M_1) and (M_2)→M_2, branch to LAB
AND	M_1,M_2,M_3	Logical AND of (M_1) and (M_2)→M_3
ORA	Imm,M_1,M_2	Logical OR of Imm and (M_1)→M_2
ORB	M_1,M_2,LAB	Logical OR of (M_1) and (M_2)→M_2, branch to LAB
ORD	M_1,M_2,M_3	Logical OR of (M_1) and (M_2)→M_3
COM	M_1,M_2,LAB	Compare M_1 and M_2. If (M_1)>(M_2) then increment M_2 and branch to LAB (M_1)<(M_2) then increment M_1 and branch to LAB+1 (M_1)=(M_2) then continue
CMT	M_1,M_2,M_3	Compare M_1 and M_2 and M_3. If (M_1)≠(M_2) skip 2 instructions (M_1)=(M_2)≠(M_3) skip 1 instruction (M_1)=(M_2)=(M_3) do not skip

COT	Imm_1,Imm_2,M	If $Imm_1 \leq$ (M)$\leq Imm_2$ skip 2 instructions
		If (M)<Imm_1 skip 1 instruction
		If (M)>Imm_2 do not skip

BR_i i=1,2.3	LAB1,LAB2, LAB3	Branch conditionally to LAB1, LAB2, or LAB3, if A_i is negative, zero, or positive, respectively

Mnemonic	Operands	Description of Operation
CHP	M_1,M_2,M_3	Change priority system by loading M_1, M_2, and M_3 into priority registers
SCH	M_1,M_2,M_3	Exchange (M_1), (M_2), and (M_3) with the contents of A_1, A_2, and A_3, respectively
RAC	1,2,3	Rotate the catenated accumulator [Acc_1, Acc_2, Acc_3] 16 bit positions. Result will be: $(A_1)\rightarrow(A_2),(A_2)\rightarrow(A_3),(A_3)\rightarrow(A_1)$. If any operand field is blank, the corresponding transfer will not be performed. (For example, RAC 1,2 will result $(A_1)\rightarrow(A_2)$ and $(A_2)\rightarrow(A_3)$)
ADS	M_1,M_2,M_3	$(M_1)+(M_2)-(M_3)\rightarrow M_3$
INC	M_1,M_2,LAB	Increment (M_1) and (M_2); branch to LAB
DEC	M_1,M_2,LAB	Decrement (M_1) and (M_2); branch to LAB
BRF	J,LAB	Branch, if Condition Code Register bit number J is set, to LAB
BNF	J,LAB	Branch to LAB if bit number J of Condition Code Register is clear
CLF	J_1,J_2,LAB	Clear bit number J_1 and J_2 of (CCR) and branch to LAB
STF	J_1,J_2,LAB	Set bit number J_1 and J_2 of CCR and branch to LAB
SCK	M,LAB1,LAB2	Store (M) into Clock Register, store LAB1 into Buffer Register and branch to LAB2
JMP	LAB1,LAB2	Jump to LAB1 and store LAB2 into Buffer Register (BR). This instruction can be used as an unconditional jump when operand field two is not specified and can be a jump to subroutine LAB1 and return to LAB2 when returning from subroutine (LAB2 can be =*+1)

Mnemonic	Operands	Description of Operation
RTN	M	Jump indirect through the contents of M. If the operand field is blank jump indirect through Buffer Register
LDA	M_1,M_2,M_3	Load accumulator: $(M_i)\rightarrow A_i$; i=1,2,3
STA	M_1,M_2,M_3	Store accumulator: $(A_i)\rightarrow M_i$; i=1,2,3
LOA	M,LAB	Load catenated accumulator: $(M)\rightarrow[A_1,A_2,A_3]$. Branch to LAB
SOA	M,LAB	Store catenated accumulator: $[A_1,A_2,A_3]\rightarrow M$, and branch to LAB
LDX	M_1,M_2,M_3	Load X registers: $(M_i)\rightarrow X_i$; i=1,2,3
STX	M_1,M_2,M_3	Store X registers: $(X_i)\rightarrow M_i$; i=1,2,3
LOX	M,LAB	Load catenated X registers: $(M)\rightarrow[X_1,X_2,X_3]$, branch to LAB
SOX	M,LAB	Store catenated X registers: $[X_1,X_2,X_3]\rightarrow M$
RTR	M_1,M_2,M_3	Rotate M_1, M_2, and M_3 one bit position to right
RTL	M_1,M_2,M_3	Rotate M_1, M_2, and M_3 one bit position to left
RBR	M_1,M_2,M_3	Rotate each word 16 bit positions to right. Op code field does not take part in rotation
RBL	M_1,M_2,M_3	Rotate each word 16 bit positions to left. Op code field does not take part in rotation
PSH	M_1,M_2,M_3	Perform three consecutive push-down stack operations over memory using the stack pointer
POP	M_1,M_2,M_3	Perform three consecutive pop-up stack operations over memory using the stack pointer
ICS	LAB	Increment stack pointer and branch to LAB

Mnemonic	Operands	Description of Operation
DCS	LAB	Decrement stack pointer and branch to LAB
HLT		Halt

Note: When the program written in assembly lauguage is converted to object codes, using an assembler program, if operand field #1 is blank the bit #57 (the most significant bit) of the memory word will be cleared. If operand field #2 and/or #3 is blank the bit #56 and/or #55 will be cleared, respectively. If all three operand fields are specified, then the three MSBs of the memory word are set. When executing each instruction in the memory, if an operand field is blank then the corresponding operation to that field will not be performed.

In the case of those instructions for which the presence of an operand field is essential, the assembler will issue an error signal when the essential operand field is blank. For example: ICS LAB increments the stack pointer and branches to LAB where ICS increments the stack pointer and goes to the next instruction. However, OUT M, ,DM will cause the assembler to issue an error signal, because the select code is missing.

CHAPTER IV

INTERRUPT AND PRIORITY SYSTEMS

4.1 The Interrupt System

To furnish sufficient, reliable and fast service to the I/O devices, three different types of interrupt routines are designed, any of which can be selected by the programmer to use [7].

4.1.1 Standard Routine

The interrupt system is disabled (no other interrupt request accepted except power fail and parity). The current program counter will be saved in location Ø. The next instruction is fetched from the location addressed by the contents of location 1 (Jump @ 1).* This is the first instruction of a master interrupt service routine and according to the device select code, which generated the interrupt, it will transfer the control to the relevant device service routine. After servicing a device, the service routine should set the interrupt flag (enable the interrupt system) and simulate an indirect jump through location Ø, to restore the pre-interrupt status of the processor and return control to the interrupted program.

---

*Jump @ 1 means "jump indirect through location 1."

In the standard routine, no secondary interrupt is accepted during the time the processor is servicing a device, because the interrupt system is disabled. However, if the programmer desires, the interrupt system can be re-enabled by setting the interrupt flag and the priority system can be used to allow higher priority devices to interrupt the current device service routine. To reduce CPU overhead and programming complexity, Branch and nest routine is used, rather than the standard routine, to handle simultaneous interrupt requests.

4.1.2 Branching Sequences

By setting the branch and nest flag (BN), the interrupt system follows a branching sequence. First, an INTS sequence is performed to determine the select code of the highest priority device interrupting the program. The address of the service routine (ISA) is obtained from the contents of memory location SC (SC = the select code of the device). If the MSB of this address (bit No. 15) is zero, a simple branch is initiated, otherwise the control will perform a branch and nest interrupt routine.

4.1.2.1 Simple Branch Sequence. The interrupt system is disabled (clear ION); the program counter is saved in location Ø and a jump to ISA is performed. The sequence is terminated by servicing the device, enabling the interrupt system, and returning to the interrupted program.

4.1.2.2 Branch and Nest Sequence. If the MSB of ISA is "1" (bit No. 15 is set), a branch and nest sequence is performed. This sequence is designed to accept new interrupts from higher priority devices, when

servicing another device. The stack pointer is used to perform push-down and pop-up operations on the memory (as a LIFO stack). The program counter is pushed down into the stack. Provided that no stack overflow results, a jump to ISA transfers the control to the device service routine. If a higher priority device requests for service (could be the result of either a new service request from a higher priority device or assigning a higher priority to a device waiting for service by changing the contents of the priority registers), the program counter is pushed down into the stack and the control is transferred to ISA of the latter device. A service routine is to be terminated by a pop-up stack operation and an indirect branch through the contents of the stack pointer returns the control to the last interrupted device service routine or the interrupted main program.

Here, there is no transfer to the main program when a higher priority device interrupts the service routine of another device. This results in a decrease on CPU overhead. Figure 2 shows the sequence of operation for three different interrupt routines.

4.1.3 Priority System

To each I/O device connected to the address bus and/or data bus, a priority bit is assigned. The priority bits can be cleared or set by the instruction CHP M_1, M_2, M_3. If one of the addresses in the operand field is missing, the corresponding priority register remains unchanged.

The priority linkage is shown in Figure 3.

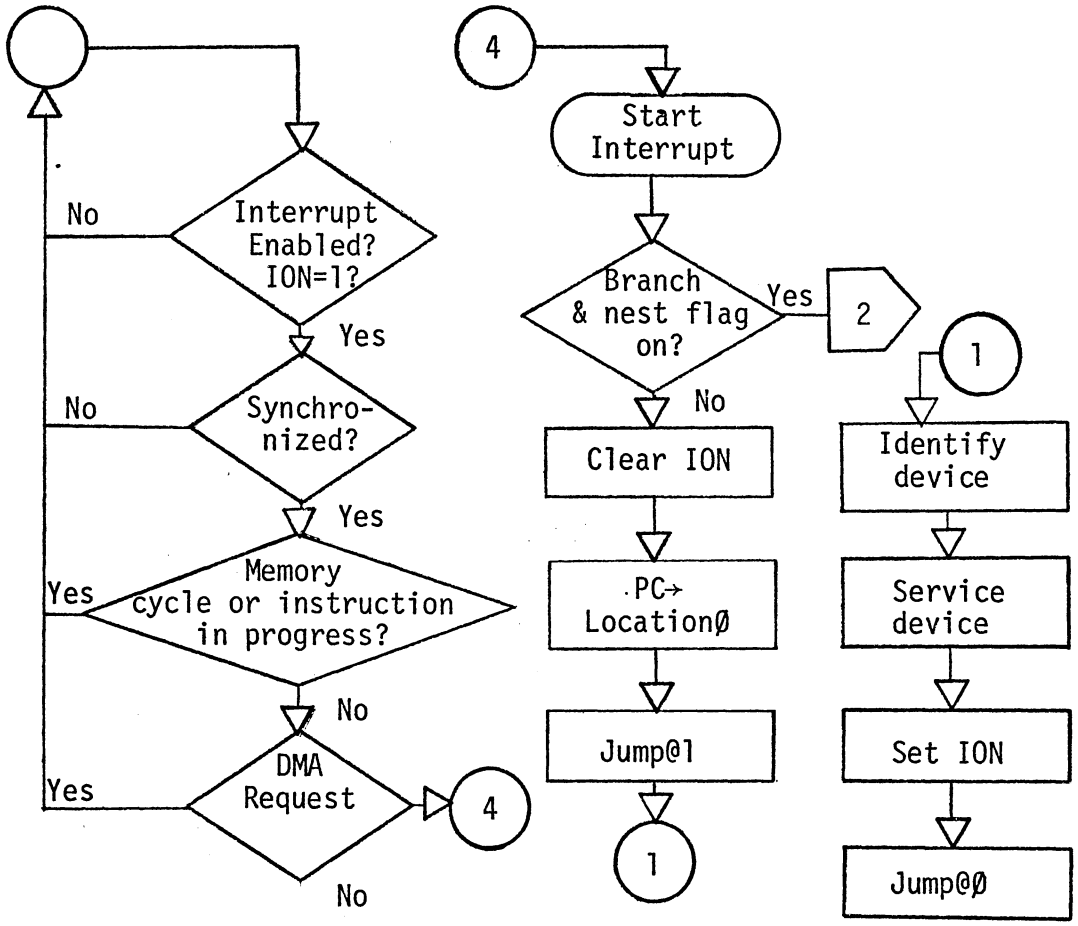


Figure 2. Interrupt Handling System

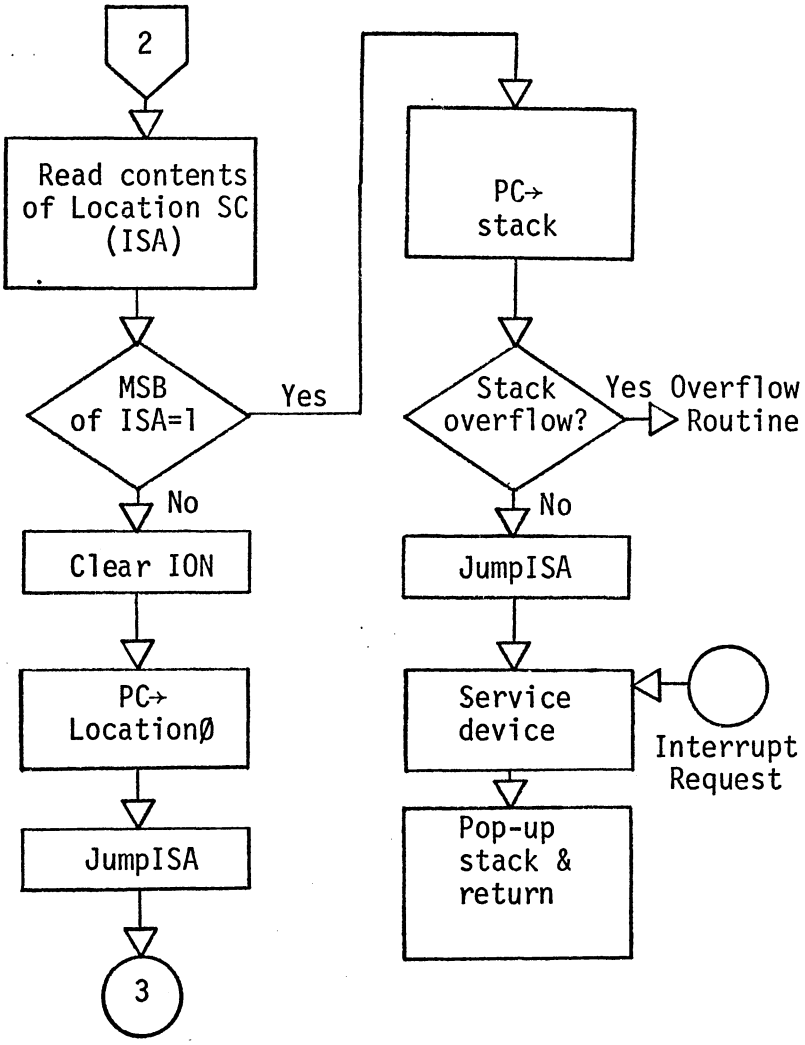


Figure 2. (Continued)

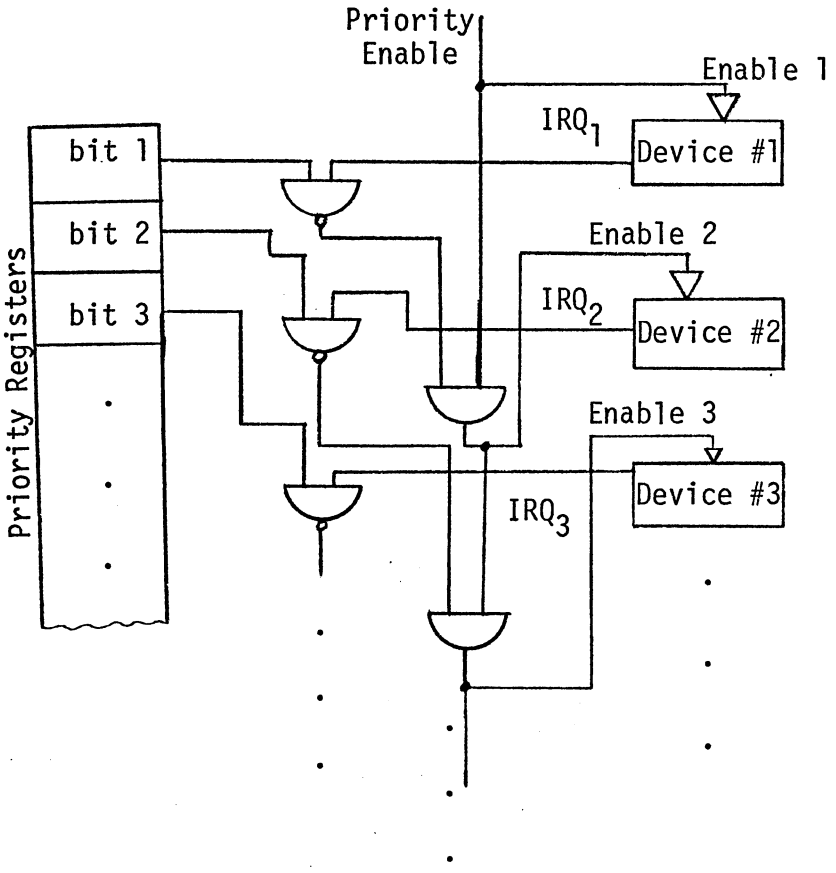


Figure 3. Priority System

CHAPTER V

ADDRESSING FEATURES

5.1 Indirect Addressing

All the instructions which have at least one address operand can be an indirect as well as a direct address instruction. In other words, there is no difference between two instructions regarding their direct addressing or indirect addressing.

Before any address is executed to fetch data, the corresponding indirect flag in condition Code Register (CCR bit numbers 8 through 10) is tested. If it is set, then the address is considered to be indirect, and if it is clear the address is considered to be a direct address (Figure 4).

5.2 Indexing

The aspect of indexing is accomplished in the same manner as the indirect addressing.

If the corresponding CCR flag (bit numbers 5 through 7) is set, the address is considered to be an indexed address. Otherwise, the address is not indexed. Figure 4 shows how each address is treated when the corresponding instruction is executed.

Note: An address can be indirect and indexed simultaneously. The effective address in such a case will be (M)+(X), where M is the raw address and X is the index register.

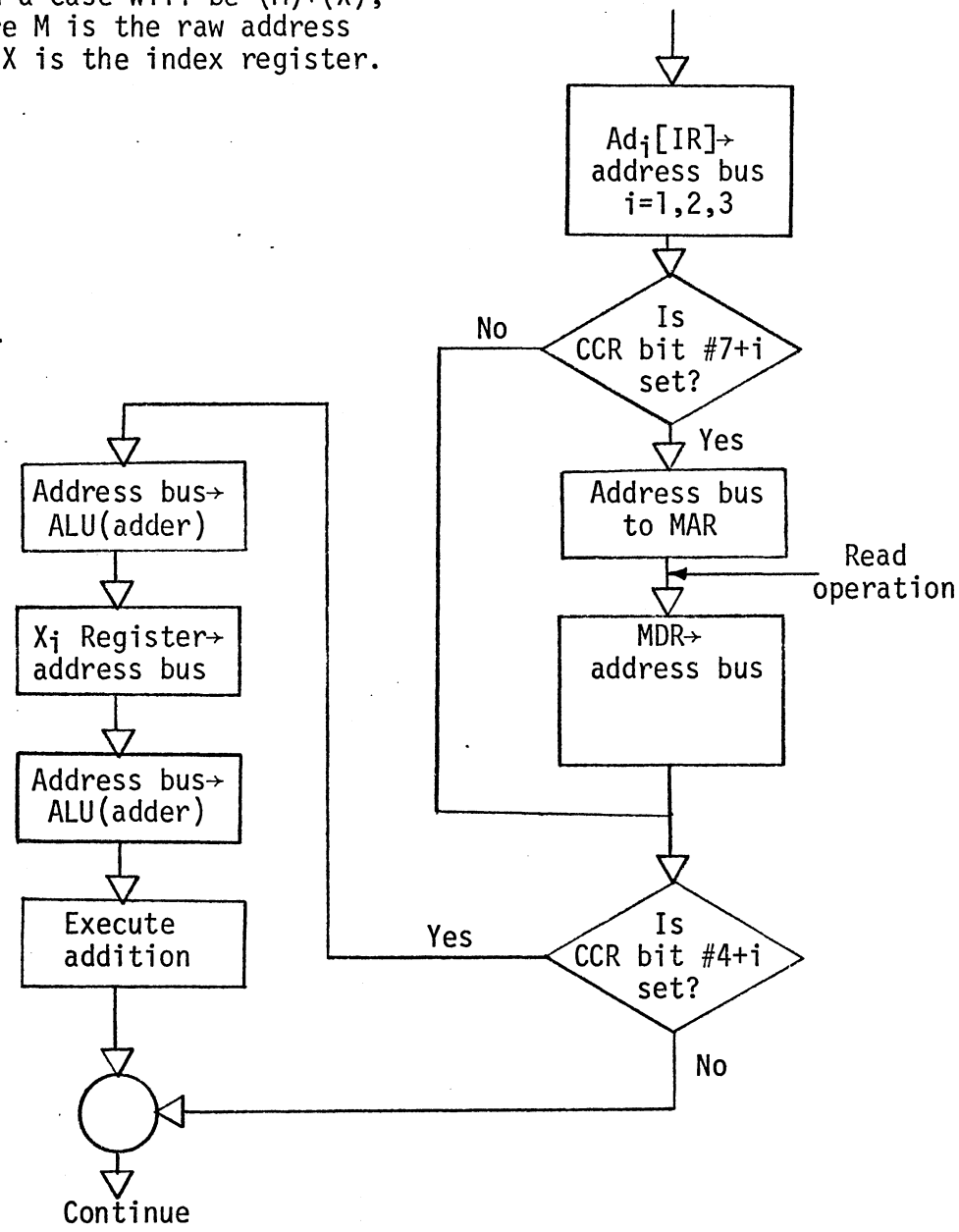


Figure 4. Addressing Options

CHAPTER VI

INSTRUCTION SET FLOW DIAGRAM

For each instruction in the instruction set, described in Chapter V, there is a micro-program stored in Read Only Memory (ROM). The portion of the ROM which holds the micro-program is referred to as the Control Memory (CM).

In this chapter the micro-programs required to execute the macro instructions are given in flow diagram form. In order to avoid redundancy, some of the microprograms which are similar to one another and differ only in one or two steps are not given in full and only their differences are mentioned. Simple instructions which involve few steps and their micro-programs are rather obvious are not described and their flow diagrams are not given.

6.1 Fetch Cycle

The micro-program which is executed to perform the fetch cycle and is common to all instructions is represented in Figure 5.

6.2 Input/Output Instructions

Figures 6 through 12 represent the flow diagrams of the Input/Output instructions.

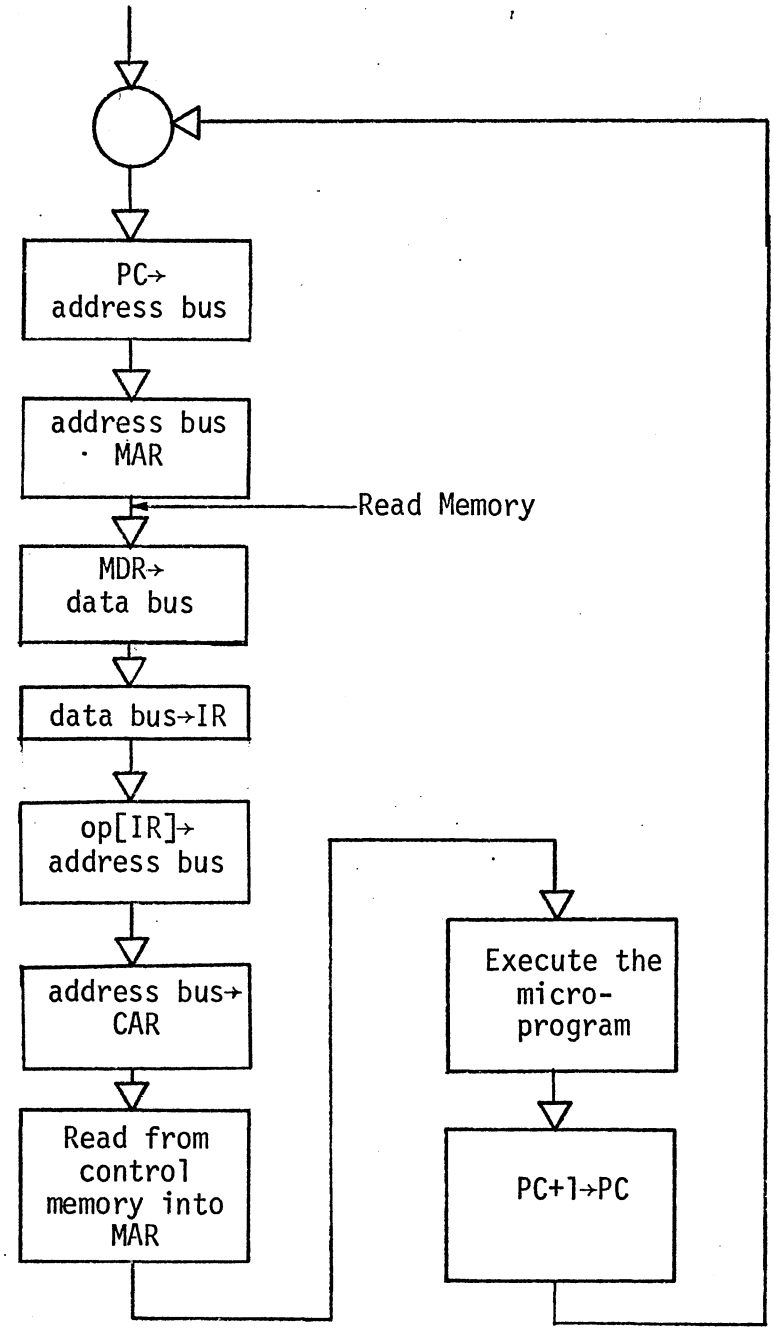


Figure 5. Fetch Cycle

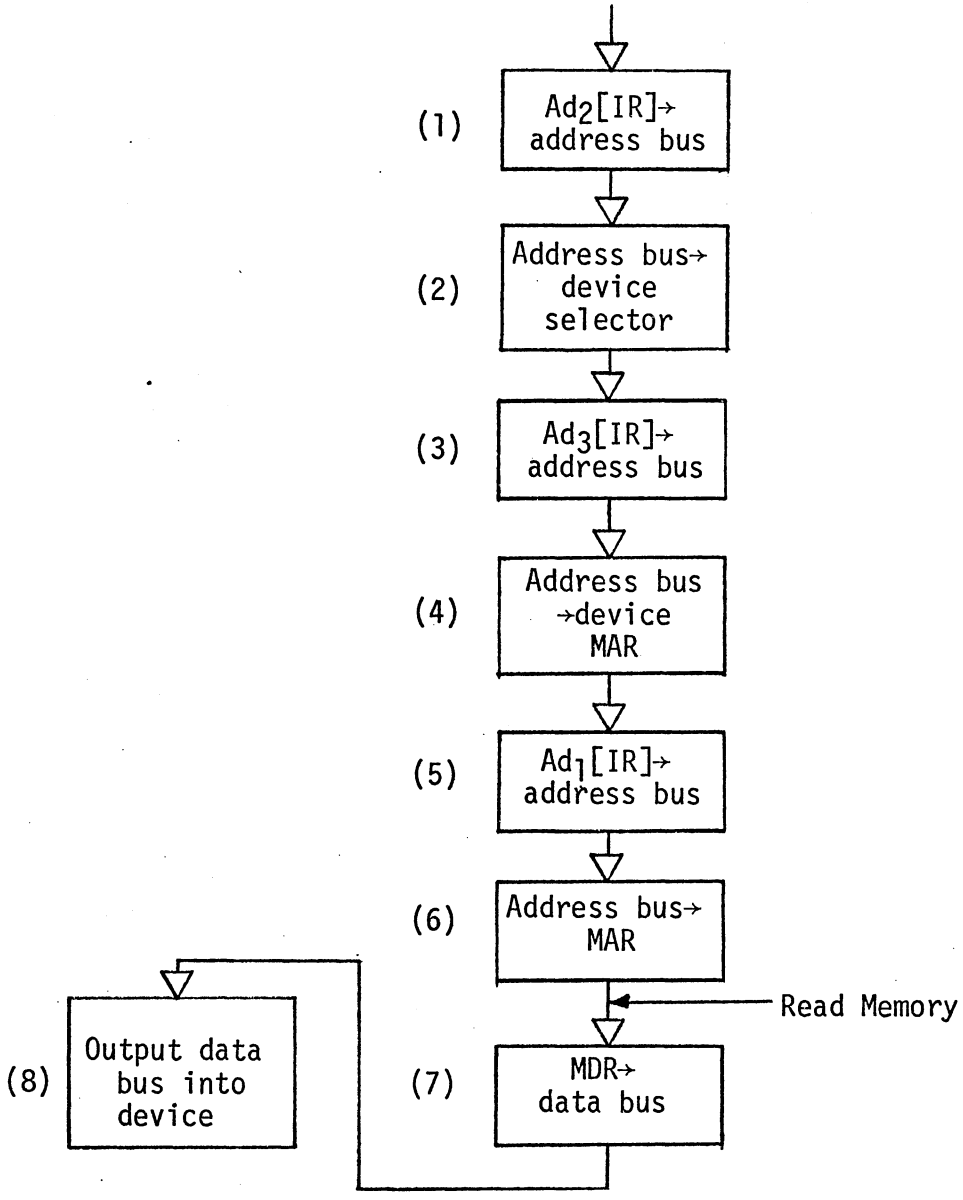


Figure 6. The Flow Diagram of OUT M,SC,DM

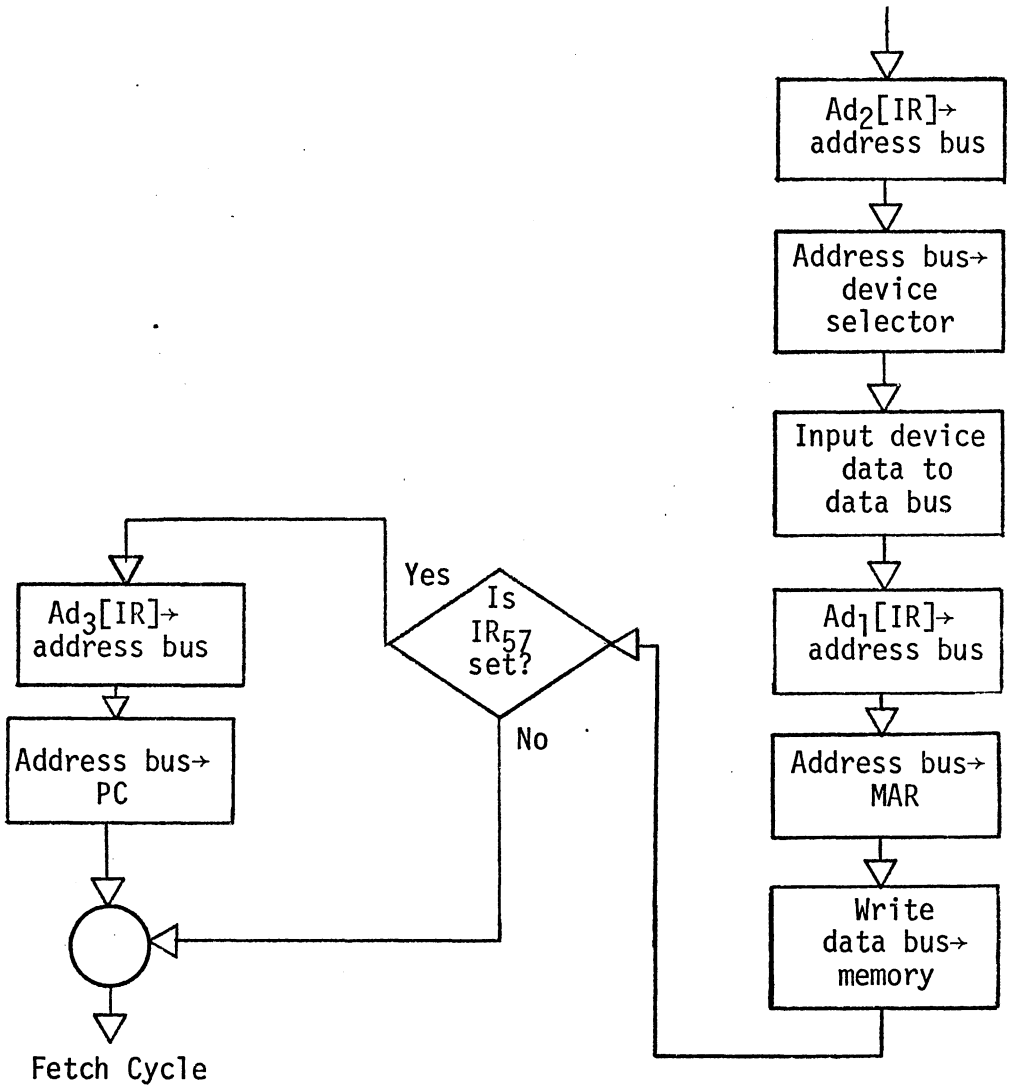


Figure 7. The Flow Diagram of INI M,SC,LAB

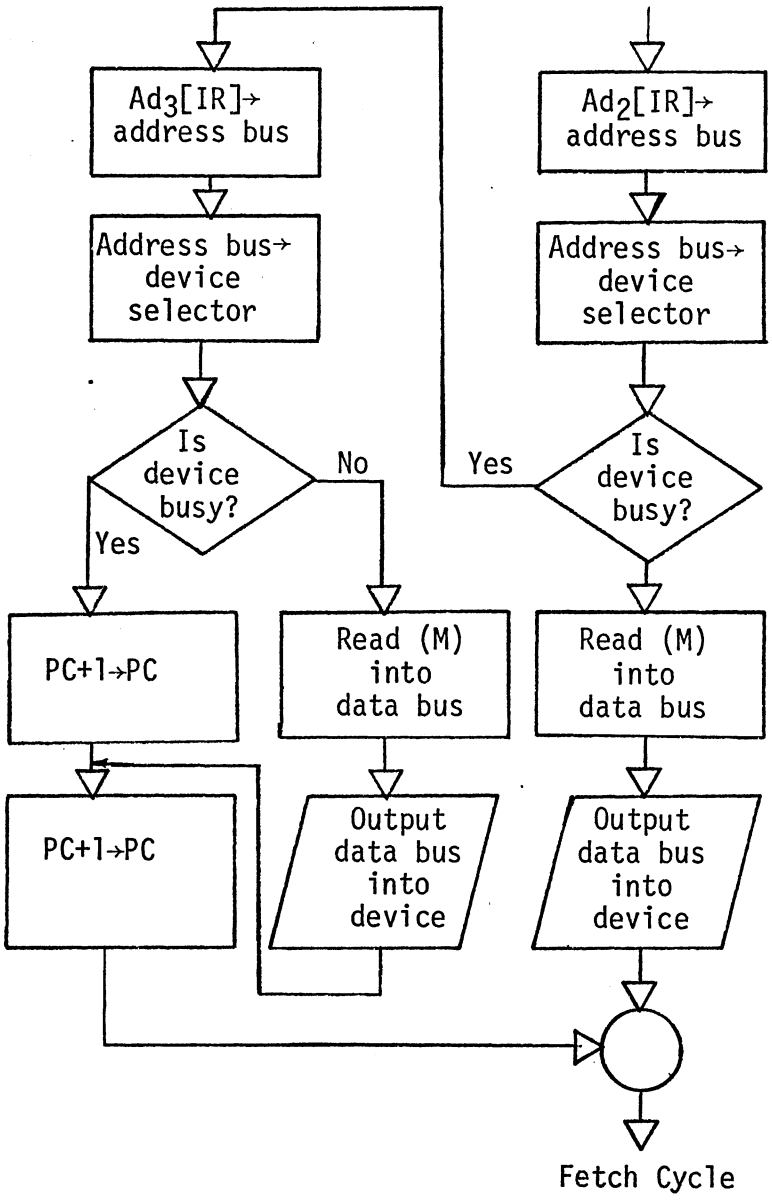


Figure 8. The Flow Diagram of OUR M,SC_1,SC_2

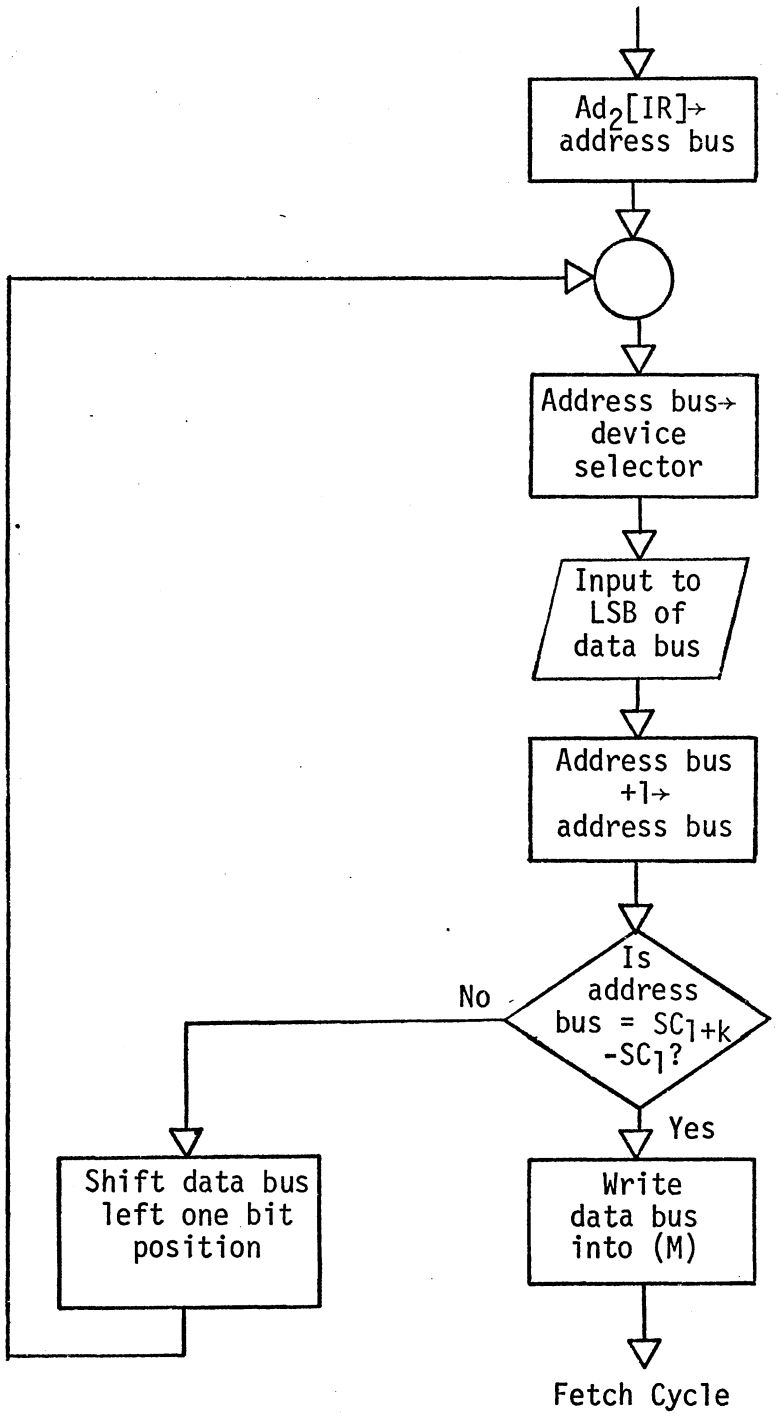


Figure 9. The Flow Diagram of INB M,SC_1,SC_{1+k}

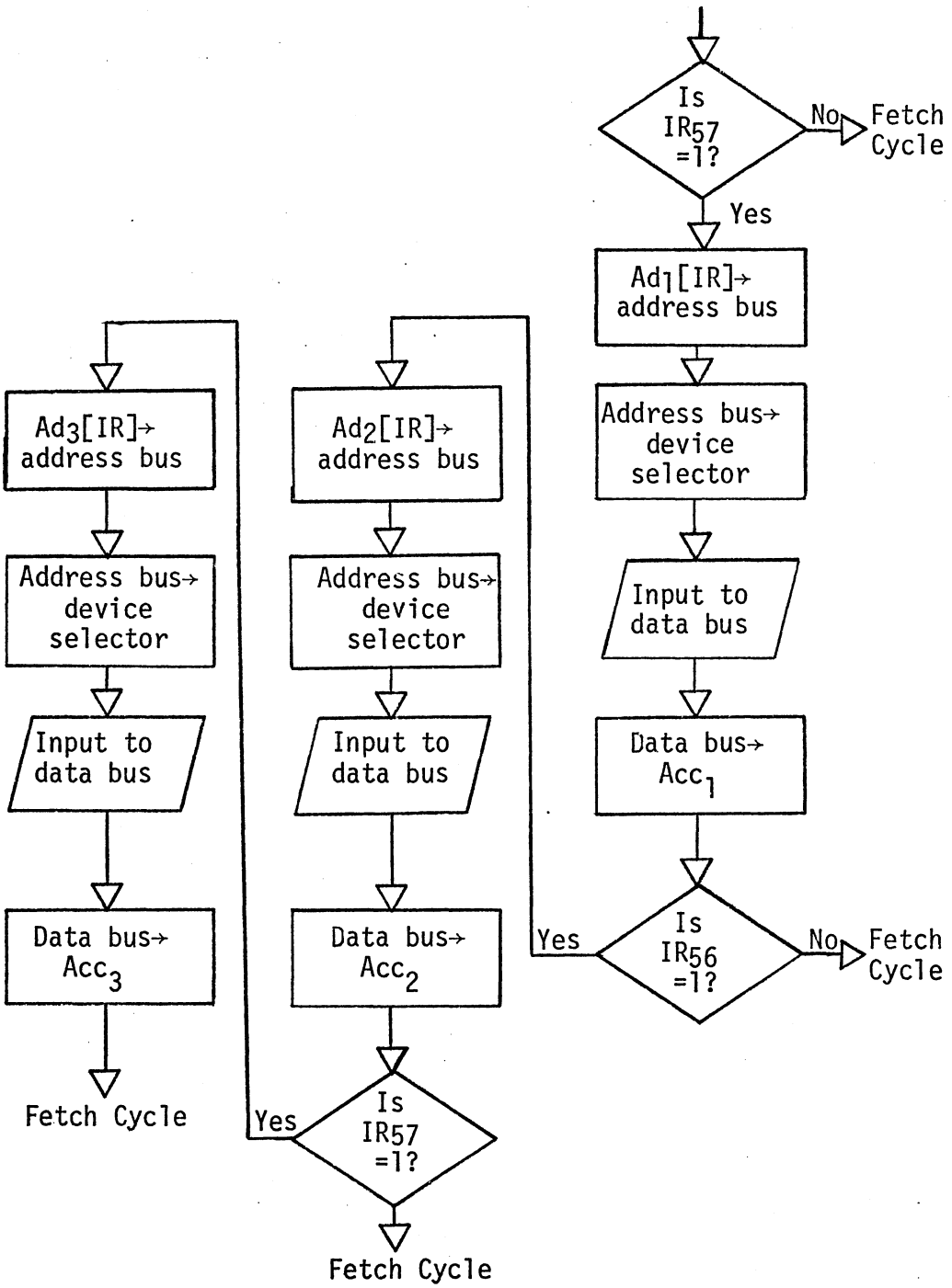


Figure 10. The Flow Diagram of IWP SC_1, SC_2, SC_3

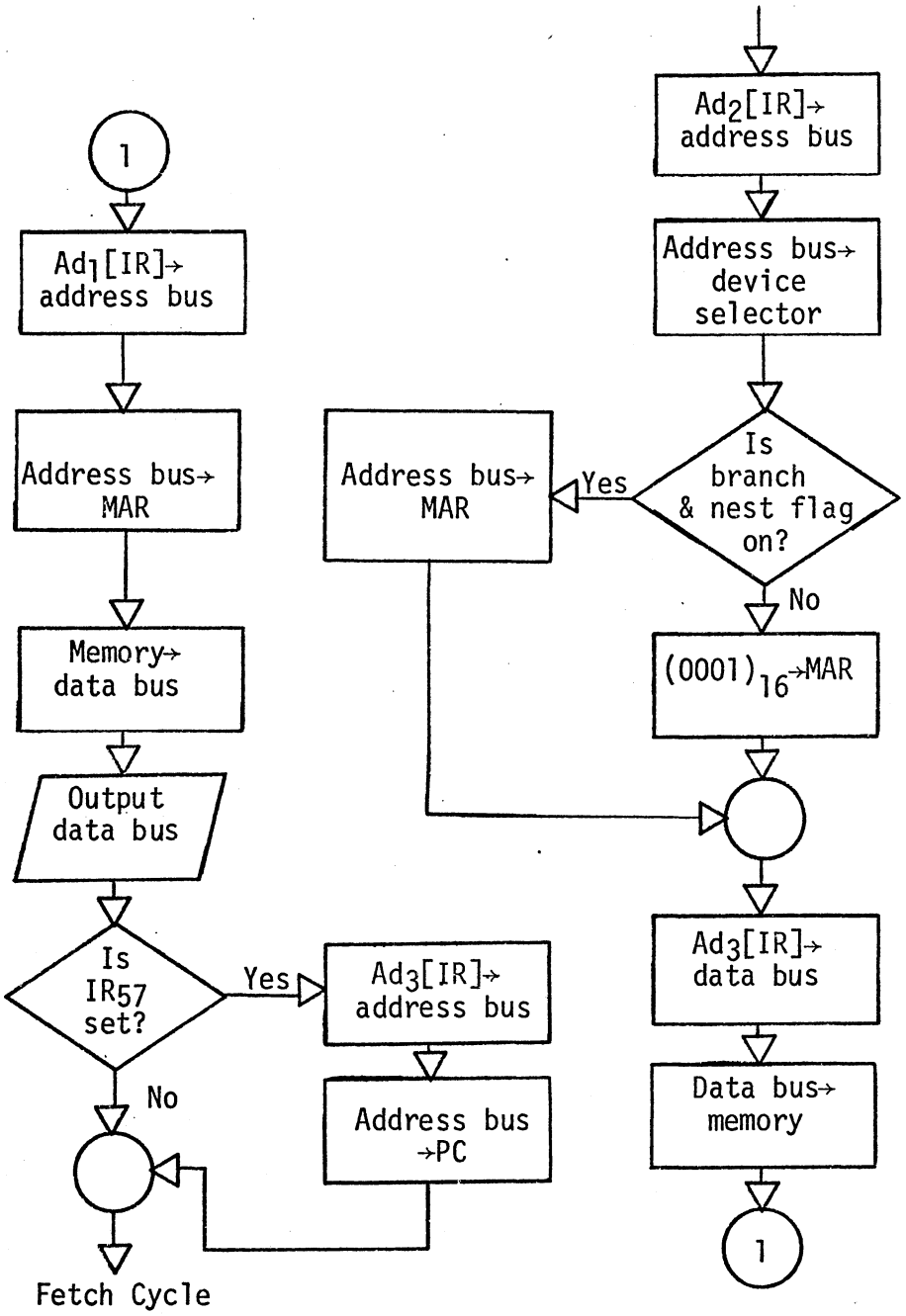


Figure 11. The Flow Diagram of OSR M,SC,LAB

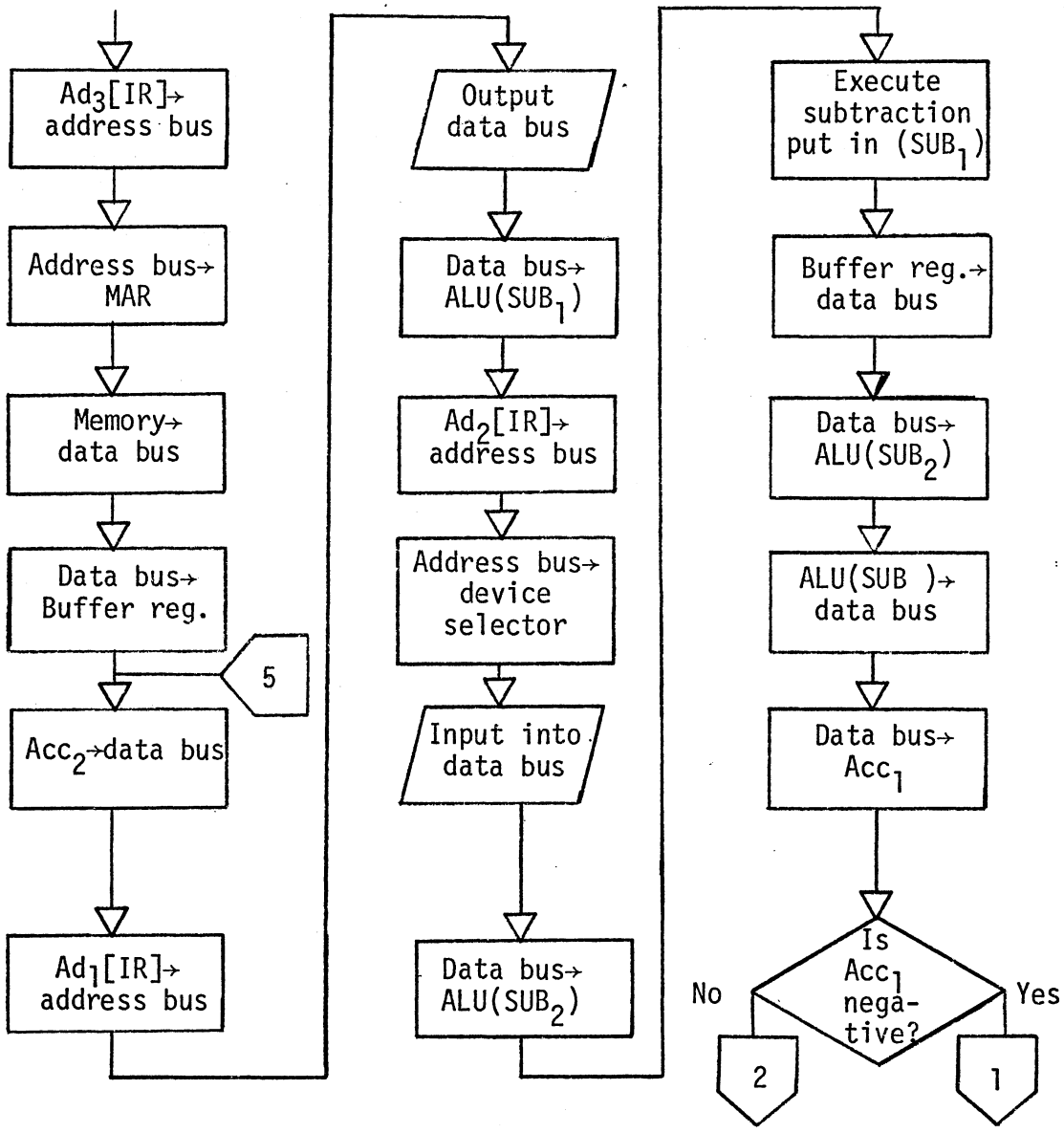


Figure 12. The Flow Diagram of FDB SC_1,SC_2,M

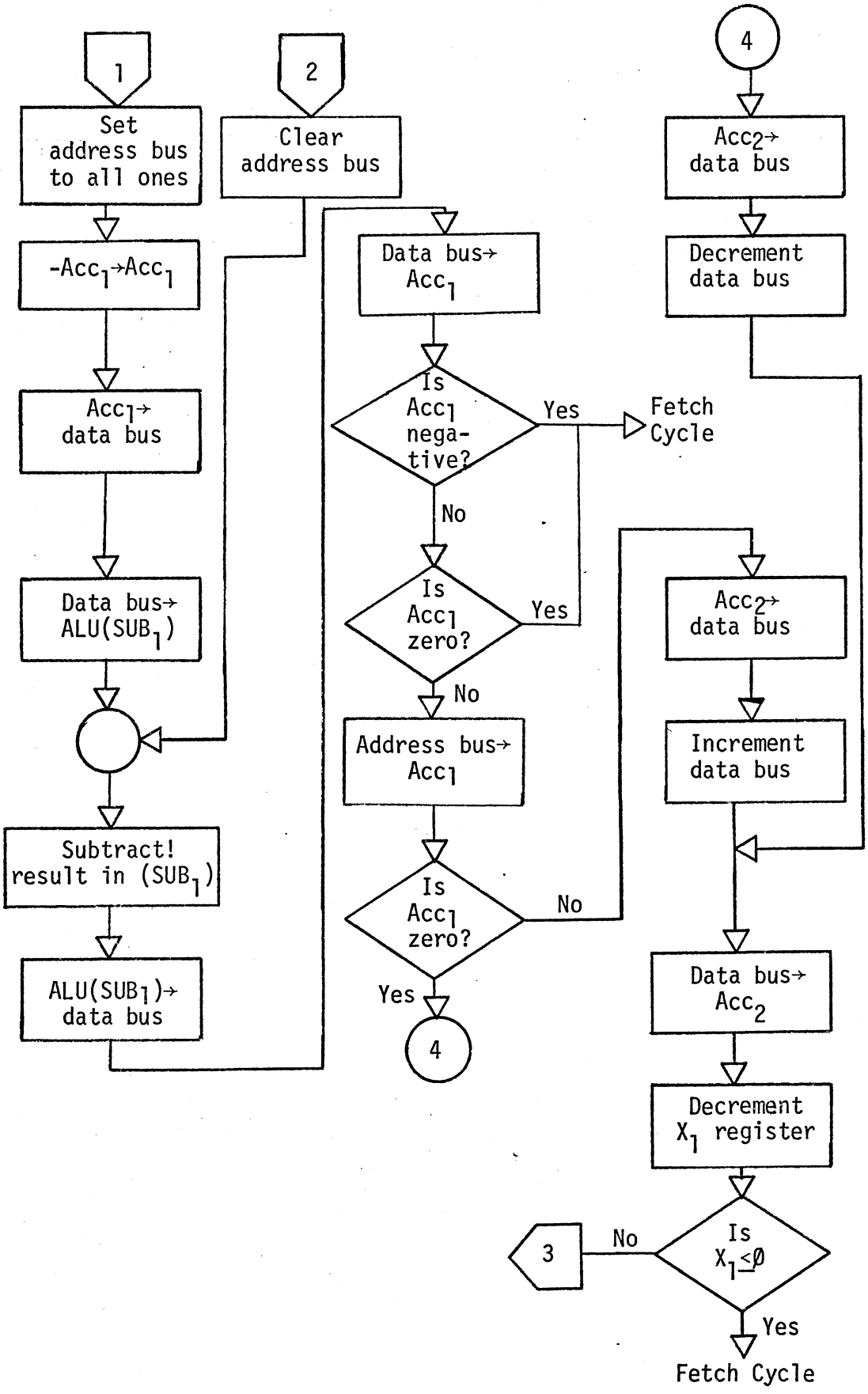


Figure 12. (Continued)

6.3 Other Instructions

Figures 13 through 15 represent the flow diagrams of some of the other instructions.

The rest of the instructions are either extensions of described instructions or well-known instructions used in most of the conventional computers and, therefore, are not described here.

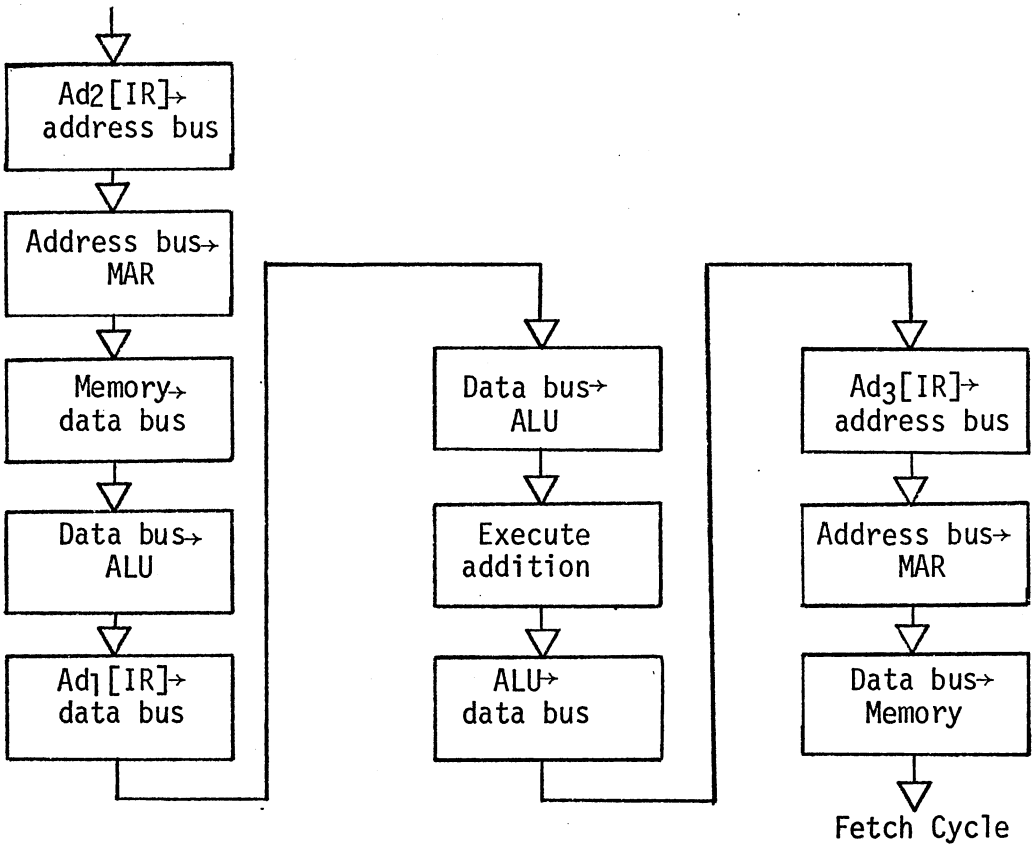


Figure 13. The Flow Diagram of ADA Imm,M_1,M_2

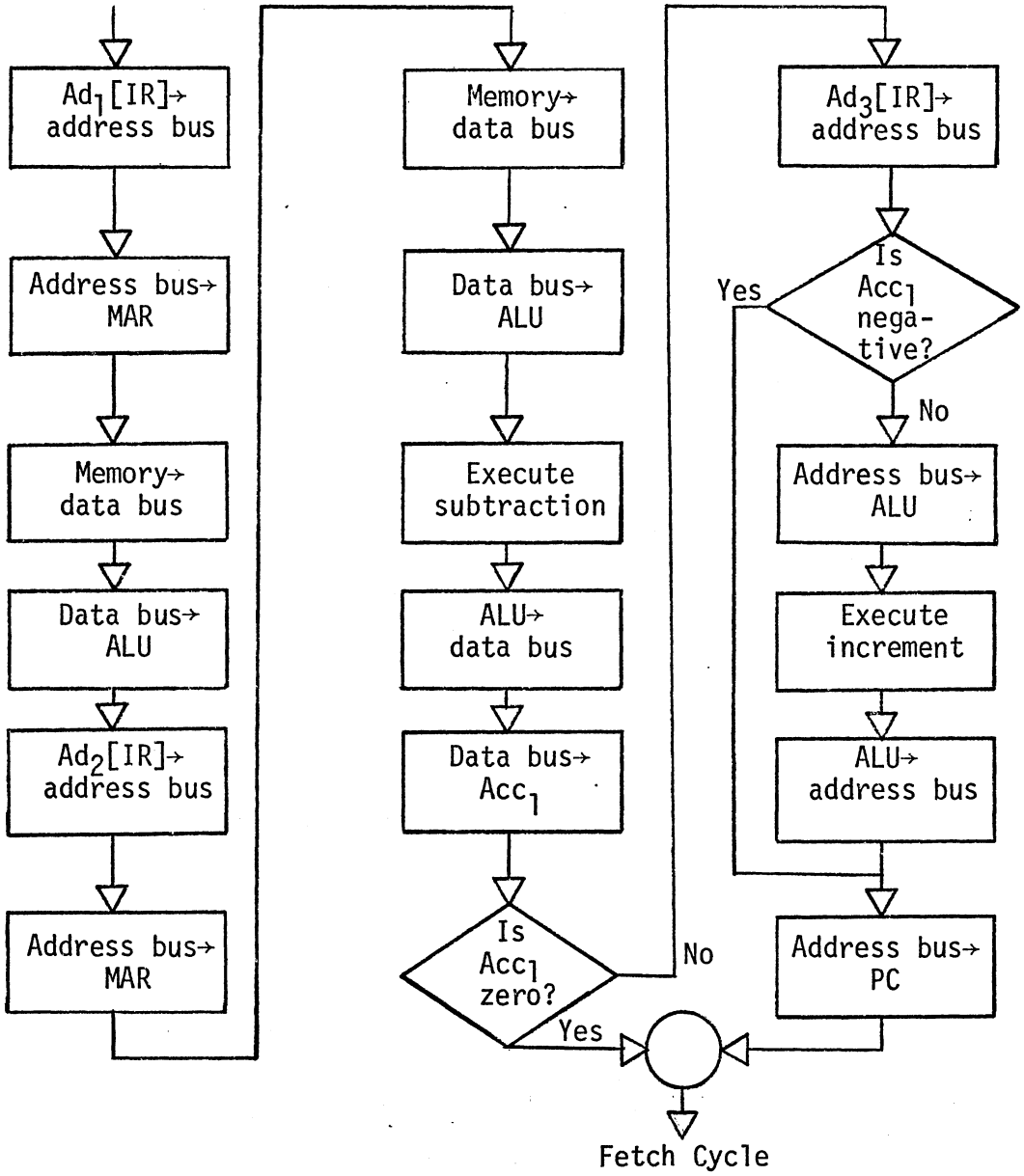
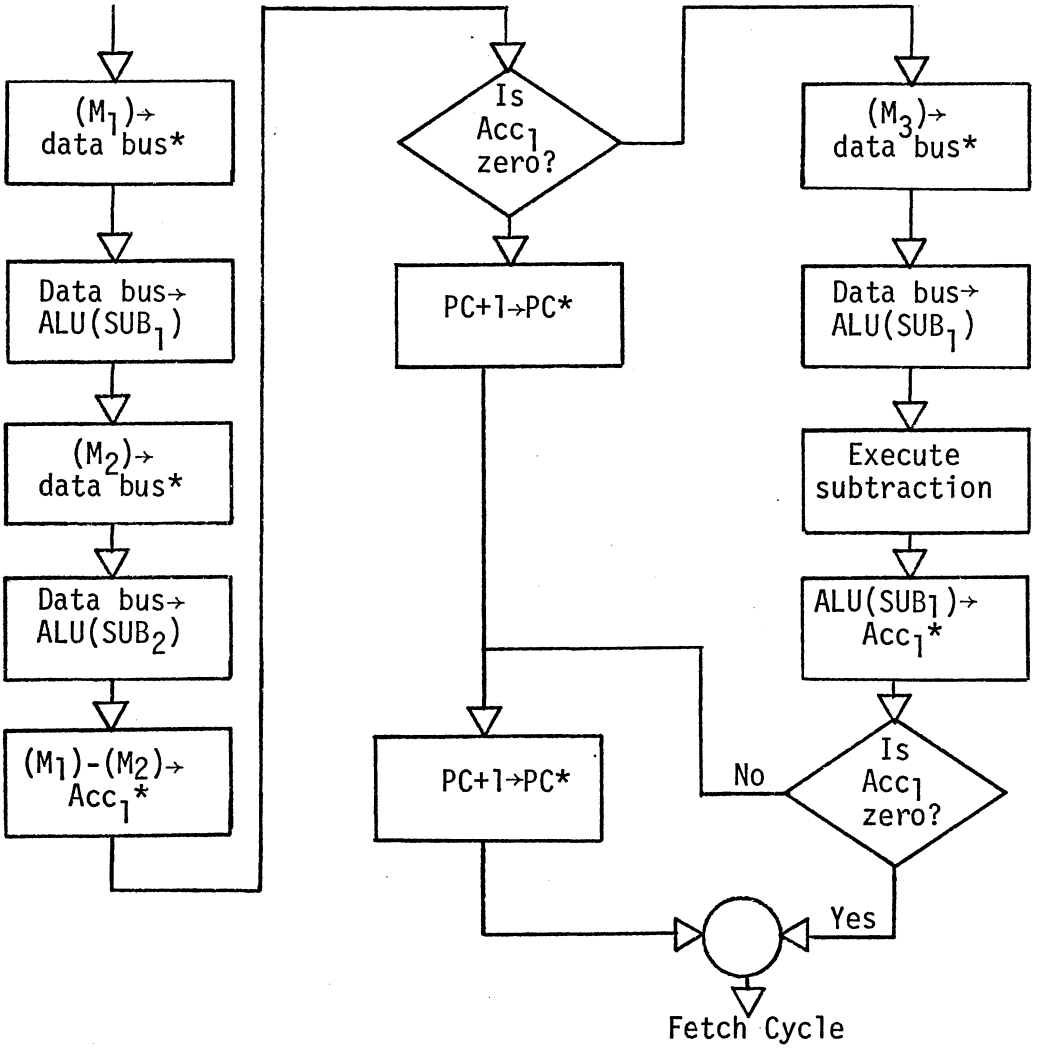


Figure 14. The Flow Diagram of COM M_1,M_2,LAB



*A multi-step micro-program given in previously drawn flow diagrams.

Figure 15. The Flow Diagram of CMT M_1, M_2, M_3

CHAPTER VII

SUMMARY AND CONCLUSIONS

A computer with a three-address instructions architecture serves to solve some of the problems involved in the programming and speed consideration.

In most of the conventional general purpose or dedicated computers delivered to the market, the number of Input/Output instructions is too small and usually little attempt is made to increase the variety of these instructions. This quantitative and qualitative limitation on the Input/Output instructions usually happens to be an inconvenience, if not a block, on the programming of the computers used for process control.

The variety of Input/Output instructions given in this effort proves to be helpful in this respect and hopefully can be considered as an aid to further studies of the improvement methods of process control computers.

To simplify the hardware required to transfer data within the registers, TAMP is designed such that all of the data transfers are performed via the buses. This will make the microprograms rather long which in turn reduces the speed of the computer. A more sophisticated hardware organization can avoid this disability.

There are also some hardware versus software trade-offs. Building

a digital comparator for compare instructions instead of using a micro-program is one example.

Considering the decreasing trend of the price of dense logic devices, and the advantages of the three-address computers, these need to be re-evaluated and they are likely to be found more suitable than single-address machines in the future.

BIBLIOGRAPHY

[1] Ivanoff, A. "Theoretical Foundations of the Automatic Regulation of Temperature." Journal of Institute of Fuel, Vol. 7 (1934), 117.

[2] Hazen, H. L. "Theory of Servomechanisms." J. Franklin Institute, Vol. 3 (1934), 279.

[3] English Electrical-Leo Computers Ltd. KDF.9 Programming Manual. Kidsgrove, England, 1963.

[4] Burroughs Corp. Burroughs B5500 Information Processing Systems Reference Manual. Detroit, Michigan, 1964.

[5] IBM Corp. IBM Reference Manual 1620 Data Processing System. White Plains, New York, 1960.

[6] Honeywell Electronic Data Processing. Honeywell Series 200, Model 8200, General Systems Description. Wellsely Hills, Mass., 1965.

[7] Allison, A. "An Improved Microcomputer Interrupt Structure." Computer Design, Vol. 13, No. 1 (January, 1974), 81-85.

VITA

Hamid Najafi

Candidate for the Degree of

Master of Science

Thesis: A THREE-ADDRESS MINICOMPUTER FOR PROCESS CONTROL

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Tehran, Iran, February 18, 1952, the son of Dr. and Mrs. H. Najafi.

Education: Graduated from Andisheh (Don Bosco) High School, Tehran, Iran, in June, 1970; received the Bachelor of Science degree in Electrical Engineering from Arya-Mehr University of Technology, Tehran, Iran, in 1975; completed requirements for the Master of Science degree at Oklahoma State University in May, 1977.

Professional Experience: Field Engineer, Burroughs Corporation, 1973-1975; graduate teaching assistant, School of Mathematical Sciences, Oklahoma State University, 1976-1977.