

INTEL 8085 MICROPROCESSOR SYSTEM  
SIMULATION ON IBM-370/115 MAINFRAME

A Thesis

by

MD. BASHIR UDDIN

Submitted to the Department of Electrical and  
Electronic Engineering, Bangladesh University  
of Engineering & Technology, Dhaka in partial  
fulfilment of the requirements for the Degree  
of

MASTER OF SCIENCE IN ENGINEERING (ELECTRICAL &  
ELECTRONIC)

November, 1984.



#61384#

INTEL 8085 MICROPROCESSOR SYSTEM  
SIMULATION ON IBM-370/115 MAINFRAME

A Thesis

by

MD. BASHIR UDDIN

Accepted as satisfactory for partial fulfilment of the requirement for the degree of Master of Science in Engineering (Electrical & Electronic), Bangladesh University of Engineering and Technology, Dhaka.

EXAMINERS



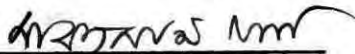
(DR. S. M. RAHMAN)  
Thesis Supervisor  
Computer Engg. Deptt.

Chairman



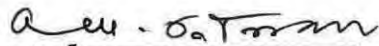
(DR. S. F. RAHMAN)  
Professor & Head,  
Electrical & Electronic  
Engineering Department

Member



(DR. M. R. KHAN)  
Professor & Head,  
Computer Engg. Deptt.

Member



(DR. A. M. PATWARI)  
Vice-Chancellor,  
BUET, Dhaka.

External  
Member

C E R T I F I C A T E

This is to certify that this thesis work has been done by me and it has not been submitted elsewhere for the award of any degree or diploma.

*Mel. Basiruddin*

Signature of the Candidate

A B S T R A C T

For the design of microprocessor based systems in an environment where the development systems are not available the development of simulators seems to be essential. The present work deals with the development of 8085 microprocessor simulator on IBM-370/115 mainframe. A powerful command set has been designed to communicate with the simulator for debugging purpose. Detailed study of intel 8085 microprocessor instruction set and its architecture has been carried out for its modelling on the mainframe. On the otherhand study was also made to incorporate the microprocessors 8-bit words with the full/half words of the mainframe. Finally a simulation program has been developed and tested from the simulation algorithm of the microprocessor model.

## ACKNOWLEDGEMENT

The author wishes to express his gratitude and indebtedness to Dr. Syed Mahbubur Rahman, Assistant Professor, Department of Computer Engineering, Bangladesh University of Engineering and Technology, Dhaka for his continuous guidance, suggestion, interest, contribution of new ideas and supervision at all stages of the research.

He is grateful to Prof. A.M. Patwari, Vice-Chancellor of Bangladesh University of Engineering & Technology, Dhaka for his suggestions at different times.

The author wishes to express his gratitude to Dr. Shamsuddin Ahmed, Dean & former Head, Dept. of Electrical and Electronic Engineering, BUET and also to his teacher Dr. Syed Fazl-i Rahman, Prof. and Head, Dept. of Electrical & Electronic Engg. BUET for constant advice and encouragement in carrying out this study.

Acknowledgement is due to Dr. A.M. Zahoorul Haque, Prof. and former Dean, Dept. of Electrical & Electronic Engg. BUET and also to Dr. Mahfuzur Rahman Khan, Prof. & Head, Dept. of Computer Engg. BUET, Dhaka for giving inspiration in carrying out the programme.

C O N T E N T S

ABSTRACT	i
ACKNOWLEDGEMENT	ii
CONTENTS	iii
LIST OF FIGURES & TABLES	v
<u>CHAPTER - I</u> : <u>INTRODUCTION</u>	1-1
1.1 What is simulation	1-1
1.2 Objective of the present Research	1-2
1.3 Expected results of the present work	1-4
<u>CHAPTER -II</u> : <u>COMMUNICATION WITH SIMULATION PROGRAM</u>	2-1
2.1 Introduction	2-1
2.2 Commands and their formats	2-1
2.3 Description of the commands	2-6
2.3.1 Input/output transfer command	2-6
2.3.2 Input-data transfer command	2-7
2.3.3 Breakpoint setting for program segment simulation	2-9
2.3.4 Memory dump	2-10
2.3.5 Program loading	2-11
2.3.6 Single step printing	2-12
2.3.7 Exit from endless loop	2-13
2.3.8 Setting of program counter	2-13
2.3.9 Setting of register contents	2-14
2.3.10 Start and end of simulation	2-15
2.4 Example of command set	2-15
<u>CHAPTER-III</u> : <u>MICROPROCESSOR MODEL ON MAINFRAME</u>	3-1
3.1 Analysis of the problem and programming language selection	3-1
3.2 The Microprocessor architecture and its image on the mainframe	3-3
3.2.1 Register structure and their images	3-3
3.2.2 Arithmetic logic unit and its image	3-10
3.2.3 Control unit and its image	3-11
3.2.4 Intel 8085 memory and its image	3-12
3.3 Summary	3-12

<u>CHAPTER - IV</u>	: <u>DEVELOPMENT OF SIMULATION PROGRAM</u>	4-1
4.1	Introduction	4-1
4.2	Command processor	4-4
4.2.1	Command recognizer	4-4
4.2.2	Command validity checker and executor	4-5
4.3	Control program	4-5
4.4	Debugger	4-8
4.5	Error handler	4-9
<u>CHAPTER - V</u>	: <u>SIMULATION PROGRAM TEST RUN</u>	5-1
5.1	Test problem	5-1
5.2	Program flowchart & listing	5-3
5.3	Test results	5-7
5.4	Error tests	5-8
<u>CHAPTER - VI</u>	: <u>DISCUSSION &amp; CONCLUSION</u>	6-1
APPENDIX - A		
APPENDIX - B		
APPENDIX - C		
REFERENCES		

LIST OF FIGURES

- Fig. 3.1 Functional block diagram of 8085 microprocessor.
- Fig. 3.2 8085 Status flags.
- Fig. 3.3 8085 Registers and memory picture on mainframe.
- Fig. 4.1 State diagram showing transition among the program units of the simulator.
- Fig. 4.2 Control program showing its different functions.
- Fig. 4.3 Address table of the instruction program segment.
- Fig. 5.1 Circuit diagram of the test problem.
- Fig. 5.2 Flowchart for test program.

LIST OF TABLES

- Table 2.1 Command format.
- Table 5.1 Test results.
- Table 5.2 Test results.
- Table 5.3 Test results.
- Table 5.4 Results of error test.
- Table 5.5 Results of error test.



CHAPTER - I  
INTRODUCTION

1-1. WHAT IS SIMULATION

The word "Simulation" does mean different things to different people according to the mode of requirements. Man has been simulating since first his brain developed the power to imagine.

According to EDICT "Simulation is the development and use of models to aid in the evaluation of ideas and study of the systems or situations\*(5). McCooy defines "Simulation is the act of representing some aspects of the real world which may be easily manipulated"(5).

The philosophical basis for the design of a simulation module is the concept of modeling a system approximately in proper method which represents the behaviour of the

system very close to the real situation.

In generalised form simulation may be defined as the technique of creating a model of experimental or practical situation that may happen in a device or in a laboratory or in a field.

The simulation may be classified in different categories. Since the purpose of the present work is the simulation of a microprocessor system, therefore the digital computer-aided simulation is the main interest of this thesis.

## 1-2 OBJECTIVE OF THE PRESENT RESEARCH

In just one decade microprocessors have found thousands of applications in commerce, industry, transport, medicine, education, defence, communications, food processing, agriculture, entertainment and domestic life. No section of human activity is untouched by the microprocessor. But the efficient utilization of this most modern and innovative technology in the various fields requires a detail understanding of the software features alongwith the hardware concept of the microprocessor. For successful operation of a microprocessor based system effective software development plays an important role. In the development of software module, it is frequently necessary to make numerous corrections and changes.

When designing a new microprocessor based system even for a simple application, software development of the

system requires debugging aids for successful execution of the programs. Usually microprocessor development system (MDS) provides these facilities on which programs may be written, translated and stored during development. It provides the means for editing and testing program module as they are written even at a stage where the prototype hardware is incomplete or totally absent( 8 ). It is therefore possible for hardware development to proceed in parallel with software development. Through application of this development system one can display or print the contents of the registers, memory locations and debug the programs using single stepping or break points. Without this system it is almost impossible to findout the various complicated types of errors that occur during program development phase. This development system is very much expensive and supplied by the manufacturers at high cost.

As such attempts are made to have simulated programs to model the microprocessor on an existing large computer in order to approximately satisfy the requirements of the software aspects of the microprocessor development system. Usually different development systems are required to develop different microprocessor softwares where as the same mainframe could be used for several microprocessor simulation. Using the simulation program the process of software testing and debugging can be speeded up because of superior capabilities of large computer in comparison to that

of microprocessor, simulations can also incorporate explicit error messages, traps for illegal conditions and diagnostics. In order to achieve all the facilities discussed above, without having the costly microprocessor development system it has been decided to attempt at the Development of a simulation module for a microprocessor system on the existing main frame.

### 1.3 EXPECTED RESULTS OF THE PRESENT WORK

As a result of the present research work a simulation module is expected to be developed, the implementation of which will provide some aids to develop the microprocessor softwares on the mainframe. The following are some of the most important aids for the above purpose.

- i. Execution of microprocessor programs in single stepping or upto certain predefined address etc.
- ii. Display or printout the contents of all the registers, memory locations, stack pointer and program counter etc. at every step or only at predefined breakpoints or from a certain address to any other addresses etc.
- iii. Setting the program counter with a specified address before starting execution of the program or after one or more segments of the

## CHAPTER - II

### COMMUNICATION WITH SIMULATION PROGRAM

#### 2.1 INTRODUCTION

This Chapter focuses on the formulation and description of the communicating tools of the simulation program which we call as 'command'. The commands have been designed for loading, checking, executing, displaying, transferring and printing etc. of microprocessor program and to provide various other facilities during its development phase.

#### 2.2 COMMANDS AND THEIR FORMATS

The commands designed to communicate with the simulation program can broadly be classified into two groups as given below:

- a) The commands in the first group are utilized to set certain parameters that will be used during execution of microprocessor instructions. These parameters are contained in the command table of that specific command.

For example if we want to define the breakpoints of a microprocessor program, the addresses for the breakpoints must be stored before execution. If more than one break point is to be defined multiple addresses are to be stored. So the command table of the 'BREAKPOINT' command contains the address parameters. The maximum size of command table depends on the maximum number of breakpoints that we would allow. After execution of each microprocessor instruction the previously defined breakpoints are compared with the address of the next instruction and if found equal, breakpoint condition is satisfied.

The command table configuration and type of parameters are dependent upon the command used. The command parameters may be microprocessor address, instruction code and data.

- b) The commands forming the second group are instantaneously utilized during their execution to set the microprocessor registers and memory contents.

The command formats are shown in Table 2-1. The following syntax are followed for writing the commands:

- i) Words with capital letters indicate the command itself.
- ii) Items surrounded by "< >" are those to be specified by the user.
- iii) Items enclosed in brackets "[ ]" are optional and can be omitted. If these items are omitted the simulation program will supply the default values or previously specified values wherever necessary.
- iv) Symbols other than those described above, e.g. commas, hyphens, equal signs etc. must be entered exactly at the positions as they are shown in the table.
- v) Optional items indicated by "....." can be repeated as many times as desired within the maximum specified limit for that command.
- vi) When two or more items appear between two vertical lines, this means that the desired function must be specified by indicating one of the items for input.

Table 2.1 Command format:

a) Commands to set the parameters of the command table

1. \* $\backslash$ OPTIONS- $\left\{ \begin{array}{l} \text{IN} = \left| \begin{array}{l} \text{DSKT} \\ \text{KBRD} \end{array} \right| \backslash \left| \begin{array}{l} \text{OUT} = \{ \text{LPRINT, CONSOL} \} \end{array} \right. \end{array} \right\}$
2. \* $\backslash$ IN- $\langle$ port address $\rangle \backslash \{ \langle$ data-1 $\rangle \backslash \dots \dots \backslash \langle$ data-n $\rangle \} \backslash \text{\$}\text{\$}$   
where n is not greater than 23
3. \* $\backslash$ STOP $\backslash$ AT $\backslash$  BREAKPOINTS- $\{ \langle$ addr.-1 $\rangle \backslash \dots \dots \backslash \langle$ addr.-n $\rangle \} \backslash \text{\$}\text{\$}\text{\$}\text{\$}$   
where n is not greater than 11.
4. \* $\backslash$ PRINT $\backslash$ AT $\backslash$  BREAKPOINTS- $\{ \langle$ addr.-1 $\rangle \backslash \dots \dots \backslash \langle$ addr.-n $\rangle \} \backslash \text{\$}\text{\$}\text{\$}\text{\$}$   
where n is not greater than 11.
5. \* $\backslash$ DUMP $\backslash$ FROM $\backslash$  ADDR- $\langle$ start-addr. $\rangle \backslash$  TO $\backslash$   $\langle$ end-addr. $\rangle \backslash$  AT $\backslash$   
 $\{ \langle$ addr.-1 $\rangle \backslash \dots \dots \backslash \langle$ addr.-n $\rangle \} \backslash \text{\$}\text{\$}\text{\$}\text{\$}$   
where n is not greater than 9.
6. \* $\backslash$ ENTER $\backslash$  TO- $\langle$ addr-1 $\rangle \backslash$  AT $\backslash$   $\langle$ addr-2 $\rangle \{ \langle$ data-1 $\rangle \backslash \dots \dots \backslash \langle$ data-n $\rangle \} \backslash \text{\$}\text{\$}$   
where n should not exceed microprocessor memory size.
7. \* $\backslash$ PRINT $\backslash$  SINGLE $\backslash$  STEPPING $\text{\$}$
8. \* $\backslash$ PRINT $\backslash$  SINGLE $\backslash$  STEP $\backslash$  FROM  $\langle$ address-1 $\rangle \backslash$  TILL $\backslash$   $\langle$ address-2 $\rangle$



(Table 2.1 cont.)

9. \*~~MAX~~INSTRUCTIONS-<max.instruction number>

10. \*~~SET~~PC~~AT~~-<address>~~WITH~~<address>

b. Commands to set microprocessor registers and memory.

11. \*~~ENTER~~FROM-<address> ~~{~~<data-1> ~~} ....<data-n> ~~} \$\$\$~~  
 where n should not exceed microprocessor memory size~~

12. \*~~SET~~PC-<address>

13. \*~~SET~~REGISTER- $\left\{ A=\langle \text{data} \rangle \mid \mid B=\langle \text{data} \rangle \mid \mid C=\langle \text{data} \rangle \mid \mid \right.$   
 $\left. D=\langle \text{data} \rangle \mid \mid E=\langle \text{data} \rangle \mid \mid H=\langle \text{data} \rangle \mid \mid L =\langle \text{data} \rangle \right\}$

14. \*~~P~~DUMP-<address-1> ~~To~~<address-2>

15. \*~~EXEC~~\$

16. \*~~END~~\$

- vii) If one or more items enclosed in braces "{ }", at least one of the items must be specified.
- viii) A blank space is indicated by the symbol " / ".

## 2.3 DESCRIPTION OF THE COMMANDS

The first column of all the commands contain an asterisk(\*) symbol .A hyphen(-) is used in the command to separate the command parameters and dollar(\$) is used to end the command. The addresses mentioned in the command format consist of 2 bytes indicating the microprocessor memory address. The data and the port address consist of 1 byte. The detail description of the design and functions of the commands are discussed in the subsequent paragraphs.

### 2.3.1 INPUT/OUTPUT SIMULATION COMMAND

The 8085 microprocessor uses I/O mapped I/O method to transfer data between microprocessor and the external world . These operations are performed by using two-byte instruction known as IN and OUT. These I/O operations may be simulated through using secondary storage(diskette,disk, tape etc.), console, console key board and line printer etc. without changing any hardware configuration of existing IBM mainframe. In the simulation module the provision of I/O operations are provided by using diskette (DSKT) and Console key board(KBRD) as input medium and line prin-

ter( LPRINT) and console(CONSOL) as output medium. The OPTION command is introduced for selecting of input and output media for data transfer between microprocessor and I/O ports.

As this command is related to I/O operations, so it may be omitted if the program does not involve any input-output data transfer. The format of the command that represents the input/output simulation is given below:

Command format:

$* \cancel{O}PTIONS \left\{ IN = \begin{array}{ l} DSKT \\ \hline KBRD \end{array} \left  \cancel{O} \right. \right\} OUT = \{ LPRINT, CONSOL \}$
---

### 2.3.2 INPUT- DATA TRANSFER COMMAND

The input to the microprocessor accumulator from an I/O port is accomplished by executing the 2-byte IN instruction. The second byte of the instruction is the port address from where the data are to be transferred. In the previous sub-section(2.3.1), the diskette and console have been selected as input media(ports) for the simulation module. The IN command is introduced in order to transfer the data from a specified port through diskette to the microprocessor accumulator whenever an IN instruction is executed for that port. Data through console keyboard may be directly obtained by displaying the appropriate message

(shown in appendix-A) upon the console at the instant of execution of the instruction. The IN command must be used in conjunction with OPTION command if diskette is used for simulating the input ports. The first parameter in this command is the 1-byte port address and the others are the port data, each of which will be serially loaded to the accumulator during execution of each microprocessor IN instruction.

This command for a particular job may be more than one and its number depends on the number of port addresses used in the program. Therefore the number of IN command should be equal to number of input port address. The simulation program allows maximum of 16 ports for a particular job and port address should be in between 00(Hex) and FF(Hex). The maximum number of data in each port should not exceed 23. This number comes from the capacity of an eighty column record containing the command and port data. The command must be ended with two consecutive dollar (\$) sign. The command format is given below :

Command format:

<pre>*IN-&lt;Port address&gt;{&lt;data-1&gt;.....&lt;data-n&gt;}\$\$</pre>
--

### 2.3.3 BREAKPOINT SETTING FOR PROGRAM SEGMENT SIMULATION

To help in the debugging and diagnosis of programs for the microprocessor it is convenient to have the whole program in the form of several segments specially when dealing with complex programs. The confirmation of one segment will lead to debug the next one. This switching function from one program segment to another can be achieved through using breakpoints.

The command "STOP AT BREAKPOINTS" allows the user to execute the section of a program until one of the breakpoint conditions is met. The execution of the program is then stopped and the control may be transferred to execute the next command statement.

The command "PRINT AT BREAKPOINTS" serves for observing the status of the microprocessor registers corresponding to the breakpoint addresses. It allows the user to obtain a printout if any one of the breakpoints is found. This printout may be to show the contents of general purpose registers, accumulator, program counter, stack pointer, different flag bits, instruction code etc.

Typical breakpoint conditions are program counter address or microprocessor data references. Each address parameter consists of two bytes followed by a blank space. The maximum number of breakpoints allowed in each command should not exceed 11. This number comes from the capacity of an

an 80 column record containing the command and its parameters. The command statement should be ended with four consecutive dollar symbol.

The commands using the breakpoints are as follows:

Command format:

```
*STOP AT BREAKPOINTS- { <addr.-1> . . . . <addr.-n> } $$$$
*PRINT AT BREAKPOINTS- { <addr.-1> . . . . <addr.-n> } $$$$
```

Where n is not greater than 11

#### 2.3.4 MEMORY DUMP

In some cases we need to dump the memory for locating the errors. The following two commands have been introduced to obtain the microprocessor memory dump for effective debugging of programs.

Command format:

```
*DUMP FROM ADDR- <start-addr.> TO <end-addr.> AT
{ <addr.-1> . . . . <addr.-n> } $$$$
*PDUMP- <address-1> To <address-2>
```

The first command allows the user to get a storage dump from the start-address to end-address when any one of the dump address is found. Each address consisting of 2 bytes is separated from the next one by a blank space. This command is also ended with four consecutive dollar symbol. A table of dump address is formed on execution of this command. Considering the space provided by an 80 column

record containing the command and its parameters, the maximum allowable number of dump addresses in each command may be upto 9.

The execution of the second command (shown above) produces a memory dump, the size which is specified in the command parameters.

### 2.3.5 PROGRAM LOADING

The following commands are designed to load the microprocessor program and the associated data to the mainframe memory.

#### Command format

<pre>*ENTER TO-&lt;addr-1&gt; AT &lt;addr-2&gt; {&lt;data-1&gt; ... &lt;data-n&gt;} b\$\$\$ *ENTER FROM-&lt;address&gt; {&lt;data-1&gt; ... &lt;data-n&gt;} \$\$\$ where n should not exceed microprocessor memory size.</pre>
--

Through execution of first command, the instruction codes and data are transferred to the command table. During the execution of microprocessor program, if the program counter attains the address-2 all the codes will be transferred from command table to the microprocessor designated mainframe memory starting at the address-1.

The execution of the second command transfers the program and data directly to the microprocessor designated mainframe memory starting at the address specified

in the command parameters.

In both commands, data consisting of 1 byte may be either instruction code or data. The maximum number of data allowed in the command should not exceeds microprocessor memory size. The codes and data must be ended with two consecutive dollar sign.

### 2.3.6 SINGLE STEP PRINTING

During program development phase, the user may require to examine the contents of the registers and memory locations, program counter, stack pointer and different flag bits etc. in every execution step of the instruction. Single step printing can provide these facilities. The following commands have the capability to provide the single-stepping facilities.

The first command allows the user to execute the program one instruction at a time and to give a printout for that when program counter attains address-1 and it will continue until address-2 is reached. Whereas the second command provides the single step printing as long as the program execution continues.

Command format:

<pre>*<del>P</del>PRINT<del>P</del>/SINGLE<del>P</del>/STEP<del>P</del>/FROM &lt;address-1&gt; <del>P</del>TILL<del>P</del> &lt;address-2&gt;</pre> <pre>*<del>P</del>PRINT<del>P</del>/SINGLE<del>P</del>/STEPPING\$</pre>
---



### 2.3.7 EXIT FROM ENDLESS LOOP

In many cases the programmer uses instruction-loop for repetition of a particular set of calculations. Normally the loop is terminated after the range has been executed which may consist of one or any number of statements. But unfortunately if the control finds no condition to exit from the loop then the repetition of cycles will be continued and never ends until it is intervened. The following command is designed to exit from such an endless loop.

Command : format:

<code>*<del>P</del>MAX<del>P</del>INSTRUCTIONS-&lt;max.instruction number&gt;</code>
--

The maximum number of instructions is mentioned by the programmer that is roughly estimated.

A counter is maintained to count the execution steps and at the end of each step its contents is compared with the maximum number, the equality stops the execution of simulation program with an appropriate error message.

### 2.3.8 SETTING OF PROGRAM COUNTER

When the system is reset by enabling the RESET line of the 8085 microprocessor the program counter is automatically forced to set with the initial address 0000H. But occasionally a segment of the program starting from any other

address may be required to be simulated.

The first command sets the program counter during the simulation process with address-1 when address-2 will be the content of the program counter. This command facilitate to make the program jump externally from one address to another predefined address.

The second command sets the program counter value to the specified address immediately when the command is encountered i.e. before the simulation starts. The commands are as follows:

Command format:

```
*hSEThAT-<addr.1> hWITHh<addr.-2>
*hSEThPC-<address>
```

### 2.3.9 SETTING OF REGISTER CONTENTS

If user wants to set the registers with initial values he can do so by using the following command. In the simulation program the default value of all the registers have been set to zero. The general purpose registers B,C,D,E,H,L, and the accumulator (A) are included in the command. The register parameters are separated from one another by a blank space. Any number of registers may be set with initial values.

Command format:

```
*hSEThREGISTER- {A=<data>|h| B=<data>|h| C<data>|h|
D=<data>|h| E=<data>|h| H=<data>|h| L =<data>}
```

### 2.3.10 START AND END OF SIMULATION

After setting the initial conditions the microprocessor program execution may be initiated by using the

Command format :

\*/EXEC\$

command. The execution starts from the memory address pointed by the contents of the simulated program counter and continues until breakpoint address is encountered. The control may then be directed to execute the next command set.

Command set consists of a number of commands ending with "EXEC".

The execution of the simulation programme is ended and the program control is returned to the mainframe supervisor only when the following command is given

\*/END\$

### 2.4 EXAMPLE OF COMMAND SET

1. \* SET PC-20C2
2. \* STOP AT BREAKPOINTS-2OFF
3. \* PRINT SINGLE STEPPING\$
4. \* EXEC\$

Command set-1

5. \* SET REGISTER-A=2D,B=54
6. \* PDUMP-20C2 TO 20FF
7. \* END\$

] Command set-2

Command-1 : Set the program counter to 20C2.

Command-2 : Store the breakpoint 20FF in its command table

Command-3 : Flag byte is set to indicate single step printing at the end of each instruction.

Command-4 : Starts the execution of microprocessor instruction

Command-5 : Set the accumulator to 2D and register B to 54.

Command-6 : Dump the memory from the location 20C2 to 20FF.

Command-7 : Stops simulation.

## CHAPTER - III

### MICROPROCESSOR MODEL ON MAINFRAME

#### 3.1 ANALYSIS OF THE PROBLEM AND PROGRAMMING LANGUAGE SELECTION

The analysis of the problem is based on the capabilities of the mainframe and the requirements of the microprocessor to meet up all the necessities of simulation function.

In recent years there has been exceptional reliance on the capabilities of large-scale, high speed IBM Computer system. These systems have rapidly developed to the point where their speed, storage capacity and logical power can provide us a large extent of facilities of memory and registers.

In order for modeling of Intel 8085 microprocessor on the IBM-370-115, the mainframe requires to have all the 8085 architectures, such as control units, ALU, Memory, registers, flag bits etc. in the same manner as the microprocessor have. But the mainframe is not supposed to have all the hardware supports of the 8085 microprocessor because it is designed to function in its own methodology. For example IBM mainframe uses the registers, each one has the minimum length of 4 bytes ( a Full Word) , although half operation is possible , whereas the 8085 microprocessor has the register length of one byte.

Even more the important is the fact that though the Intel 8085 microprocessor is a byte oriented machine it includes some instructions that require bit manipulations such as STC, RAL, RAR, RLC, RRC and logical operation AND, OR and Exclusive-OR (all these instructions have been shown in Appendix - A ). The solution of these problems are discussed in details in the subsequent sections of this Chapter.

As regarding the selection of the programming language for simulation purpose in light of the above problems it seems to be very difficult or almost impossible to solve these by using the available high level languages. But Assembly offer most suitable advantages for the simulation work, Because it is the symbolic representation of machine language and all kinds of manipulations are poss-

ible with the help of this language. As such considering all these aspects, IBM Assembly has been selected as the programming language for the development of simulation module of the Intel 8085 microprocessor system.

### 3.2 THE MICROPROCESSOR ARCHITECTURE AND ITS IMAGE ON THE MAINFRAME

Before going to the solution of the problem we should focus in brief on the Intel 8085 microprocessor architecture on the modeling point of view. The Figure 3-1 shows the functional block diagram of this microprocessor. The architecture of the microprocessor can be divided into three main sections:

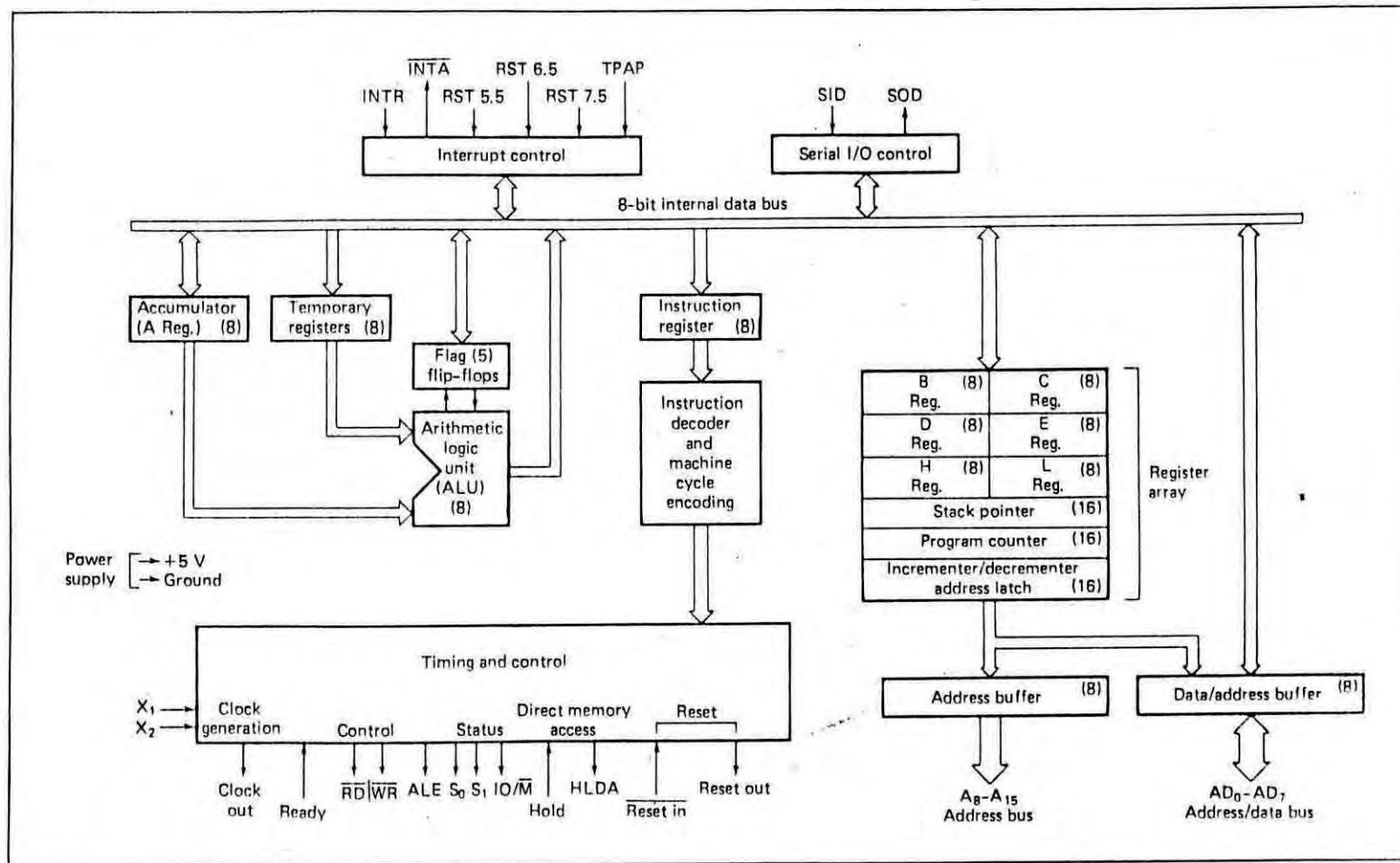
1. Register Section
2. Arithmetic and Logic Unit (ALU )
3. Control unit.

#### 3.2.1 REGISTER STRUCTURES AND THEIR IMAGES

The 8085 microprocessor uses both 8-bit and 16-bit registers. It has eight addressable 8-bit registers. Six of these can be used as 8-bit registers or 16 bit register pairs. In addition , the 8085 contains two more 16-bit registers. The 8085's registers are as follows:

- i) The accumulator(A) is an 8-bit register and has its usual meaning. Most arithmetic and Logic operations are performed using the accumulator. All

Figure 3.1 8085 Microprocessor Functional Block Diagram





I/O data transfers between the 8085 and the I/O devices are performed via the accumulator. Also there are a number of instructions that move data between accumulator and memory. The accumulator is simulated on the mainframe by allocating 1 byte of the mainframe memory and its location is addressed by MFA (Microprocessor Accumulator on Mainframe).

- ii) The General purpose registers B,C,D,E,H and L are each 8-bit long and may be used as six 8-bit or as three 16-bit registers depending on the instruction being executed. Each of these 8-bit registers can be incremented or decremented by a single instruction. There are a number of instructions which combine two of these 8-bit registers to form 16-bit register pairs as follows:

A	and	PSW
B	and	C
D	and	E
H	and	L
high-order byte		low-order byte

The 16-bit register pair obtained by combining the accumulator and the program status word (PSW) is used only for stack operations. Ari-

ithmetic operations use B and C, or D and E or H and L as 16-bit data registers.

The HL register pair (called a data pointer by Intel 8085) can be used for address pointing. This is the implied or register indirect addressing mode. There are a number of instructions, such as Mov reg,M and Mov M, reg, which move data between any register and memory location addressed by HL. A few instructions use the BC and DE register pairs as address pointers but normally they are used as general purpose data registers.

The simulations of the general purpose registers may be accomplished in the same manner as that of accumulator by allocating 1 byte of mainframe memory for each of this 8-bit registers. The simulated general purpose registers designated as follows:

<u>Microprocessor</u>	<u>Mainframe</u>
B	MFB
C	MFC
D	MFD
E	MFE
H	MFH
L	MFL

(iii) The Program Status Word (PSW) consists of five flags. These flags are used by conditional jump, call and return from subroutine instructions. The figure 3-2 represents these five flags.

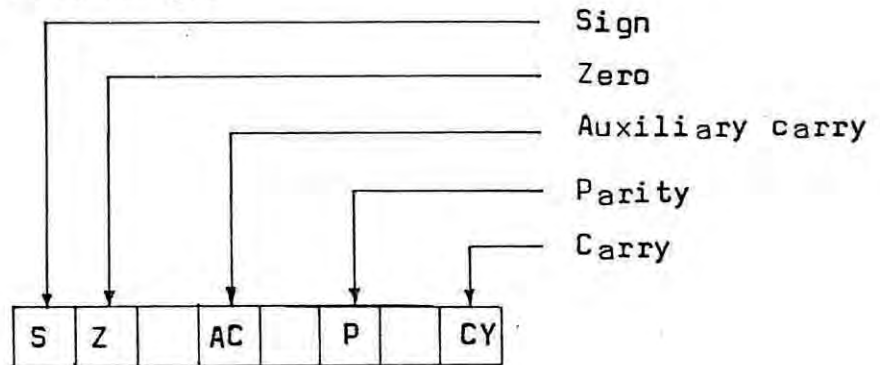


Fig. 3-2 8085 flags.

The carry flag(CY) is set or reset by arithmetic operations. Its status is then may be tested by program instructions if required . In subtraction the CY flag acts as a "borrow"flag, indicating the minuend is less than the subtrahend if the flag is set.

The zero flag(Z) is set if the result of certain instructions is zero. The zero flag is cleared if the result is not zero.

The sign flag(S) is set to the condition of the most significant bit of the accumulator following the execution of arithmetic

or logical instructions. These instructions use the MSB of data to represent the sign of the number contained in the accumulator. A set sign flag represents a negative number. Whereas a reset flag means a positive number.

The auxiliary carry flag(AC) indicating carryout of the bit 3 of the accumulator. This flag is commonly used in BCD (binary-coded-decimal) arithmetic.

The parity flag(P) tests for the number of 1s in the accumulator. If the accumulator holds an even number of 1s, it is said that even parity exists and the parity flag is set to 1. However, if the accumulator holds an odd number of 1s (called odd parity), the parity flag on the 8085 is reset to 0.

Dealing with each flag bit in simulating the PSW would require a number of mainframe instructions, the using of which are cumbersome and time consuming. But only two instructions require PSW register for:

- i) Storing the flag bits on the stack in one byte(PUSH PSW).

- ii) retrieving the PSW and set the flag bits according to the contents of the PSW (POP PSW).

All other instructions handle each of the flag bits independently. Therefore, for ease of manipulation, each flag bit of the PSW is represented by 1 byte of mainframe memory. They are designated as follows:

<u>Microprocessor</u>	<u>Mainframe</u>
CY	MFCY
Z	MFZ
S	MFS
AC	MFAC
P	MFP

One byte of the mainframe memory is also reserved to represent the PSW of the microprocessor and designated as MFPSW.

- iv) The Stack Pointer (SP) is 16 bits long. All stack operations with the 8085 use 16-bit register pairs. The stack pointer contains the address of the last data byte written into the stack. It is decremented by 2 each time 2 bytes of data are written or pushed onto the stack and is incremented by 2 each time 2 bytes of data are read from or pulled off the stack, that is, the top of the

stack has the lowest address in the stack that grows downward. It occupies two bytes on the mainframe memory(MFSP).

- v) Program Counter(PC): The program counter Contains the address of the instruction or operational(OP) Code. The program counter usually points to the next instruction location, that is, it normally contains the address of the next instruction to be executed. On the mainframe memory the two bytes are reserved for the program counter and address as MFPC.

### 3.2.2 ARITHMETIC LOGIC UNIT AND ITS IMAGE:

The ALU performs all the data manipulations, such as arithmetic and logic operations, inside the microprocessor. The size of ALU conforms to the word length of the microprocessor. This means that an 8-bit microprocessor will have a 8-bit ALU. Typically, the ALU performs the following functions:

- 1) Binary addition and logic operations
- 2) Finding 1's complement of data
- 3) Shifting or rotating the contents of the accumulator 1-bit to the left or right through carry.

The arithmetic and logic instructions of IBM-370/115 simulate the functions of the ALU of the 8085 microprocessor.

### 3.2.3. CONTROL UNIT AND ITS IMAGE:

The main purpose of this microprocessor section is to read and decode instructions from the program memory. In order to execute an instruction, the control unit steps through the appropriate blocks of the ALU based on the opcodes contained in the instruction register. The opcodes define the operations to be performed by the control unit in order to execute an instruction. The control unit interprets the contents of the instruction register and then responds to the instruction by generating a sequence of enable signals. These signals activate the appropriate ALU logic block to perform the required operation.

In general, the control section fetches and decodes instruction from memory and generates all the necessary control signals for the registers and ALU in order to execute them.

The instruction fetching and decoding functions of the control unit can be simulated by a control program on the mainframe. The main functions of the control program are:

- i) to fetch an instruction from the memory location addressed by the simulated microprocessor program counter (MFPC).

- ii) to decode the opcode and calculate the address of the program section which will perform the functions of the specified instruction.
- iii) to update the MFPC with the next instruction location of the mainframe.
- iv) to execute the required instruction.

The development of the control program is discussed in detail in the Chapter-4.

#### 3.2.4 INTEL 8085 MEMORY AND ITS IMAGE:

Program steps (instructions) and data must be stored in the memory and recalled at the appropriate time in order for the computer to perform its function. Intel 8085 is 8-bit microprocessor, that is, it uses 8-bit word. It has the 16-bit address bus. This provides a maximum of  $2^{16} = 65,536$  memory.

64 Kilo bytes of mainframe memory is reserved to represent microprocessor memory. The starting address is designated as MICROMEM.

#### 3.3 SUMMARY

The 8085 microprocessor image on the mainframe as discussed above may be summarized in the following Figure 3-3.



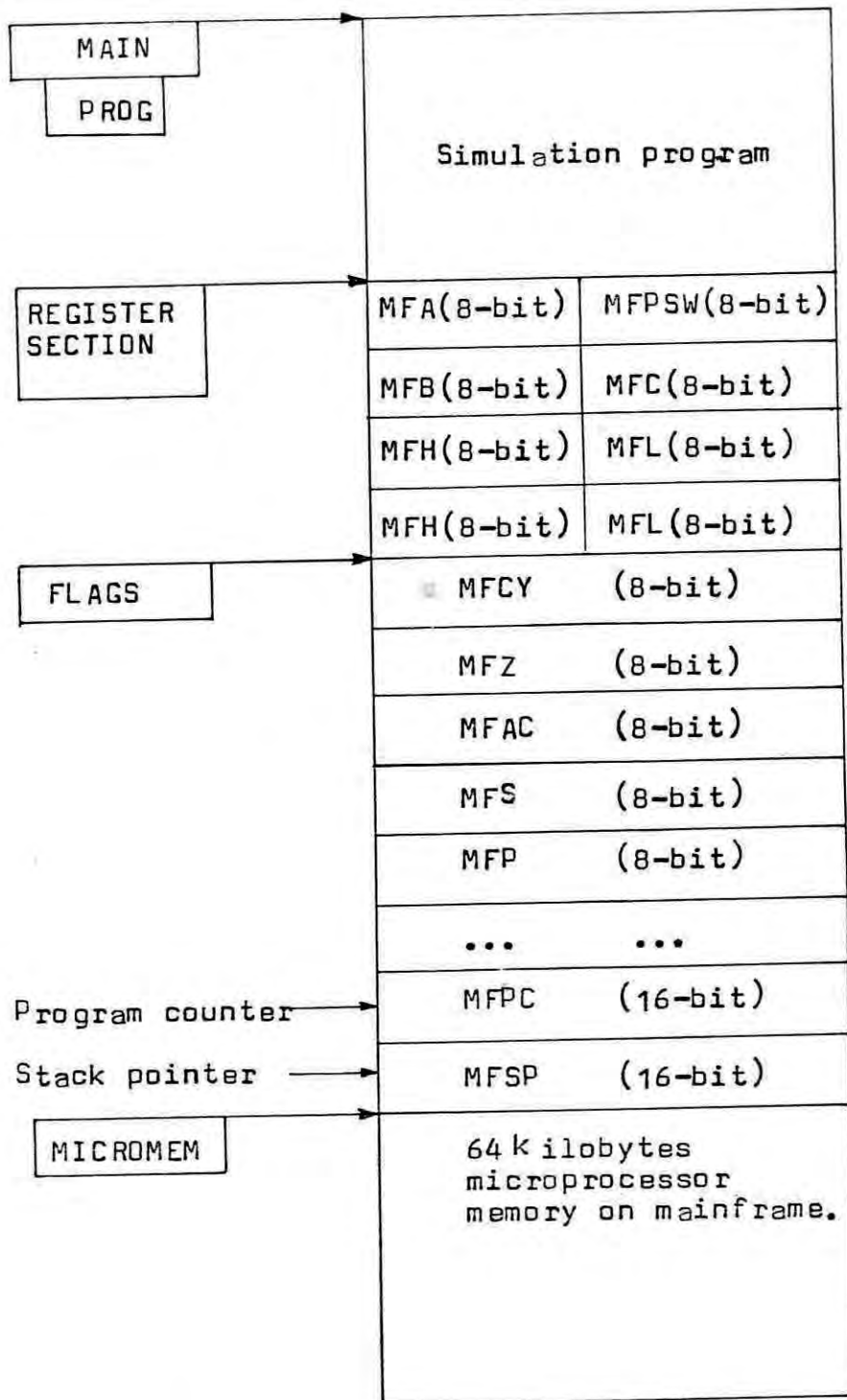


Fig. 3-3 8085 Registers and memory picture on mainframe.

## CHAPTER -IV

### DEVELOPMENT OF SIMULATION PROGRAM

#### 4.1 INTRODUCTION

The technique and procedure adopted in the development of simulation program have been described in this chapter. The simulation program may be grouped into the following four primary sections:

1. Command processor
2. Control program
3. Debugger
4. Error handler

The combination of these program units represents the simulation program of the Intel 8085 microprocessor system on IBM-370/115 mainframe. The transition of program control



In order to start the execution of simulation module the program control is first received by the command processor. The commands mentioned by the user in the command set are processed in sequential order until an "EXEC". This Command transfers the program control to the control program where each of the microprocessor instructions is processed. On receiving the control from the command processor, the control program starts fetching, decoding, and executing the microprocessor instruction and at the end of each instruction the control is directed to the Debugger. In the debugger unit, the command parameters stored in the command tables are utilized accordingly.

After debugging is over the control is transferred to the control program to process the next instruction if no breakpoint condition is met. When the breakpoint condition is encountered the program control is directed to the command processor to process the next command set. If it is an "END" command, the execution of simulation program is stopped and the control passes over to the mainframe supervisor.

If an error occurs in any of the program

units, the control is directed to the error handler to end the simulation program with an appropriate error message for the user.

## 4.2 COMMAND PROCESSOR

The command processor program processes the commands designed in the simulation program. This unit may be divided into two sections as follows:

- i. Command recognizer
- ii. Command validity checker & executor

### 4.2.1 COMMAND RECOGNIZER

The command recognizer uses a special method for identification of a command. It first counts the number of characters in a command until command separator(-) or command terminator (\$) is reached, which is used to point the address of the command validity checker. The command address table (CMNADTAB) in the program consists of the addresses of all the sections of the command validity checkers. The number of command characters counted must remain in between 0 and 20 otherwise the control passes over to the error handler.

The counting of command characters does not authenticate the validity of the command. Therefore the validity of the individual command is checked in the corresponding command validity checker. An invalid command causes to branch to the error handler.

#### 4.2.2 COMMAND VALIDITY CHECKER & EXECUTOR

It executes the valid command identified by the command recognizer. All these commands have been described in chapter-2. The execution of commands 1 to 10 (table 2.1) sets the command table with the corresponding command parameters. The parameters of the command table are utilized during the processing of the debugger unit. On the otherhand the execution of the rest of the commands immediately sets the registers and memory of the microprocessor.

At the end of each command execution, the program control jumps to the command recognizer. Except is the case of the commands EXEC and END. The EXEC command transfers the control to the control program for executing the microprocessor instruction whereas "END" stops the execution of simulation program. The flowchart of the programs to execute each of these commands are shown in Appendix-B from Fig.B-1 to B-16.

#### 4.3 CONTROL PROGRAM

The control unit of the microprocessor is simulated by the control program. The Control program fetches, decodes and executes a microprocessor instruction and also updates the program counter. The functions of the control program is summarized in the Fig. 4.3

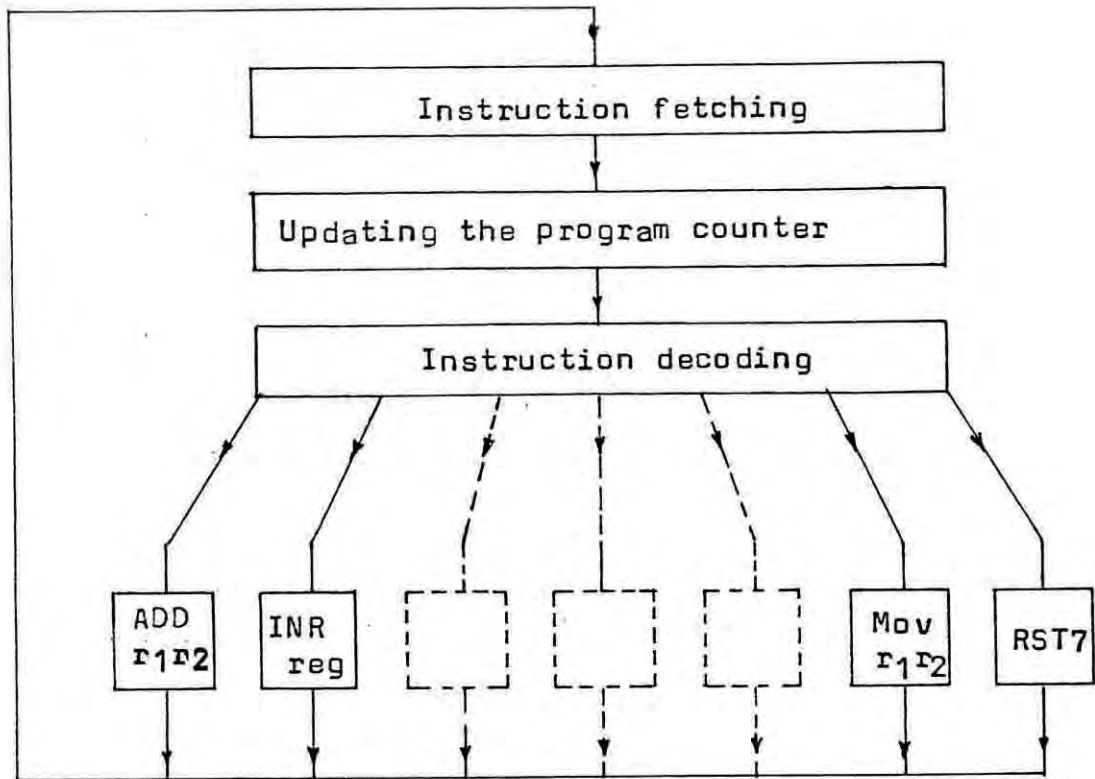


Fig. 4.2 Control program showing its different functions.

Upon receiving the control from the command processor the control unit starts to fetch the microprocessor instruction from the microprocessor designated mainframe memory location addressed by the program counter (MFPC). The instruction is fetched from the memory address, computed by adding the contents of the program counter (MFPC) to the address of the MICROMEM (starting address of the microprocessor designated on the mainframe memory). The program counter is updated through incrementing its contents before executing each microprocessor instruction.

The instruction sets of the 8085 microprocessor are given in the Appendix-C. The 8-bit machine language opcode is in between 00H to FFH excluding 0BH, 10H, 18H, 28H, 38H, CBH, DDH, EDH and FDH. We utilize the advantages of sequential order of the opcode. In order to perform the decoding function, an instruction address table is established using A-type address constant. This table contains the address of the program segment of all the instructions, each one is 4 bytes long in order to satisfy full-word (32-bit) boundary condition. We might think of the table as consisting of a series of full-word addresses, the address of each of which is 4 higher than that of its predecessor. The starting address of the table is designated by ADNOP. The figure 4.4 represents an example of such a table.

ADNOP	Address of NOP (32-bit) (NOP instruction executing segment)	opcode 00
	Address of LXIB (32-bit) (LXIB instruction executing setment)	opcode 01
	⋮	
	Address of RST7 (32-bit) (RST7 instruction executing setment)	opcode FF

Fig. 4.3 Address table of the instruction program segment.



The decoding of an instruction is accomplished through multiplying the opcode value by 4 and result is added to the address of ADNOP(initial address of the above table) in order to obtain the addresses of the program segment of the corresponding instruction from the table. Having the address located in the address table the control jumps to execute the corresponding program segment.

The execution function of the control program is simulated by executing a program segment. When an instruction (program segment) is executed the control passes over to the Debugger unit. The flow diagram of all the program segments representing the execution of instructions of the 8085 microprocessor are shown in the Appendix-B from Figure B-18 to B-88.

#### 4.4 DEBUGGER

The name implies the function of the section. This unit is intended to help the user in debugging and diagnosis of the microprocessor program. It may be recalled that the commands furnished in the table 2.1 are categorized in the two groups. At this point we are concerned with the parameter values set by the command of the first group. Comparing the MFPC contents with appropriate address parameter (flag) of the command used in the command set, the debugger may perform different functions. A parameter value from a command table is checked by the debugger only if a command

was used to set any of the parameter values by that command.

This is accomplished by checking a flag status for each command, which is set if that command was executed. The command table parameters are actually utilized in this program section where the debugger can provide the print-out of the microprocessor register contents using single stepping or breakpoints and memory dump. The loading, transferring, checking, executing and displaying etc. of program can also be accomplished according to the commands mentioned by the user. The Figure B-17 (Appendix-B) shows a flowchart representing the program section of the debugger unit. This unit transfers the control to the control program if no breakpoint condition is met. If breakpoint is found the control passes over to the command processor to process the next command set.

#### 4.5 ERROR HANDLER

The error handler deals with the various sorts of errors. As discussed earlier (section 4.1) if an error occurs in any program unit the control jumps to error handler. It is important to notice that the error handler stops the simulation program and returns the control to the mainframe supervisor. According to the number of program units, the program segments representing the error handler may be classified into two categories as follows:

- i. Command processor error routine
- ii. Control program error routine

The errors and the messages are shown in the simulation program in Appendix-A.

## CHAPTER - V

### SIMULATION PROGRAM TEST RUN

The simulation function of the developed program has been tested by simulating an 8085 microprocessor program on the IBM-370/115 mainframe with different command sets. The test problem has been designed so as to include all the possible instruction groups. The program of this problem controls the hardware circuit connected to the ports of the microcomputer depending on the data from input ports. The exact definition of the problem is given below:

#### 5.1 TEST PROBLEM

The LEDs connected to bit 0 of port 0 and port 1 depend on the input conditions set by switches on bit 1 of

port 0 and port 1, as shown in Figure 5.1. The I/O conditions are as follows:

1. If the input to bit 1 of port 0 is HIGH and input to bit 1 of port 1 is LOW, then the LED connected to port 0 will be on and the LED connected to port 1 will be off.

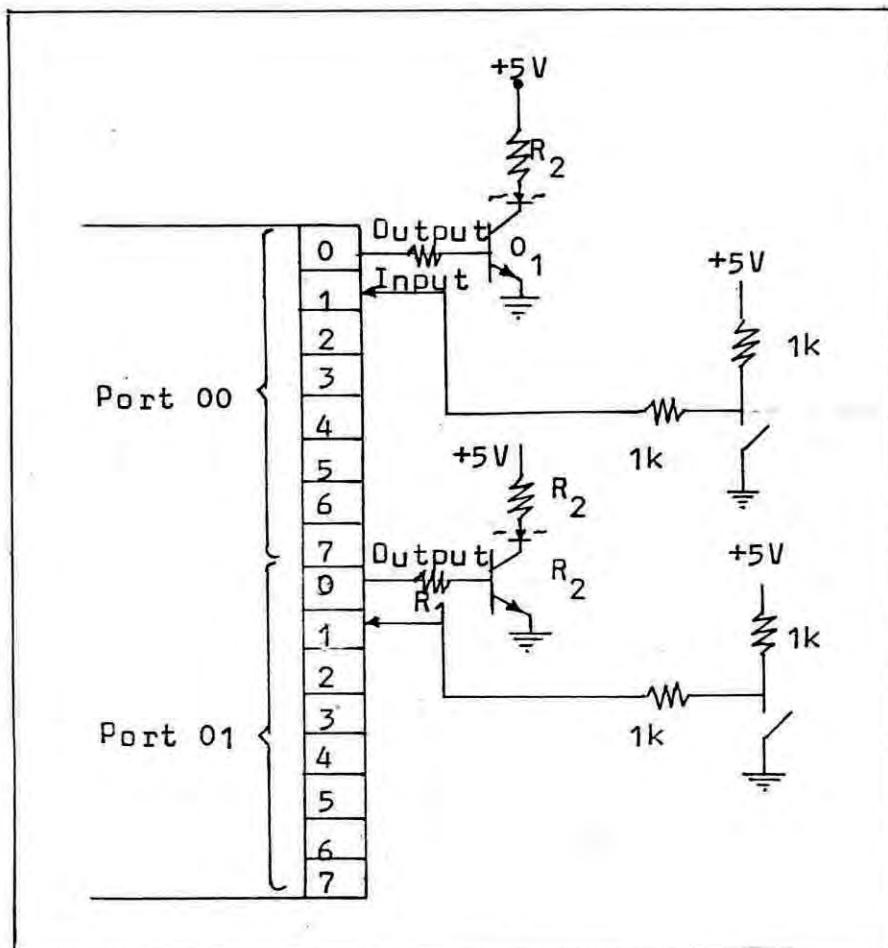


Fig. 5.1 Circuit diagram of the test program.

2. If the input to bit 1 of port 0 is LOW and that of port 1 is HIGH, then the LED of port 0 will be off and that of port 1 will be on.
3. If the bit 1 inputs of both ports 0 and 1 are the same (either both HIGH or both LOW) then LEDs of ports 0 and 1 will be on:
  - (a) Flowchart the problem
  - (b) Convert the flowchart to an 8085 program and execute it.

## 5.2 PROGRAM FLOWCHART AND LISTING

A port when used for input operation all the bits are set up as input. The same port when used for output operation all the bits are configured as output. The flowcharts of the test program are shown in Fig. 5.2.

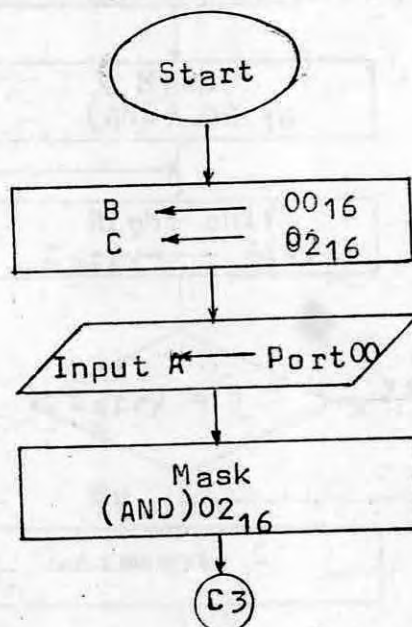


Fig. 5.2 Flowchart for test program

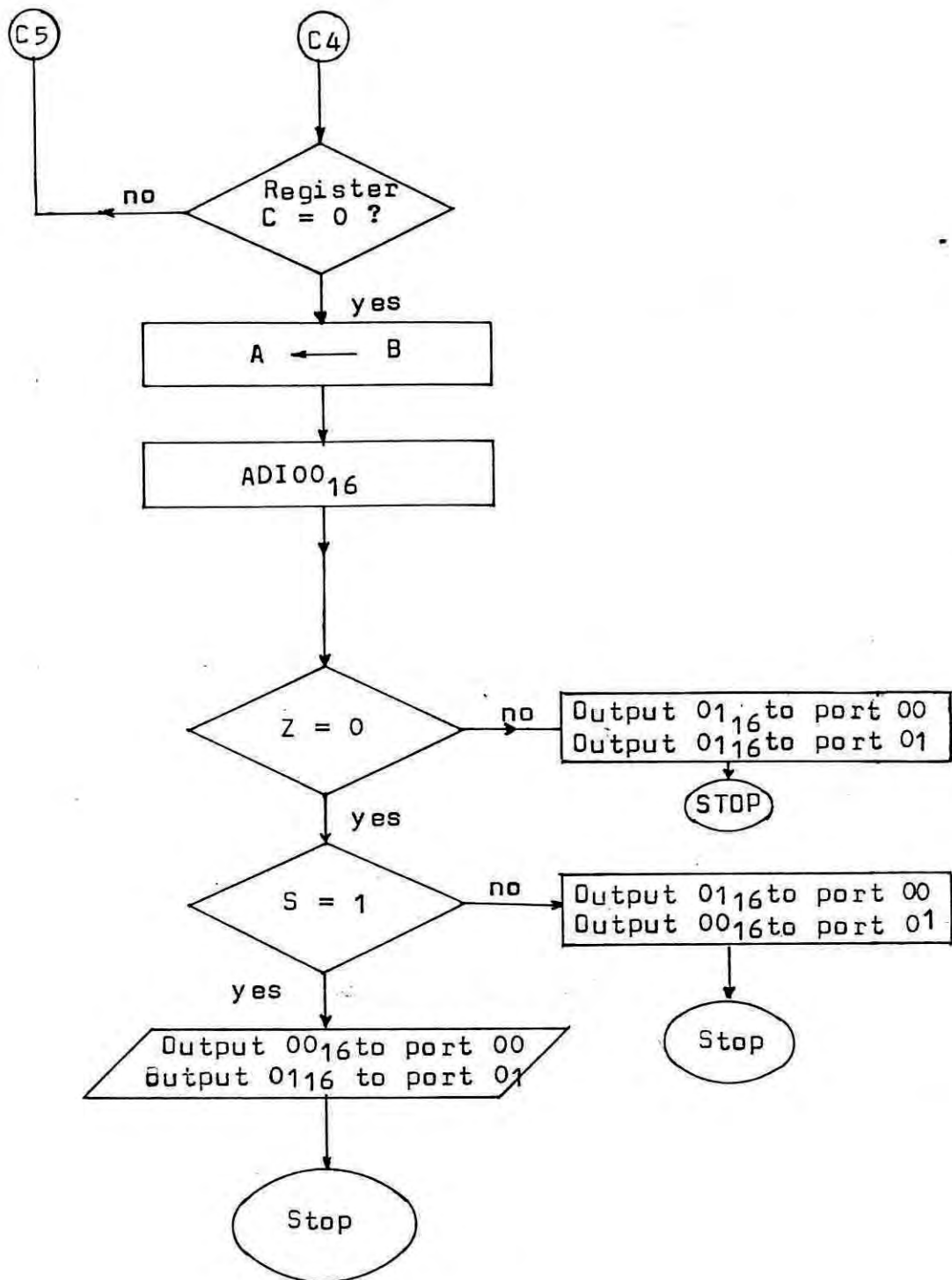


Fig. 5.2 Cont.

The flowchart can be translated into an 8085 program, the listing of which is presented below:

ASSM 2000

2000 0E00	0010 MVI B,0
2002 0E02	0020 MVI C,2
2004 DB02	0030 IN 0
2006 5602	0040 ANI 2
2008 1F	0050 RRGCK RAR
2009 D20D20	0060 JNC PRCK
200C 04	0070 INR B
200D 0D	0080 PRCK DCR C
200E C20820	0090 JNZ RECK
2011 0E02	0100 MVI C,2
2013 DB01	0110 IN 1
2015 5602	0111 ANI 2
2017 1F	0120 ZERCK RAR
2018 D21C20	0125 JNC CARCK
201B 05	0127 DCR B
201C 0D	0130 CARCK DCR C
201D C21720	0140 JNZ ZERCK
2020 76	0150 MOV A,B
2021 C600	0160 ADI 0
2023 CA3220	0170 JZ EDUCK
2026 FA3820	0180 JM MINCK
2029 3E01	0190 MVI A,1
202B D302	0200 OUT 2
202D 3E00	0210 MVI A,0
202F D301	0220 OUT 1
2031 76	0230 HLT
2032 3E01	0240 EDUCK MVI A,1
2034 D302	0250 OUT 0
2035 D301	0260 OUT 1
2038 76	0270 HLT
2039 3E00	0280 MINCK MVI A,0
203B D302	0290 OUT 0
203D 3E01	0300 MVI A,1
203F D301	0310 OUT 1
2041 76	0320 HLT



### 5.3 TEST RESULTS

The test results are obtained by simulating the 8085 program using Diskette as input port and line printer as output port. The selection of commands and command sets are so as to exhibit the necessary simulation results of the different conditions assigned in the test problem. Each of the command sets are shown along with the corresponding output results. Since the problem consists of three conditions, we obtain three individual sets of output results each of which is separately discussed as follows:

Case-1 Table 5.1 shows the output results when the input to bit 1 of port 00 is HIGH(F2 HEX) and input to bit 1 of port 1 LDW(C0 HEX) . The commands necessary to load the program and data for the above conditions and for the simulation purpose are shown in the command set. The output results show that the data transferring from accumulator to the output port 0 is 01 (turning on the LED) and that of the output port 1 is 00(turning off the LED). This satisfies the output conditions to be set for the given input conditions and verifies the correctness of the program segment to perform the function.

Case-2 The results of second case are shown in the Table 5.2. The problem condition is just opposite to the case-1. The data transferring from accumulator to the

output ports show the desired results.

Case-3 Table 5.3 shows the output results when the input to bit 1 of both port is either HIGH or LOW. The bit 1 of both ports should be such as to turn on the LEDS connected to both ports.

The test results of Table 5.3 reveal the expected outputs and proves that the program segment is error free.

Observing the input data of the two ports and the test results of the above three cases we can conclude that we have obtained the simulated output which were exactly expected.

## 5.5 ERROR TESTS

Case-1 As we have discussed earlier(Chapter-4), error handler routines have been developed in the simulation program to trap the illegal conditions. On detection of an error in the program the error handler stops the simulation program with appropriate error message for the user. To show the function of the error handler an intentional error is introduced by punching (DF)H in place of(DD)H at memory location 200A in the test program, which is indicated in the result table (5.4). This error is trapped by the error handler.

Table 5.1 Test Results of Case-1

\* CP110MS-IN=DSKT,CUT=LFRINT

\* IN-CC F2 11

\* IN-C1 C0 11

\* SET FC-2000

\* ENTER FROM-2000 06 00 0E 02 DB 00 E6 03 1F D2 C0 20 04 0D C2 C8 20 0E 02 C8 01  
 E6 02 1F D2 1C 20 05 0D C2 17 20 78 C6 00 CA 32 2C FA 39 2C 3E C1 03 00 3E 00 02  
 01 7E 3E 01 C3 00 D3 01 76 3E 0C D3 00 3E 01 D3 01 76 \$\$\$

\* PRINT SINGLE STEP FROM-2000 TILL 2020

\* STOP AT BREAKPOINTS-202F 1111

\* PRINT AT BREAKPOINTS-2021 2029 2120 1111

\* EXEC

PC=2000-06 00 0E 02 DB 00 E6 03	A=C0 B=00 C=00 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2002-0E 02 DB 00 E6 03 1F D2	A=C0 B=00 C=02 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2004-0B 00 E6 02 1F D2 00 20	A=F2 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2006-E6 C2 1F D2 00 20 04 0D	A=C2 B=00 C=02 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
FC=2008-1F C2 0D 20 04 0D C2 7E	A=C1 B=00 C=02 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
FC=2009-D2 00 20 04 0D C2 08 20	A=C1 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
FC=200E-00 C2 08 20 0E 02 00 01	A=C1 B=00 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=200E-C2 08 20 0E 02 08 01 E6	A=C1 B=00 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2008-1F C2 0D 20 04 0D 2 7E	A=00 B=00 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
FC=2009-D2 00 20 04 0D C2 08 20	A=C0 B=C0 C=11 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
FC=200C-04 00 C2 08 20 0E 02 08	A=C0 B=01 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
PC=200D-0D C2 08 20 0E 02 08 01	A=C0 B=01 C=00 D=C0 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
FC=200E-C2 08 20 0E 02 08 01 E6	A=C0 B=01 C=02 C=C0 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
FC=2011-0E 02 C8 01 E6 02 1F D2	A=C0 B=01 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=1	SP=C000-80 94 DA 91
PC=2013-0E 01 E6 02 1F D2 1C 20	A=C0 B=01 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
PC=2015-E6 02 1F D2 1C 20 05 0D	A=C0 B=01 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
PC=2017-1F D2 1C 20 05 0D C2 17	A=C0 B=01 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=C000-80 94 DA 91
FC=2018-D2 1C 20 05 0D C2 17 20	A=C0 B=01 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=201C-00 C2 17 20 78 C6 00 CA	A=C0 B=01 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=C CY=0	SP=C000-80 94 DA 91
FC=201D-C2 17 20 78 C6 00 CA 32	A=C0 B=01 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2017-1F D2 1C 20 05 0D C2 17	A=C0 B=01 C=01 D=C0 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
PC=2018-D2 1C 20 05 0D C2 17 20	A=C0 B=01 C=00 D=C0 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
PC=201C-00 C2 17 20 78 C6 00 CA	A=C0 B=C1 C=00 D=C0 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=201D-C2 17 20 78 C6 00 CA 32	A=C1 B=01 C=00 D=C0 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=C000-80 94 DA 91
FC=2020-78 C6 00 CA 32 20 FA 39			

PC=2021 C6 00 CA 32 20 FA 39 2C  
 SP=0001 80 94 DA 91 04 C0 2A 47  
 REGISTERS A=C1 B=C1 C=C0 D=00 E=00 H=00 L=00  
 FLAGS S=C Z=C AC=C P=0 CY=C

PC=2120 3E 01 D3 00 3E C0 03 01  
 SP=0001 80 94 DA 91 04 C0 2A 47  
 REGISTERS A=C1 B=C1 C=C0 D=00 E=00 H=C0 L=00  
 FLAGS S=C Z=C AC=C P=0 CY=C

8085 ACCUMULATOR CONTENT OUTPUTTING FOR PORT (0) IS C1 ← Turns LED on

PC=2020 3E 00 D3 01 76 3E 01 D3  
 SP=0000 80 94 CA 91 04 C0 2A 47  
 REGISTERS A=C0 B=C1 C=00 L=00 E=00 H=00 L=C0  
 FLAGS S=C Z=C AC=C P=0 CY=C

8085 ACCUMULATOR CONTENT OUTPUTTING FOR PORT 01 IS C0 ← Turns LED off

Table 5.2 Test results of Case- 2

```
* CFICNS-IN=DSKT,OUT=LPRINT
* IP-C0 F0 11
* IP-C1 C2 11
* SET FC-2000
* PRINT SINGLE STEP FROM-2000 TILL 2020
* STOP AT BREAKPOINTS-200F 1111
* PRINT AT BREAKPOINTS-2021 2029 202D 1111
* EXEC
```

FC=2000-06 00 0E J2 DB 00 E6 12	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2002-0E 02 0B J0 EC 02 1F 02	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2004-00 00 0E 02 1F 02 00 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2006-E6 02 1F 02 0D 20 04 0D	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2008-1F 02 0D 20 04 0D 03 0E	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2009-02 0D 20 04 0D 02 08 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2010-00 C2 09 20 0E J2 DB 11	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2011-E-C2 08 20 1E 02 DB 01 1C	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2013-1F 02 0D 20 04 0D 02 0E	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2014-02 0D 20 04 0D 02 08 20	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2015-00 C2 09 20 0E J2 DB 11	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2016-E-C2 08 20 0E 02 DB 01 1E	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2017-1E 02 0D 11 04 02 1F 02	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2018-00 01 E6 12 1F 02 1C 10	A=02 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2019-E6 02 1F 02 1C 10 05 0D	A=C2 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2017-1F 02 1C 20 05 0D 02 17	A=C1 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2018-02 1C 20 05 0D 02 17 20	A=C1 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=201C-00 C2 17 20 78 C6 00 CA	A=C1 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=201C-C2 17 20 78 C6 00 CA 32	A=C1 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=CCCC-80 94 DA 91
FC=2017-1F 02 1C 20 05 0D 02 17	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=1	SP=CCCC-80 94 DA 91
FC=2018-02 1C 20 05 0D 02 17 20	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=1	SP=CCCC-80 94 DA 91
FC=201E-05 0D C2 17 20 78 C6 00	A=00 B=FF C=01 D=00 E=00 H=00 L=00	S=1 Z=0 AC=1 P=1 CY=1	SP=CCCC-80 94 DA 91
FC=201C-00 C2 17 20 78 C6 00 CA	A=00 B=FF C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=1 CY=1	SP=CCCC-80 94 DA 91
FC=201D-C2 17 20 78 C6 00 CA 32	A=00 B=FF C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=1 CY=1	SP=CCCC-80 94 DA 91
FC=2020-78 C6 00 CA 32 20 FA 4F	A=FF B=FF C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=1 CY=1	SP=CCCC-80 94 DA 91

```
PC=2021 C6 00 CA 32 20 FA 4F 20
SP=0100 E0 94 DA 91 04 C0 2A 47
REGISTERS A=FF B=FF C=CC D=00 E=00 H=00 L=00
FLAGS S=1 Z=C AC=C P=1 CY=0
```

```
PC=2029 3E 00 D3 00 3E 01 D3 01
SP=0100 E0 94 DA 91 04 C0 2A 47
REGISTERS A=00 B=FF C=CC D=00 E=00 H=00 L=00
FLAGS S=1 Z=C AC=C P=1 CY=0
```

EC05 ACCUMULATOR CONTENT OUTPUTTING FOR PORT 00 IS 00 ← Turns LED off

```
PC=202D 3E 01 D3 01 76 11 B1 60
SP=0100 E0 94 DA 91 04 C0 2A 47
REGISTERS A=C1 B=FF C=CC D=00 E=00 H=00 L=00
FLAGS S=1 Z=C AC=C P=1 CY=0
```

EC05 ACCUMULATOR CONTENT OUTPUTTING FOR PORT 01 IS 01 ← Turns LED on

1111BREAKPOINT OCCURED AT-203F1111



Due to this error the instruction D2(Hex) at memory location 2009H causes to jump to the address 200F where the one byte data (08)H is treated as opcode to the simulation program. Since 8085 microprocessor does not have any opcode like(08)H, so the error handler of simulation program stopped simulation showing the error message"invalid opcode".

This error can be detected with the help of the informations provided in the error message, where we get the memory address 200FH containing the data(08)A as the location of error. If we trace back from this address we can surely find out the exact location of error.

Case-2 An error is intentionally introduced by punching (39)H in place of (32)H at memory location 2024H in the test program and the erroneous program is simulated. The simulation results tabulated in the table 5.5 shows the corresponding output. The data of the input ports depict that the LEDS connected to bit 1 of both ports should be turned on (01 Hex). But the simulation program turned off (00 Hex) the LED connected to the port 0 and turned on (01 Hex) that of port 1, which are the expected outputs of the simulation program because

Table 5.4 Results of Error test(Case-1)

```

* CPTICMS-IN=DSKT,OUT=LPRINT
* IN-00 F2 11
* IN-01 C0 11
* SET FC-2000
* ENTER FROM-2000 06 00 0E 02 0B 00 E6 02 1F 02 20 04 0D C2 08 20 0E 02 00 01
E6 02 1F 02 1C 20 15 0D C2 17 20 7B C6 01 CA 32 21 FA 39 20 3E 01 03 01 3E 00 03
01 7E 3E 01 C3 00 03 01 76 3E 00 03 00 3E 11 03 01 76 11
* PRINT SINGLE STEP FROM-2000 TILL 2020
* STEP AT BREAKPOINTS-202F 1111
* PRINT AT BREAKPOINTS-2021 2029 202D 1111
* CLRF FROM ADDR-2000 TO 2041 AT 2006 1111
* EXEC1
    
```

Wrong data punched  
Actual data → 0D (As seen in listing)

FC=2000-06 00 0E 02 0B 00 E6 02	A=00 B=00 C=00 D=00 E=00	H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-01 F4 6E 1B
FC=2002-0E 02 0B 00 E6 02 1F 02	A=C0 B=00 C=02 D=00 E=00	H=JJ L=00	S=J Z=0 AC=0 P=0 CY=C	SP=0000-01 F4 6E 1B
FC=2004-0B 00 E6 02 1F 02 0F 20	A=F2 B=0C C=02 D=0C E=C0	H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-01 F4 6E 1B

MEMORY DUMP	MA=2000	0 1 2 3 4 5 6 7	8 9 A B C D E F
	MA=2010	06 1C 0E 02 0B 00 E6 02	1F 02 0F 21 04 0D C2 08
	MA=2020	20 0E 02 0B 01 E6 02 1F	D2 1C 20 05 0D C2 17 20
	MA=2030	7B C6 00 CA 32 2C FA 39	2C 3E 01 C3 00 3E 00 C3
	MA=2040	01 7E 3E 01 D3 00 03 01	76 3E 00 03 00 3E 01 C3
		01 7E 0D A5 74 1E 0D A5	78 18 0C A5 7C 18 0D A5

Jump addr. 200F in place of 200D

Jumping here causes an invalid op code exception

without the error jumping would have occurred here.

FC=2006-E6 02 1F 02 0F 21 04 0D	A=02 B=00 C=02 D=00 E=00	H=00 L=00	S=1 Z=0 AC=1 P=0 CY=0	SP=0000-01 F4 6E 1B
FC=2008-1F 02 0F 20 04 0D 02 0E	A=01 B=00 C=02 D=00 E=00	H=00 L=00	S=0 Z=0 AC=1 P=0 CY=1	SP=0000-01 F4 6E 1B
FC=2009-02 0F 21 04 0D C2 08 20	A=C1 B=0C C=02 D=00 E=00	H=00 L=00	S=0 Z=0 AC=1 P=0 CY=0	SP=0000-01 F4 6E 1B

\*\*\*\*\*EFFECT MESSAGE- 08 IS INVALID OPCODE MA=200F\*\*\*\*\*

Table 5.5 Results of error test (Case-2)

\* OPTIONS-IN=DSKT,OUT=LPRINT

\* IN=00 DB \$\$

\* IN=01 AB \$\$

\* SET PC=2100

\* ENTER FROM=2007 06 07 0E 02 0B 11 0E 02 1F 22 0D 21 04 07 02 0B 20 0E 02 0B 01  
 E5 02 1F 02 1C 21 05 00 C2 17 20 79 06 00 CA 09 20 FA 39 20 3E 01 03 00 0E 00 03  
 01 7C 3E 01 03 00 03 01 76 3E 01 03 01 76 \$\$

Wrong data punched  
 Actual data → 22 (As seen in listing)

Jumping would have occurred here without error. Jumping here causes unexpected results

\* PRINT SINGLE STEP FROM=2000 TILL 2121

\* STOP AT BREAKPOINTS=203F \$\$\$

\* PRINT AT BREAKPOINTS=2021 2039 203D \$\$\$

\* EXEC

PC=2000-35 00 0E 02 0B 00 0E 02	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2002-3E 02 03 10 0E 02 1F 02	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2004-0B 00 05 12 1F 02 00 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2006-E6 02 1F 02 00 20 04 00	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=2008-1F 02 00 20 04 00 C2 19	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=2009-32 00 20 04 00 C2 08 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=200B-0D C2 0B 20 0E 02 00 01	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=200E-02 00 20 0E 02 00 01 0E	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2008-1F 02 00 20 04 00 C2 08	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2009-02 00 20 04 00 C2 08 20	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=200D-3D C2 09 20 0E 02 00 01	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=200E-C2 0B 20 1E 02 00 01 0E	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2011-0E 02 09 01 0E 02 1F 02	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2013-0B 01 0E 02 1F 02 1C 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=2015-E6 02 1F 02 1C 21 05 0D	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=2017-1F 02 1C 21 05 00 C2 17	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=2018-32 1C 20 05 00 C2 17 20	A=00 B=00 C=02 D=00 E=00 H=00 L=00	S=0 Z=1 AC=1 P=0 CY=0	SP=0000-00 41 40 00
PC=201C-0D C2 17 20 79 06 00 CA	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2010-C2 17 20 79 06 00 CA 19	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2017-1F 02 1C 20 05 00 C2 17	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2018-32 1C 20 05 00 C2 17 20	A=00 B=00 C=01 D=00 E=00 H=00 L=00	S=0 Z=0 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=201C-3D C2 17 20 79 06 00 CA	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=201D-C2 17 20 79 06 00 CA 19	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00
PC=2020-7B C5 00 CA 39 20 FA 39	A=00 B=00 C=00 D=00 E=00 H=00 L=00	S=0 Z=1 AC=0 P=0 CY=0	SP=0000-00 41 40 00

PC=2021 C5 00 CA 39 20 FA 39 20  
 SP=0000 00 41 40 00 09 43 34 F3  
 REGISTERS A=00 B=00 C=00 D=00 E=00 H=00 L=00  
 FLAGS S=0 Z=1 AC=0 P=0 CY=0

PC=2030 3E 00 03 00 3E 01 03 01  
 SP=0000 00 41 40 00 09 43 34 F3  
 REGISTERS A=00 B=00 C=00 D=00 E=00 H=00 L=00  
 FLAGS S=0 Z=1 AC=0 P=0 CY=0

8085 ACCUMULATOR CONTENT 0 INPUTTING FOR PORT 00 IS 00 ← Turns LED off

PC=203D 3E 01 03 01 76 F0 03 A4  
 SP=0000 00 41 40 00 09 43 34 F3  
 REGISTERS A=01 B=00 C=00 D=00 E=00 H=00 L=00  
 FLAGS S=0 Z=1 AC=0 P=0 CY=0

8085 ACCUMULATOR CONTENT 0 INPUTTING FOR PORT 01 IS 01 ← Turns LED on

\$\$\$BREAKPOINT OCCURED AT=203F\$\$\$



the instruction CA(Hex) at memory location 2023 causes to branch to the address 2039 where these outputs are desired.

This proves the ability of the simulation program to help in tracing the errors in the microprocessor program. On the otherhand the unexpected output for the microprocessor program makes it necessary for the programmer to choose the appropriate command sets from the simulation commands for debugging purpose.

Now we can detect the errors in program by applying breakpoint condition at different points to check the status of different microprocessor registers or taking the dump of microprocessor memory to see the overall status of the program and data.

The results of the above two cases reveal the fact that the errors which have been made in the test program are exactly trapped by the simulation program.

## CHAPTER - VI

### DISCUSSION AND CONCLUSION

In an environment where the prototype hardware is incomplete or totally absent and the microprocessor development system is also not available the simulation program in conjunction with the Cross-Assembler proved to be very useful tool for the software development of microprocessor system. This also paves the way for multitype microprocessor instruction simulations.

During the development phase of simulation program we have designed a number of commands which can be used to communicate with the simulation module. These commands are strong enough to help the programmer in developing the microprocessor software. With the help of the simulation program we can display or print the

contents of a register or memory location, execute programs, and debug the programs using single-stepping or breakpoint.

All the 8085 microprocessor instructions have been successfully simulated in such way, as if they are being processed in the microprocessor itself except input/output operation and interrupt control.

Input/output operation is very difficult to simulate. However, testing of the program on the actual microprocessor and input/output equipments is essential. In present work an attempt has been made to simulate the I/O operations by using the auxiliary storage of the mainframe giving a series of the expected input data for each of the input ports. Among the available storage devices the diskette and console keyboard are selected to simulate the input ports and whereas the line printer and console display to simulate output ports.

The interrupt required for reviving the microprocessor from the wait state (using HLT instruction) has been simulated using the console keyboard. The HLT instruction causes the mainframe CPU in the wait state and display a message for interrupting through the console keyboard. Interrupting is simulated by entering the vector address through keyboard. This address is addressed in the MFPC and starts the execution accordingly.

However, input/output system as well as interrupt

control can be exactly simulated using in-circuit emulator which provides one of the most sophisticated hardware debugging techniques.

The simulation program that we have developed works on the existing batch processing mode. Though we have obtained satisfactory results of the simulation performance through processing in batch mode but more efficient simulation is possible if we develop the programs using interactive mode. It will effectively reduce the time required to develop the microprocessor software.

Under interactive mode the breakpoint results can be used for instantaneous corrections of the registers, memory contents and an appropriate command set may be used for subsequent simulation work. This gives a tremendous advantage over the batch mode, where a series of trial command sets are given independent of the previous result. Another important facility is the simulation of the hardware interrupts in the interactive mode by allocating individual key for each of them. The interrupt key can be scanned at the end of simulation of each microprocessor instruction. The implementation of terminal facilities in BUET Computer Centre could open the door way of further works using interactive mode as the extension of the present work.

However, the present work provides a set of commands which proved to be the powerful aids to develop the Intel

8085 microprocessor software. The important feature of the present research is that the implementation of this simulation module will partially replace an extensive costly microprocessor development system.

APPENDIX - A  
SIMULATION PROGRAM

PROGRAM- INTEL 8085 MICROPROCESSOR SYSTEM SIMULATION  
ON IBM-370/115 MAINFRAME

PREPARED BY- MD. BASHIR UDDIN

SUPERVISED BY- DR. SYED MAHBUBUR RAHMAN  
ASSISTANT PROFESSOR  
DEPARTMENT OF COMPUTER ENGINEERING  
BUET, DHAKA

=====

ABSTRACT

=====

THIS IS A SIMULATION PROGRAM OF INTEL 8085 MICROPROCESSOR SYSTEM. THE PROGRAM IS DEVELOPED FOR LOADING, CHECKING, TRANSERRING, DISPLAYING, AND EXECUTING ETC. OF MICROPROCESSOR PROGRAM ON IBM-370/115 MAINFRAME BY EXECUTING SOME COMMANDS. THE PROGRAM IS PROVIDED WITH ERROR HANDLING CAPABILITIES. THROUGH APPLICATION OF THIS PROGRAM ONE CAN DISPLAY OR PRINT THE CONTENTS OF REGISTERS, MEMORY LOCATION AND DEBUG THE MICROPROCESSOR PROGRAM USING SINGLE STEPPING OR BREAKPOINTS.

=====

PROGRAM ORGANIZATION

=====

THE SIMULATION PROGRAM CONSISTS OF A MAIN ROUTINE 'MAIN' AND NINE SUBROUTINES. THE SUBROUTINE ARE CALLED THE 'MAIN' WHENEVER REQUIRED. THE PROGRAM IS ORGANIZED AS FOLLOWS.

- 1) COMMAND PROCESSOR
    - A. COMMAND RECOGNIZER
    - B. COMMAND VALIDITY CHECKER & EXECUTOR
  - 2) CONTROL PROGRAM
  - 3) DEBUGGER
  - 4) ERROR HANDLER
- =====

XXXXXXXXXXXXXXXXXXXXXXXXX  
X COMMAND RECOGNIZER X  
XXXXXXXXXXXXXXXXXXXXXXXXX

THIS IS THE STARTING SECTION OF THE SIMULATION PROGRAM. THE COMMANDS ARE RECOGNIZED IN THIS SECTION. THE PROCESSING OF A COMMAND IS STARTED THROUGH READING A COMMAND CARD. THE PRELIMINARY IDENTIFICATION OF A COMMAND IS ACCOMPLISHED BY COUNTING THE COMMAND AND CHARACTERS. A COMMAND ADDRESS TABLE IS ESTABLISHED WHICH CONTAINS THE ADDRESSES OF ALL THE COMMAND VALIDITY CHECKER AND EXECUTOR. WE MIGHT THINK OF THE TABLE AS CONSISTING OF A SERIES OF FULL-WORD ADDRESSES, THE ADDRESS OF EACH OF WHICH IS 4 HIGHER THAN ITS PREDECESSOR. ON IDENTIFICATION OF A COMMAND THE CONTROL BRANCHES TO THE COMMAND ADDRESS TABLE TO OBTAIN THE ADDRESS OF THE COMMAND VALIDITY CHECKER AND EXECUTOR OF THAT COMMAND.

=====

MAIN	START
	PRINT NOGEN
	ENTRY MFA, MFSP, MICROMEM
DISKT	DTFCD DEVADDR=SYSRDR, EOFADDR=END, IOAREA1=INPUT, WORKA=YES
	COMOD WORKA=YES
PRINT	DTFPR DEVADDR=SYSLST, IOAREA1=OUTPUT, CONTROL=YES
	PRMOD CONTROL=YES

```

BEGIN   BALR   12,0
        USING *,12
        LA    10,INXH
        USING INXH,10
        LA    11,INXH+4092
        USING INXH+4092,11
        _A   9,INXH+8184
        USING INXH+8184,9
NEXT    GET   DISKT,PRINT
        MVC   OUTPUT(132),SPACE
        MVC   OUTPUT(80),INWORK
        CNTRL PRINT,SP,2
        PUT   PRINT
        LA    2,INWORK+4
        SR    3,3
NEXTTB  CLI   0(2),C'- '
        BC    8,MBRCH
        CLI   0(2),C'$ '
        BC    8,MBRCH
        A     3,=F'1'
        C     3,=F'20'
        BC    2,STOPA
        A     2,=F'1'
        BC    15,NEXTB
MBRCH   LA    7,CMNADTAB
        SR    5,5
        _R   5,3
        M     4,=F'4'
        AR    7,5
        L     7,0(7)
        BR    7

```

R9,R10,R11,AND R12 ARE USED AS BASE REGISTERS.

A COMMAND CARD READ AND PRINTED FOR CHECKING ITS CONTENTS. R2 LOADED WITH COMMAND ADDRESS.

COMMAND CHARRACTERS ARE COUNTED IN R3 UNTIL COMMAND SEPERATOR(-) OR COMMAND TERMINATOR(\$) ENCOUNTERED. IF COUNTER EXCEEDS MAX. COUNT, ERROR HANDLER IS TO BE PROCESSED.

COUNTER CONTENTS ARE MULTIPLIED BY 4 AND ADDED TO STARTING ADDRESS OF COMMAND ADDRESS TABLE(CMNADTAB) TO OBTAIN THE ADDRESS OF VALIDITY CHECKER AND EXECUTOR OF THAT COMMAND.

\*\*\*\*\*

```

*
*
*          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*          X  COMMAND VALIDITY CHECKER&EXECUTOR X
*          XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
*
*
*

```

```

*
* THE VALIDITY OF A COMMAND IS FIRST CHECKED AND THEREAFTER THE
* THE VA_ID COMMAND IS EXECUTED IN A PROGRAM KNOWN AS ' COMMAND
* VALIDITY CHECKER AN EXECUTOR' THE INVALID COMMAND LEADS THE CONT-
* RO_ TO THE ERRJR HANDLER. AT THE END OF PROCESSING OF EACH COMMAND
* THE CONTROL JUMPS TO THE COMMAND RECONIZER. EXECPT IS THE CASE OF
* THE COMMAND 'EXEC' AND 'END'. THE 'EXEC' COMMAND CAUSES TO BRANCH
* TO CONTROL PROGRAM WHERE ' ND' COMMAND STOPS SIMULATION.
*
*

```

=====

```

OPTION  CLC   INWORK(10),=C'* OPTIONS-'
        BC    7,STOPA
        LA    8,INWORK+10
        CLC   0(3,8),=C'IN='
        BC    8,JPTB
        CLC   0(4,8),=C'OUT='
        BC    7,STOPC
        A     8,=F'4'
        CLC   0(6,8),=C'LPRINT'
        BC    8,JPTA
        CLC   0(6,8),=C'CONSOL'
        BC    7,STOPC
        MVC   DUTCN(4),=C'CNSL'
        B     NEXT
OPTA    MVC   OUTPT(4),=C'PRNT'
        A     8,=F'7'
        CLC   0(6,8),=C'CONSOL'

```

THIS COMMAND CHECKS FOR I/O SIMULATION, WHETHER INPUT OR OUTPUT OR BOTH TO BE SIMULATED OR NOT. THE FLAG 'DSKT' IS SET IF INPUT PORT IS SIMULATED THROUGH DISKETTE & 'KBRD' IS SET IF INPUT IS SIMULATED THROUGH CONSOLE KEYBOARD. THE FLAG 'PRNT' IS SET IF OUTPUT PORT IS SIMULATED ON LINE PRINTER AND 'CNSL' IF IT IS SIMULATED UPON CONSOLE. THE END OF PROCESSING OF COMMAND CAUSES TO BRANCH TO 'NEXT' TO PROCESS THE NEXT COMMAND.





BEXEC	CLC	INWORK(7),=C* EXEC\$*	
	BC	7,STOPA	SIMULATION IS STARTED BY EXECUTING
	MVC	ADRES,=F'0'	'EXEC' COMMAND. INSTRUCTION ADDRESS
	LA	6,MICROMEM	IS LOCATED BY ADDING PROGRAM COUNTER
	LA	7,ADNOP	CONTENTS WITH STARTING ADDRESS(MICR-
	MVC	ADRES+2(2),MFPC	OMEM) OF MICROPROCESSOR MEMORY ON
	A	6,ADRES	MAINFRAME. OPCODE VALUE IS MULTIPLI-
	MVC	MEPC(2),MFPC	ED BY 4 AND ADDED TO STARTING ADDR-
	L	5,=F'0'	SS(ADNOP) OF INSTRUCTION ADDRESS
	MVC	KLT+3(1),0(6)	TABLE TO OBTAIN THE ADDRESS OF THE
	-	5,KLT	PROGRAM SEGMENT OF THAT INSTRUCTION.
	M	4,=F'4'	THE CONTROL JUMPS TO THE EVALUATED
	AR	7,5	ADDRESS TO SIMULATE THE INSTRUCTION.
	L	7,0(7)	
	L	5,=F'1'	
	AH	5,MFPC	
	STH	5,MFPC	
	BR	7	
B>DMP	CLC	INWORK(8),=C* PDUMP-	
	BC	7,STOPA	TO OBTAIN MEMORY DUMP, ADDRESS ARE
	LA	8,INWORK+8	CONVERTED FROM EBCDIC TO BINARY AND
	LA	3,MICROMEM	ADDED TO STARTING ADDRESS(MICROMEM)
	MVC	ADRES,=F'0'	OF MICROPROCESSOR MEMORY ON MAINFR-
	MVC	JK(4),0(8)	AME. R2 LOADED WITH DUMP START ADDR-
	TR	JK(4),TTAB-X'C1'	ESS AND R3 LOADED WITH DUMP END ADDR-
	MVC	JKL(5),JK	ESS AND BRANCHES TO SUBROUTINE
	PACK	JKT(3),JKL	(SUBDMP) FOR PRINTING MICROPROCESSOR
	CLC	JKT(2),=X'FFFF'	MEMORY DUMP. RETURNING FROM SUBROUT-
	BC	2,STOPL	INE BRANCHES TO PROCESS NEXT COMMAND
	NI	JKT+1,X'F0'	
	MVC	DPCA(2),JKT	
	A	8,=F'4'	
	CLC	0(4,8),=C' TO '	
	BC	7,STOPA	
	A	8,=F'4'	
	MVC	JK(4),0(8)	
	TR	JK(4),TTAB-X'C1'	
	MVC	JKL(5),JK	
	PACK	JKT(3),JKL	
	CLC	JKT(2),=X'FFFF'	
	BC	2,STOPL	
	DI	JKT+1,X'0F'	
	MVC	DPCB(2),JKT	
	LR	2,3	
	MVC	ADRES+2(2),DPCA	
	A	2,ADRES	
	MVC	ADRES+2(2),DPCB	
	A	3,ADRES	
	LA	13,SAVE	
	L	15,=V(SUBDMP)	
	BALR	14,15	
	B	NEXT	
B>SET	CLC	INWORK(9),=C* SET PC-	
	BC	7,STOPA	THE PROGRAM COUNTER (MFPC) IS SET BY
	MVC	JK(4),INWORK+9	EXECUTING THIS PROGRAM SEGMENT. THE
	TR	JK(4),TTAB-X'C1'	ADDRESS IS CONVERTED TO BINARY AND
	MVC	JKL(5),JK	MOVED TO 'MFPC' TO SET WITH GIVEN
	PACK	JKT(3),JKL	ADDRESS.
	CLC	JKT(2),=X'FFFF'	
	BC	2,STOPL	
	MVC	MFPC(2),JKT	
	B	NEXT	
BENTRT	C_C	INWORK(11),=C* ENTER TO-	
	BC	7,STOPA	TO LOAD THE MICROPROCESSOR PROGRAM/
	L	4,=F'19'	DATA IN THE COMMAND TABLE, ADDRESS
	LA	8,INWORK+11	CONVERTED TO BINARY AND STORED IN

```

MVC JK(4),0(8)
TR JK(4),TTAB-X'C1'
MVC JKL(5),JK
PACK JKT(3),JKL
CLC JKT(2),=X'FFFF'
BC 2,STOPL
MVC ENTAD(2),JKT
A 8,=F'4'
CLC 0(4,8),=C' AT '
BC 7,STOPA
A 8,=F'4'
MVC JK(4),0(8)
TR JK(4),0(8)
TR JK(4),TTAB-X'C1'
MVC JKL(5),JK
PACK JKT(3),JKL
CLC JKT(2),=X'FFFF'
BC 2,STOPL
MVC ATADR(2),JKT
A 8,=F'5'
L 5,=A(MICROMEM+65536)
ESTP1 MVC TPK(2),0(8)
CLC TPK(2),=C'$$'
BE ESTP4
CLI TPK,C' '
BC 8,STOPD
TR TPK(2),TTAB-X'C1'
MVC PTL(3),TPK
PACK TPK(2),PTL
CLI TPK,X'FF'
BC 2,STOPB
MVC 0(1,5),TPK
A 5,=F'1'
A 8,=F'3'
BCT 4,ESTP1
L 2,=F'19'
ESTP2 GET DISKT,INWORK
MVC OUTPUT(132),SPACE
MVC OUTPUT(80),INWORK
PUT PRINT
LA 8,INWORK
L 3,=F'27'
ESTP3 MVC TPK(2),0(8)
CLI TPK,C' '
BC 8,STOPD
CLC TPK(2),=C'$$'
BE ESTP4
TR TPK(2),TTAB-X'C1'
MVC PTL(3),TPK
PACK TPK(2),PTL
CLI TPK,X'FF'
BC 2,STOPB
MVC 0(1,5),TPK
A 2,=F'1'
C 2,=F'65536'
BC 2,STOPE
A 5,=F'1'
A 8,=F'3'
BCT 3,ESTP3
B ESTP2
ESTP4 STH 2,DATCNT
B NEXT
BPSSET CLC INWORK(12),=C'* SET PC AT-'
BC 7,STOPA
-A 8,INWORK+12
MVC JK(4),0(8)

```

COMMAND TABLE. TWO BYTES OF EBCDIC PROGRAM/DATA CODE CONVERTED TO 1 BYTE OF BINARY CODE AND STORED IN THE COMMAND TABLE EACH TIME AND THIS IS CONTINUED TILL THE FIRST CARD HAS BEEN PROCESSED. THE PROCESSING OF NEXT CARDS STARTS FROM 'ESTP2' SHOWN BELOW.

THE SECOND AND ONWARD PROGRAM/DATA CARDS ARE PROCESSED HERE. AFTER PROCESSING IS OVER THE CONTROL BRANCHES TO 'NEXT' TO PROCESS NEXT COMMAND.

EXECUTION OF THIS SEGMENT SETS THE COMMAND TABLE WITH FLAG AND COMMAND PARAMETERS TO SET PROGRAM COUNTER.

```

TR      JK(4),TTAB-X'C1'
MVC     JKL(5),JK
PACK    JKT(3),JKL
CLC     JKT(2),=X'FFFF'
BC      2,STOPL
MVC     ATPC(2),JKT
A       8,=F'4'
C_C     0(6,8),=C' WITH '
BC      7,STOPA
A       8,=F'6'
MVC     JK(4),0(8)
TR      JK(4),TTAB-X'C1'
MVC     JKL(5),JK
PACK    JKT(3),JKL
C_C     JKT(2),=X'FFFF'
BC      2,STOPL
MVC     WITHPC(2),JKT
B       NEXT
BENTRF  CLC     INWORK(13),=C'* ENTER FROM-'
BC      7,STOPA
LA      3,MICROMEM
MVC     ADRES,=F'0'
-       4,=F'21'
-A      8,INWORK+13
MVC     JK(4),0(8)
TR      JK(4),TTAB-X'C1'
MVC     JKL(5),JK
PACK    JKT(3),JKL
C_C     JKT(2),=X'FFFF'
BC      2,STOPL
MVC     ENTADF(2),JKT
MVC     ADRES+2(2),ENTADF
A       3,ADRES
A       8,=F'5'
EPNT1   MVC     TPK(2),0(8)
CLC     TPK(2),=C'$$'
BE      EPNT4
TR      TPK(2),TTAB-X'C1'
MVC     PTL(3),TPK
PACK    TPK(2),PTL
CLI     TPK,X'FF'
BC      2,STOPB
MVC     0(1,3),TPK
A       3,=F'1'
A       8,=F'3'
BCT     4,EPNT1
L       5,=F'21'
EPNT2   GET     DISKT,INWORK
MVC     OUTPUT(132),SPACE
MVC     OUTPUT(80),INWO K
PUT     PRINT
LA      8,INWORK
L       4,=F'27'
EPNT3   MVC     TPK(2),0(8)
CLC     TPK(2),=C'$$'
BE      EPNT4
TR      TPK(2),TTAB-X'C1'
MVC     PTL(3),TPK
PACK    TPK(2),PTL
CLI     TPK,X'FF'
BC      2,STOPB
MVC     0(1,3),TPK
A       5,=F'1'
C       5,=F'65536'
BC      2,STOPE
A       3,=F'1'

```

\*MFPC\* WHEN A PARTICULAR ADDRESS IS REACHED. ALL ADDRESSES ARE CONVERTED TO BINARY AND STORED IN THE COMMAND TABLE.

EXECUTION OF THIS PROGRAM SEGMENT LOADS PROGRAM/DATA DIRECTLY TO MICROPROCESSOR MEMORY DESIGNATED ON MAIN FRAME. THE LOADING ADDRESS IS COMPUTED BY ADDING THE GIVEN ADDRESS TO THE STARTING ADDRESS(MICROMEM) OF THE MICROPROCESSOR MEMORY ON MAINFRAME. TWO BYTES OF EBCDIC PROGRAM/DATA CODE CONVERTED TO 1 BYTE OF BINARY CODE AND STORED IN THE MEMORY EACH TIME AND CONTINUED TILL THE FIRST CARD HAS BEEN PROCESSED. THEN PROCESSING OF NEXT CARDS STARTS FROM \*EPNT2\* SHOWN BELOW.

THE SECOND AND ONWARD PROGRAM/DATA CARDS ARE PROCESSED HERE. AFTER PROCESSING IS OVER,THE CONTROL BRANCHES TO PROCESS THE NEXT COMMAND.

	A	8,=F'3'	
	BCT	4,EPNT3	
	B	EPNT2	
EPNT4	B	NEXT	
PRSTPEP	CLC	INWORK(20),=C'* PRINT SINGLE STEPPINGS'	
	BC	7,STOPA	FOR SINGLE STEP PRINTING A FLAG'PSS'
	MVC	SSCNT(3),=C'PSS'	IS SET IN THE COMMAND TABLE.
	B	NEXT	
BDJMP	CLC	INWORK(17),=C'* DUMP FROM ADDR--'	
	BC	7,STOPA	ALL THE ADDRESSES IN THIS TYPE OF
	L	4,=F'0'	DUMP COMMAND ARE CONVERTED TO BINARY
	LA	5,DMP TAB	AND STORED IN THE COMMAND TABLE. THE
	_A	8,INWORK+17	JUMP START ADDRESS IS LOADED IN R2
	MVC	JK(4),0(8)	AND DUMP END ADDRESS IN R3. THEN
	TR	JK(4),TTAB-X'C1'	BRANCHES TO A SUBROUTINE(SUBDMP) FOR
	MVC	JKL(5),JK	PRINTING MICROPROCESSOR MEMORY DUMP.
	PACK	JKT(3),JKL	
	CLC	JKT(2),=X'FFFF'	
	BC	2,STOPL	
	NI	JKT+1,X'F0'	
	MVC	DMPSTART(2),JKT	
	A	8,=F'4'	
	CLC	0(4,8),=C' TO '	
	BC	7,STOPA	
	A	8,=F'4'	
	MVC	JK(4),0(8)	
	TR	JK(4),TTAB-X'C1'	
	MVC	JKL(5),JK	
	PACK	JKT(3),JKL	
	CLC	JKT(2),=X'FFFF'	
	BC	2,STOPL	
	DI	JKT+1,X'0F'	
	MVC	DMPEND(2),JKT	
	A	8,=F'4'	
	CLC	0(4,8),=C' AT '	
	BC	7,STOPA	
	A	8,=F'4'	
DSTP1	MVC	JK(4),0(8)	
	CLC	JK(4),=C'\$\$\$\$'	
	BE	DSTP2	
	TR	JK(4),TTAB-X'C1'	
	MVC	JKL(5),JK	
	PACK	JKT(3),JKL	
	CLC	JKT(2),=X'FFFF'	
	BC	2,STOPL	
	MVC	0(2,5),JKT	
	A	5,=F'2'	
	A	8,=F'5'	
	A	4,=F'1'	
	C	4,=F'9'	
	BC	4,DSTP1	
DSTP2	STH	4,DMPCNT	
	B	NEXT	
MXINST	CLC	INWORK(18),=C'* MAX INSTRUCTION--'	
	BC	7,STOPA	THE GIVEN MAXIMUM INSTRUCTION NUMBER
	LA	8,INWORK+18	IS CONVERTED TO BINARY AND STORED IN
	MVC	MX,=D'0'	THE COMMAND TABLE.
	PACK	MX+5(3),0(4,8)	
	CVB	4,MX	
	ST	4,MAXNO	
	B	NEXT	
STBP	CLC	INWORK(22),=C'* STOP AT BREAKPOINTS--'	
	BC	7,STOPA	THE BREAKPOINT ADDRESSES IN THE COM-
	L	4,=F'0'	MAND ARE CONVERTED TO BINARY AND
	LA	5,SBP TAB	STORED IN THE COMMAND TABLE.
	LA	8,INWORK+22	

```

SSTP1  MVC      JK(4),0(8)
        CLC      JK(4),=C'$$$$'
        BE       SSTP2
        TR       JK(4),TTAB-X'C1'
        MVC      JKL(5),JK
        PACK     JKT(3),JKL
        CLC      JKT(2),=X'FFFF'
        BC       2,STOPL
        MVC      0(2,5),JKT
        A        5,=F'2'
        A        8,=F'5'
        A        4,=F'1'
        C        4,=F'11'
        BC       4,SSTP1
SSTP2  STH      4,SBPCNT
        B        NEXT
PRNTB  CLC      INWRK(23),=C'* PRINT AT BREAKPOINTS-'
        BC       7,STOPA
        L        4,=F'0'
        LA      5,PRNTTAB
        LA      8,INWRK+23
        MVC      JK(4),0(8)
        CLC      JK(4),=C'$$$$'
        BE       PSTP2
        TR       JK(4),TTAB-X'C1'
        MVC      JKL(5),JK
        PACK     JKT(3),JKL
        CLC      JKT(2),=X'FFFF'
        BC       2,STOPL
        MVC      0(2,5),JKT
        A        5,=F'2'
        A        8,=F'5'
        A        4,=F'1'
        C        4,=F'11'
        BC       4,PSTP1
PSTP1  STH      4,SBPCNT
        B        NEXT
PSTP2  CLC      INWRK(25),=C'* PRINT SINGLE STEP FROM-'
        BC       7,STOPA
        L        4,=F'0'
        LA      8,INWRK+25
        MVC      JK(4),0(8)
        TR       JK(4),TTAB-X'C1'
        MVC      JK(5),JK
        PACK     JKT(3),JKL
        CLC      JKT(2),=X'FFFF'
        BC       2,STOPL
        MVC      PRNTADR(2),JKT
        A        8,=F'4'
        CLC      0(6,8),=C' TILL '
        BC       7,STOPA
        A        8,=F'6'
        MVC      JK(4),0(8)
        TR       JK(4),TTAB-X'C1'
        MVC      JKL(5),JK
        PACK     JKT(3),JKL
        CLC      JKT(2),=X'FFFF'
        BC       2,STOPL
        MVC      TILADR(2),JKT
        MVC      SPSNT(3),=C'SSP'
        B        NEXT

```

```

*****
*
*
*
*
XXXXXXXXXXXXXXXXXXXXX
X CONTROL PROGRAM X
*
*
*
*

```



	MVC	MFB(1),HOUS+1	
	B	INCRM	
DCRB	MVC	HOUS(2),=H'0'	LOAD R4 WITH CONTENTS OF REGISTER
	MVC	ADD(2),=H'0'	'MFB' AND DECREASE ITS CONTENTS BY 1
	L	4,=F'0'	THROUGH HALF-WORD OPERATION. BRANCH
	MVC	ADD+1(1),MFB	TO A COMMON ROUTINE 'DECRM' TO SET
	LH	4,ADD	DIFFERENT FLAGS.
	SH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFB(1),HOUS+1	
	B	DECRM	
MVIB	A	6,=F'1'	MOVE IMMEDIATE DATA TO REGISTER
	MVC	MFB(1),0(6)	'MFB' AND UPDATE PROGRAM COUNTER.
	A	6,=F'1'	'MFPC'.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
R_C	MVC	ADD,=H'0'	LOAD R4 WITH ACCUMULATOR (MFA) CONT-
	L	4,=F'0'	ENTS AND ROTATE ONE POSITION TO THE
	MVC	ADD+1(1),MFA	LEFT, BY MOVING ITS MOST SIGNIFICANT
	LH	4,ADD	BIT OF 'MFA' INTO ITS LEAST SIGNIFI-
	SL	4,1	CANT BIT AND CARRY FLAG(MFCY) IS SET
	STH	4,ADD	BY MSB.
	MVC	RA(1),ADD	
	MVC	MFCY(1),RA	
	OC	ADD+1(1),RA	
	MVC	MFA(1),ADD+1	
	A	6,=F'1'	
	B	XEXT	
DADB	MVC	AD(4),=F'0'	MOVE THE CONTENTS OF REGISTER PAIR
	MVC	AD+2(2),MFB	'MFB MFC' TO A FULL-WORD MEMORY LOC-
	B	DADCOM	ATION ADDRESSED BY 'AD' AND BRANCH
*			TO A COMMON ROUTINE 'DADCOM' TO PRO-
*			CESS THE REST OPERATIONS.
LDAXB	MVC	ADRES,=F'0'	ADD REGISTER PAIR (MFB MFC) CONTENTS
	MVC	ADRES+2(2),MFB	TO STARTING ADDRESS (MICROMEM) OF
	LA	8,MICROMEM	MICROPROCESSOR DESIGNATED MAINFRAME
	A	8,ADRES	MEMORY TO LOCATE THE ADDRESS FROM
	MVC	MFA(1),0(8)	WHERE ACCUMULATOR IS TO BE LOADED.
	A	6,=F'1'	
	B	XEXT	
DCXB	MVC	ADD,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	L	4,=F'0'	TER PAIR 'MFB MFC' AND DECREASE ITS
	MVC	ADD(2),MFB	CONTENTS BY 1 THROUGH HALF-WORD
	LH	4,ADD	OPERATION.
	SH	4,=H'1'	
	STH	4,ADD	
	MVC	MFB(2),ADD	
	A	6,=F'1'	
	B	XEXT	
INRC	MVC	HOUS,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD(2),=H'0'	TER 'MFC' AND INCREASE ITS CONTENTS
	L	4,=F'0'	BY 1 THROUGH HALF-WORD OPERATION.
	MVC	ADD+1(1),MFC	BRANCH TO A COMMON ROUTINE 'INCRM'
	LH	4,ADD	TO SET DIFFERENT FLAGS.
	AH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFC(1),HOUS+1	
	B	INCRM	
DCRC	MVC	HOUS(2),=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD(2),=H'0'	TER 'MFC' AND BRANCH TO A COMMON
	L	4,=F'0'	ROUTINE 'DECRM' TO SET DIFFERENT
	MVC	ADD+1(1),MFC	FLAGS.
	LH	4,ADD	
	SH	4,=H'1'	



	STH	4,HOUS	
	MVC	MFC(1),HOUS+1	
	B	DECRM	
MVIC	A	6,=F'1'	MOVE IMMEDIATE DATA TO REGISTER
	MVC	MFC(1),0(6)	'MFC' AND UPDATE PROGRAM COUNTER.
	A	6,=F'1'	
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
RRC	MVC	ADD,=H'0'	LOAD R4 WITH ACCUMULATOR CONTENTS
	L	4,=F'0'	AND ROTATE ONE POSITION TO THE RIGHT
	MVC	ADD+1(1),MFA	AND TRANSFERRING THE LEAST SIGNIFICANT
	MVC	RA(1),MFA	BIT OF 'MFA' TO ITS MOST SIGNIFICANT
	NI	RA,X'01'	BIT. CARRY FLAG(MFCY) IS SET
	MVC	ADD(1),RA	BY LEAST SIGNIFICANT BIT.
	LH	4,ADD	
	SRL	4,1	
	STH	4,ADD	
	MVC	MFA(1),ADD+1	
	MVC	MFCY(1),RA	
	A	6,=F'1'	
	B	XEXT	
LXID	A	6,=F'1'	MOVE LOW ORDER IMMEDIATE DATA TO REG
	MVC	MFE(1),0(6)	ISTER 'MFE' AND HIGHER ORDER DATA TO
	A	6,=F'1'	REGISTER 'MFD'. UPDATE PROGRAM COUN-
	MVC	MFD(1),0(6)	TER(MFPC) BY INCREASING ITS PRESENT
	A	6,=F'1'	CONTENTS.
	L	4,=F'2'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
STAXD	MVC	ADRES,=F'0'	ADD REGISTER (MFD MFE) CONTENTS TO
	MVC	ADRES+2(2),MFD	STARTING ADDRESS (MICROMEM) OF MICR-
	LA	8,MICROMEM	PROCESSOR DESIGNATED MAINFRAME MEMORY
	A	8,ADRES	TO LOCATE THE ADDRESS WHERE ACCUMUL-
	MVC	0(1,8),MFA	ATOR(MFA) CONTENTS IS TO BE STORED.
	A	6,=F'1'	
	B	XEXT	
INXD	MVC	ADD,=H'0'	LOAD R4 WITH THE CONTENTS OF 'MFD'
	L	4,=F'0'	'MFE' REGISTER PAIR AND INCREASE ITS
	MVC	ADD(2),MFD	CONTENTS BY 1 THROUGH HALF-WORD OPE-
	LH	4,ADD	RATION.
	AH	4,=H'1'	
	STH	4,ADD	
	MVC	MFD(2),ADD	
	A	6,=F'1'	
	B	XEXT	
INRD	MVC	HOUS(2),=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD(2),=H'0'	TER 'MFD' AND INCREASE ITS CONTENTS
	L	4,=F'0'	BY 1 THROUGH HALF-WORD OPERATION.
	MVC	ADD+1(1),MFD	BRANCH TO A COMMON ROUTINE 'INCRM'
	_H	4,ADD	TO SET THE DIFFERENT FLAGS.
	AH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFD(1),HOUS+1	
	B	INCRM	
DCRD	MVC	HOUS,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD,=H'0'	TER 'MFD' AND DECREASE ITS CONTENTS
	L	4,=F'0'	BY 1 THROUGH HALF-WORD OPERATION.
	MVC	ADD+1(1),MFD	BRANCH TO A COMMON ROUTINE 'DECRM'
	_H	4,ADD	TO SET DIFFERENT FLAGS.
	SH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFD(1),HOUS+1	
	B	DECRM	

MVID	A	6,=F'1'	MOVE IMMEDIATE DATA TO REGISTER
	MVC	MFD(1),0(6)	'MFD' AND UPDATE THE PROGRAM COUNTER
	A	6,=F'1'	'MFPC' BY INCREASING ITS PRESENT
	-	4,=F'1'	CONTENTS.
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
RAL	MVC	ADD,=H'0'	LOAD R4 WITH ACCUMULATOR(MFA) CONTE-
	-	4,=F'0'	NENTS, ROTATE ONE POSITION TO THE LEFT
	MVC	ADD+1(1),MFA	AND SET CARRY WITH MSB AND LSB WITH
	LH	4,ADD	CARRY FLAG(MFCY).
	SLL	4,1	
	STH	4,ADD	
	MVC	RA(1),ADD	
	MVC	MFA(1),ADD+1	
	OC	MFA(1),MFCY	
	NI	RA,X'01'	
	MVC	MFCY(1),RA	
	A	6,=F'1'	
	B	XEXT	
DADD	MVC	AD,=F'0'	MOVE THE CONTENTS OF REGISTER PAIR
	MVC	AD+2(2),MFD	'MFD MFE' TO A FULL-WORD MEMORY LOC-
	B	DADCOM	ATION ADDRESSED BY 'AD' AND BRANCH
*			TO A COMMON ROUTINE 'DADCOM' TO PRO-
*			CESS THE REST OPERATIONS.
LJAXD	MVC	ADRES,=F'0'	ADD REGISTER PAIR(MFD MFE) CONTENTS
	MVC	ADRES+2(2),MFD	TO STARTING ADDRESS (MICROMEM) OF
	LA	8,MICROMEM	MICROPROCESSOR MEMORY ON MAINFRAME
	A	8,ADRES	TO LOCATE THE ADDRESS FROM WHERE
	MVC	MFA(1),0(8)	ACCUMULATOR(MFA) CONTENTS IS TO BE
	A	6,=F'1'	LOADED.
	B	XEXT	
DCXD	MVC	ADD,=H'0'	LOAD R4 WITH THE CONTENTS OF 'MFD' &
	-	4,=F'0'	'MFE' REGISTER PAIR AND DECREASE ITS
	MVC	ADD(2),MFD	CONTENTS THROUGH HALF-WORD OPERA-
	_H	4,ADD	TION.
	SH	4,=H'1'	
	STH	4,ADD	
	MVC	MFD(2),ADD	
	A	6,=F'1'	
	B	XEXT	
INRE	MVC	HOUS,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD,=H'0'	TER 'MFE' CONTENTS THROUGH HALF-WORD
	-	4,=F'0'	OPERATION. BRANCH TO A COMMON ROUTINE
	MVC	ADD+1(1),MFE	'INCRM' TO SET DIFFERENT FLAGS.
	LH	4,ADD	
	AH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFE(1),HOUS+1	
	B	INCRM	
DCRE	MVC	HOUS,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIS-
	MVC	ADD,=H'0'	TER 'MFE' AND DECREASE ITS CONTENTS
	L	4,=F'0'	BY 1 THROUGH HALF-WORD OPERATION.
	MVC	ADD+1(1),MFE	BRANCH TO A COMMON ROUTINE 'DECRM'
	_H	4,ADD	TO SET DIFFERENT FLAGS.
	SH	4,=H'1'	
	STH	4,HOUS	
	MVC	MFE(1),HOUS+1	
	B	DECRM	
MVIE	A	6,=F'1'	MOVE IMMEDIATE DATA TO REGISTER 'MFE'
	MVC	MFE(1),0(6)	AND UPDATE THE PROGRAM COUNTER(MFPC)
	A	6,=F'1'	BY INCREASING ITS PRESENT CONTENTS.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	

```

RAR      MVC      ADD,=H'0'
          L        4,=F'0'
          MVC      ADD+1(1),MFA
          MVC      RA(1),MFA
          NI       RA,X'01'
          MVC      ADD(1),MFCY
          LH       4,ADD
          SR_      4,1
          STH      4,ADD
          MVC      MFA(1),ADD+1
          MVC      MFCY(1),RA
          A        6,=F'1'
          B        XEXT
RIM      MVC      MFA(1),MASK
          A        6,=F'1'
          B        XEXT
LXIH     A        6,=F'1'
          MVC      MFL(1),0(6)
          A        6,=F'1'
          MVC      MFH(1),0(6)
          A        6,=F'1'
          L        4,=F'2'
          AH       4,MFPC
          STH      4,MFPC
          B        XEXT
S-ILD    MVC      ADRES,=F'0'
          A        6,=F'1'
          MVC      ADRES+3(1),0(6)
          A        6,=F'1'
          MVC      ADRES+2(1),0(6)
          LA       8,MICROMEM
          A        8,ADRES
          MVC      0(1,8),MFL
          A        8,=F'1'
          MVC      0(1,8),MFH
          A        6,=F'1'
          L        4,=F'2'
          AH       4,MFPC
          STH      4,MFPC
          B        XEXT
INXH     L        4,=F'0'
          MVC      ADD,=H'0'
          MVC      ADD(2),MFH
          LH       4,ADD
          AH       4,=H'1'
          STH      4,ADD
          MVC      MFH(2),ADD
          A        6,=F'1'
          B        XEXT
INRH     MVC      HJUS,=H'0'
          MVC      ADD,=F'0'
          L        4,=F'0'
          MVC      ADD+1(1),MFH
          LH       4,ADD
          AH       4,=H'1'
          STH      4,HOUS
          MVC      MFH(1),HOUS+1
          B        INCRM
DCRH     MVC      HOUS,=H'0'
          MVC      ADD,=H'0'
          L        4,=F'0'
          MVC      ADD+1(1),MFH
          _H      4,ADD
          SH       4,=H'1'
          STH      4,HOUS
          MVC      MFH(1),HOUS+1

```

LOAD R4 WITH ACCUMULATOR (MFA) CONTENTS, ROTATE ONE POSITION TO THE RIGHT. SET MSB BY CARRY FLAG CONTENT AND CARRY FLAG(MFCY) WITH LSB.

LOAD ACCUMULATOR(MFA) WITH 8 BITS OF INTERRUPT MASK DATA CONTAINED IN 'MASK'.  
MOVE LOW ORDER IMMEDIATE DATA TO REGISTER 'MFL' AND HIGH ORDER DATA TO REGISTER 'MFH'. UPDATE PROGRAM COUNTER(MFPC) BY INCREASING ITS PRESENT CONTENTS.

ADD NEXT TWO INSTRUCTION BYTES TO THE STARTING ADDRESS (MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME. REGISTER R8 CONTAINS THIS ADDRESS. STORE REGISTER 'MFL' AT LOCATION ADDRESSED BY R8. INCREASE R8 AND STORE 'MFH' AT LOCATION ADDRESSED BY R8.

LOAD R4 WITH CONTENTS OF REGISTER 'MFH MFL' REGISTER PAIR AND INCREASE ITS CONTENTS BY 1 THROUGH HALF-WORD OPERATION.

LOAD R4 WITH THE CONTENTS OF REGISTER 'MFH' AND INCREASE ITS CONTENTS BY 1 THROUGH HALF-WORD OPERATION. BRANCH TO A COMMON ROUTINE 'INCRM' TO SET DIFFERENT FLAGS.

LOAD R4 WITH THE CONTENTS OF REGISTER 'MFH' AND DECREASE ITS CONTENTS BY 1 THROUGH HALF-WORD OPERATION. BRANCH TO A COMMON ROUTINE 'DECRM' TO SET DIFFERENT FLAGS.

```

MVIH  B   DECRM
      A   6,=F'1'
      MVC MFH(1),0(6)
      A   6,=F'1'
      L   4,=F'1'
      AH  4,MFPC
      STH 4,MFPC
      B   XEXT
DAA   MVC HOUS,=H'0'
      MVC ADD,=H'0'
      MVC SUB,=H'0'
      SR  8,8
      SR  7,7
      MVC ADD+1(1),MFA
      MVC SUB+1(1),MFA
      LH  7,ADD
      NC  ADD,=X'000F'
      LH  8,ADD
      CH  8,=H'9'
      BC  13,DA1
      AH  7,=X'0006'
      MVI MFA C,X'01'
DA1   VC  SUB,=X'00F0'
      LH  8,SUB
      CH  8,=X'0090'
      BC  13,DA2
      AH  7,=X'0060'
DA2   STH 7,HOUS
      MVC MFA(1),HOUS+1
      MVC CYC,=H'0'
      L   13,SAVE
      L   15,=V(SUBFLG)
      LA  3,=A(HOUS,CYC)
      BALR 14,15
      L   5,0(3)
      MVC MFZ(1),0(5)
      L   5,04(3)
      MVC MFS(1),0(5)
      L   5,08(3)
      VC  HOUS,=X'00FF'
      LH  7,HOUS
      CH  7,=H'0'
      BC  8,DA3
      MVC MFP(1),0(5)
DA3   L   5,0(4)
      MVC MFCY(1),0(5)
      A   6,=F'1'
      B   XEXT
DADH  MVC AD,=F'0'
      MVC AD+2(2),MFH
      B   DADCOM
LHLD  MVC ADRES,=F'0'
      A   6,=F'1'
      MVC ADRES+3(1),0(6)
      A   6,=F'1'
      MVC ADRES+2(1),0(6)
      LA  8,MICROMEM
      A   8,ADRES
      MVC MFL(1),0(8)
      A   8,=F'1'
      MVC MFH(1),0(8)
      A   6,=F'1'
      L   4,=F'2'
      AH  4,MFPC
      STH 4,MFPC
      B   XEXT

```

MOVE IMMEDIATE DATA TO REGISTER 'MFH' AND UPDATE PROGRAM COUNTER 'MFPC' BY INCREASING ITS CONTENTS.

CHECK FOR LEAST SIGNIFICANT NIBBLE OF ACCUMULATOR(MFA) IS CONTENTS, WHETHER IT IS GREATER THAN 9 OR NOT IF IT EXCEEDS 9 ADD 6 TO 'MFA' CONTENTS. ALSO ADD 50H TO (MFA) IF MOST SIGNIFICANT NIBBLE OF 'MFA' CONTENTS EXCEEDS 9. BRANCH TO SUBROUTINE 'SUBFLG' TO SET THE FLAGS ZERO(MFZ), PARITY(MFP) AND CARRY(MFCY).

MOVE THE CONTENTS OF 'MFH MFL' TO FULL WORD 'AD'. BRANCH TO 'DADCOM' TO PROCESS THE REST OPERATIONS. ADD NEXT TWO INSTRUCTION BYTE TO THE STARTING ADDRESS (MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME. R8 CONTAINS THIS ADDRESS. LOAD REGISTER 'MFL' WITH CONTENTS ADDRESSED BY R8. INCREASE R8 AND LOAD 'MFH' WITH CONTENTS ADDRESSED BY R8.

DCXH	L	4,=F'0'	LOAD R4 WITH THE CONTENTS OF 'MFH'
	MVC	ADD,=H'0'	'MFL' REGISTER PAIR AND DECREASE ITS
	MVC	ADD(2),MFH	CONTENTS BY 1 THROUGH HALF-WORD OPE-
	LH	4,ADD	RATION.
	SH	4,=H'1'	
	STH	4,ADD	
	MVC	MFH(2),ADD	
	A	6,=F'1'	
	B	XEXT	
INRL	MVC	HJUS,=H'0'	LOAD R4 WITH THE CONTENTS OF REGIST-
	MVC	ADD,=H'0'	ER 'MFL' AND INCREASE ITS CONTENTS
	L	4,=F'0'	BY 1 THROUGH HALF-WORD OPERATION.
	MVC	ADD+1(1),MFL	BRANCH TO A COMMON ROUTINE 'INCRM'
	LH	4,ADD	TO SET DIFFERENT FLAGS.
	AH	4,=H'1'	
	STH	4,HJUS	
	MVC	MFL(1),HJUS+1	
	B	INCRM	
DCRL	MVC	HJUS,=H'0'	LOAD R4 WITH CONTENTS OF REGISTER
	MVC	ADD,=H'0'	'MFL' AND DECREASE ITS CONTENTS BY 1
	L	4,=F'0'	THROUGH HALF-WORD OPERATION. BRANCH
	MVC	ADD+1(1),MFL	TO A COMMON ROUTINE 'DECRM' TO SET
	LH	4,ADD	DIFFERENT FLAGS.
	SH	4,=H'1'	
	STH	4,HJUS	
	MVC	MFL(1),HJUS+1	
	B	DECRM	
MVIL	A	6,=F'1'	MOVE IMMEDIATE DATA TO REGISTER 'MFL'
	MVC	MFL(1),0(6)	AND UPDATE THE PROGRAM COUNTER
	A	6,=F'1'	'MFPC'.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
CMA	XI	MFA,X'FF'	COMPLEMENT ACCUMULATOR(MFA) CONTENTS
	A	6,=F'1'	BY LOGICAL EXCLUSIVE OR OPERATION.
	B	XEXT	
SIM	MVC	MASK(1),MFA	SET INTERRUPT MASK WITH ACCUMULATOR.
	A	6,=F'1'	(MFA) CONTENTS.
	B	XEXT	
LXISP	A	6,=F'1'	MOVE LOW ORDER IMMEDIATE DATA TO
	MVC	MFSP+1(1),0(6)	LOW ORDER POSITION OF STACKPOINTER
	A	6,=F'1'	'MFSP' AND HIGH ORDER DATA TO HIGH
	MVC	MFSP(1),0(6)	ORDER POSITION. UPDATE PROGRAM CONU-
	A	6,=F'1'	TER BY INCREASING ITS CONTENTS.
	L	4,=F'2'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
STA	MVC	ADRES,=F'0'	ADD NEXT TWO BYTE TO STARTING ADDRE-
	A	6,=F'1'	SS (MICROMEM) OF MICROPROCESSOR MEM-
	MVC	ADRES+3(1),0(6)	ORY ON MAINFRAME TO LOCATE THE ADDR-
	A	6,=F'1'	ESS, WHERE ACCUMULATOR CONTENTS IS
	MVC	ADRES+2(1),0(6)	TO BE STORED.
	LA	8,MICROMEM	
	A	8,ADRES	
	MVC	0(1,8),MFA	
	A	6,=F'1'	
	L	4,=F'2'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
INXSP	L	4,=F'0'	LOAD R4 WITH CONTENTS OF STACKPOINT-
	LH	4,MFSP	ER 'MFSP' AND INCREASE ITS CONTENTS
	AH	4,=H'1'	BY 1 THROUGH HALF-WORD OPERATION.
	STH	4,MFSP	

	A	6,=F'1'	
	B	XEXT	
INRM	MVC	ADRES,=F'0'	ADD MEMORY POINTING REGISTER PAIR
	MVC	ADD,=H'0'	(MFH MFL) CONTENTS TO STARTING ADDR-
	L	4,=F'0'	ESS(MICROMEM) OF MICROPROCESSOR MEM-
	MVC	ADRES+2(2),MFH	ORY ON MAINFRAME. LOAD R4 WITH DATA
	LA	8,MICROMEM	LOCATED AT COMPUTED ADDRESS. INCREA-
	A	8,ADRES	SE ITS CONTENTS BY 1 AND BRANCH TO A
	MVC	ADD+1(1),0(8)	COMMON ROUTINE 'INCRM' TO SET DIFFE-
	LH	4,ADD	RENT FLAGS.
	AH	4,=H'1'	
	STH	4,HOUS	
	MVC	0(1,8),HOUS+1	
	B	INCRM	
DCRM	MVC	ADRES,=F'0'	ADD MEMORY POINTING REGISTER(MFH MFL
	MVC	ADD,=H'0'	) CONTENTS TO STARTING ADDRESS(MICR-
	L	4,=F'0'	OMEM) OF MICROPROCESSOR MEMORY ON
	MVC	ADRES+2(2),MFH	MAINFRAME. LOAD R4 WITH DATA LOCATED
	LA	8,MICROMEM	AT COMPUTED ADDRESS. DECREASE ITS
	A	8,ADRES	CONTENTS BY 1.BRANCH TO A COMMON
	MVC	ADD+1(1),0(8)	ROUTINE 'DECRM' TO SET DIFFERENT
	_H	4,ADD	FLAGS.
	SH	4,=H'1'	
	STH	4,HOUS	
	MVC	0(1,8),HOUS+1	
	B	DECRM	
MVIM	MVC	ADRES,=F'0'	ADD MEMORY POINTING REGISTER PAIR
	A	6,=F'1'	(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM	ESS(MICROMEM) OF MICROPROCEESOR
	MVC	ADRES+2(2),MFH	MEMORY ON MAINFRAME. MOVE IMMEDIATE
	A	8,ADRES	DATA TO THIS COMPUTED LOCATION. UPD-
	MVC	0(1,8),0(6)	ATE THE PROGRAM COUNTER.
	A	6,=F'1'	
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
STC	MVI	MFCY,X'01'	SET THE CARRY FLAG(MFCY).
	A	6,=F'1'	
	B	XEXT	
DADSP	MVC	AD,=F'0'	MOVE STACKPOINTER(MFSP) CONTENTS TO
	MVC	AD+2(2),MFSP	A FULL-WORD MEMORY LOCATION 'AD' AND
	B	DADCOM	BRANCH TO A COMMON ROUTINE'DADCOM'TO
			PROCESS THE REST OPERATIONS.
*			ADD NEXT TWO INSTRUCTION BYTES TO
LDA	MVC	ADRES,=F'0'	STARTING ADDRESS(MICROMEM) OF MICRO-
	A	6,=F'1'	PROCESSOR MEMORY DESIGNATED ON MAIN-
	MVC	ADRES+3(1),0(6)	FRAME. MOVE THE CONTENTS LOCATED AT
	A	6,=F'1'	THE COMPUTED MEMORY ADDRESS INTO
	MVC	ADRES+2(1),0(6)	ACCUMULATOR(MFA).UPDATE PROGRAM
	LA	8,MICROMEM	COUNTER AND BRANCH TO 'DEBUGGER'.
	A	8,ADRES	
	MVC	MFA(1),0(8)	
	A	6,=F'1'	
	L	4,=F'2'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
DCXSP	L	4,=F'0'	LOAD R4 WITH THE CONTENTS OF STACK-
	LH	4,MFSP	POINTER(MFSP) AND DECREASE ITS CONT-
	SH	4,=H'1'	ENTS BY 1 THROUGH HALF-WORD OPERAT-
	STH	4,MFSP	ION.
	A	6,=F'1'	
	B	XEXT	
INRA	MVC	HJUS,=H'0'	LOAD R4 WITH CONTENTS OF REGISTER
	MVC	ADD,=H'0'	'MFA' AND INCREASE ITS CONTENTS BY 1
	L	4,=F'0'	THROUGH HALF-WORD OPERATION. BRANCH

```

MVC ADD+1(1),MFA
LH 4,ADD
AH 4,=H'1'
STH 4,HOUS
MVC MFA(1),HOUS+1
B INCRM
DCRA MVC HJUS,=H'0'
MVC ADD,=H'0'
- 4,=F'0'
MVC ADD+1(1),MFA
LH 4,ADD
SH 4,=H'1'
STH 4,HOUS
MVC MFA(1),HOUS+1
B DECRM
MVA A 6,=F'1'
MVC MFA(1),0(6)
A 6,=F'1'
L 4,=F'1'
AH 4,MFPC
STH 4,MFPC
B XEXT
CMC XI MFCY,X'01'
A 6,=F'1'
B XEXT
MOVBB MVC MFB,MFB
A 6,=F'1'
B XEXT
MOVBC MVC MFB,MFC
A 6,=F'1'
B XEXT
MOVBD MVC MFB,MFD
A 6,=F'1'
B XEXT
MOVBE MVC MFB,MFE
A 6,=F'1'
B XEXT
MOVBH MVC MFB,MFH
A 6,=F'1'
B XEXT
MOVBL MVC MFB,MFL
A 6,=F'1'
B XEXT
MOVBM MVC ADRES,=F'0'
MVC ADRES+2(2),MFH
LA 8,MICROMEM
A 8,ADRES
MVC MFB(1),0(8)
A 6,=F'1'
B XEXT
MOVBA MVC MFB,MFA
A 6,=F'1'
B XEXT
MOVCB MVC MFC,MFB
A 6,=F'1'
B XEXT
MOVCC MVC MFC,MFC
A 6,=F'1'
B XEXT
MOVCD MVC MFC,MFD
A 6,=F'1'
B XEXT
MOVCE MVC MFC,MFE
A 6,=F'1'
B XEXT
MOVCH MVC MFC,MFH

```

TO A COMMON ROUTINE 'INCRM' TO SET DIFFERENT FLAGS.

LOAD R4 WITH CONTENTS OF REGISTER 'MFA' AND DECREASE ITS CONTENTS BY 1 THROUGH HALF-WORD OPERATION. BRANCH TO A COMMON ROUTINE 'DECRM' TO SET DIFFERENT FLAGS.

MOVE IMMEDIATE DATA TO REGISTER 'MFA' AND UPDATE THE PROGRAM COUNTER 'MFPC'

COMPLEMENT CARRY FLAG THROUGH LOGICAL EXCLUSIVE OR (IMMEDIATE) OPERATION.

ADD MEMORY POINTING REGISTER PAIR (MFH MFL) CONTENTS TO STARTING ADDRESS (MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME TO LOCATE ADDRESS FROM WHERE REGISTER (MFB) IS TO LOAD.

```

A      6,=F'1'
B      XEXT
MOVCL  MVC  MFC,MFL
A      6,=F'1'
B      XEXT
MOVCM  MVC  ADRES,=F'0'
MVC  ADRES+2(2),MFH
LA     8,MICROMEM
A      8,ADRES
MVC  MFC(1),0(8)
A      6,=F'1'
B      XEXT
MOVCA  MVC  MFC,MFA
A      6,=F'1'
B      XEXT
MOVDB  MVC  MFD,MFB
A      6,=F'1'
B      XEXT
MOVDC  MVC  MFD,MFC
A      6,=F'1'
B      XEXT
MOVDD  MVC  MFD,MFD
A      6,=F'1'
B      XEXT
MOVDE  MVC  MFD,MFE
A      6,=F'1'
B      XEXT
MOV DH  MVC  MFD,MFH
A      6,=F'1'
B      XEXT
MOV DL  MVC  MFD,MFL
A      6,=F'1'
B      XEXT
MOVDM  MVC  ADRES,=F'0'
MVC  ADRES+2(2),MFH
LA     8,MICROMEM
A      8,ADRES
MVC  MFD(1),0(8)
A      6,=F'1'
B      XEXT
MOVDA  MVC  MFD,MFA
A      6,=F'1'
B      XEXT
MOVEB  MVC  MFE,MFB
A      6,=F'1'
B      XEXT
MOVEC  MVC  MFE,MFC
A      6,=F'1'
B      XEXT
MOVE D  MVC  MFE,MFD
A      6,=F'1'
B      XEXT
MOVEE  MVC  MFE,MFE
A      6,=F'1'
B      XEXT
MOVEH  MVC  MFE,MFH
A      6,=F'1'
B      XEXT
MOVE L  MVC  MFE,MFL
A      6,=F'1'
B      XEXT
MOVEM  MVC  ADRES,=F'0'
MVC  ADRES+2(.2),MFH
LA     8,MICROMEM
A      8,ADRES
MVC  MFE(1),0(8)

```

ADD MEMORY POINTING REGISTER PAIR(MFH MFL) CONTENTS TO STARTING ADDRESS (MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME TO LOCATE ADDRESS FROM WHERE REGISTER 'MFC' IS TO BE LOADED

ADD MEMORY POINTING REGISTER PAIR (MFH MFL) CONTENTS TO STARTING ADDRESS(MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME TO LOCATE ADDRESS FROM WHERE 'MFD' IS TO BE LOADED.

ADD MEMORY POINTING REGISTER PAIR (MFH MFL) CONTENTS TO STARTING ADDRESS(MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME TO LOCATE ADDRESS FROMWHERE 'MFE' IS TO BE LOADED.



```

A      6,=F'1'
B      XEXT
MOV EA MVC MFH,MFA
A      6,=F'1'
B      XEXT
MOV HB MVC MFH,MFB
A      6,=F'1'
B      XEXT
MOV HC MVC MFH,MFC
A      6,=F'1'
B      XEXT
MOV HD MVC MFH,MFD
A      6,=F'1'
B      XEXT
MOV HE MVC MFH,MFE
A      6,=F'1'
B      XEXT
MOV HH MVC MFH,MFH
A      6,=F'1'
B      XEXT
MOV HL MVC MFH,MFL
A      6,=F'1'
B      XEXT
MOV HM MVC ADRES,=F'0'
MVC ADRES+2(2),MFH
LA 8,MICROMEM
A 8,ADRES
MVC MFH(1),0(8)
A 6,=F'1'
B XEXT
MOV HA MVC MFH,MFA
A 6,=F'1'
B XEXT
MOV LB MVC MFL,MFB
A 6,=F'1'
B XEXT
MOV LC MVC MFL,MFC
A 6,=F'1'
B XEXT
MOV LD MVC MFL,MFD
A 6,=F'1'
B XEXT
MOV LE MVC MFL,MFE
A 6,=F'1'
B XEXT
MOV LH MVC MFL,MFH
A 6,=F'1'
B XEXT
MOV LL MVC MFL,MFL
A 6,=F'1'
B XEXT
MOV LM MVC ADRES,=F'0'
MVC ADRES+2(2),MFH
LA 8,MICROMEM
A 8,ADRES
MVC MF_(1),0(8)
A 6,=F'1'
B XEXT
MOV LA MVC MFL,MFA
A 6,=F'1'
B XEXT
MOV MB MVC ADRES,=F'0'
MVC ADRES+2(2),MFH
LA 8,MICROMEM
A 8,ADRES
MVC 0(1,8),MFB

```

ADD MEMORY POINTING REGISTER PAIR  
{MFH MFL} CONTENTS TO STARTING ADDR-  
ESS(MICROMEM) OF MICROPROCESSOR MEM-  
ORY ON MAINFRAME FROM WHERE 'MFH' IS  
TO BE LOADED.

ADD MEMORY POINTING REGISTER PAIR  
{MFH MFL} CONTENTS TO STARTING ADDR-  
ESS(MICROMEM) OF MICROPROCESSOR MEM-  
ORY ON MAINFRAME FROM WHERE 'MFL'  
CONTENTS IS TO BE LOADED.

ADD MEMORY POINTING REGISTER PAIR  
{MFH MFL} CONTENTS TO STARTING ADDR-  
ESS(MICROMEM) OF MICROPROCESSOR MEM-  
ORY ON MAINFRAME TO LOCATE THE ADDR-  
ESS WHERE 'MFB' CONTENTS IS TO BE

	A	6,=F'1'		
	B	XEXT		MOVED.
M3V MC	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS(MICROMEM) OF MICROPROCESSOR MEM-
	A	8,ADRES		ORY ON MAINFRAME TO LOCATE THE ADDR-
	MVC	0(1,8),MFC		ESS WHERE 'MFC' CONTENTS IS TO BE
	A	6,=F'1'		MOVED.
	B	XEXT		
M3V MD	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS (MICROMEM) OF MICROPROCESSOR
	A	8,ADRES		MEMORY ON MAINFRAME TO LOCATE THE
	MVC	0(1,8),MFD		ADDRESS WHERE 'MFD' CONTENTS IS TO
	A	6,=F'1'		BE MOVED.
	B	XEXT		
M3V ME	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS (MICROMEM) OF MICROPROCESSOR
	A	8,ADRES		MEMORY WHERE 'MFE' CONTENTS IS TO BE
	MVC	0(1,8),MFE		MOVED.
	A	6,=F'1'		
	B	XEXT		
M3V MH	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS (MICROMEM) OF MICROPROCESSOR
	A	8,ADRES		MEMORY ON MAINFRAME TO LOCATE THE
	MVC	0(1,8),MFH		ADDRESS WHERE 'MFH' CONTENTS IS TO
	A	6,=F'1'		BE MOVED.
	B	XEXT		
M3V ML	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS (MICROMEM) OF MICROPROCESSOR
	A	8,ADRES		MEMORY ON MAINFRAME TO LOCATE THE
	MVC	0(1,8),MFL		ADDRESS WHERE 'MFL' CONTENTS IS TO
	A	6,=F'1'		BE MOVED.
	B	XEXT		
HLT	LA	13,SAVE		SIMULATION OF 'HLT' INSTRUCTION CAU-
	L	15,=V(OUTPUT)		SES TO BRANCH TO SUBROUTINE(OUTPUT)
	L	3,=F'2'		TO OBTAIN THE APPROPRIATE DATA FROM
	BA LR	14,15		CONSOLE KEYBOARD. THE DATA 'ENDS'
	L	5,0(3)		ENDS SIMULATION AND DATA 'JUMP' CON-
	A	6,=F'1'		TINUES SIMULATION. VECTOR ADDRESS
	CLC	0(4,5),=C'ENDS'		IS CONVERTED TO BINARY AND SET(MFPC)
	BC	8,END		WITH IT TO SIMULATE NEXT INSTRUCTION
	CLC	0(4,5),=C'JUMP'		
	BC	7,STOPH		
	A	5,=F'5'		
	CLC	0(4,5),=C'		
	BC	8,XEXT		
	MVC	JK(4),0(5)		
	TR	JK(4),TTAB-X'C1'		
	MVC	JKL(5),JK		
	PACK	JKT(3),JKL		
	CLC	JKT(2),=X'FFFF'		
	BC	2,STOPL		
	MVC	MFPC(2),JKT		
	MVC	ADRES,=F'0'		
	MVC	ADRES+2(2),MFPC		
	LA	8,MICROMEM		
	A	8,ADRES		
	LR	6,8		
	B	XEXT		
M3V MA	MVC	ADRES,=F'0'		ADD MEMORY POINTING REGISTER PAIR
	MVC	ADRES+2(2),MFH		(MFH MFL) CONTENTS TO STARTING ADDR-
	LA	8,MICROMEM		ESS(MICROMEM) OF MICROPROCESSOR

```

A      8,ADRES
MVC   0(1,8),MFA
A      6,=F'1'
B      XEXT
MDVAB MVC   MFA,MFB
A      6,=F'1'
B      XEXT
MDVAC MVC   MFA,MFC
A      6,=F'1'
B      XEXT
MDVAD MVC   MFA,MFD
A      6,=F'1'
B      XEXT
MDVAE MVC   MFA,MFE
A      6,=F'1'
B      XEXT
MDVAH MVC   MFA,MFH
A      6,=F'1'
B      XEXT
MDVAL MVC   MFA,MFL
A      6,=F'1'
B      XEXT
MDVAM MVC   ADRES,=F'0'
MVC   ADRES+2(2),MFH
- A    8,MICROMEM
MVC   8,ADRES
A      MFA(1),0(8)
A      6,=F'1'
B      XEXT
MDVAA MVC   MFA,MFA
A      6,=F'1'
B      XEXT
ADDB  MVC   ADD,=H'0'
MVC   ADD+1(1),MFB
MVC   CYY,=X'00'
B      ADCOM
ADDC  MVC   ADD,=H'0'
MVC   ADD+1(1),MFC
MVC   CYY,=X'00'
B      ADCOM
ADDD  MVC   ADD,=H'0'
MVC   ADD+1(1),MFD
MVC   CYY,=X'00'
B      ADCOM
ADDE  MVC   ADD,=H'0'
MVC   ADD+1(1),MFE
MVC   CYY,=X'00'
B      ADCOM
ADDH  MVC   ADD,=H'0'
MVC   ADD+1(1),MFH
MVC   CYY,=X'00'
B      ADCOM
ADDL  MVC   ADD,=H'0'
MVC   ADD+1(1),MFL
MVC   CYY,=X'00'
B      ADCOM
ADDM  MVC   ADD,=H'0'
MVC   ADRES,=F'0'
MVC   ADRES+2(2),MFH
LA    8,MICROMEM
A      8,ADRES
MVC   ADD+1(1),0(8)
MVC   CYY,=X'00'
B      ADCOM
ADDA  MVC   ADD,=H'0'
MVC   ADD+1(1),MFA

```

MEMORY ON MAINFRAME TO LOCATE THE ADDRESS WHERE 'MFA' CONTENTS IS BE MOVED.

ADD MEMORY POINTING REGISTER PAIR (MFH MFL) CONTENTS TO STARTING ADDRESS (MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME TO LOCATE THE ADDRESS FROM WHERE ACCUMULATOR IS TO BE LOADED.

MOVE REGISTER(MFB) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PROCESS REST OPERATIONS.  
 MOVE REGISTER(MFC) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO COMMON ROUTINE 'ADCOM' TO PROCESS REST OPERATIONS.  
 MOVE REGISTER(MFD) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PROCESS THE REST OPERATIONS.  
 MOVE REGISTER(MFE) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PROCESS THE REST OPERATIONS.  
 MOVE REGISTER(MFH) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PROCESS THE REST OPERATIONS.  
 MOVE REGISTER(MFL) TO A MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PRCESS THE REST OPERATIONS.  
 ADD MEMORY POINTING REGISTER PAIR (MFH MFL) TO STARTING ADDRESS OF MICROPROCESSOR MEMORY ON MAINFRAME FROM WHERE DATA IS MOVED TO A MEMORY LOCATION 'ADD'. MOVE ZERO TO CYY AND BRANCH TO A COMMON ROUTINE 'ADCOM' TO PROCESS THE REST OPERATIONS.  
 MOVE ACCUMULATOR CONTENTS TO MEMORY LOCATION 'ADD' AND ZERO TO 'CYY' AND

```

MVC      CYY,=X'00'
B        ADCOM
ADCB    MVC      ADD,=H'0'
MVC      ADD+1(1),MFB
MVC      CYY(1),MFCY
B        ADCOM
ADCC    MVC      ADD,=H'0'
MVC      ADD+1(1),MFC
MVC      CYY(1),MFCY
B        ADCOM
ADCD    MVC      ADD,=H'0'
MVC      ADD+1(1),MFD
MVC      CYY(1),MFCY
B        ADCOM
ADCE    MVC      ADD,=H'0'
MVC      ADD+1(1),MFE
MVC      CYY(1),MFCY
B        ADCOM
ADCH    MVC      ADD,=H'0'
MVC      ADD+1(1),MFH
MVC      CYY(1),MFCY
B        ADCOM
ADCL    MVC      ADD,=H'0'
MVC      ADD+1(1),MFL
MVC      CYY(1),MFCY
B        ADCOM
ADCM    MVC      ADRES,=F'0'
MVC      ADD,=H'0'
MVC      ADRES+2(2),MFH
_A      8,MICROMEM
A        8,ADRES
MVC      ADD+1(1),0(8)
MVC      CYY(1),MFCY
B        ADCOM
ADCA    MVC      ADD,=H'0'
MVC      ADD+1(1),MFA
MVC      CYY(1),MFCY
B        ADCOM
ADCOM  MVC      SUB,=H'0'
MVC      HOUS,=H'0'
L        4,=F'0'
MVC      SUB+1(1),MFA
_H      4,ADD
AH      4,SUB
MVC      HOUS+1(1),CYY
AH      4,HOUS
STH     4,HOUS
MVC      MFA(1),HOUS+1
_A      13,SAVE
L        15,=V(ACFLGI)
LA      3,=A(ADD,SUB,CYY)
BALR   14,15
L        5,0(3)
MVC      MFAC(1),0(5)
MVC      CYC,=H'0'
LA      13,SAVE
L        15,=V(SUBFLG)
LA      3,=A(HOUS,CYC)
BALR   14,15
L        5,0(3)
MVC      MFZ(1),0(5)
L        5,4(3)
MVC      MFS(1),0(5)
L        5,8(3)
NC      HOUS,=X'00FF'
LH     7,HOUS

```

```

BRANCH TO A COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFB) TO A MEMORY LOCA-
TION 'ADD' AND CARRY FLAG TO 'CYY'
AND BRANCH TO COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFC) TO A MEMORY LOCA-
TION 'ADD' AND CARRY FLAG TO 'CYY'
AND BRANCH TO COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFD) TO MEMORY LOCAT-
ION 'ADD' AND CARRY FLAG TO 'CYY' AND
BRANCH TO COMMON ROUTINE 'ADCOM' TO
PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFE) TO MEMORY LOCA-
TION 'ADD' AND CARRY FLAG TO 'CYY'
AND BRANCH TO COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFH) TO MEMORY LOCA-
TION 'ADD' AND CARRY FLAG TO 'CYY'
AND BRANCH TO COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFL) TO MEMORY LOCAT-
ION 'ADD' AND CARRY FLAG TO 'CYY'
AND BRANCH TO COMMON ROUTINE 'ADCOM'
TO PROCESS THE REST OPERATIONS.
ADD MEMORY POINTING REGISTER PAIR
(MFH MFL) TO STARTING ADDRESS OF
MICROPROCESSOR MEMORY ON MAINFRAME
FROM WHERE DATA IS MOVED TO MEMORY
LOCATION 'ADD'. MOVE CARRY FLAG TO
'CYY' AND BRANCH TO COMMON ROUTINE
'ADCOM' TO PROCESS THE REST OPERA-
TIONS.
MOVE ACCUMULATOR CONTENTS TO MEMORY
LOCATION 'ADD' AND CARRY FLAG TO
'CYY' AND BRANCH TO COMMON ROUTINE
'ADCOM' TO PROCESS REST OPERATIONS.
ADD ACCUMULATOR(MFA) CONTENTS TO
'ADD' AND 'CYY'. STORE THE RESULT IN
'MFA'. BRANCH TO A SUBROUTINE
'ACFLGI' TO SET AUXILIARY CARRY
(MFAC) AND THEN BRANCH TO ANOTHER
SUBROUTINE(SUBFLG) TO SET THE FLAGS
'MFZ', 'MFS', 'MFP' AND 'MFCY'.

```

```

CH      7,=H'0'
BC      8,PADCOM
MVC    MFP(1),0(5)
PADCOM L      5,0(4)
MVC    MFCY(1),0(5)
A       6,=F'1'
B       XEXT
SJBB   MVC    SJB,=H'0'
MVC    SUB+1(1),MFB
MVC    CYY(1),=X'00'
B       SUBCOM
SJBC   MVC    SUB,=H'0'
MVC    SUB+1(1),MFC
MVC    CYY,=X'00'
B       SUBCOM
SJBD   MVC    SUB,=H'0'
MVC    SUB+1(1),MFD
MVC    CYY,=X'00'
B       SUBCOM
SJBE   MVC    SUB,=H'0'
MVC    SUB+1(1),MFE
MVC    CYY,=X'00'
B       SUBCOM
SJBH   MVC    SUB,=H'0'
MVC    SJB+1(1),MFH
MVC    CYY,=X'00'
B       SUBCOM
SJBL   MVC    SUB,=H'0'
MVC    SUB+1(1),MFL
MVC    CYY,=X'00'
B       SUBCOM
SJBM   MVC    ADRES,=F'0'
MVC    SUB,=H'0'
MVC    ADRES+2(2),MFH
LA      8,MICROMEM
A       8,ADRES
MVC    SUB+1(1),0(8)
MVC    CYY,=X'00'
B       SUBCOM
SJBA   MVC    SUB,=H'0'
MVC    SUB+1(1),MFA
MVC    CYY,=X'00'
B       SUBCOM
SJBCOM MVC    ADD,=H'0'
MVC    HOUS,=H'0'
L       4,=F'0'
MVC    ADD+1(1),MFA
LH      4,ADD
SH      4,SUB
MVC    HOUS+1(1),CYY
SH      4,HOUS
STH     4,HOUS
MVC    MFA(1),HOUS+1
LA      13,SAVE
-       15,=V(ACFLGD)
LA      3,=A(ADD,SUB,CYY)
BALR   14,15
L       5,0(3)
MVC    MFAC(1),0(5)
MVC    CYC,=H'0'
LA      13,SAVE
L       15,=V(SUBFLG)
LA      3,=A(HOUS,CYC)
BALR   14,15
L       5,0(3)
MVC    MFZ(1),0(5)

```

```

MOVE REGISTER(MFB) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFC) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFD) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFE) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFH) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO COMMON ROUTINE 'SUBCOM' TO
PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFL) TO MEMORY LOCAT-
ION 'SUB' AND ZERO TO 'CYY' AND BRA-
NCH TO A COMMON ROUTINE 'SUBCOM' TO
PROCESS THE REST OPERATIONS.
ADD MEMORY POINTING REGISTER PAIR
(MFH MFL) TO STARTING ADDRESS OF
MICROPROCESSOR MEMORY ON MAINFRAME
FROM WHERE DATA TO BE MOVED TO MEMO-
RY LOCATION 'SUB' AND ZERO TO 'CYY'
AND BRANCH TO A COMMON ROUTINE
'SUBCOM' TO PROCESS THE REST OPERAT-
IONS.
MOVE ACCUMULATOR CONTENTS TO MEMORY
LOCATION 'SUB' AND ZERO TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
SUBTRACT 'ADD' AND 'CYY' FROM ACCUM-
ULATOR. STORE THE RESULTS IN ACCUMU-
LATOR. BRANCH TO A SUBROUTINE
'SUBFLGD' TO SET AUXILIARY CARRY
FLAG AND THEN BRANCH TO ANOTHER
SUBROUTINE 'SUBFLG' TO SET THE FLAGS
'MFZ', 'MFS', 'MFP' AND 'MFCY'.

```

```

L      5,4(3)
MVC   MFS(1),0(5)
-     5,8(3)
NC    HOUS,=X'00FF'
LH    7,HOUS
CH    7,=H'0'
BC    8, SUBCOM
MVC   MFB(1),0(5)
PJBCOM
L     5,0(4)
MVC   MFCY(1),0(5)
A     6,=F'1'
B     XEXT
S3BB  MVC   SUB,=H'0'
MVC   SUB+1(1),MFB
MVC   CYY(1),MFCY
B     SUBCOM
S3BC  MVC   SUB,=H'0'
MVC   SUB+1(1),MFC
MVC   CYY(1),MFCY
B     SUBCOM
S3BD  MVC   SUB,=H'0'
MVC   SUB+1(1),MFD
MVC   CYY(1),MFCY
B     SUBCOM
S3BE  MVC   SUB,=H'0'
MVC   SUB+1(1),MFE
MVC   CYY(1),MFCY
B     SUBCOM
S3BH  MVC   SUB,=H'0'
MVC   SUB+1(1),MFH
MVC   CYY(1),MFCY
B     SUBCOM
S3BL  MVC   SUB,=H'0'
MVC   SUB+1(1),MFL
MVC   CYY(1),MFCY
B     SUBCOM
S3BM  MVC   ADRES,=F'0'
MVC   SUB,=H'0'
MVC   ADRES+2(2),MFH
-A    8,MICROMEM
A     8,ADRES
MVC   SUB+1(1),0(8)
MVC   CYY(1),MFCY
B     SUBCOM
S3BA  MVC   SUB,=H'0'
MVC   SUB+1(1),MFA
MVC   CYY(1),MFCY
B     SUBCOM
ANAB  NC    MFA(1),MFB
B     ANCOM
ANAC  NC    MFA(1),MFC
B     ANCOM
ANAD  NC    MFA(1),MFD
B     ANCOM
ANAE  NC    MFA(1),MFE
B     ANCOM
ANAH  NC    MFA(1),MFH
B     ANCOM
ANAL  NC    MFA(1),MFL
B     ANCOM
ANAM  MVC   ADRES,=F'0'
MVC   ADRES+2(2),MFH
LA    8,MICROMEM
A     8,ADRES
NC    MFA(1),0(8)
B     ANCOM

```

```

MOVE REGISTER(MFB) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
AND BRANCH TO A COMMON ROUTINE 'SUB-
COM' TO PROCESS REST OPERATIONS.
MOVE REGISTER(MFC) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFD) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFE) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFH) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
MOVE REGISTER(MFL) TO MEMORY LOCAT-
ION 'SUB' AND CARRY FLAG TO 'CYY' AND
BRANCH TO A COMMON ROUTINE 'SUBCOM'
TO PROCESS THE REST OPERATIONS.
ADD MEMORY POINTING REGISTER PAIR
(MFH MFL) TO STARTING ADDRESS OF
MICROPROCESSOR MEMORY ON MAINFRAME
TO LOCATE ADDRESS FROM WHERE DATA IS
TO BE MOVED TO MEMORY LOCATION 'SUB'
AND CARRY FLAG TO 'CYY' AND BRANCH
TO A COMMON SUBROUTINE 'SUBCOM' TO
PROCESS THE REST OPERATIONS.
MOVE ACCUMULATOR CONTENTS TO MEMORY
LOCATION 'SUB' AND CARRY FLAG TO
'CYY' AND BRANCH TO A COMMON ROUTINE
'SUBCOM' TO PROCESS REST OPERATIONS
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFB) AND ACCUMULATOR.
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFC) AND ACCUMULATOR.
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFD) AND ACCUMULATOR.
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFE) AND ACCUMULATOR.
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFH) AND ACCUMULATOR.
LOGICAL AND OPERATION BETWEEN REGIS-
TER(MFL) AND ACCUMULATOR.
ADD MEMORY POINTING REGISTER(MFH MFL
) TO STARTING ADDRESS OF MICROPROCE-
SSOR MEMORY ON MAINFRAME. DATA FROM
THAT ADDRESS IS USED FOR LOGICAL AND
OPERATION WITH ACCUMULATOR.

```

ANAA	VC	MFA(1),MFA
	B	ANCOM
ANCOM	MVI	MFAC,X'01'
	B	LOGCOM
LOGCOM	MVI	MFCY,X'00'
	MVC	CYC,=X'FFF0'
	MVC	HJUS+1(1),MFA
	LA	13,SAVE
	L	15,=V(SUBFLG)
	LA	3,=A(HOUS,CYC)
	BALR	14,15
	-	5,0(3)
	MVC	MFZ(1),0(5)
	L	5,04(3)
	MVC	MFS(1),0(5)
	-	5,08(3)
	NC	HJUS,=X'00FF'
	LH	7,HOUS
	CH	7,=H'0'
	BC	8,KOGCOM
	MVC	MFD(1),0(5)
KOGCOM	A	6,=F'1'
	B	XEXT
XRAB	XC	MFA(1),MFB
	B	DXRCOM
XRAC	XC	MFA(1),MFC
	B	DXRCOM
XRAD	XC	MFA(1),MFD
	B	DXRCOM
XRAE	XC	MFA(1),MFE
	B	DXRCOM
XRAH	XC	MFA(1),MFH
	B	DXRCOM
XRAL	XC	MFA(1),MFL
	B	DXRCOM
XRAM	XC	ADRES,=F'0'
	MVC	ADRES+2(2),MFH
	LA	8,MICROMEM
	L	8,ADRES
	XC	MFA(1),0(8)
	B	DXRCOM
XRAA	XC	MFA(1),MFA
	B	DXRCOM
ORAB	XC	MFA(1),MFB
	B	DXRCOM
ORAC	XC	MFA(1),MFC
	B	DXRCOM
ORAD	XC	MFA(1),MFD
	B	DXRCOM
ORAE	XC	MFA(1),MFE
	B	DXRCOM
ORAH	XC	MFA(1),MFH
	B	DXRCOM
ORAL	XC	MFA(1),MFL
	B	DXRCOM
ORAM	MVC	ADRES,=F'0'
	MVC	ADRES+2(2),MFH
	LA	8,MICROMEM
	A	8,ADRES
	XC	MFA(1),0(8)
	B	DXRCOM
ORAA	XC	MFA(1),MFA
	B	DXRCOM
DXRCOM	MVI	MFAC,X'00'
	B	LOGCOM
CMPB	MVC	SUB,=H'0'

LOGICAL AND OPERATION BETWEEN REGISTER(MFA) AND ACCUMULATOR.  
AUXILIARY CARRY(MFAC) IS SET TO 1.

CARRY FLAG IS SET TO ZERO. BRANCH TO A SUBROUTINE 'SUBFLG' TO SET THE FLAGS 'MFZ', 'MFS' AND 'MFP'.

LOGICAL EXCLUSIVE OR OPERATION BETWEEN REGISTER(MFB) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE-OR OPERATION BETWEEN REGISTER(MFC) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE-OR OPERATION BETWEEN REGISTER(MFD) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE-OR OPERATION BETWEEN REGISTER(MFE) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE-OR OPERATION BETWEEN REGISTER(MFH) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE-OR OPERATION BETWEEN REGISTER(MFL) AND ACCUMULATOR.  
LOGICAL EXCLUSIVE OR OPERATION BETWEEN MEMORY CONTENTS AND ACCUMULATOR.

LOGICAL EXCLUSIVE-OR OPERATION BETWEEN ACCUMULATOR AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFB) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFC) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFD) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFE) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFH) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN REGISTER(MFL) AND ACCUMULATOR.  
LOGICAL OR OPERATION BETWEEN MEMORY CONTENTS AND ACCUMULATOR.

LOGICAL OR OPERATION BETWEEN ACCUMULATOR AND ACCUMULATOR.  
SET AUXILIARY CARRY FLAG WITH 0.

MOVE REGISTER(MFB) CONTENTS TO MEMO-

```

MVC SUB+1(1),MFB
B COMP
CMPC MVC SUB,=H'0'
MVC SUB+1(1),MFC
B COMP
CMPD MVC SUB,=H'0'
MVC SUB+1(1),MFD
B COMP
CMPE MVC SUB,=H'0'
MVC SUB+1(1),MFE
B COMP
CMPH MVC SUB,=H'0'
MVC SUB+1(1),MFH
B COMP
CMPL MVC SUB,=H'0'
MVC SUB+1(1),MFL
B COMP
CMPM MVC ADRES,=F'0'
MVC SUB,=H'0'
MVC ADRES+2(2),MFH
LA 8,MICROMEM
A 8,ADRES
MVC SUB+1(1),0(8)
B COMP
CMPA MVC SUB,=H'0'
MVC SUB+1(1),MFA
B COMP
COMP MVC ADD,=H'0'
MVC HOUS,=H'0'
L 4,=F'0'
MVC ADD+1(1),MFA
-H 4,ADD
SH 4,SUB
STH 4,HOUS
MVC CYY,=X'00'
LA 13,SAVE
L 15,=V(ACFLGD)
LA 3,=A(ADD,SUB,CYY)
BALR 14,15
L 5,0(3)
MVC MFAC(1),0(5)
MVC CYC,=H'0'
LA 13,SAVE
L 15,=V(SUBFLG)
LA 3,=A(HOUS,CYC)
BALR 14,15
L 5,0(3)
MVC MFZ(1),0(5)
L 5,4(3)
MVC MFS(1),0(5)
L 5,8(4)
NC HOUS,=X'00FF'
-H 7,HOUS
CH 7,=H'0'
BC 8,PCOMP
MVC MFP(1),0(5)
L 5,0(4)
PCOMP MVC MFCY(1),0(5)
A 6,=F'1'
B XEXT
INCRM MVC SUB,=H'1'
MVC CYY,=X'00'
-A 13,SAVE
L 15,=V(ACFLGI)
LA 3,=A(ADD,SUB,CYY)
BALR 14,15

```

RY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. MOVE REGISTER(MFC) CONTENTS TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. MOVE REGISTER(MFD) CONTENTS TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. MOVE REGISTER(MFE) CONTENTS TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS THE REST OPERATIONS. MOVE REGISTER(MFH) CONTENTS TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. MOVE REGISTER(MFL) CONTENTS TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. MOVE MEMORY CONTENTS POINTED BY (MFH MFL) TO MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS THE REST OPERATIONS.

MOVE ACCUMULATOR CONTENTS TO A MEMORY LOCATION 'SUB' AND BRANCH TO 'SUBCOM' TO PROCESS REST OPERATIONS. SUBTRACT 'SUB' AND 'CYY' FROM ACCUMULATOR CONTENTS. STORE THE RESULTS IN ACCUMULATOR. BRANCH TO A SUBROUTINE 'ACFLGD' TO SET AUXILIARY CARRY 'MFAC' AND BRANCH TO ANOTHER SUBROUTINE 'SUBFLG' TO SET THE FLAGS 'MFZ' 'MFS', 'MFP' AND 'MFCY'.

MOVE 1 TO 'SUB' AND BRANCH TO A SUBROUTINE 'ACFLGI' TO SET AUXILIARY CARRY(MFAC). BRANCH TO ANOTHER SUBROUTINE 'SUBFLG' TO SET THE FLAGS 'MFZ' 'MFS', 'MFP' AND 'MFCY'.



```

L      5,0(3)
MVC   MFAC(1),0(5)
MVC   CYC,=X'FFF0'
LA    13,SAVE
L     15,=V(SUBFLG)
LA    3,=A(HOUS,CYC)
BALR  14,15
L     5,0(3)
MVC   MFZ(1),0(5)
L     5,04(3)
MVC   MFS(1),0(5)
L     5,8(3)
NC    HOUS,=X'00FF'
LH    7,HOUS
CH    7,=H'0'
BC    8,INRMD
MVC   MFP(1),0(5)
INRMD A    6,=F'1'
      B    XEX 1
DECRM MVC   SUB,=H'1'
      MVC   CYY,=X'00'
      LA    13,SAVE
      L     15,=V(ACFLGD)
      -A   3,=A(ADD,SUB,CYY)
      BALR  14,15
      L     5,0(3)
      MVC   MFAC(1),0(5)
      MVC   CYC,=X'FFF0'
      LA    13,SAVE
      L     15,=V(SUBFLG)
      -A   3,=A(HOUS,CYC)
      BALR  14,15
      L     5,0(3)
      MVC   MFZ(1),0(5)
      L     5,04(3)
      MVC   MFS(1),0(5)
      L     5,08(3)
      NC    HOUS,=X'00FF'
      LH    7,HOUS
      CH    7,=H'0'
      BC    8,DCRMD
DCRMD MVC   MFP(1),0(5)
      A    6,=F'1'
      B    XEX 1
DADCOM MVC   SB(4),=F'0'
      L     8,=F'0'
      MVC   SB+2(2),MFH
      -A   8,AD
      ST    8,AD
      MVC   MFH(2),AD+2
      NC    AD,=X'00010000'
      -C   8,AD
      C    8,=F'0'
      BC    8,CYD1
      MVI   MFCY,X'01'
      B    CYD2
      MVI   MFCY,X'00'
      A    6,=F'1'
      B    XEX 1
      MVC   ADRES,=F'0'
      LA    8,MICROMEM
      MVC   ADRES+2(2),MFPC
      A    8,ADRES
      MVC   ADRES+3(1),0(8)
      A    8,=F'1'

```

MOVE 1 TO MEMORY LOCATION 'SUB' AND BRANCH TO A SUBROUTINE 'ACFLGD' TO SET AUXILIARY CARRY 'MFAC' AND THEN BRANCH TO ANOTHER SUBROUTINE 'SUBFLG' TO THE FLAGS 'MFZ', 'MFS', 'MFP' AND 'MFCY'.

THE CONTENTS OF MEMORY LOCATION POINTED BY 'MFH MFL' IS ADDED THE REGISTER PAIR AND THE NINETH BIT OF THE RESULT FROM THE RIGHT IS CHECKED. SET CARRY FLAG WITH 0 IF IT IS 0 OTHERWISE SET WITH 1.

LOAD PROGRAM COUNTER (MFPC) WITH NEXT TWO INSTRUCTION BYTES.

	MVC	ADRES+2(1),0(8)	
	MVC	MFPC(2),ADRES+2	
	LA	8,MICROMEM	
	A	8,ADRES	
	LR	6,8	
	B	XEXT	
JCOM1	-	4,=F'0'	IF FLAG IS 1 BRANCH TO 'JMP' OTHER-
	LH	4,ADD	WISE BRANCH TO 'JCOM3'.
	CH	4,=H'1'	
	BE	JMP	
JCOM3	L	4,=F'0'	THE PROGRAM COUNTER IS UPDATED THRO-
	A	6,=F'3'	UGH INCREASING ITS CONTENTS BY 2.
	-H	4,=H'2'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	XEXT	
JCOM2	-	4,=F'0'	IF FLAG IS 0 BRANCH TO 'JMP' OTHER-
	LH	4,ADD	WISE BRANCH TO 'JCOM3'.
	CH	4,=H'0'	
	BE	JMP	
	B	JCOM3	
JM	MVC	ADD,=H'0'	JUMP IF SIGN FLAG IS 1.
	MVC	ADD+1(1),MFS	
	B	JCOM1	
JNC	MVC	ADD,=H'0'	JUMP IF CARRY FLAG IS 0.
	MVC	ADD+1(1),MFCY	
	B	JCOM2	
JNZ	MVC	ADD,=H'0'	JUMP IF ZERO FLAG IS 0.
	MVC	ADD+1(1),MFZ	
	B	JCOM2	
JP	MVC	ADD,=H'0'	JUMP IF SIGN FLAG IS 0.
	MVC	ADD+1(1),MFS	
	B	JCOM2	
JC	MVC	ADD,=H'0'	JUMP IF CARRY FLAG IS 1.
	MVC	ADD+1(1),MFCY	
	B	JCOM1	
JPE	MVC	ADD,=H'0'	JUMP IF PARITY FLAG IS EVEN.
	MVC	ADD+1(1),MFP	
	B	JCOM1	
JPO	MVC	ADD,=H'0'	JUMP IF PARITY FLAG IS ODD.
	MVC	ADD+1(1),MFP	
	B	JCOM2	
JZ	MVC	ADD,=H'0'	JUMP IF ZERO FLAG IS 1.
	MVC	ADD+1(1),MFZ	
	B	JCOM1	
PCHL	MVC	MFPC(1),MFH	THE CONTENTS OF MEMORY POINTING REG-
	MVC	MFPC+1(1),MFL	ISTER PAIR(MFH MFL) IS LOADED TO
	A	6,=F'1'	PROGRAM COUNTER(MFPC).
	B	XEXT	
CALL	-	5,=F'0'	THE NEXT TWO INSTRUCTION BYTE IS
	-	4,=F'0'	LOADED TO THE PROGRAM COUNTER AND
	MVC	ADRES,=F'0'	THE ADDRESS OF THE NEXT INSTRUCTION
	LA	8,MICROMEM	IS STORED IN THE STACK.
	MVC	ADRES+2(2),MFPC	
	A	8,ADRES	
	MVC	ADRES+3(1),0(8)	
	A	8,=F'1'	
	MVC	ADRES+2(1),0(8)	
	-H	5,MFPC	
	AH	5,=H'2'	
	STH	5,MFPC	
	LH	5,MFSP	
	SH	5,=H'2'	
	STH	5,MFSP	
	LA	8,MICROMEM	
	AR	5,8	

	MVC	0(1,5),MFPC+1	
	A	5,=F'1'	
	MVC	0(1,5),MFPC	
	MVC	MFPC(2),ADRES+2	
	A	8,ADRES	
	LR	6,8	
	B	XEXT	
CCOM1	L	4,=F'0'	IF FLAG IS 1 BRANCH TO 'CALL' OTHER-
	LH	4,ADD	WISE BRANCH TO 'CCOM3'.
	CH	4,=H'1'	
	BC	8,CALL	
	B	CCOM3	
CCOM2	L	4,=F'0'	IF FLAG IS 0 BRANCH TO 'CALL' OTHER-
	LH	4,ADD	WISE BRANCH TO 'CCOM3'.
	CH	4,=H'0'	
	BC	8,CALL	
CCOM3	A	6,=F'3'	THE PROGRAM COUNTER (MFPC) IS UPDAT-
	LH	4,=H'2'	ED THROUGH INCREASING ITS CONTENTS
	AH	4,MFPC	BY 2.
	STH	4,MFPC	
	B	XEXT	
CC	MVC	ADD,=H'0'	CALL IF CARRY FLAG IS 1.
	MVC	ADD+1(1),MFCY	
	B	CCOM1	
CNC	MVC	ADD,=H'0'	CALL IF CARRY FLAG IS 0.
	MVC	ADD+1(1),MFCY	
	B	CCOM2	
CM	MVC	ADD,=H'0'	CALL IF SIGN FLAG IS 1.
	MVC	ADD+1(1),MFS	
	B	CCOM1	
CNZ	MVC	ADD,=H'0'	CALL IF ZERO FLAG IS 0.
	MVC	ADD+1(1),MFZ	
	B	CCOM2	
C <sup>2</sup>	MVC	ADD,=H'0'	CALL IF SIGN FLAG IS 0.
	MVC	ADD+1(1),MFS	
	B	CCOM2	
C <sup>2</sup> E	MVC	ADD,=H'0'	CALL IF PARITY FLAG IS EVEN.
	MVC	ADD+1(1),MFP	
	B	CCOM1	
C <sup>2</sup> O	MVC	ADD,=H'0'	CALL IF PARITY FLAG IS ODD.
	MVC	ADD+1(1),MFP	
	B	CCOM2	
CZ	MVC	ADD,=H'0'	CALL IF ZERO FLAG IS 1.
	MVC	ADD+1(1),MFZ	
	B	CCOM1	
RET	LA	8,MICROMEM	THE PROGRAM COUNTER IS LOADED FROM
	MVC	ADRES+2(2),MFSP	THE LOCATION ADDRESSED BY STACK POI-
	A	8,ADRES	NTER. STACK POINTER IS INCREASED BY 2
	MVC	MFPC+1(1),0(8)	
	A	8,=F'1'	
	MVC	MFPC(1),0(8)	
	LH	5,MFSP	
	AH	5,=H'2'	
	STH	5,MFSP	
	MVC	ADRES+2(2),MFPC	
	LA	8,MICROMEM	
	A	8,ADRES	
	LR	6,8	
	B	XEXT	
RCOM1	L	4,=F'0'	IF FLAG IS 1 BRANCH TO 'RET' OTHER-
	LH	4,ADD	WISE BRANCH TO 'RCOM3'.
	CH	4,=H'1'	
	BC	8,RET	
	B	RCOM3	
RCOM2	L	4,=F'0'	IF FLAG IS 0 BRANCH TO 'RET' OTHER-
	LH	4,ADD	WISE BRANCH TO 'RCOM3'.

	CH	4,=F'0'	
	BC	8,RET	
	B	RCOM3	
RCOM3	A	6,=F'1'	BRANCH TO DEBUGGER.
	B	XEXT	
RC	MVC	ADD,=H'0'	RETURN IF CARRY FLAG IS SET TO 1.
	MVC	ADD+1(1),MFCY	
	B	RCOM1	
RM	MVC	ADD,=H'0'	RETURN IF SIGN FLAG IS 1.
	MVC	ADD+1(1),MFS	
	B	RCOM1	
RNC	MVC	ADD,=H'0'	RETURN IF CARRY FLAG IS 0.
	MVC	ADD+1(1),MFCY	
	B	RCOM2	
RNZ	MVC	ADD,=H'0'	RETURN IF ZERO FLAG IS 0.
	MVC	ADD+1(1),MFZ	
	B	RCOM2	
R <sup>S</sup>	MVC	ADD,=H'0'	RETURN IF SIGN FLAG IS 0.
	MVC	ADD+1(1),MFS	
	B	RCOM2	
R <sup>PE</sup>	MVC	ADD,=H'0'	RETURN IF PARITY IS EVEN.
	MVC	ADD+1(1),MFP	
	B	RCOM1	
R <sup>PO</sup>	MVC	ADD,=H'0'	RETURN IF PARITY IS ODD.
	MVC	ADD+1(1),MFP	
	B	RCOM2	
RZ	MVC	ADD,=H'0'	RETURN IF ZERO FLAG IS 1.
	MVC	ADD+1(1),MFZ	
	B	RCOM1	
PJPCOM	LH	4,=H'2'	STACK POINTER IS INCREMENTED BY 2.
	AH	4,MFSP	
	STH	4,MFSP	
	A	6,=F'1'	
	B	XEXT	
XCHG	MVC	M(1),MFD	THE CONTENTS OF {MFD,MFE} IS EXCHANGED WITH {MFH,MFL}.
	MVC	N(1),MFE	
	MVC	MFD(1),MFH	
	MVC	MFE(1),MFL	
	MVC	MFH(1),M	
	MVC	MFL(1),N	
	A	6,=F'1'	
	B	XEXT	
PJSHCOM	L	4,=F'0'	THE STACK POINTER IS DECREMENTED BY 2.
	-H	4,MFSP	
	SH	4,=H'2'	
	STH	4,MFSP	
	A	6,=F'1'	
	B	XEXT	
S <sup>PHL</sup>	MVC	MFSP(1),MFH	THE CONTENTS OF REGISTER PAIR {MFH MFL} IS STORED IN THE LOCATION ADDRESSED BY STACK POINTER.
	MVC	MFSP+1(1),MFL	
	A	6,=F'1'	
	B	XEXT	
X <sup>THL</sup>	MVC	ADRES,=F'0'	THE REGISTER PAIR {MFH MFL} IS EXCHANGED WITH THE DATA ADDRESSED BY THE STACK POINTER.
	MVC	ADRES+2(2),MFSP	
	LA	8,MICROMEM	
	LR	4,8	
	A	4,ADRES	
	MVC	O(1,4),MFL	
	A	4,=F'1'	
	MVC	O(1,4),MFH	
	A	8,ADRES	
	MVC	MFL(1),O(8)	
	A	8,=F'1'	
	MVC	MFH(1),O(8)	
	A	6,=F'1'	
	B	XEXT	

ADI	MVC	ADD,=H'0'	MOVE IMMEDIATE DATA TO 'ADD' AND 0
	A	6,=F'1'	TO 'CYY' AND BRANCH TO ROUTINE
	MVC	ADD+1(1),0(6)	'ADCOM' TO PROCESS THE REST OPERAT-
	MVC	CYY,=X'00'	IONS.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	ADCOM	
ACI	MVC	ADD,=H'0'	MOVE IMMEDIATE DATA TO 'ADD' AND
	A	6,=F'1'	CARRY FLAG TO 'CYY' AND BRANCH TO A
	MVC	ADD+1(1),0(6)	ROUTINE 'ADCOM' TO PROCESS THE REST
	MVC	CYY(1),MFCY	OPERATIONS.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	ADCOM	
SJI	MVC	SUB,=H'0'	MOVE IMMEDIATE DATA TO 'SUB' AND 0
	A	6,=F'1'	TO 'CYY' AND BRANCH TO A ROUTINE
	MVC	SUB+1(1),0(6)	'SUBCOM' TO PROCESS THE REST OPERAT-
	MVI	CYY,X'00'	IONS.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	SUBCOM	
SBI	MVC	SUB,=H'0'	MOVE IMMEDIATE DATA TO 'SUB' AND
	A	6,=F'1'	CARRY FLAG TO 'CYY' AND BRANCH TO A
	MVC	SUB+1(1),0(6)	ROUTINE 'SUBCOM' TO PROCESS THE REST
	MVC	CYY(1),MFCY	OPERATIONS.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	SUBCOM	
ANI	A	6,=F'1'	LOGICAL AND OPERATION OF IMMEDIATE
	NC	MFA(1),0(6)	DATA WITH ACCUMULATOR.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	ANCOM	
XRI	A	6,=F'1'	LOGICAL EXCLUSIVE-OR OPERATION OF
	XC	MFA(1),0(6)	IMMEDIATE DATA WITH ACCUMULATOR.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	QXRCOM	
ORI	A	6,=F'1'	LOGICAL OR OPERATION OF IMMEDIATE
	OC	MFA(1),0(6)	DATA WITH ACCUMULATOR.
	L	4,=F'1'	
	AH	4,MFPC	
	STH	4,MFPC	
	B	QXRCOM	
COI	MVC	SUB,=H'0'	MOVE IMMEDIATE DATA TO 'SUB' AND
	A	6,=F'1'	BRANCH TO A ROUTINE 'COMP' TO PROC-
	MVC	SUB+1(1),0(6)	ESS THE REST OPERATIONS.
	L	4,=F'1'	
	A	4,MFPC	
	STH	4,MFPC	
	B	COMP	
RSTO	L	5,=F'0'	THE PROGRAM COUNTER CONTENTS IS STO-
	LH	5,MFSP	RED AT THE LOCATION ADDRESSED BY THE
	SH	5,=H'2'	STACK POINTER AND IT IS LOADED WITH
	STH	5,MFSP	THE VECTOR ADDRESS (0000)H.
	-A	8,MICROMEM	
	AR	5,8	
	MVC	0(1,5),MFPC+1	
	A	5,=F'1'	
	MVC	0(1,5),MFPC	

```

RST 1
MVC MFPC(2),=X'0000'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
LA 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0008'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
-A 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0010'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
LA 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0018'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
-A 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0020'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
-A 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0028'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
-A 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC
MVC MFPC(2),=X'0030'
B RSTCOM
- 5,=F'0'
LH 5,MFSP
SH 5,=H'2'
STH 5,MFSP
-A 8,MICROMEM
AR 5,8
MVC 0(1,5),MFPC+1
A 5,=F'1'
MVC 0(1,5),MFPC

```

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0008)H.

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0010)H.

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0018)H.

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY THE STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0020)H.

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0028)H.

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0030)H.

```

RST7   MVC   MFPC(2),=X'0030'
        B     RSTCOM
        -     5,=F'0'
        LH    5,MFSP
        SH    5,=H'2'
        STH   5,MFSP
        LA    8,MICROMEM
        AR    5,8
        MVC   0(1,5),MFPC+1
        A     5,=F'1'
        MVC   0(1,5),MFPC
        MVC   MFPC(2),=X'0038'
RSTCOM B     RSTCOM
        MVC   ADRES,=F'0'
        MVC   ADRES+2(2),MFPC
        A     8,ADRES
        LR    6,8
        B     XEXT
PJPB   MVC   ADRES,=F'0'
        LA    8,MICROMEM
        MVC   ADRES+2(2),MFSP
        A     8,ADRES
        MVC   MFC(1),0(8)
        A     8,=F'1'
        MVC   MFB(1),0(8)
        B     PJP COM
PJPD   MVC   ADRES,=F'0'
        LA    8,MICROMEM
        MVC   ADRES+2(2),MFSP
        A     8,ADRES
        MVC   MFE(1),0(8)
        A     8,=F'1'
        MVC   MFD(1),0(8)
        B     PJP COM
PJPH   MVC   ADRES,=F'0'
        LA    8,MICROMEM
        MVC   ADRES+2(2),MFSP
        A     8,ADRES
        MVC   MFL(1),0(8)
        A     8,=F'1'
        MVC   MFH(1),0(8)
        B     PJP COM
PJPPSW MVC   ADRES,=F'0'
        LA    8,MICROMEM
        MVC   ADRES+2(2),MFSP
        A     8,ADRES
        MVC   MFPSW(1),0(8)
        MVC   ADD,=H'0'
        MVC   ADD+1(1),MFPSW
        LH    4,ADD
        SLL   4,1
        STH   4,ADD
        MVC   MFS(1),ADD
        MVC   PSD(1),MFPSW
        NI    PSD,X'40'
        MVC   ADD,=H'0'
        MVC   ADD+1(1),PSD
        LH    4,ADD
        SLL   4,2
        STH   4,ADD
        MVC   MFZ(1),ADD
        MVC   PSD(1),MFPSW
        NI    PSD,X'10'
        MVC   ADD,=H'0'
        MVC   ADD+1(1),PSD
        LH    4,ADD

```

THE PROGRAM COUNTER CONTENTS IS STORED AT THE LOCATION ADDRESSED BY STACKPOINTER AND IT IS LOADED WITH THE VECTOR ADDRESS (0038)H.

THE CONTENTS OF STACK ARE LOADED IN REGISTERS 'MFC' AND 'MFB' RESPECTIVELY.

THE CONTENTS OF THE STACK ARE LOADED IN REGISTERS 'MFE' AND 'MFD' RESPECTIVELY.

THE CONTENTS OF THE STACK ARE LOADED IN REGISTERS 'MFL' AND 'MFH' RESPECTIVELY.

THE CONTENTS OF THE STACK ARE LOADED IN 'PSW' AND ACCUMULATOR RESPECTIVELY. ALL THE FLAGS ARE SET WITH THE CORRESPONDING BIT OF THE 'PSW'.

	SLL	4,4	
	STH	4,ADD	
	MVC	MFC(1),ADD	
	MVC	PSD(1),MFPSW	
	NI	PSD,X'04'	
	MVC	ADD,=H'0'	
	MVC	ADD+1(1),PSD	
	LH	4,ADD	
	SLL	4,6	
	STH	4,ADD	
	MVC	MFC(1),ADD	
	MVC	PSD(1),MFPSW	
	NI	PSD,X'01'	
	MVC	ADD,=H'0'	
	MVC	ADD+1(1),PSD	
	LH	4,ADD	
	SLL	4,8	
	STH	4,ADD	
	MVC	MFCY(1),ADD	
	A	8,=F'1'	
	MVC	MFA(1),0(8)	
	B	PUSHCOM	
PJSHB	L	4,=F'0'	THE CONTENTS OF REGISTER PAIR 'MFB
	MVC	ADRES,=F'0'	MFC' ARE STORED IN THE STACK.
	LA	8,MICROMEM	
	MVC	ADRES+2(2),MFSP	
	A	8,ADRES	
	S	8,=F'1'	
	MVC	0(1,8),MFB	
	S	8,=F'1'	
	MVC	0(1,8),MFC	
	B	PUSHCOM	
PJSHD	L	4,=F'0'	THE CONTENTS OF REGISTER PAIR 'MFD
	MVC	ADRES,=F'0'	MFE' ARE STORED IN THE STACK.
	LA	8,MICROMEM	
	MVC	ADRES+2(2),MFSP	
	A	8,ADRES	
	S	8,=F'1'	
	MVC	0(1,8),MFD	
	S	8,=F'1'	
	MVC	0(1,8),MFE	
	B	PUSHCOM	
PJSHH	L	4,=F'0'	THE CONTENTS OF REGISTER PAIR 'MFH
	MVC	ADRES,=F'0'	MFL' ARE STORED IN THE STACK.
	LA	8,MICROMEM	
	MVC	ADRES+2(2),MFSP	
	A	8,ADRES	
	S	8,=F'1'	
	MVC	0(1,8),MFH	
	S	8,=F'1'	
	MVC	0(1,8),MFL	
	B	PUSHCOM	
PJSHPSW	L	4,=F'0'	ALL THE 8085 FLAGS ARE INSERTED INTO
	MVC	ADRES,=F'0'	THE 'PSW' AND THE CONTENTS OF ACCUM-
	LA	8,MICROMEM	ULATOR AND 'PSW' ARE STORED IN THE
	MVC	ADRES+2(2),MFSP	STACK.
	A	8,ADRES	
	S	8,=F'1'	
	MVC	0(1,8),MFA	
	S	8,=F'1'	
	NI	MFC SW,X'FE'	
	OC	MFC SW,MFCY	
	MVC	MK,=F'0'	
	MVC	MK+3(1),MFC	
	L	4,MK	
	SLL	4,4	



```

ST      4,MK
NI      MFP SW,X*EF*
DC      MFP SW(1),MK+3
MVC     MK,=F'0'
MVC     MK+3(1),MFS
-
SLL     4,7
ST      4,MK
NI      MFP SW,X*7F*
DC      MFP SW(1),MK+3
MVC     MK,=F'0'
MVC     MK+3(1),MFZ
L       4,MK
SLL     4,6
ST      4,MK
NI      MFP SW,X*BF*
DC      MFP SW(1),MK+3
MVC     MK,=F'0'
MVC     MK+3(1),MFP
L       4,MK
SLL     4,2
ST      4,MK
NI      MFP SW,X*FB*
DC      MFP SW(1),MK+3
MVC     0(1,8),MFP SW
B       PUSHCOM
EI      A      6,=F'1'
        B      XEXT
DI      A      6,=F'1'
        B      XEXT
IN      L      4,=F'1'
        AH     4,MFPC
        STH    4,MFPC
        CLC    INPT(4),=C'DSKT'
        BC     8,INA
        CLC    INPT(4),=C'KBRD'
        BC     8,INB
INA     BC     7,STOPC
        -A    13,SAVE
        L      15,=V(SUBDSK)
        BALR  14,15
        A      6,=F'2'
INB     B      XEXT
        LA    13,SAVE
        L      15,=V(OUTPUT)
        L      3,=F'0'
        BALR  14,15
        -     5,0(3)
        MVC   TPK(2),0(5)
        TR    TPK(2),TTAB-X'C1'
        MVC   PTL(3),TPK
        PACK  TPK(2),PTL
        MVC   MFA(1),TPK
        A      6,=F'2'
OJT     B      XEXT
        -     4,=F'1'
        AH     4,MFPC
        STH    4,MFPC
        A      6,=F'1'
        CLC    OUTPT(4),=C'PRNT'
        BC     7,OUTA
        MVC   XB(2),0(6)
        UNPK  XL(3),XB
        MVC   XB(2),XL
        TR    XB(2),TRTAB-X'F0'
        MVC   XC(2),MFA

```

IF INPUT PORT IS SIMULATED THROUGH DISKETTE BRANCH TO SUBROUTINE (SUBDSK).  
IF IT IS SIMULATED THROUGH CONSOLE-KEYBOARD BRANCH TO SUBROUTINE (OUTPUT).

IF OUTPUT PORT IS SIMULATED ON LINE PRINTER THEN UNPACK PORT ADDRESS AND DATA, CONVERT TO EBCDIC FORM AND PRINT ON THE LINE-PRINTER. IF IT IS SIMULATED UPON CONSOLE KEYBOARD THEN BRANCH TO SUBROUTINE 'OUTPUT' TO DISPLAY UPON CONSOLE.



```

BPPRNT  B      DMPX
        LA     13,SAVE
        L      15,=V(SUBBPP)
        BALR   14,15
DMPX    _H     3,DMPCNT
        CH     3,=H'0'
        BE     ADENT
        LA     2,DMP TAB
        LH     5,MEPC
DMPY    CH     5,0(2)
        BE     DMPZ
        A      2,=F'2'
        BCT   3,DMPY
        B      ADENT
DMPZ    LA     3,MICROMEM
        MVC   ADRES,=F'0'
        MVC   ADRES+2(2),DMPSTART
        LR    2,3
        A     2,ADRES
        MVC   ADRES+2(2),DMPEND
        A     3,ADRES
        LA    13,SAVE
        L     15,=V(SUBDMP)
        BALR  14,15
ADENT   _H     3,DATCNT
        CH     3,=F'0'
        BC     8,ADPSSF
        MVC   ADRES,=F'0'
        LH     5,MEPC
        CH     5,ATADR
        BC     7,ADPSSF
        LA     4,MICROMEM
        MVC   ADRES+2(2),ENTAD
        A     4,ADRES
        L     5,=A(MICROMEM+65536)
ENTA    MVC   0(1,4),0(5)
        A     4,=F'1'
        A     5,=F'1'
        BCT   3,ENTA
ADPSSF  SR     5,5
        SR    3,3
        CLC   SPSNT(3),=C'000'
        BE     PCSAD
        LH     5,MEPC
        CH     5,PRNTADR
        BC     4,PCSAD
        CH     5,TILADR
        BC     2,PCSAD
SSPRNT  LA     13,SAVE
        CLC   SSTF(3),=C'TNT'
        BC     8,XYPRNT
        CNTRL PRINT,SP,2
        MVC   SSTF(3),=C'TNT'
XYPRNT  L      15,=V(SUBPRINT)
        BALR  14,15
PCSAD   _H     3,ATPC
        CH     3,MEPC
        BC     7,ADDSST
        LH     5,WITHPC
        STH   5,MFPC
ADDSST  C_C    SSCNT(3),=C'000'
        BE     MXAD
SSTPRNT LA     13,SAVE
        CLC   SPTF(3),=C'TBT'
        BC     8,WXPRNT
        CNTRL PRINT,SP,2

```

THE FLAG FOR MEMORY DUMP IS TESTED. IS DUMP REQUIRED AT THIS VALUE OF PROGRAM COUNTER(MFPC). IF NO, BRANCH TO TEST THE FLAG OF THE NEXT COMMAND TABLE OTHERWISE GO TO THE SUBROUTINE(SUBDMP) FOR PRINTING THE MICROPROCESSOR MEMORY DUMP.

THE FLAG IS TESTED WHETHER PROGRAM/ DATA TO BE LOADED IN THE MICROPROCESSOR MEMORY FROM COMMAND TABLE. IF YES, COMPUTE THE LOADING ADDRESS BY ADDING THE GIVEN ADDRESS 'ENTAD' TO STARTING ADDRESS(MICROMEM) OF MICROPROCESSOR MEMORY ON MAINFRAME AND PROGRAM/DATA CODE LOADED ACCORDINGLY IF NO, BRANCH TO TEST THE NEXT FLAG OF THE NEXT COMMAND TABLE.

THE FLAG IS TESTED WHETHER SINGLE STEP PRINTING IS REQUIRED. IF YES, BRANCH TO SUBROUTINE(SUBPRINT) FOR SINGLE STEP PRINTING OTHERWISE GO TO TEST THE FLAG OF THE NEXT COMMAND TABLE.

WHETHER PROGRAM COUNTER(MFPC) IS TO BE SET. IF NO, BRANCH TO PROCESS THE NEXT COMMAND TABLE OTHERWISE SET 'MFPC' WITH ADDRESS STORED IN 'WITHPC'

THE FLAG IS TESTED WHETHER SINGLE STEP PRINTING IS REQUIRED. IF YES, BRANCH TO SUBROUTINE(SUBPRINT) FOR SINGLE STEP PRINTING OTHERWISE GO TO TEST THE FLAG OF THE NEXT COMMAND TABLE.

```

WXPRNT  MVC      BPTF(3),=C'TBT'
        L        15,=V(SUBPRINT)
        BALR     14,15
MXAD    L        5,MAXND
        C        5,=F'0'
        BE      SBPAD
        -        4,PRESD
        A        4,=F'1'
        ST      4,PRESD
        CR      4,5
        BC      2,STOPG
SBPAD   SR      5,5
        SR      3,3
        LH      3,SBPCNT
        CH      3,=H'0'
        BC      8,ZEXT
        LA      2,SBPTAB
        LH      5,MEPC
BSTPA   CH      5,0(2)
        BE      BSTPB
        A        2,=F'2'
        BCT     3,BSTPA
        B        ZEXT
BSTPB   MVC      XL(3),MEPC
        UNPK    PL(5),XL
        MVC     B'X(4),PL
        TR      B'X(4),TRTAB-X'F0'
        MVC     JUTPUT(132),SPACE
        MVC     OUTPUT(90),BPRINT
        CNTRL   PRINT,SP,2
        PUT     PRINT
        MVC     SSTF(3),=C'000'
        MVC     BPTF(3),=C'000'
        B*     NEXT

```

WHETHER MAXIMJM NUMBER OF INSTRUCTION STEPS HAVE BEEN EXECUTED. IF NO, BRANCH TO TEST THE FLAG OF NEXT COMMAND TABLE OTHERWISE STOP SIMULATION GIVING APPROPRIATE MESSAGE.

IS BRAKPOINT ENCOUNTERED. IF NO, BRANCH TO SIMULATE THE NEXT INSTRUCTION OTHERWISE GO TO 'NEXT' TO PROCESS THE NEXT COMMAND SET.

\*\*\*\*\*

```

*
*                               XXXXXXXXXXXXXXXXXXXX
*                               X  ERROR HANDLER  X
*                               XXX XXXXXXXXXXXXXXXXXXX
*

```

```

*   IF AN ERROR OCCURS IN ANY OF THE PROGRAM UNITS THE CONTROL BRANC-
*   HES TO THE ERROR HANDLER. EXECUTION OF THIS SECTION STOPS SIMULA-
*   TION WITH APPROPRIATE ERROR MESSAGE FOR THE USER. THE ERROR HANDLER
*   MAY BE CALSSIFIED INTO TWO CATAGORIES AS FOLLOWS.
*   1) CMMAND PROCESSOR ERROR ROUTINE
*   2) CONTROL PROGRAM ERROR ROUTINE

```

```

*   THE COMMAND PROCESSOR ERROR ROUTINES INCLUDE 'STOPA', 'STOPC',
*   'STOPD', 'STOPE', 'STOPG', 'STOPL', AND CONTROL PROGRAM ERROR
*   ROUTINES INCLUDE 'STOPB' AND 'STOPH'.

```

=====

```

STOPA   MVC      JUTPUT(132),SPACE
        MVC      INV(25),INWORK      ROUTINE FOR ILLEGAL COMMAND.
        MVC      JUTPUT(132),INVCMD
        CNTRL   PRINT,SP,3
        PUT     PRINT
        CLOSE   DISKT,PRINT
        EOJ
STOPB   MVC      TPK(2),0(6)          ROUTINE FOR INVALID OPCODE
        _A     3,MICROMEM
        SR      6,3
        ST      6,AD
        MVC     PK(2),AD+2
        MVC     JKT(3),PK

```

```

UNPK   JKL(5),JKT
MVC    TMP(4),JKL
TR     TMP(4),TRTAB-X'F0'
MVC    PK(2),TPK
UNPK   XL(3),PK
MVC    PK(2),XL
TR     PK(2),TRTAB-X'F0'
CNTRL  PRINT,SP,3
MVC    OUTPUT(132),SPACE
MVC    OUTPUT(60),WPRINT
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPC  MVC    OUTPUT(132),SPACE      ROUTINE FOR ILLEGAL OPTION.
MVC    OUTPUT(70),CPRI T
CNTRL  PRINT,SP,3
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPD  MVC    OUTPUT(132),SPACE      ROUTINE FOR ABSENCE OF COMMAND TER-
MVC    OUTPUT(100),DPRINT          MINATING SYMBOL.
CNTRL  PRINT,SP,3
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPH  MVC    OUTPUT(132),SPACE      ILLEGAL INFORMATION FROM CONSOLE
MVC    OUTPUT(90),HPRINT           FOR 'HLT' INSTRUCTION.
CNTRL  PRINT,SP,3
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPE  MVC    OUTPUT(132),SPACE      ROUTINE DECLEARs THE ERROR MESSAGE
MVC    OUTPUT(100),EPRINT          WHEN NUMBER OF DATA EXCEEDS MICROP-
CNTRL  PRINT,SP,3                  ROCESSOR MEMORY SIZE.
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPG  MVC    OUTPUT(132),SPACE      EXIT ROUTINE FROM ENDLESS LOOP.
MVC    OUTPUT(50),PRINT
CNTRL  PRINT,SP,3
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ
STOPL  MVC    XL(3),JKT              ROUTINE FOR MICROPROCESSOR ADDRESS
UNPK   P_(5),XL                    EXCEPTION.
MVC    LX(4),PL
TR     LX(4),TRTAB-X'F0'
CNTRL  PRINT,SP,3
MVC    OUTPUT(132),SPACE
MVC    OUTPUT(80),LPRINT
PUT    PRINT
CLOSE  DISKT,PRINT
EOJ

```

```

*****
*                                     DECLARATIVES                                     *
*                                     -----                                     *
*=====

```

```

ADNOP  DS      F
DC      A_4(NOP)                    ADDRESS TABLE OF THE PROGRAM
DC      A_4(LXIB)                   SEGMENTS OF MICROPROCESSOR
DC      A_4(STAXB)                  INSTRUCTIONS.
DC      AL4(INXB)
DC      A_4(INRB)
DC      AL4(DCRB)
DC      AL4(MVIB)
DC      AL4(RLC)

```

DC A\_4{STP08}  
DC AL4{DADB}  
DC AL4{LDAXB}  
DC AL4{DCXB}  
DC A\_4{INRC}  
DC AL4{DCRC}  
DC AL4{MVIC}  
DC AL4{RRC}  
DC AL4{STP10}  
DC AL4{LXID}  
DC AL4{STAXD}  
DC A\_4{INXD}  
DC AL4{INRD}  
DC AL4{DCRD}  
DC A\_4{MVID}  
DC AL4{RAL}  
DC AL4{STP18}  
DC AL4{DADD}  
DC AL4{LDAXD}  
DC AL4{DCXD}  
DC AL4{INRE}  
DC AL4{DCRE}  
DC A\_4{MVIE}  
DC AL4{RAR}  
DC A\_4{RIM}  
DC AL4{LXIH}  
DC AL4{SHLD}  
DC A\_4{INXH}  
DC A\_4{INRH}  
DC AL4{DCRH}  
DC AL4{MVIH}  
DC AL4{DAA}  
DC AL4{STP28}  
DC AL4{DADH}  
DC AL4{LHLD}  
DC AL4{DCXH}  
DC AL4{INRL}  
DC A\_4{DCRL}  
DC AL4{MVIL}  
DC AL4{CMA}  
DC A\_4{SIM}  
DC AL4{LXI SP}  
DC A\_4{STA}  
DC AL4{INXSP}  
DC AL4{INRM}  
DC AL4{DCRM}  
DC AL4{MVIM}  
DC AL4{STC}  
DC AL4{STP38}  
DC AL4{DADSP}  
DC AL4{LDA}  
DC A\_4{DCXSP}  
DC A\_4{INRA}  
DC AL4{DCRA}  
DC AL4{MVIA}  
DC AL4{CMC}  
DC AL4{MOVBB}  
DC AL4{MOVBC}  
DC AL4{MOVBD}  
DC AL4{MOVBE}  
DC AL4{MOVBH}  
DC AL4{MOVBL}  
DC AL4{MOVBM}  
DC AL4{MOVBA}  
DC AL4{MOVCB}  
DC AL4{MOVCC}

DC	AL 4 {MOVCD}
DC	AL 4 {MOVCE}
DC	AL 4 {MOVCH}
DC	AL 4 {MOVCL}
DC	AL 4 {MOVCM}
DC	AL 4 {MOVCA}
DC	AL 4 {MOVDB}
DC	AL 4 {MOVDC}
DC	AL 4 {MOVDD}
DC	AL 4 {MOVDE}
DC	AL 4 {MOVDH}
DC	A_4 {MOVDL}
DC	A_4 {MOVDM}
DC	AL 4 {MOVDA}
DC	AL 4 {MOVEB}
DC	AL 4 {MOVEC}
DC	AL 4 {MOVED}
DC	AL 4 {MOVEE}
DC	AL 4 {MOVEH}
DC	AL 4 {MOVEL}
DC	A_4 {MOVEM}
DC	AL 4 {MOVEA}
DC	AL 4 {MOVHB}
DC	AL 4 {MOVHC}
DC	AL 4 {MOVHD}
DC	AL 4 {MOVHE}
DC	AL 4 {MOVHH}
DC	AL 4 {MOVHL}
DC	AL 4 {MOVHM}
DC	AL 4 {MOVHA}
DC	AL 4 {MOVLB}
DC	AL 4 {MOVLC}
DC	AL 4 {MOVLD}
DC	AL 4 {MOVLE}
DC	AL 4 {MOVLH}
DC	AL 4 {MOVLL}
DC	AL 4 {MOVLM}
DC	AL 4 {MOVLA}
DC	AL 4 {MOVMB}
DC	AL 4 {MOVMC}
DC	A_4 {MOVMD}
DC	A_4 {MOVME}
DC	AL 4 {MOVMH}
DC	AL 4 {MOVML}
DC	AL 4 {HLT}
DC	A_4 {MOVMA}
DC	AL 4 {MOVAB}
DC	AL 4 {MOVAC}
DC	AL 4 {MOVAD}
DC	AL 4 {MOVAE}
DC	AL 4 {MOVAH}
DC	AL 4 {MOVAL}
DC	AL 4 {MOVAM}
DC	A_4 {MOVAA}
DC	AL 4 {ADDB}
DC	AL 4 {ADDC}
DC	AL 4 {ADDD}
DC	AL 4 {ADDE}
DC	AL 4 {ADDH}
DC	AL 4 {ADDL}
DC	AL 4 {ADDM}
DC	AL 4 {ADDA}
DC	AL 4 {ADCB}
DC	AL 4 {ADCC}
DC	AL 4 {ADCD}
DC	AL 4 {ADCE}

DC	AL 4 { ADCH }
DC	AL 4 { ADCL }
DC	A_ 4 { ADCM }
DC	AL 4 { ADCA }
DC	AL 4 { SUBB }
DC	AL 4 { SUBC }
DC	AL 4 { SUBD }
DC	AL 4 { SUBE }
DC	AL 4 { SUBH }
DC	AL 4 { SUBL }
DC	AL 4 { SUBM }
DC	AL 4 { SUBA }
DC	AL 4 { SBBB }
DC	AL 4 { SBBC }
DC	AL 4 { SBBD }
DC	AL 4 { SBBE }
DC	AL 4 { SBBH }
DC	AL 4 { SBBL }
DC	AL 4 { SBBM }
DC	AL 4 { SBBA }
DC	AL 4 { ANAB }
DC	AL 4 { ANAC }
DC	AL 4 { ANAD }
DC	AL 4 { ANAE }
DC	AL 4 { ANAH }
DC	AL 4 { ANA_ }
DC	AL 4 { ANAM }
DC	AL 4 { ANAA }
DC	AL 4 { XRAB }
DC	AL 4 { XRAC }
DC	AL 4 { XRAD }
DC	AL 4 { XRAE }
DC	AL 4 { XRAH }
DC	AL 4 { XRA_ }
DC	AL 4 { XRAM }
DC	AL 4 { XRAA }
DC	AL 4 { ORAB }
DC	AL 4 { ORAC }
DC	AL 4 { ORAD }
DC	AL 4 { ORAE }
DC	AL 4 { ORAH }
DC	AL 4 { ORAL }
DC	AL 4 { ORAM }
DC	AL 4 { ORAA }
DC	AL 4 { CMPB }
DC	AL 4 { CMPC }
DC	AL 4 { CMPD }
DC	AL 4 { CMPE }
DC	AL 4 { CMPL }
DC	A_ 4 { CMPL }
DC	AL 4 { CMPH }
DC	AL 4 { CMPM }
DC	AL 4 { CMPA }
DC	AL 4 { RNZ }
DC	AL 4 { POPB }
DC	AL 4 { JNZ }
DC	AL 4 { JMP }
DC	AL 4 { CNZ }
DC	AL 4 { PUSHB }
DC	AL 4 { ADI }
DC	AL 4 { RST0 }
DC	AL 4 { RZ }
DC	AL 4 { RET }
DC	AL 4 { JZ }
DC	AL 4 { STPCB }
DC	AL 4 { CZ }
DC	AL 4 { CALL }



DC	AL4{ACI}
DC	AL4{RST1}
DC	AL4{RNC}
DC	AL4{POPD}
DC	AL4{JNC}
DC	AL4{OUT}
DC	AL4{CNC}
DC	AL4{PUSHD}
DC	AL4{SUI}
DC	AL4{RST2}
DC	AL4{RC}
DC	AL4{STPD9}
DC	AL4{JC}
DC	AL4{IN}
DC	AL4{CC}
DC	AL4{STPDD}
DC	AL4{SBI}
DC	AL4{RST3}
DC	AL4{RPO}
DC	AL4{POPH}
DC	AL4{JPD}
DC	AL4{XTHL}
DC	AL4{CPD}
DC	AL4{PUSHH}
DC	AL4{ANI}
DC	AL4{RST4}
DC	AL4{RPE}
DC	AL4{PCHL}
DC	AL4{JPE}
DC	AL4{XCHG}
DC	AL4{CPE}
DC	AL4{STPED}
DC	AL4{XRI}
DC	AL4{RST5}
DC	AL4{RP}
DC	AL4{POPPSW}
DC	AL4{JP}
DC	AL4{DI}
DC	AL4{CP}
DC	AL4{PUSHPSW}
DC	AL4{ORI}
DC	AL4{RST6}
DC	AL4{RM}
DC	AL4{SPHL}
DC	AL4{JM}
DC	AL4{EI}
DC	AL4{CM}
DC	AL4{STPFD}
DC	AL4{CPI}
DC	AL4{RST7}
DC	AL4{ADPORT}
DC	AL4{BEND}
DC	AL4{BEXEC}
DC	AL4{BPDMP}
DC	AL4{BPSET}
DC	AL4{OPTION}
DC	AL4{BENTRT}
DC	AL4{BPSETT}
DC	AL4{BENTRF}
DC	AL4{STAT09}
DC	AL4{SETREG}
DC	AL4{STAT11}
DC	AL4{BDUMP}
DC	AL4{MXINST}
DC	AL4{STAT14}
DC	AL4{STAT15}

DRST7  
CMNADTAB

ADDRESS TABLE OF COMMAND VALIDITY  
CHECKER&EXECUTOR.

	DC	AL4{STAT16}
	DC	AL4{STPBP}
	DC	AL4{PRNTBP}
	DC	AL4{PRSTEPP}
	DC	AL4{PRSSTF}
SAVE	DS	18F
AD	DC	F'0'
S3	DC	F'0'
HDUS	DC	H'0'
CYC	DC	H'0'
PRESND	DC	F'0'
INPT	DS	CL4
DJ5PT	DS	CL4
DJTCN	DS	CL4
CYY	DC	X'0'
RA	DC	X'0'
PCC	DS	CL3
X	DS	CL3
PL	DS	CL5
ZERO	DS	F'0'
TR AB	DC	X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'
FLAG	DC	X'0'
MFA	DC	X'0'
MFB	DC	X'0'
MFC	DC	X'0'
MFD	DC	X'0'
MFE	DC	X'0'
	DS	CL2
MFH	DC	X'0'
	DS	CL3
MFL	DC	X'0'
MFS	DC	X'0'
MFZ	DC	X'0'
MFAC	DC	X'0'
MFP	DC	X'0'
MFCY	DC	X'0'
MF SW	DC	X'0'
MK	DC	F'0'
JADD	DS	F
P3D	DC	X'0'
T3K	DS	CL2
	DS	CL3
PTL	DS	CL3
MASK	DC	X'00'
TTAB	DC	X'FABFCFDFF'
	DC	C'
	DC	X'F0F1F2F3F4F5F6F7F8F9'
KL T	DC	F'0'
ADRES	DC	F'0'
SQ RCE	DC	F'0'
AK	DS	CL2
ADD	DC	H'0'
SJB	DC	H'0'
MFSP	DC	H'0'
MEPC	DC	H'0'
M	DS	X
N	DS	X
MFPC	DC	H'0'
INPUT	DS	CL80
	DC	C'
DJTPUT	DS	CL132
INWORK	DS	CL80
OPRINT	DS	CL60
	DC	C'
	DC	8085 ACCUMULATOR CONTENT OUTPUTTING FOR PORT'
XB	DS	CL2
	DC	C' IS'

XC	DS	CL2
	DC	20C' '
C>PRINT	DS	0CL70
	DC	C'*****ERROR MESSAGE-'
	DC	C' ILLEGAL OPTION OR OPTION MISSING*****'
	DC	20C' '
L>PRINT	DS	0CL80
	DC	C'*****ERROR MESSAGE-'
	DC	C' 8085 MICROPROCESSOR ADDRESS '
LX	DS	CL4
	DC	C' EXCEPTION*****'
	DC	30C' '
LAD	DS	F
JAD	DS	D
KAD	DS	H
NAD	DS	CL3
W>PRINT	DS	0CL60
	DC	C'*****ERROR MESSAGE-'
PK	DS	CL2
	DC	C' IS INVALID OPCODE MA='
TMP	DS	CL4
	DC	C'*****'
	DC	20C' '
INVCMD	DS	0CL132
	DC	C'*****ERROR MESSAGE-'
INV	DS	CL25
	DC	C' INVALID COMMAND*****'
S>PACE	DC	132C' '
J<	DS	CL4
J<L	DS	CL5
J<T	DS	CL3
	DS	F
D>CA	DC	H'0'
D>CB	DC	H'0'
	DS	H
ENTAD	DC	H'0'
ATADR	DC	H'0'
DATCNT	DC	F'0'
DATADR	DC	F'0'
E>PRINT	DS	0CL100
	DC	C'*****ERROR MESSAGE- NO. OF DATA IN ENTRY MODE '
	DC	C' EXCEEDS MICROPROCESSOR MEMORY *****'
	DC	20C' '
	DS	H
ATPC	DC	H'0'
WITHPC	DC	H'0'
ENTADF	DS	H'0'
UP>TIOPC	DC	H'0'
H>PRINT	DS	0CL90
	DC	C'*****ERROR MESSAGE- DATA FROM CONSOLE FOR HALT IS '
	DC	C' ILLEGAL OR MISSING*****'
	DC	20C' '
B>PRINT	DS	0CL90
	DC	49C' '
	DC	4C'\$'
	DC	C' BREAKPOINT OCCURED AT-'
B>X	DS	CL4
	DC	4C'\$'
	DC	50C' '
S>SNT	DC	C'000'
SS NT	DC	C'000'
	DS	F
DMPSTART	DS	H
DMFEND	DS	H
DMPCNT	DC	H'0'
DMPTAB	DS	XL60

```

MX          DS      F
MAXNO      DC      D'0'
SBPCNT     DC      H'0'
PBPCNT     DC      H'0'
PRNTADR    DC      H'0'
TILADR     DC      H'0'
PBPTAB     DS      XL60
SBPTAB     DS      XL60
DPRINT     DS      0CL100
           DC      C'*****ERROR MESSAGE- DATA IN ENTRY MODE IS EITHER '
           DC      C'BLANK OR NOT ENDED WITH DOLLAR($) SIGN*****'
           DC      20C' '
GPRINT     DS      0CL50
           DC      C'*****ERROR MESSAGE- EXECUTION EXCEEDS LIMIT*****'
           DC      10C' '
SSTF       DC      C'000'
BPTF       DC      C'000'
           LTORG
MICROMEM   DS      64XL1024
           END      BEGIN

```

\*\*\*\*\*

```

XXXXXXXXXXXXX
X ACFLGI X
XXXXXXXXXXXXX

```

```

* THIS IS A SUBROUTINE PROGRAM. THE AUXILIARY CARRY FLAG IS SET IN *
* THIS PROGRAM SEGMENT FOR ADD AND INCREMENT OPERATIONS AMONG OPER- *
* ANDS. THESE OPERANDS MAY BE REGISTERS OR MEMORY CONTENTS OR IMME- *
* DIATE DATA. THE OPERATION IS PERFORMED ON THE LEAST SIGNIFICANT *
* NIBBLE OF 1 BYTE DATA. EACH DATA IS INSERTED INTO 2 BYTES OF MEM- *
* ORY AREA FOR HALF-WORD OPERATION. ALL THE NIBBLES EXCEPT THE LEAST *
* SIGNIFICANT ONE IS MADE ZERO AND THEN ADDED. THE FIFTH BIT OF THE *
* RESULT FROM THE RIGHT IS ACCOUNTED WHICH REPRESENTS THE ACTUAL VAL- *
* UE OF AUXILIARY CARRY(MFAC). THE AUXILIARY CARRY IS EITHER SET TO 1 *
* OR 0 ACCORDING TO THE STATUS OF THAT BIT. *
*
=====

```

```

ACFLGI      CSECT
           PRINT NOGEN
           STM 14,12,12(13)
BEGIN       BALR 12,0
           USING *,12
           ST 13,SAVE+4
           L 5,0(3)
           MVC ADD(2),0(5)
           L 5,04(3)
           MVC SUB(2),0(5)
           L 5,08(3)
           MVC CYY+1(1),0(5)
           NC CYY,=X'0001'
           NC ADD,=X'000F'
           NC SUB,=X'000F'
           L 9,=F'0'
           LH 9,ADD
           AH 9,SUB
           AH 9,CYY
           STH 9,ADD
           NC ADD,=X'0010'
           LH 9,ADD
           CH 9,=H'0'
           BE AC1
           MJC AC(1),=X'01'

```

```

ALL THE REGISTERS ARE SAVED.
R12 IS USED AS BASE REGISTER.

THE OPERANDS OF ALU OPERATION IS
EXTRACTED FROM MAIN PROGRAM AND
STORED AT THE ADDRESS OF 'ADD' AND
'SUB'. CARRY FLAG(MFCY) IS STORED IN
'CYY'. ALL THE NIBBLES OF THESE THR-
EE EXCEPT THE LEAST SIGNIFICANT ONE
MADE 0. (ADD) IS LOADED IN R9 THROU-
GH HALF-WORD OPERATION AND IS ADDED
WITH 'SUB' AND 'CYY'. THE FIFTH
BIT OF THE RESULT FROM THE RIGHT
CHECKED. IF IT IS ZERO 'MFAC' IS SET
TO 0 OTHERWISE IT IS SET TO 1.

```

```

AC1      B      AC2
AC2      MVC    AC(1),=X'00'
          LA    3,=A(AC)
          -    13,SAVE+4
          LM   14,2,12(13)
          LM   4,12,36(13)
          BR   14

```

THE REGISTER CONTENTS ARE RESTORED AND FINALLY RETURNS TO THE CALLING PROGRAM.

```

*          DECLARATIVES
SAVE     DS    18F
          DS    F
ADD      DC    H'0'
SJB      DC    H'0'
CYY      DC    H'0'
AC       DC    X'0'
          END  BEGIN

```

\*\*\*\*\*

```

*          XXXXXXXXXXXX
*          X ACFLGD X
*          XXXXXXXXXXXX
*
*
*
*
*

```

```

* THIS IS A SUBROUTINE PROGRAM. THE AUXILIARY CARRY IS SET IN THIS
* PROGRAM SEGMENT FOR SUBTRACT AND DECREMENT OPERATION. THE DATA MAY
* BE THE REGISTER OR MEMORY CONTENTS OR IMMEDIATE DATA. THE OPERATION
* IS PERFORMED ON THE LEAST SIGNIFICANT NIBBLE OF 1 BYTE DATA. EACH
* DATA IS INSERTED INTO 2 BYTE OF MEMORY AREA FOR HALF-WORD OPERATION
* AL- THE NIBBLES EXCEPT THE LEAST SIGNIFICANT ONE IS MADE ZERO AND
* THEN SUBTRACTED. THE FIFTH BIT OF THE RESULT FROM THE RIGHT IS ACC-
* QUNTED WHICH REPRESENTS THE ACTUAL VALUE OF AJXILIARY CARRY(MFAC).
* THE AUXILIARY CARRY IS SET TO 1 OR 0 ACCORDING TO THE STATUS OF
* THAT BIT.
*
*
*

```

=====

```

ACFLGD   CSECT
          PRINT NOGEN
          STM   14,12,12(13)
BEGIN    BALR  12,0
          USING *,12
          ST   13,SAVE+4
          -   5,0(3)
          MVC ADD(2),0(5)
          -   5,04(3)
          MVC SUB(2),0(5)
          L   5,08(3)
          MVC CYY+1(1),0(5)
          VC  CYY,=X'0001'
          NC  ADD,=X'000F'
          VC  SUB,=X'000F'
          -   9,=F'0'
          -H  9,ADD
          SH  9,SUB
          SH  9,CYY
          STH 9,ADD
          NC  ADD,=X'0010'
          LH  9,ADD
          CH  9,=H'0'
          BE  AC1
          MVC AC(1),=X'01'
          B   AC2
AC1      MVC    AC(1),=X'00'
AC2      LA    3,=A(AC)
          -    13,SAVE+4
          LM   14,2,12(13)
          LM   4,12,36(13)
          BR   14

```

ALL THE REGISTERS ARE SAVED. R12 IS USED AS BASE REGISTER.

THE OPERANDS OF ALU OPERATION ARE EXTRACTED FROM MAIN PROGRAM AND STORED AT THE ADDRESS OF 'ADD' AND 'SUB'. CARRY FLAG(MFCY) IS STORED IN CYY. ALL THE NIBBLES OF THESE THREE EXCEPT THE LEAST SIGNIFICANT IS MADE 0. (ADD) IS LOADED IN R9 THROUGH HALF-WORD OPERATION AND OTHER TWO ARE SUBTRACTED FROM IT. THE FIFTH BIT OF THE RESULT FROM THE RIGHT IS CHECKED. IF IT IS ZERO 'MFAC' IS SET TO 0 OTHERWISE IT IS SET TO 1.

THE REGISTER CONTENTS ARE RESTORED AND FINALLY RETURNS TO CALLING PROGRAM.

```

*      DECLARATIVES
SAVE   DS      18F
       DS      F
ADD    DC      H'0'
SJB    DC      H'0'
CY     DC      H'0'
AC     DC      X'0'
       END     BEGIN

```

\*\*\*\*\*

```

*
*
*      XXXXXXXXXXXXX
*      X SUBFLG X
*      XXXXX)XXXXXX
*
*

```

THIS IS THE SUBROUTINE WHICH SETS ZERO FLAG(MFZ), SIGN FLAG(MFS), PARITY FLAG(MFP) AND CARRY FLAG(MFCY). THE RESULT OF THE ALU OPERATION IS USED TO DETERMINE THE THESE FLAGS. THE ACCUMULATOR CONTENTS IS INSERTED INTO A HALF-WORD MEMORY LOCATION 'HOUS'.

=====

```

SJBFLG  CSECT
        PRINT  NOGEN
        STM    14,12,12(13)      ALL THE REGISTERS ARE SAVED.
BEGIN    BALR  12,0              R12 IS USED AS BASE REGISTER.
        USING *,12
        ST     13,SAVE+4
        L     4,0(3)
        MVC   HJUS(2),0(4)      ACCUMULATOR CONTENTS ARE EXTRACTED
        L     4,04(3)          FROM MAIN PROGRAM AND STORED IN THE
        MVC   CYC(2),0(4)      HALF-WORD LOCATION 'HOUS'. ALL THE
        MVC   HJUSS(2),HOUS    BITS OF THE ACCUMULATOR CONTENTS ARE
        NC    HJUSS,=X'0080'   MADE ZERO EXCEPT THE LEAST SIGNIFIC-
        LH    9,HOUSS          ANT EIGHTH BIT WHICH REPRESENTS THE
        CH    9,=H'0'         SIGN FLAG. 'MFS' IS SET TO 1 IF THIS
        BE    S1              BIT IS 1 OTHERWISE IT IS SET TO 0.
        MVC   S(1),=X'01'
        B     S2
S1      MVC   S(1),=X'00'
S2      MVC   HOUSZ(2),HOUS
        L     4,=F'0'
        LH    4,HOUSZ
        CH    4,=H'0'
        BE    Z1
        MVC   Z(1),=X'00'
        B     Z2
Z1      MVC   Z(1),=X'01'
Z2      LH    4,CYC
        CH    4,=X'FFF0'
        BC    8,PAR1
        MVC   HOUSC(2),HOUS
        L     4,=F'0'
        NC    HOUSC,=X'0100'
        LH    4,HOUSC
        CH    4,=H'0'
        BE    CY1
        MVC   CY(1),=X'01'
        B     CY2
CY1     MVC   CY(1),=X'00'
CY2     LA    4,=A(CY)
PAR1    L     10,=F'8'
        L     5,=F'0'
        L     7,=F'0'
        L     9,=F'0'
        MVC   HJUSP(2),HOUS

```

THE NINETH BIT OF 'HOUS' CONTAINING THE ALU OPERATION IS TESTED, IF IT IS 1 CARRY FLAG IS SET TO 1 OTHERWISE IT IS SET TO 0.

THE STATUS OF PARITY FLAG IS TESTED BY CHECKING EACH BIT OF THE ACCUMULATOR.



```

OPEN PRINT
MVC KLT,=F'0'
- 2,=A(MFSP)
MVC SPK(2),0(2)
LA 8,STK
MVC KLT+2(2),SPK
- 4,=A(MICROMEM)
-R 3,4
A 3,KLT
L 7,=F'4'
SPW MVC PK(2),0(3)
UNPK XL(3),PK
MVC PK(2),XL
TR PK(2),TRTAB-X'F0'
MVC 0(2,8),PK
A 8,=F'3'
A 3,=F'1'
BCT 7,SPW
MVC ML(3),SPK
UNPK SM(5),ML
MVC SP(4),SM
TR SP(4),TRTAB-X'F0'
MVC PCK(2),2(2)
MVC KLT+2(2),PCK
A 4,KLT
-A 8,CODE
L 7,=F'8'
PCW MVC PK(2),0(4)
JNPK XL(3),PK
MVC PK(2),XL
TR PK(2),TRTAB-X'F0'
MVC 0(2,8),PK
A 8,=F'3'
A 4,=F'1'
BCT 7,PCW
MVC ML(3),PCK
UNPK SM(5),ML
MVC PC(4),SM
TR PC(4),TRTAB-X'F0'
-A 8,REG
L 2,=A(MFA)
L 7,=F'5'
REGW MVC PK(2),0(2)
JNPK XL(3),PK
MVC PK(2),XL
TR PK(2),TRTAB-X'F0'
MVC 0(2,8),PK
A 8,=F'5'
A 2,=F'1'
BCT 7,REGW
A 2,=F'2'
MVC PK(2),0(2)
UNPK XL(3),PK
MVC PK(2),XL
TR PK(2),TRTAB-X'F0'
MVC 0(2,8),PK
A 8,=F'5'
A 2,=F'4'
MVC PK(2),0(2)
UNPK XL(3),PK
MVC PK(2),XL
TR PK(2),TRTAB-X'F0'
MVC 0(2,8),PK
A 2,=F'1'
L 7,=F'5'
LA 8,FLAG

```

THE DATA CONTAINED IN STACK ARE UNPACKED AND CONVERTED TO EBCDIC FORM AND STORED AT THE ADDRESS CONTAINED IN R8.

THE STACK POINTING ADDRESS IS UNPACKED AND CONVERTED TO EBCDIC MODE IN 'SP'.

THE INSTRUCTION CODE STARTING FROM ADDRESS INDICATED BY THE PROGRAM COUNTER IS UNPACKED AND CONVERTED TO EBCDIC MODE AND STORED AT ADDRESS CONTAINED IN R8.

THE PROGRAM COUNTER CONTENTS IS UNPACKED AND CONVERTED TO EBCDIC MODE IN 'PC'.

THE CONTENTS OF MICROPROCESSOR GENERAL PURPOSE REGISTERS AND ACCUMULATOR ARE UNPACKED AND CONVERTED TO EBCDIC MODE AND STORED AT THE ADDRESS CONTAINED R8.



```

F_FLAG MVC PK(2),0(2)
        UNPK XL(3),PK
        MVC PK(2),XL
        TR PK(2),TRTAB-X'F0'
        MVC 0(1,8),PK+1
        A 8,=F'5'
        A 2,=F'1'
        BCT 7,FLAGW
        MVC JUTPUT(132),SPACE
        MVC OUTPUT(132),SST
        PUT PRINT
        CLOSE PRINT
        L 13,SAVE+4
        -M 14,12,12(13)
        BR 14
XL DS CL3
PK DS CL2
SAVE DS 18F
KL1 DC F'0'
SPK DS C_2
ML DS CL3
SM DS CL5
SST DS 0CL132
DC C' PC='
PC DS CL4
DC C'- '
CJDE DS CL2
DC C' '
DS C_2
DC C' '
DS C_2
DC C' '
DS CL2
DC C' '
DS C_2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' '
DS CL2
DC C' A='
REG DS CL2
DC C' B='
DS CL2
DC C' C='
DS CL2
DC C' D='
DS C_2
DC C' E='
DS CL2
DC C' H='
DS C_2
DC C' L='
DS CL2
DC C' S='
F_FLAG DS CL1
DC C' Z='
DS C_1
DC C' AC='
DS CL1
DC C' P='
DS C_1
DC C' CY='
DS C_1
DC C' SP='

```

ALL THE FLAG VALUES ARE UNPACKED AND CONVERTED TO EBCDIC MODE.

THE REGISTERS CONTENTS ARE RESTORED AND RETURNS TO CALLING PROGRAM.

RECORD FOR SINGLE STEP PRINTING.

```

SP      DS      CL4
        DC      C'-'
STK     DS      CL2
        DC      C' '
        DS      CL2
        DC      C' '
        DS      CL2
        DC      C' '
        DS      CL2
        DC      132C' '
SPACE  TRTAB   DC      X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C'
PCK     DS      C_12
PRINT  DTFPR   DEVADDR=SYSLST,IOAREA1=OUTPUT,BLKSIZE=132,CONTROL=YES
        PRMOD  CCONTROL=YES
OUTPUT DS      CL132
        END    BEGIN

```

```

*****
*
*                               XXXXXXXXXXXXX
*                               X  SUBDMP  X
*                               XXXXXXXXXXXXX
*
* THIS SUBROUTINE IS DEVELOPED FOR PRINTING THE DUMP OF THE MICROPROCESSOR-
* MEMORY DESIGNATED IN MAINFRAME. IN THE MAIN PROGRAM THE
* DUMP START ADDRESS HAS BEEN LOADED IN R2 AND DUMP END ADDRESS IN R3
* THE DATA CONTAINED IN THE MEMORY BETWEEN THESE ADDRS ARE SHOWN IN
* THE DUMP PRINT.
*
=====

```

```

SJBDM  CSECT
        PRINT  NOGEN
        EXTRN MICROMEM
BEGIN   STM     14,12,12(13)      ALL THE REGISTERS ARE SAVED.
        BA_R   12,0              R12 IS USED AS BASE REGISTER.
        USING  *,12
        ST     13,SAVE+4
        OPEN  PRINT
        CNTRL PRINT,SP,2
        MVC   JUTPUT(132),SPACE  THE FIRST LINE IN THE MEMORY DUMP
        MVC   JUTPUT(100),SLN    CONTAINS THE SERIAL NO. IS PRINTED.
        PUT   PRINT
        LR    5,2
        L     8,=A(MICROMEM)
        SR    5,8
MEMD1   MVC   MEMN(17),DUMP
        STH   5,PKC
        MVC   M_(3),PKC
        UNPK  SM(5),ML
        MVC   PC(4),SM
        TR    PC(4),TRTAB-X'F0'
        L     7,=F'8'
        SR    4,4
MEMD2   LA    8,MEMW
        MVC   PK(2),0(2)
        UNPK  XL(3),PK
        MVC   PK(2),XL
        TR    PK(2),TRTAB-X'F0'
        MVC   0(2,8),PK
        A     4,=F'1'
        A     8,=F'3'
        A     2,=F'1'
        BCT  7,MEMD2
        C     4,=F'16'
        BC   8,MEMD3
        A     8,=F'3'

```

	L	7,=F'8'	
	B	MEMD2	
MEMD3	MVC	OUTPUT(132),SPACE	
	MVC	OUTPUT(95),DUMP	
	PUT	PRINT	
	CR	2,3	
	BC	2,MEMD4	
	A	5,=F'16'	
	MVC	MEMN(17),SPACE	
	B	MEMD1	
MEMD4	CLOSE	PRINT	
	_I	13,SAVE+4	REGISTERS ARE RESTORED AND BRANCHES TO THE CALLING PROGRAM.
	LM	14,12,12(13)	
	BR	14	
OUTPUT	DS	C_132	OUTPUT WORK AREA FOR PRINT.
PK	DS	H	
DJMP	DC	C'MEMORY DUMP	
SAVE	DS	18F	REGISTERS SAVE AREA.
M_	DS	CL3	
SM	DS	CL5	
PK	DS	CL2	
X_	DS	CL3	
PRINT	DTFPR	DEVADDR=SYSLST,IOAREA1=OUTPUT,BLKSIZE=132,CONTROL=YES	
	PRMOD	CONTROL=YES	
S_NO	DS	0C_100	THE RECORD OF THE FIRST LINE OF THE MEMORY DUMP.
	DC	39C' '	
	DC	C'0' '	
	DC	2C' '	
	DC	C'1' '	
	DC	2C' '	
	DC	C'2' '	
	DC	2C' '	
	DC	C'3' '	
	DC	2C' '	
	DC	C'4' '	
	DC	2C' '	
	DC	C'5' '	
	DC	2C' '	
	DC	C'6' '	
	DC	2C' '	
	DC	C'7' '	
	DC	5C' '	
	DC	C'8' '	
	DC	2C' '	
	DC	C'9' '	
	DC	2C' '	
	DC	C'A' '	
	DC	2C' '	
	DC	C'B' '	
	DC	2C' '	
	DC	C'C' '	
	DC	2C' '	
	DC	C'D' '	
	DC	2C' '	
	DC	C'E' '	
	DC	2C' '	
	DC	C'F' '	
	DC	20C' '	
DJMPL	DS	0CL95	RECORD FOR MEMORY DUMP PRINT.
	DC	13C' '	
MEMN	DS	CL17	
	DC	C'MA=''	
PC	DS	CL4	
	DC	C' '	
MEMW	DS	CL2	
	DC	C' '	



```

MVC      KLT+3(1),TPK
L        5,KLT
M        4,=F'4'
AR       7,5
L        7,0(7)
A        8,=F'8'
BR       7
PJRT00  - 4,=F'23'
         LA 5,ADP00
         ST 5,AD00
A31      MVC TPK(2),0(8)
         CLC TPK(2),=C'$$'
         BC 8,AB2
         TR TPK(2),TTAB-X'C1'
         MVC PTL(3),TPK
         PACK TPK(2),PTL
         CLI TPK,X'FF'
         BC 2,ST0PB
         MVC 0(1,5),TPK
         A 5,=F'1'
         A 8,=F'3'
A32      BCT 4,AB1
         ST 3,CNT00
         L 13,SAVE+4
         LM 14,12,12(13)
         BR 14
PJRT01  - 4,=F'23'
         LA 5,ADP01
         ST 5,AD01
BB1      MVC TPK(2),0(8)
         C_C TPK(2),=C'$$'
         BC 8,BB2
         TR TPK(2),TTAB-X'C1'
         MVC PTL(3),TPK
         PACK TPK(2),PTL
         CLI TPK,X'FF'
         BC 2,ST0PB
         MVC 0(1,5),TPK
         A 5,=F'1'
         A 8,=F'3'
B32      BCT 4,BB1
         ST 3,CNT01
         L 13,SAVE+4
         LM 14,12,12(13)
         BR 14
PJRT02  - 4,=F'23'
         LA 5,ADP02
         ST 5,AD02
C31      MVC TPK(2),0(8)
         CLC TPK(2),=C'$$'
         BC 8,CB2
         TR TPK(2),TTAB-X'C1'
         MVC PTL(3),TPK
         PACK TPK(2),PTL
         C_I TPK,X'FF'
         BC 2,ST0PB
         MVC 0(1,5),TPK
         A 5,=F'1'
         A 8,=F'3'
C32      BCT 4,CB1
         ST 3,CNT02
         L 13,SAVE+4
         LM 14,12,12(13)
         BR 14
PJRT03  - 4,=F'23'
         LA 5,ADP03

```

ALL THE DATA ARE PROCESSED SERIALLY.  
EACH OF THE POR DATA IS CONVERTED  
TO BINARY AND PACKED AND STORED IN  
THE COMMAND TABLE.

THE REGISTERS CONTENTS ARE RESTORED  
AND RETURNS TO THE CALLING PROGRAM.

DB1	ST	5,PAD03
	MVC	TPK(2),0(8)
	C_C	TPK(2),=C'\$\$'
	BC	8,DB2
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,DB1
D32	ST	3,CNT03
	-	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT04	-	4,=F'23'
	LA	5,ADP04
	ST	5,PAD04
EB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,EB2
	CLI	TPK,C' '
	BC	8,STOPD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,EB1
EB2	ST	3,CNT04
	-	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT05	L	4,=F'23'
	LA	5,ADP05
	ST	5,PAD05
FB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,FB2
	CLI	TPK,C' '
	BC	8,STOPD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,FB1
FB2	ST	3,CNT05
	L	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT06	-	4,=F'23'
	LA	5,ADP06
	ST	5,PAD06
GB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,GB2
	CLI	TPK,C' '

	BC	8, STOPD
	TR	TPK(2), TTAB-X'C1'
	MVC	PTL(3), TPK
	PACK	TPK(2), PTL
	C_I	TPK, X'FF'
	BC	2, STOPB
	MVC	0(1,5), TPK
	A	5, =F'1'
	A	8, =F'3'
GB2	BCT	4, GB1
	ST	3, CNT06
	L	13, SAVE+4
	LM	14, 12, 12(13)
	BR	14
PORT07	-	4, =F'23'
	LA	5, ADP07
	ST	5, PAD07
HB1	MVC	TPK(2), 0(8)
	C_C	TPK(2), =C'\$\$'
	BC	8, HB2
	CLI	TPK, C' '
	BC	8, STOPD
	TR	TPK(2), TTAB-X'C1'
	MVC	PTL(3), TPK
	PACK	TPK(2), PTL
	CLI	TPK, X'FF'
	BC	2, STOPB
	MVC	0(1,5), TPK
	A	5, =F'1'
	A	8, =F'3'
HB2	BCT	4, HB1
	S1	3, CNT07
	-	13, SAVE+4
	LM	14, 12, 12(13)
	BR	14
PORT08	L	4, =F'23'
	LA	5, ADP08
	ST	5, PAD08
JB1	MVC	TPK(2), 0(8)
	CLC	TPK(2), =C'\$\$'
	BC	8, JB2
	CLI	TPK, C' '
	BC	8, STOPD
	TR	TPK(2), TTAB-X'C1'
	MVC	PTL(3), TPK
	PACK	TPK(2), PTL
	C_I	TPK, X'FF'
	BC	2, STOPB
	MVC	0(1,5), TPK
	A	5, =F'1'
	A	8, =F'3'
JB2	BCT	4, JB1
	ST	3, CNT08
	L	13, SAVE+4
	-M	14, 12, 12(13)
	BR	14
PORT09	L	4, =F'23'
	LA	5, ADP09
	ST	5, PAD09
KB1	MVC	TPK(2), 0(8)
	CLC	TPK(2), =C'\$\$'
	BC	8, KB2
	CLI	TPK, C' '
	BC	8, STOPD
	TR	TPK(2), TTAB-X'C1'
	MVC	PTL(3), TPK

	PACK	TPK(2),PTL
	C_I	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,KB1
KB2	ST	3,CNT09
	L	13,SAVE+4
	_M	14,12,12(13)
	BR	14
PORT0A	L	4,=F'23'
	_A	5,ADP0A
	ST	5,PAD0A
LB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,LB2
	CLI	TPK,C' '
	BC	8,STOPD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CL_I	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,LB1
LB2	ST	3,CNT0A
	L	13,SAVE+4
	_M	14,12,12(13)
	BR	14
PORT0B	L	4,=F'23'
	_A	5,ADP0B
	ST	5,PAD0B
MB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,MB2
	CLI	TPK,C' '
	BC	8,STOPD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CL_I	TPK,X'FF'
	BC	2,STOPB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
	BCT	4,MB1
MB2	ST	3,CNT03
	L	13,SAVE+4
	_M	14,12,12(13)
	BR	14
PORT0C	L	4,=F'23'
	_A	5,ADP0C
	ST	5,PAD0C
NB1	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,NB2
	CLI	TPK,C' '
	BC	8,STOPD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CL_I	TPK,X'FF'
	BC	2,STOPB



	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
NB2	BCT	4,NB1
	ST	3,CNT0C
	L	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT0D	L	4,=F'23'
	LA	5,ADP0D
	ST	5,PA00D
P31	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,PB2
	CLI	TPK,C' '
	BC	8,ST0PD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,ST0PB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
PB2	BCT	4,PB1
	ST	3,CNT0D
	L	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT0E	L	4,=F'23'
	LA	5,ADP0E
	ST	5,PA00E
Q31	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,QB2
	CLI	TPK,C' '
	BC	8,ST0PD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,ST0PB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'
QB2	BCT	4,QB1
	ST	3,CNT0E
	L	13,SAVE+4
	LM	14,12,12(13)
	BR	14
PJRT0F	L	4,=F'23'
	LA	5,ADP0F
	ST	5,PA00F
R31	MVC	TPK(2),0(8)
	CLC	TPK(2),=C'\$\$'
	BC	8,RB2
	CLI	TPK,C' '
	BC	8,ST0PD
	TR	TPK(2),TTAB-X'C1'
	MVC	PTL(3),TPK
	PACK	TPK(2),PTL
	CLI	TPK,X'FF'
	BC	2,ST0PB
	MVC	0(1,5),TPK
	A	5,=F'1'
	A	8,=F'3'

```

R32      BCT      4,RB1
         ST       3,CNTOF
         L        13,SAVE+4
         LM       14,12,12(13)
STOPB    BR       14
         MVC      PK(2),TPK          ERROR ROUTINE FOR INVALID OPCODE.
         UNPK    X_(3),PK
         MVC      PK(2),XL
         TR      PK(2),TRTAB-X*F0*
         CNTRL   PRINT,SP,2
         MVC     OUTPUT(132),SPACE
         MVC     OUTPUT(80),BPRINT
         PUT     PRINT
         CLOSE   PRINT
STOPD    EOJ
         MVC     OUTPUT(132),SPACE
         MVC     OUTPUT(100),DPRINT
         CNTRL   PRINT,SP,2
         PUT     PRINT
         CLOSE   PRINT
         EOJ
BPRINT   DS       0CL80
         DC      C*          ERROR MESSAGE-
PK        DS       CL2
         DC      C* IS INVALID OPCODE
DPRINT   DS       0CL100
         DC      C*          ERROR MESSAGE- DATA IN ENTRY MODE IS
         DC      C*BLANK OR NOT ENDED WITH DOLLAR SIGN*
         DC      20C*
PJRTAD   DS       F
         DC      AL4(PORT00)        PORT PROGRAM SEGMENT ADDRESS TABLE.
         DC      AL4(PORT01)
         DC      AL4(PORT02)
         DC      AL4(PORT03)
         DC      AL4(PORT04)
         DC      AL4(PORT05)
         DC      AL4(PORT06)
         DC      AL4(PORT07)
         DC      AL4(PORT08)
         DC      AL4(PORT09)
         DC      AL4(PORT0A)
         DC      AL4(PORT0B)
         DC      AL4(PORT0C)
         DC      AL4(PORT0D)
         DC      AL4(PORT0E)
         DC      AL4(PORT0F)
TTAB     DC      X*FAFBFCFDFEFF*
         DC      C*
         DC      X*F0F1F2F3F4F5F6F7F8F9*
DJTPUT   DS       CL132
SAVE     DS       18F
K_       DC      F'0'
TPK      DS       CL2
PTL      DS       CL3
PAD00    DC      F'0'
PAD01    DC      F'0'
PAD02    DC      F'0'
PAD03    DC      F'0'
PAD04    DC      F'0'
PAD05    DC      F'0'
PAD06    DC      F'0'
PAD07    DC      F'0'
PAD08    DC      F'0'
PAD09    DC      F'0'
PAD0A    DC      F'0'
PAD0B    DC      F'0'

```

```

PAD0C      DC      F'0'
PA: 0D     DC      F'0'
PAD0E     DC      F'0'
PA 0F     DC      F'0'
ADP00     DS      X_60
ADP01     DS      X_60
ADP02     DS      X_60
ADP03     DS      XL60
ADP04     DS      XL60
ADP06     DS      X_60
ADP07     DS      X_60
ADP08     DS      XL60
ADP09     DS      X_60
ADP0A     DS      XL60
ADP0B     DS      XL60
ADP0C     DS      X_60
ADP0D     DS      XL60
ADP0E     DS      X_60
ADP0F     DS      X_60
ADP05     DS      XL60
TRTAB     DC      X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C'
CNT00     DC      F'0'
CNT01     DC      F'0'
CNT02     DC      F'0'
CNT03     DC      F'0'
CNT04     DC      F'0'
CNT05     DC      F'0'
CNT06     DC      F'0'
CNT07     DC      F'0'
CNT08     DC      F'0'
CNT09     DC      F'0'
CNT0A     DC      F'0'
CNT0B     DC      F'0'
CNT0C     DC      F'0'
CNT0D     DC      F'0'
CNT0E     DC      F'0'
CNT0F     DC      F'0'
PR NT     DTFR  DEVADDR=SYSLST, IDAREA1=OUTPUT, CONTROL=YES
          PRMOD CONTROL=YES
X_        DS      CL3
SPACE     DC      132C' '
          END      BEGIN

```

COMMAND TABLES FOR THE PORT RELATED  
COMMAND.

\*\*\*\*\*

```

*
*
*          XXXXXXXXXXXX
*          X  SUBDSK  X
*          XXXXX)XXXXXX
*
*

```

```

* THE SUBROUTINE IS INTENDED FOR TRANSFERRING A PORT DATA TO THE
* ACCUMULATOR. THE EXECUTION F 'IN' INSTRUCTION CAUSES TO BRANCH TO
* THIS PROGRAM IF PORT DATA IS PROVIDED IN DISKETTE. THIS PROGRAM IS
* REFLECTION OF THE SUBROUTINE 'SUBIN'. SO THIS SUBROUTINE CONSISTS
* OF 16 PROGRAM SEGMENTS, EACH ONE IS PROCESSED AT A TIME WHEN THE
* DATA OF THAT PARTICULAR PORT IS REQUIRED TO LOAD INTO ACCUMULATOR
* THE PROCESSING PROCEDURE OF ALL THE PROGRAM SEGMENTS ARE SAME.
*

```

=====

```

SJ DSK      CSECT
            PRINT NOGEN
            EXTRN PAD00,PAD01,PAD02,PAD03,PAD04,PAD05,PAD06,PAD07,PAD08
            EXTRN PAD09,PAD0A,PAD0B,PAD0C,PAD0D,PAD0E,PAD0F,MFA
            STM 14,12,12(13) ALL THE REGISTERS ARE SAVED.
            BALR 12,0 R12 IS USED AS BASE REGISTER.
BEGIN      USING *,12

```

```

ST      13,SAVE+4
A       6,=F'1'
MVC    KLT,=F'0'
MVC    KLT+3(1),0(6)
SR      5,5
-       5,KLT
M       4,=F'4'
BTAB   BC      15,BTA3(5)
BC      15,BAD00
BC      15,BAD01
BC      15,BAD02
BC      15,BAD03
BC      15,BAD04
BC      15,BAD05
BC      15,BAD06
BC      15,BAD07
BC      15,BAD08
BC      15,BAD09
BC      15,BAD0A
BC      15,BAD0B
BC      15,BAD0C
BC      15,BAD0D
BC      15,BAD0E
BC      15,BAD0F
BAD00  L       2,=A(PAD00)
-       3,0(2)
-       4,=A(MFA)
MVC    0(1,4),0(3)
A       3,=F'1'
ST      3,0(2)
L       13,SAVE+4
-M     14,12,12(13)
BR      14
BAD01  L       2,=A(PAD01)
L       3,0(2)
L       4,=A(MFA)
MVC    0(1,4),0(3)
A       3,=F'1'
ST      3,0(2)
L       13,SAVE+4
LM     14,12,12(13)
BR      14
BAD02  L       2,=A(PAD02)
L       3,0(2)
L       4,=A(MFA)
MVC    0(1,4),0(3)
A       3,=F'1'
ST      3,0(2)
-       13,SAVE+4
LM     14,12,12(13)
BR      14
BAD03  L       2,=A(PAD03)
L       3,0(2)
L       4,=A(MFA)
MVC    0(1,4),0(3)
A       3,=F'1'
ST      3,0(2)
-       13,SAVE+4
LM     14,12,12(13)
BR      14
BAD04  -       2,=A(PAD04)
-       3,0(2)
L       4,=A(MFA)
MVC    0(1,4),0(3)
A       3,=F'1'
ST      3,0(2)

```

THE PORT ADDRESS IS EXTRACTED FROM THE ADDRESS CONTAINED IN R6 AND MULTIPLIED BY 4 TO OBTAIN THE ADDRESSES OF THE PROGRAM SEGMENT FOR THE CORRESPONDING PORT.

BRANCHING TABLE.

THE PORT DATA FROM THE LOCATION THE ADDRESS OF WHICH IS CONTAINED IN 'PAD00' IS LOADED INTO ACCUMULATOR.

THE REGISTERS CONTENTS ARE RESTORED AND RETURNS TO THE CALLING PROGRAM.

	L	13, SAVE+4
	LM	14, 12, 12(13)
BAD05	BR	14
	L	2, =A(PAD05)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	L	13, SAVE+4
	-M	14, 12, 12(13)
BAD06	BR	14
	L	2, =A(PAD06)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	L	13, SAVE+4
	-M	14, 12, 12(13)
BAD07	BR	14
	L	2, =A(PAD07)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	L	13, SAVE+4
	-M	14, 12, 12(13)
BAD08	BR	14
	L	2, =A(PAD08)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	-	13, SAVE+4
	-M	14, 12, 12(13)
BAD09	BR	14
	L	2, =A(PAD09)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	L	13, SAVE+4
	LM	14, 12, 12(13)
BAD0A	BR	14
	L	2, =A(PAD0A)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	-	13, SAVE+4
	-M	14, 12, 12(13)
BAD0B	BR	14
	L	2, =A(PAD0B)
	-	3, 0(2)
	-	4, =A(MFA)
	MVC	0(1, 4), 0(3)
	A	3, =F' 1'
	ST	3, 0(2)
	-	13, SAVE+4
	LM	14, 12, 12(13)
	BR	14



```

C          3,=F'2'
BC         8,HALT
MVC       MK(2),0(6)
JNPK      XL(3),MK
MVC       MK(2),XL
TR        MK(2),TRTAB-X'F0'
L         7,0(3)
MVC       PK(2),0(7)
UNPK      XL(3),PK
MVC       PK(2),XL
TR        PK(2),TRTAB-X'F0'
PUT       CJSOOLE,DSPLY1
CLOSE    CJSOOLE
L         13,SAVE+4
LM        14,12,12(13)
BR        14
DSPIN     A          6,=F'1'
MVC       TK(2),0(6)
UNPK      XL(3),TK
MVC       TK(2),XL
TR        TK(2),TRTAB-X'F0'
PUTR      CJSOOLE,DSPLY2,CAREA
MVC       AA(2),CAREA
CLOSE    CONSOLE
LA        3,=A(AA)
L         13,SAVE+4
-M        14,2,12(13)
-M        4,12,36(13)
BR        14
HALT     PUTR      CONSOLE,DSPLY4,CAREA
MVC       ZL(10),CAREA
CLOSE    CONSOLE
LA        3,=A(ZL)
L         13,SAVE+4
LM        14,2,12(13)
LM        4,12,36(13)
BR        14
CJSOOLE  DTFCN    DEVAADR=SYSLOG,IOAREA1=OAREA,BLKSIZE=80,INPSIZE=80,
CAREA    DS       CL80
OAREA    DS       CL80
XL        DS       CL3
SAVE     DS       18F
DSPLY1   DS       0CL80
DC       C'      8085 ACCUMULATOR CONTENT OUTPUTTING FOR'
DC       C'OUTPUT PORT '
MK        DS       CL2
DC       C'IS '
PK        DS       CL2
DC       50C' '
Z_        DS       CL10
DSPLY2   DS       0CL80
DC       C' KEY A DATA FROM GIVEN VALUES THROUGH KEYBOARD FOR '
TK        DS       CL2
DC       C'INPUT PORT'
DC       50C' '
DSPLY3   DS       0CL80
DC       C'      KEY A DATA FROM GIVEN VALUES FOR INTERRUPT MASK'
DC       50C' '
TRTAB    DC       X'F0F1F2F3F4F5F6F7F8F9C1C2C3C4C5C6'
AA        DS       CL2
DSPLY4   DS       0CL80
DC       C'      ENTER CORRECT DATA THROUGH CONSOLE KEY FOR HALT'
DC       50C' '
END      BEGIN
*****

```

OBTAINING INPUT DATA THROUGH CONSOLE KEYBOARD. IF BRANCH IS FROM 'HLT' INSTRUCTION, GO TO 'HALT' FOR OBTAINING DATA FROM CONSOLE KEYBOARD, OTHERWISE UNPACK AND CONVERT THE PORT ADDRESS AND DATA FOR DISPLAYING UPON CONSOLE.

INPUT PORT ADDRESS IS UNPACKED AND CONVERT TO EBCDIC MODE TO DISPLAY THE MESSAGE 'DSPLY2' FOR THE USER. DATA ENTERED THROUGH KEYBOARD IS TAKEN TO MAIN PROGRAM WITH HELP OF R3. REGISTERS CONTENTS ARE RESTORED AND RETURNS TO MAIN PROGRAM.

IN RESPONSE TO THE MESSAGE 'DSPLY4' DATA IS ENTERED THROUGH CONSOLE KEYBOARD AND TAKEN TO MAIN PROGRAM WITH HELP OF R3. REGISTERS ARE RESTORED AND RETURNS TO MAIN PROGRAM.

\*

\*\*\*\*\*

```

XXXXXXXXXXXXX
X  SUBBPP  X
XXXXX)XXXXXX

```

```

* THIS SUBROUTINE IS INTENDED FOR BREAKPOINT PRINTING. THE PARAMET- *
* ERS OF THE COMMAND TABLE FOR 'PRINT AT BREAKPOINTS' COMMAND ARE UTI- *
* LIZED IN THIS SECTION. THIS SUBROUTINE RECEIVES THE CONTROL FROM *
* THE DEBUGGER WHEN USER MENTIONS ABOVE COMMAND IN HIS PROGRAM. IT *
* ALLOWS THE USER TO OBTAIN A PRINTOUT OF THE CONTENTS OF GENERAL- *
* PURPOSE REGISTERS, ACCUMULATOR, PROGRAM COUNTER, STACK POINTER, *
* DIFFERENT FLAGS AND INSTRUCTION CODES ETC. FOR DEBUGGING CONVINIE- *
* NCE PROGRAM COUNTER IS MADE TO REPRESENT THE ADDRESS OF THE INSTR- *
* UCTION JUST SIMULATED. EIGHT BYTES OF PROGRAM STARTING FROM THE *
* MEMORY ADDRESS INDICATED BY PROGRAM COUNTER ARE SHOWN AGAINST IT *
* IN BREAKPOINT PRINTING. *
*
*=====

```

```

SJBPP      CSECT
           PRINT NOGEN
           EXTRN MFA,MFSP,MICROMEM
BEGIN      STM      14,12,12(13)      ALL THE REGISTERS ARE SAVED.
           BALR    12,0              R12 IS USED AS BASE REGISTER.
           USING  *,12
           ST      13,SAVE+4
           OPEN   PRINT
           MVC     KLT,=F'0'
           L       2,=A(MFSP)
           MVC     SPK(2),0(2)
           LA      8,STK
           MVC     KLT+2(2),SPK
           L       4,=A(MICROMEM)
           LR      3,4
           A       3,KLT
           L       7,=F'8'
SPW        MVC     PK(2),0(3)        THE DATA CONTAINED IN STACK ARE UNPA-
           UNPK   XL(3),PK          CKED AND CONVERTED TO EBCDIC FORM
           MVC     PK(2),XL          AND STORED AT THE LOCATION ADDRESSED
           TR      PK(2),TRTAB-X'F0' BY R8.
           MVC     0(2,8),PK
           A       8,=F'3'
           A       3,=F'1'
           BCT    7,SPW
           MVC     M_(3),SPK        THE STACK POINTING ADDRESS IS UNPA-
           UNPK   SM(5),ML          CKED AND CONVERTED TO EBCDIC MODE IN
           MVC     SP(4),SM          'SP'.
           TR      SP(4),TRTAB-X'F0'
           MVC     PCK(2),2(2)
           MVC     KLT+2(2),PCK
           A       4,KLT
           -A     8,CODE
PCW        L       7,=F'8'
           MVC     PK(2),0(4)        THE INSTRUCTION CODE STARTING FROM
           UNPK   XL(3),PK          ADDRESS INDICATED BY THE PROGRAM
           MVC     PK(2),XL          COUNTER IS UNPACKED AND CONVERTED TO
           TR      PK(2),TRTAB-X'F0' EBCDIC MODE AND STORED AT THE LOCAT-
           MVC     0(2,8),PK          ION ADDRESSED BY R8.
           A       8,=F'3'
           A       4,=F'1'
           BCT    7,PCW
           MVC     M_(3),PCK        THE PROGRAM COUNTER CONTENTS IS UNP-
           UNPK   SM(5),ML          ACKED AND CONVERTED TO EBCDIC MODE
           MVC     PC(4),SM          IN 'PC'.
           TR      PC(4),TRTAB-X'F0'

```



```

REGW  LA      8,REG
      -      2,=A(MFA)
      L      7,=F'5'
      MVC    PK(2),0(2)
      UNPK   XL(3),PK
      MVC    PK(2),XL
      TR     PK(2),TRTAB-X'F0'
      MVC    0(2,8),PK
      A      8,=F'6'
      A      2,=F'1'
      BCT    7,REGW
      A      2,=F'2'
      MVC    PK(2),0(2)
      UNPK   XL(3),PK
      MVC    PK(2),XL
      TR     PK(2),TRTAB-X'F0'
      MVC    0(2,8),PK
      A      8,=F'6'
      A      2,=F'4'
      MVC    PK(2),0(2)
      UNPK   XL(3),PK
      MVC    PK(2),XL
      TR     PK(2),TRTAB-X'F0'
      MVC    0(2,8),PK
      A      2,=F'1'
      L      7,=F'5'
F_FLAGW LA     8,FLAG
      MVC    PK(2),0(2)
      UNPK   XL(3),PK
      MVC    PK(2),XL
      TR     PK(2),TRTAB-X'F0'
      MVC    0(1,8),PK+1
      A      8,=F'6'
      A      2,=F'1'
      BCT    7,FLAGW
      CNTRL  PRINT,SP,2
      MVC    OUTPUT(132),SPACE
      MVC    JUTPUT(100),PCL
      PUT    PRINT
      MVC    JUTPUT(110),SPL
      PUT    PRINT
      MVC    JUTPUT(100),REGL
      PUT    PRINT
      MVC    JUTPUT(132),SPACE
      MVC    JUTPUT(100),FLAGL
      PUT    PRINT
      CLOSE  PRINT
      -      13,SAVE+4
      -M     14,12,12(13)
      BR     14
X_     DS    CL3
PK     DS    CL2
K_T    DC    F'0'
SP_    DS    0CL100
      DC    30C' '
      DC    C'SP='
SP     DS    CL4
      DC    C' '
STK    DS    CL2
      DC    C' '
      DS    C_2
      DC    C' '
      DS    CL2
      DC    C' '
      DS    CL2
      DC    C' '

```

THE CONTENTS OF THE MICROPROCESSOR GENERAL PURPOSE REGISTERS AND ACCUMULATOR ARE UNPACKED AND CONVERTED TO EBCDIC MODE AND STORED AT THE LOCATION ADDRESSED BY R8.

ALL THE FLAG VALUES ARE UNPACKED AND CONVERTED TO EBCDIC MODE AND STORED AT THE LOCATION ADDRESSED BY R8.

BREAKPOINT STATUS IS PRINTED.

THE REGISTERS CONTENTS ARE RESTORED AND BRANCHES TO CALLING PROGRAM.

RECORD FOR PRINTING THE STACK POINTER AND ITS CONTENTS.

	DS	CL 2	
	DC	C' '	
	DS	C_2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
SPACE	DC	132C' '	
SP	DS	CL 2	
M_	DS	CL 3	
S4	DS	CL 5	
SAVE	DS	18F	
OUTPUT	DS	CL 132	
TRTAB	DC	X' F0F 1F 2F 3F 4F 5F 6F 7F 8F 9C1C2C3C4C5'	
PRINT	DTFPR	DEVADDR=SYSLST,IOAREA1=OUTPUT,BLKSIZE=132,CONTROL=YES	
	PRMOD	CONTROL=YES	
PCL	DS	0C_110	RECORD FOR PRINTING THE PROGRAM
	DC	30C' '	COUNTER CONTENTS AND HEXADECIMAL
	DC	C'PC='	PROGRAM.
PC	DS	CL 4	
	DC	C' '	
CODE	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	C' '	
	DS	CL 2	
	DC	50C' '	
PCK	DS	C_2	
REGL	DS	0CL100	RECORD TO PRINT REGISTERS CONTENTS.
	DC	30C' '	
	DC	C'REGISTERS	
REG	DC	C'A='	
	DS	C_2	
	DC	C' B='	
	DS	C_2	
	DC	C' C='	
	DS	CL 2	
	DC	C' D='	
	DS	CL 2	
	DC	C' E='	
	DS	C_2	
	DC	C' H='	
	DS	CL 2	
	DC	C' L='	
	DS	CL 2	
	DC	50C' '	
FLAGL	DS	0C_100	FLAG BIT PRINTING RECORD.
	DC	30C' '	
	DC	C'FLAGS	
FLAG	DS	C_1	S='
	DC	C' Z='	
	DS	CL 1	
	DC	C' AC='	
	DS	CL 1	
	DC	C' P='	
	Di	CL 1	

```
DC      C  CY=  
DS      C_1  
DC      50C  
END     BEGIN
```

APPENDIX - B  
FLOWCHART OF SIMULATION PROGRAM

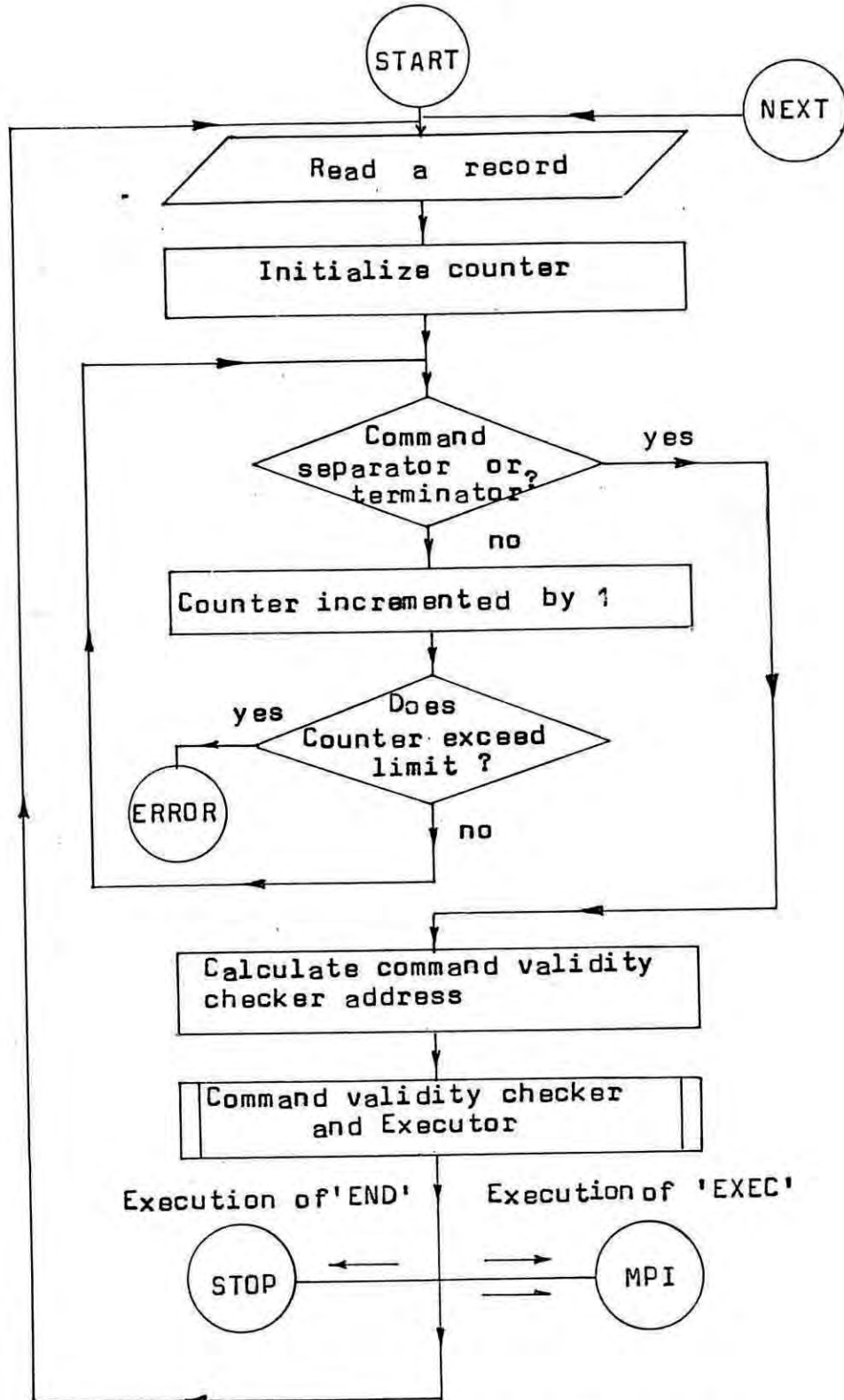


Fig. B-1 Flowchart of command processor.

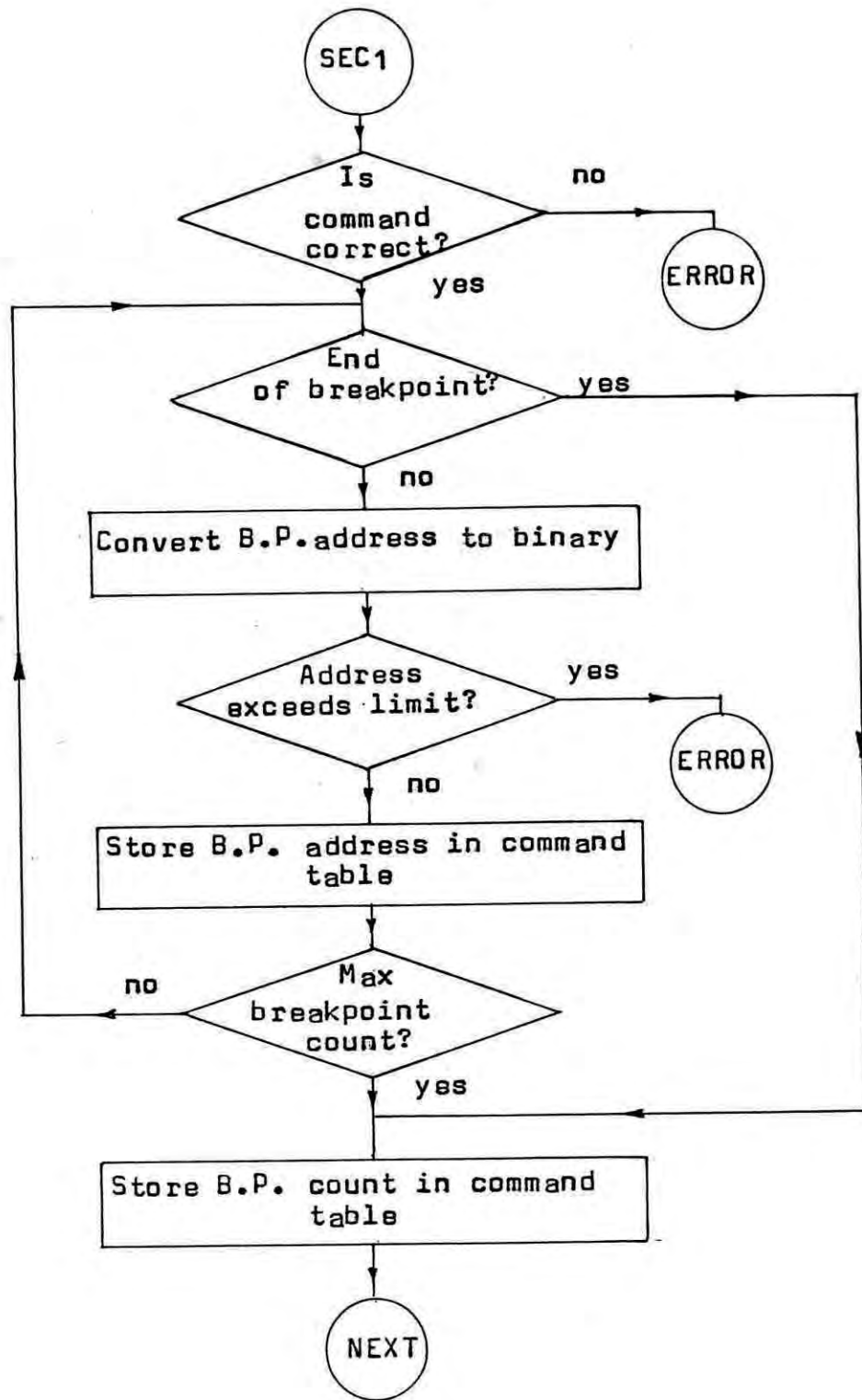


Fig. B-2 Breakpoint flowchart.

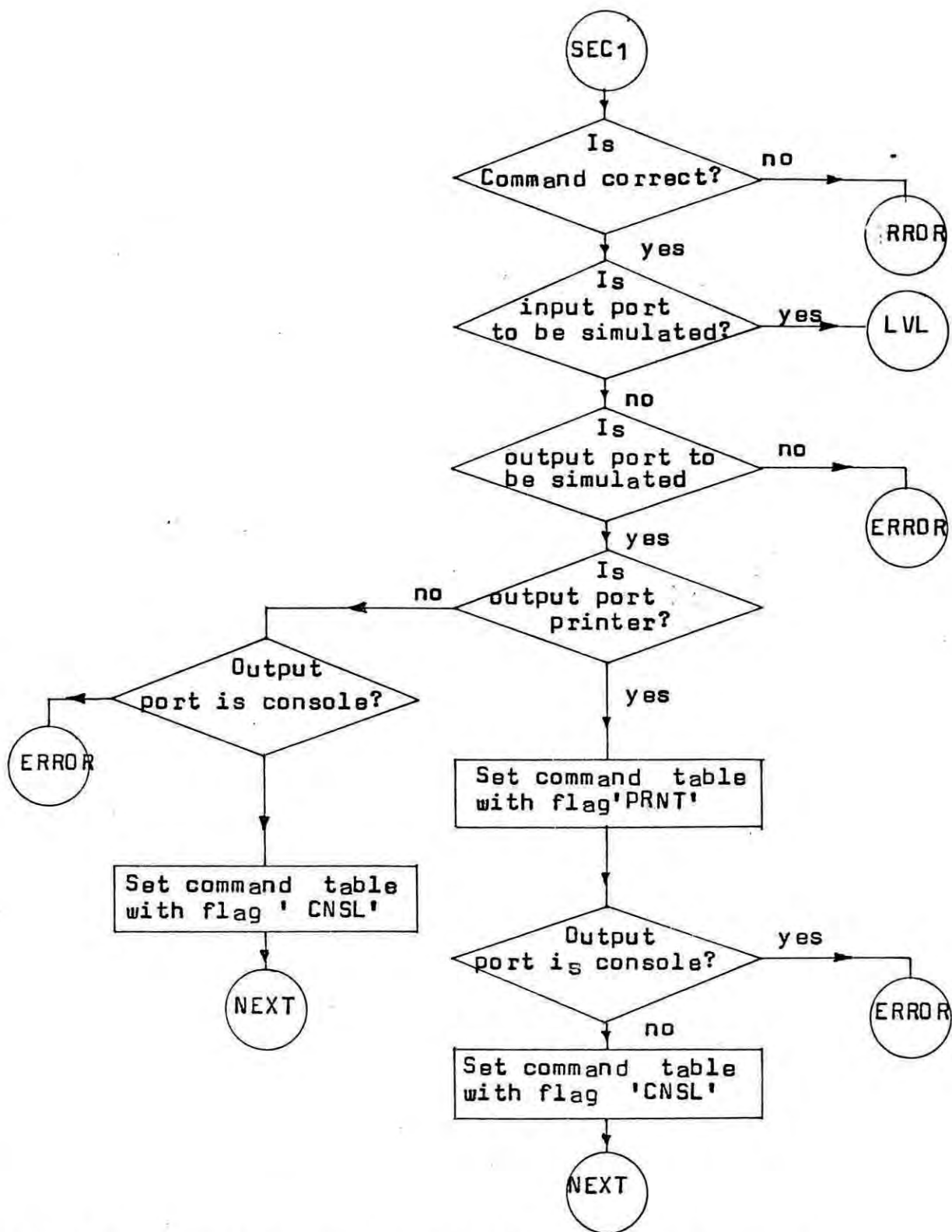


Fig. B-3. Flowchart to show the I/O simulation command

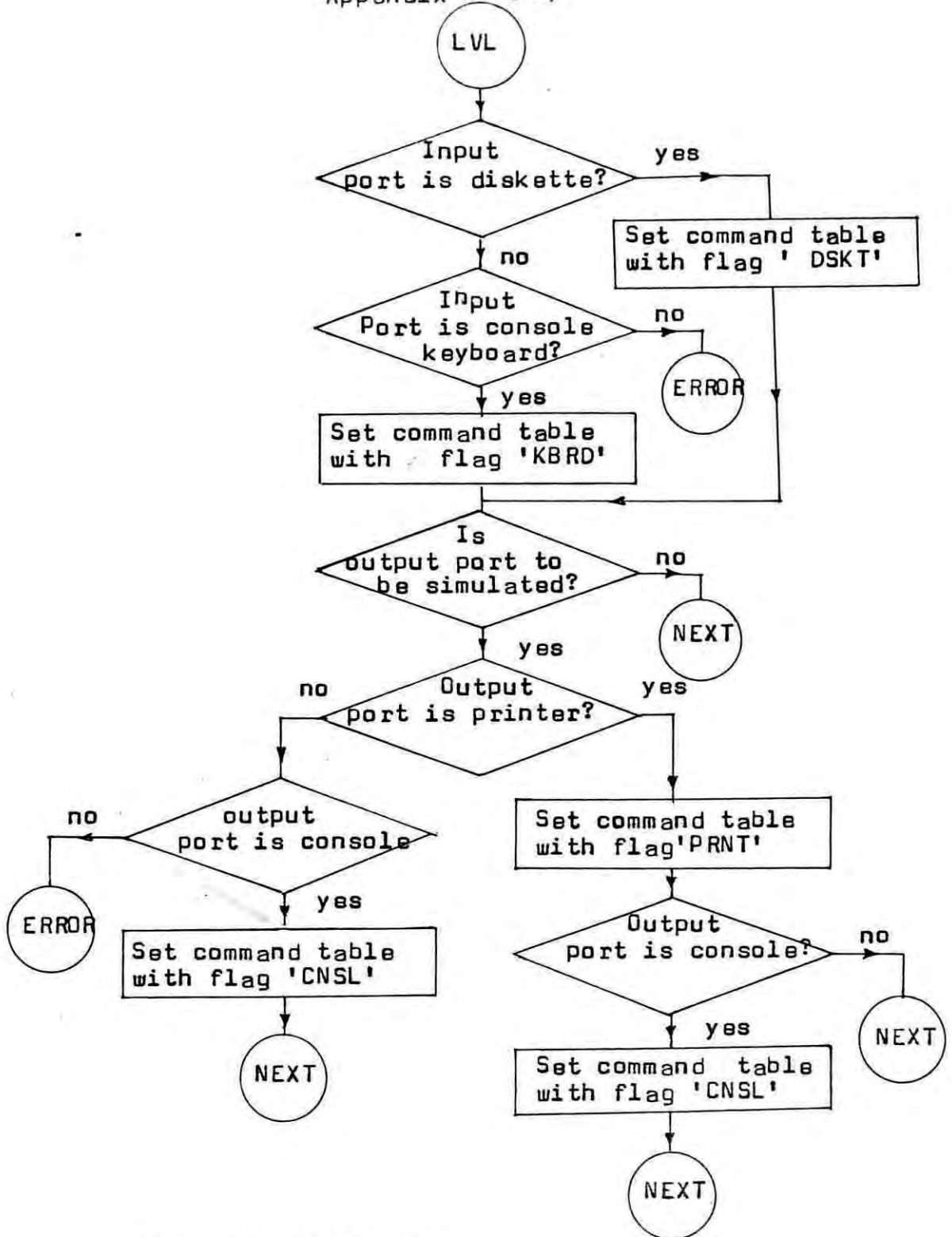


Fig. B-3. continuation



Appendix B-5

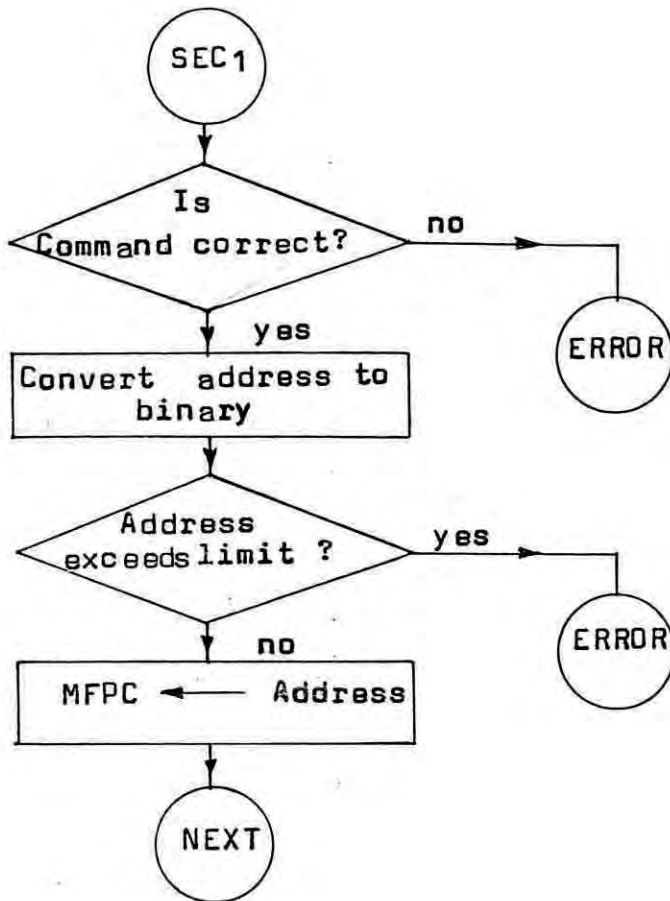


Fig. B-4 Flowchart for setting the program counter

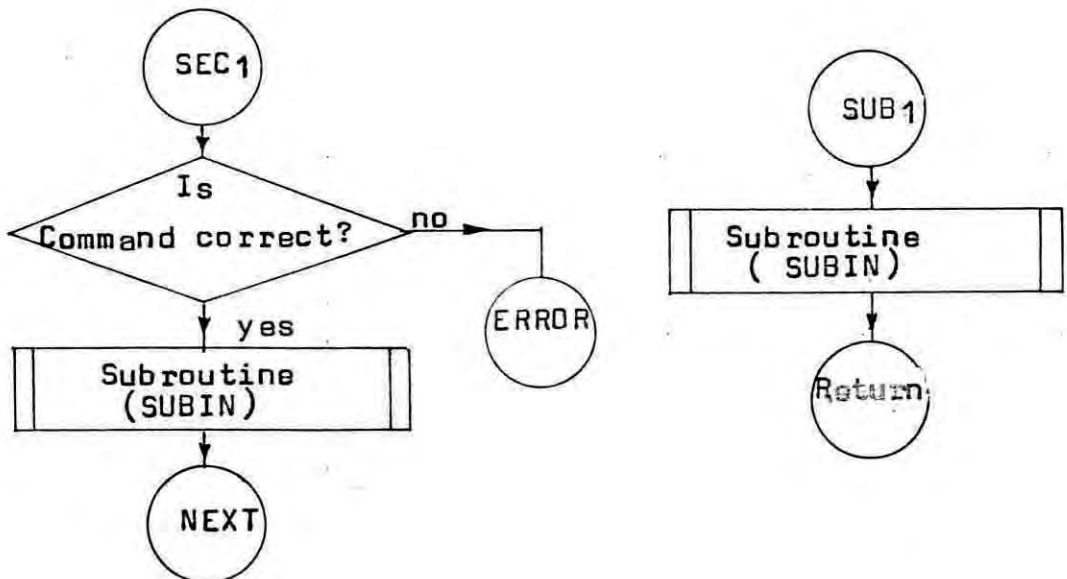


Fig. B-5. Flowchart to simulate Diskette as input port.

Appendix B-6

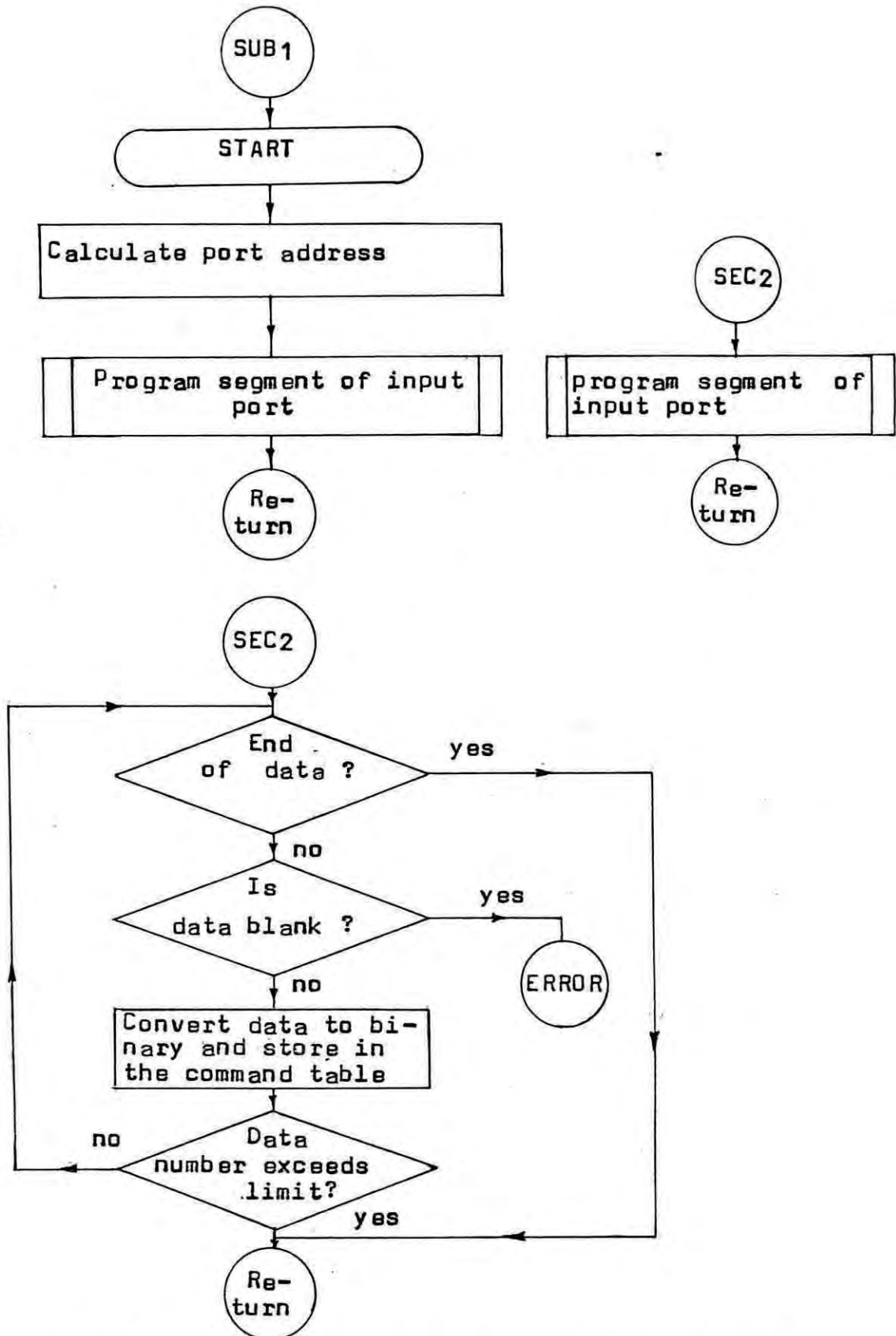


Fig. B-6 Subroutine SUBIN for Diskette (simulated) port.

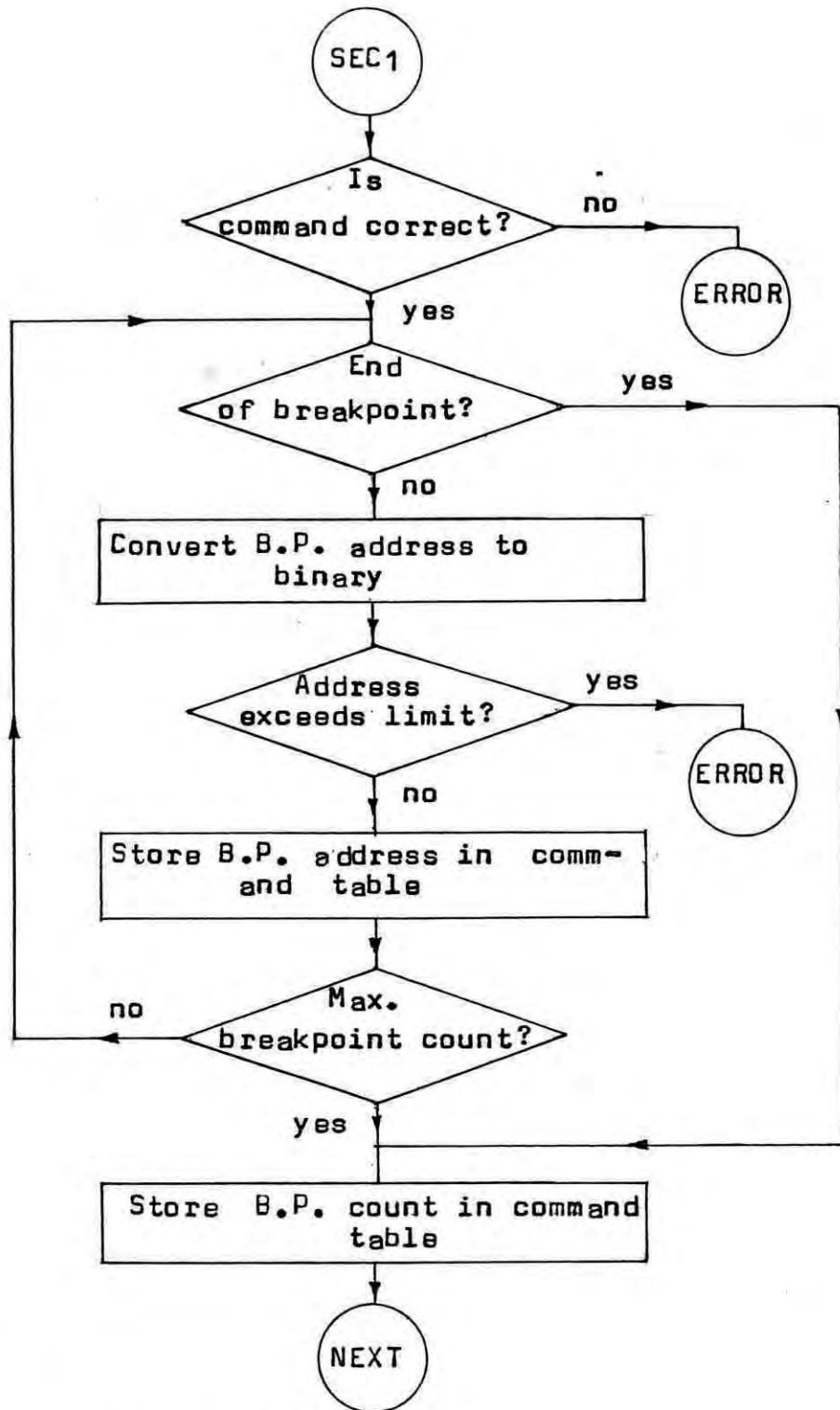


Fig. B-7 Flowchart for breakpoint pringing.

Appendix-B-8

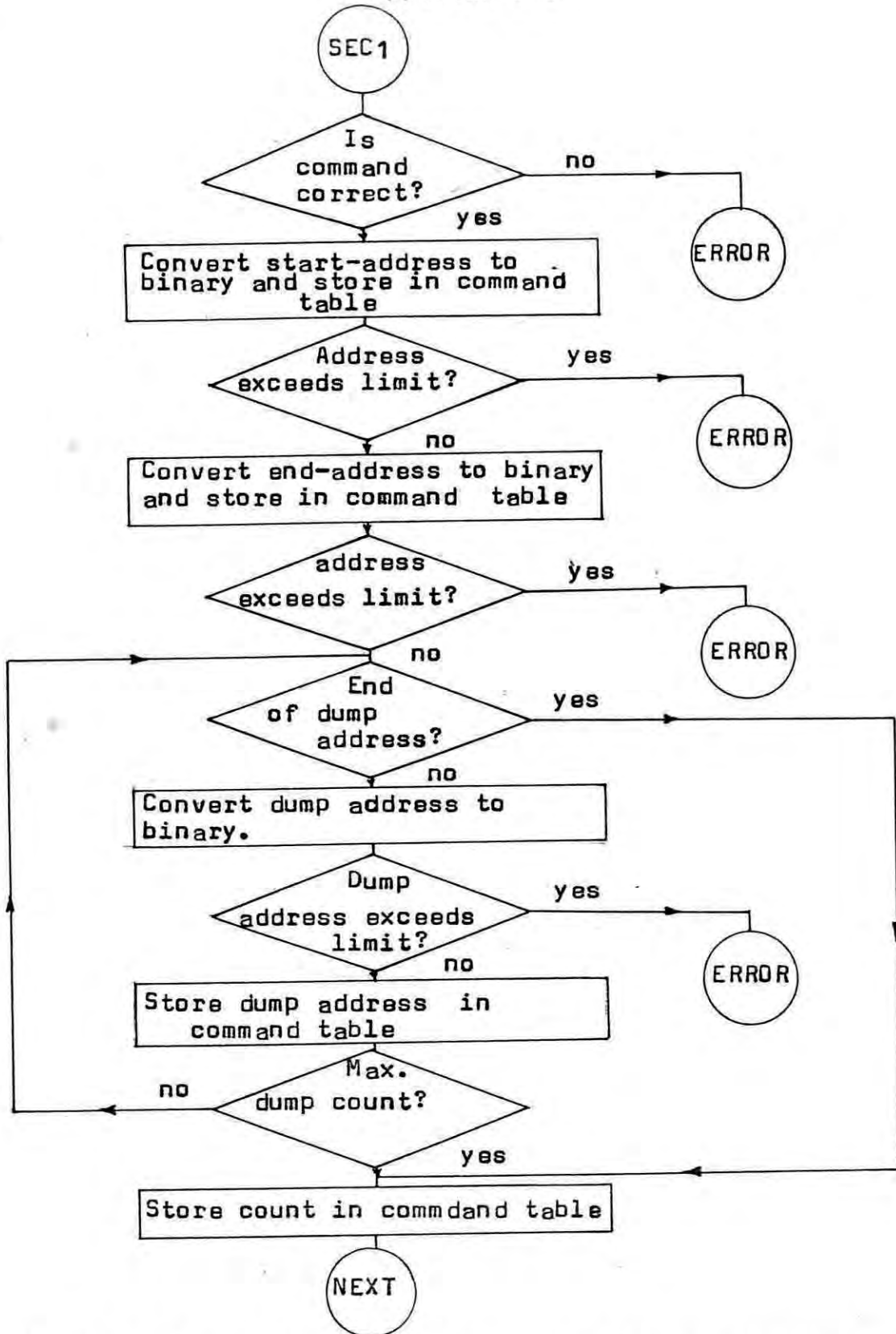


Fig. B-8 Flowchart to set the command table with the parameters of memory dump command.

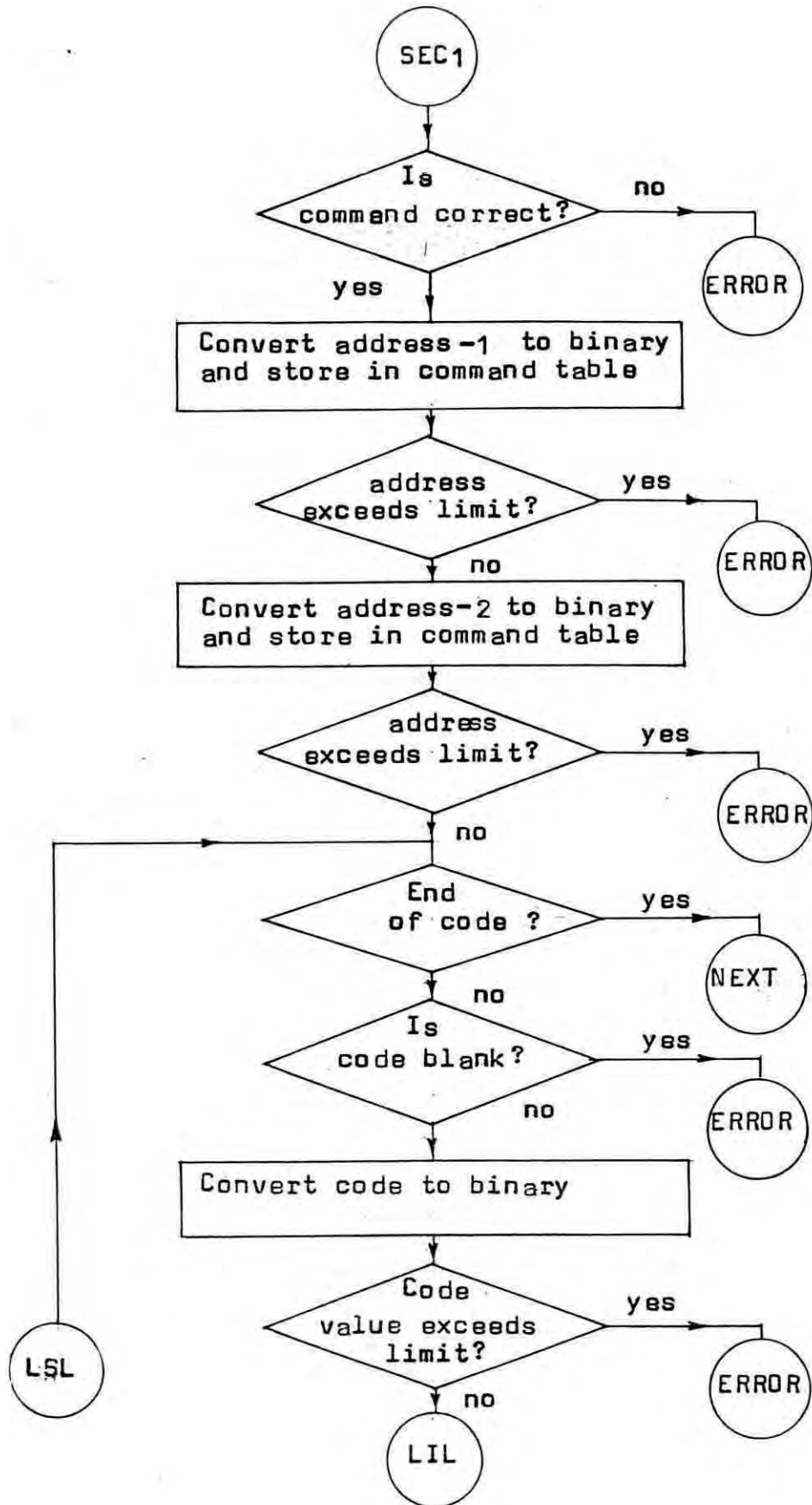


Fig.B-9 Flowchart to load program in command table

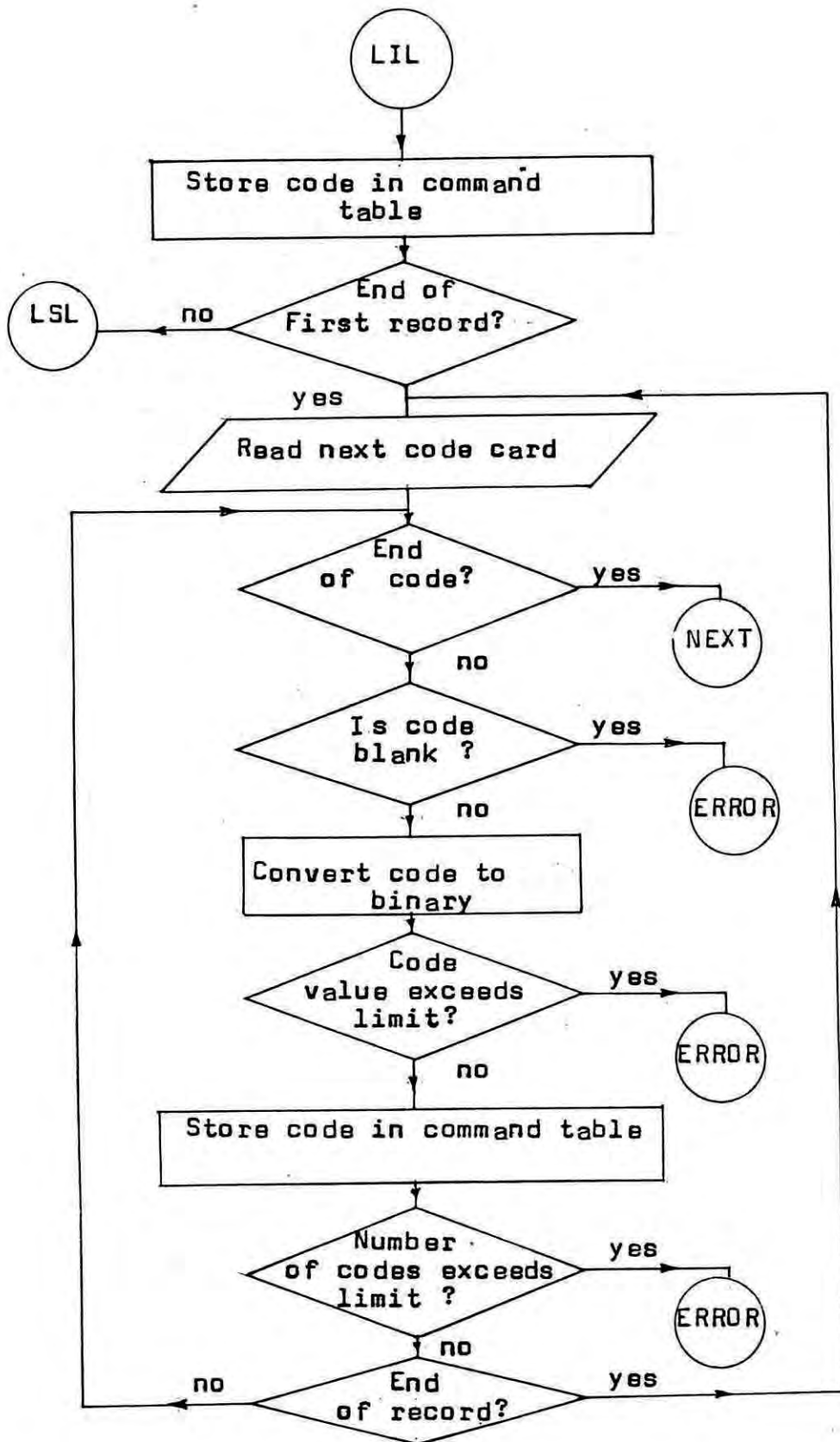


Fig. B-9 continuation.

Appendix-B-11

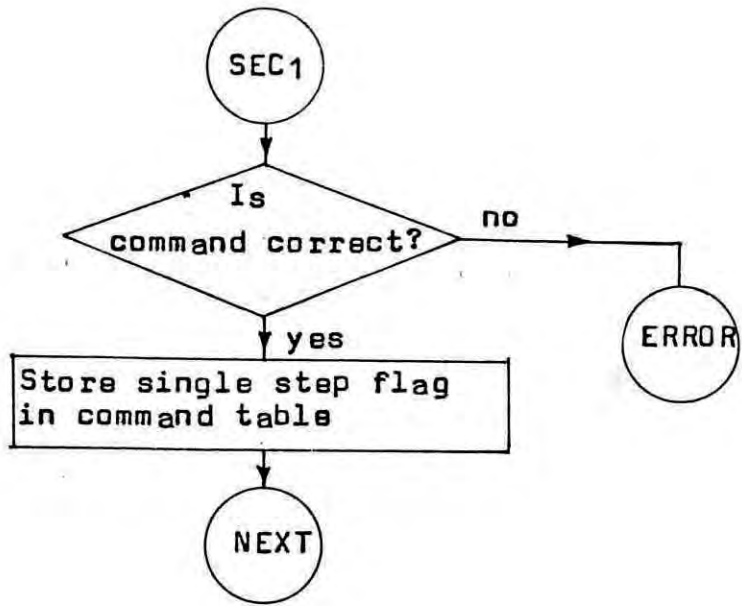


Fig. B-10. Single stepping flowchart

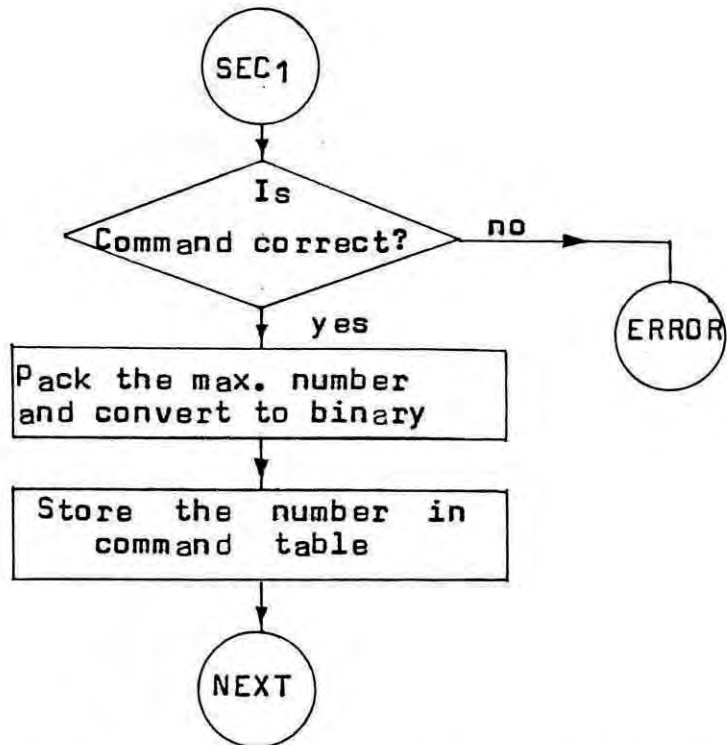


Fig. B-11 Flowdiagram for exit from endless loop

Appendix- B-12

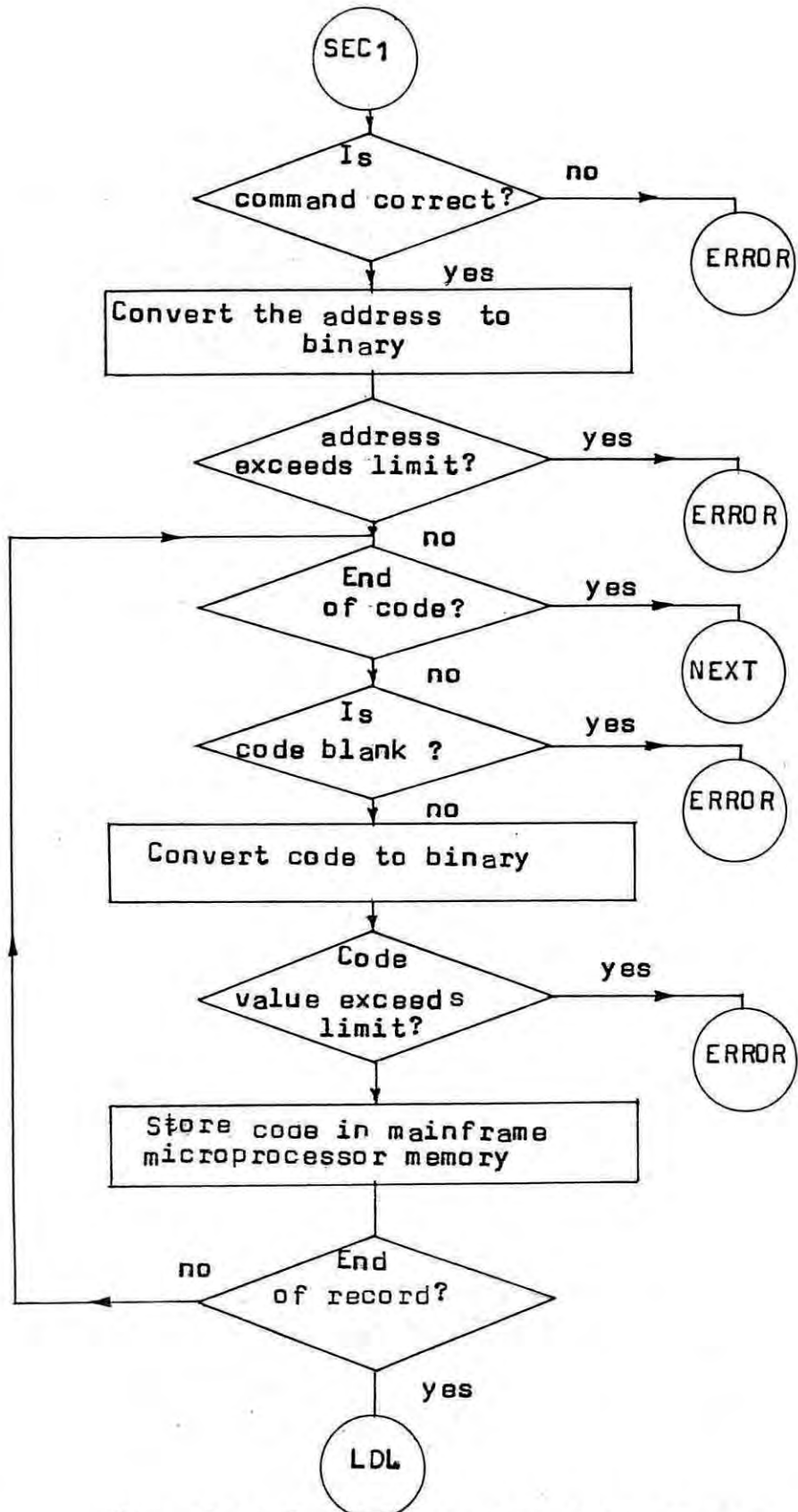


Fig. B-12 Program loading flow diagram.



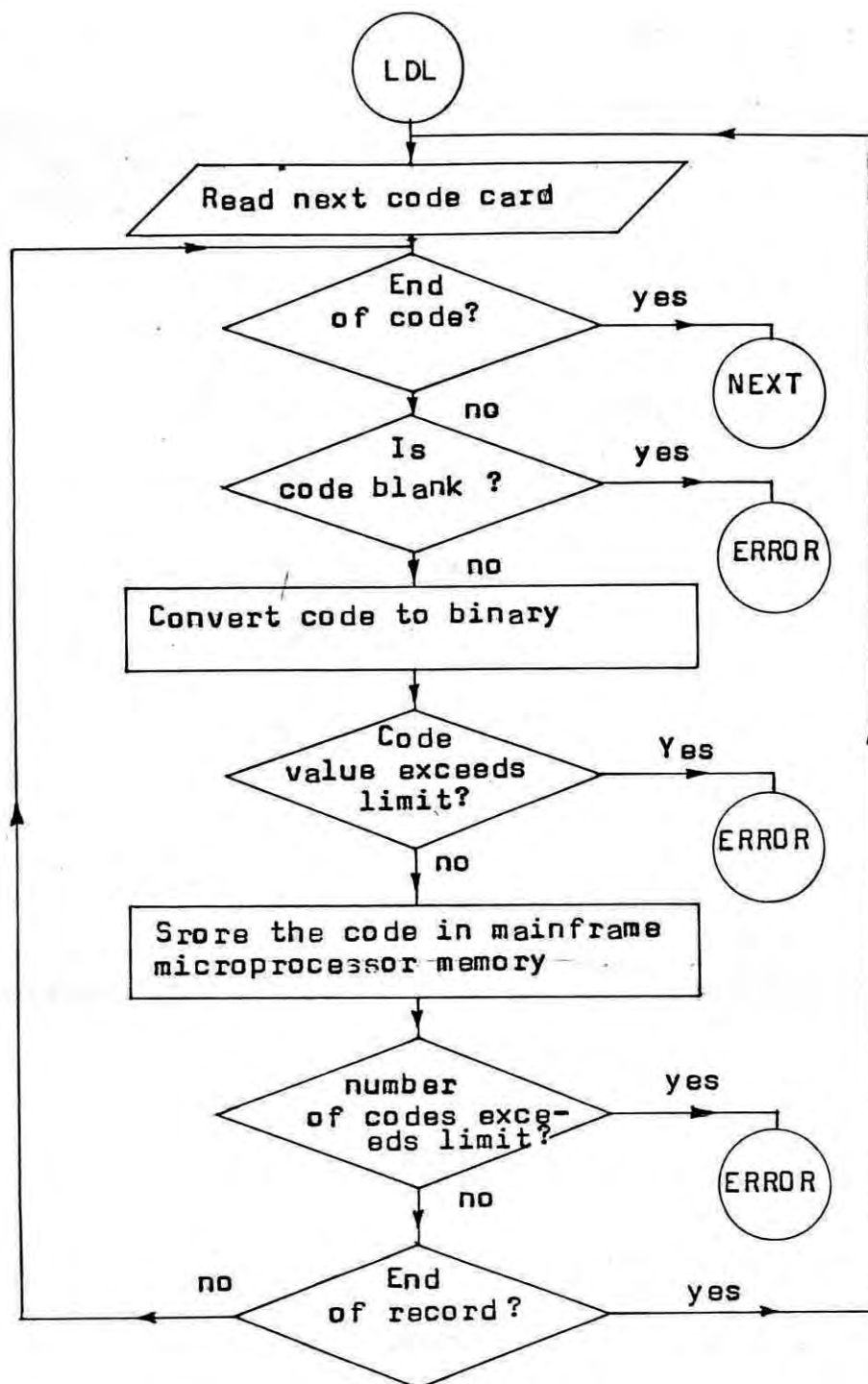


Fig. B-12 continuation

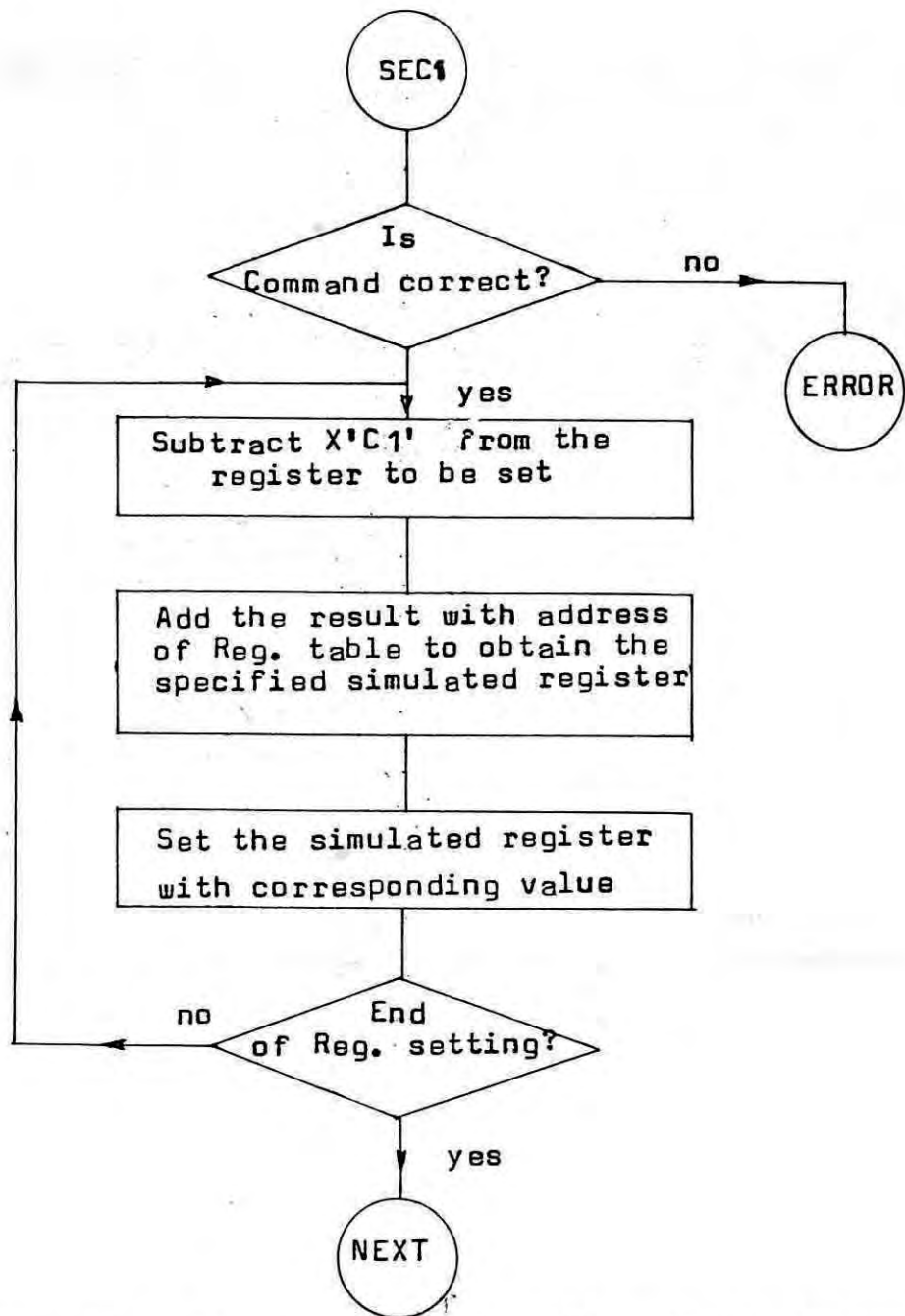


Fig. B-13 Flow diagram of setting the registers.

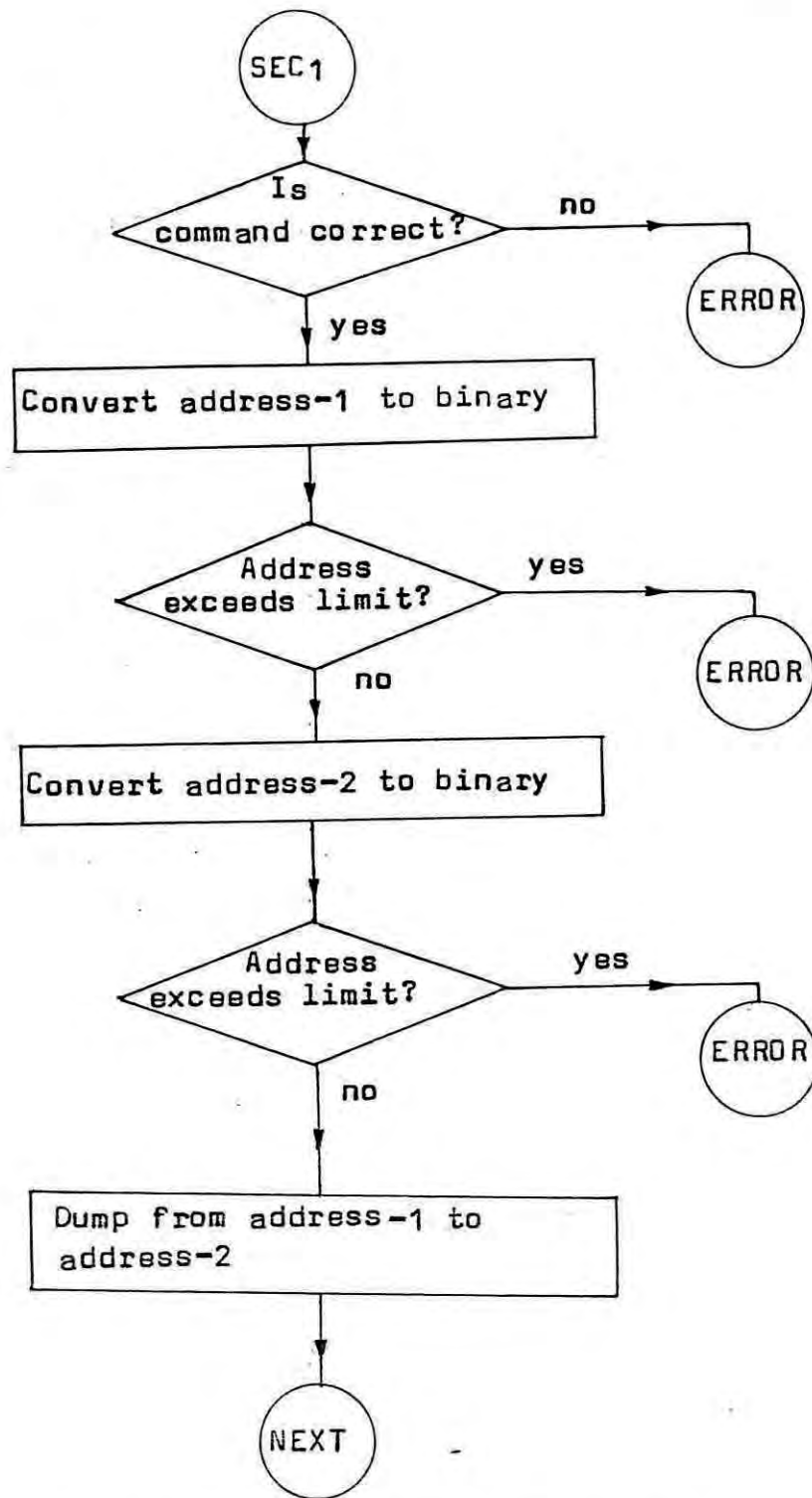


Fig. B-14 Flowchart to dump the memory

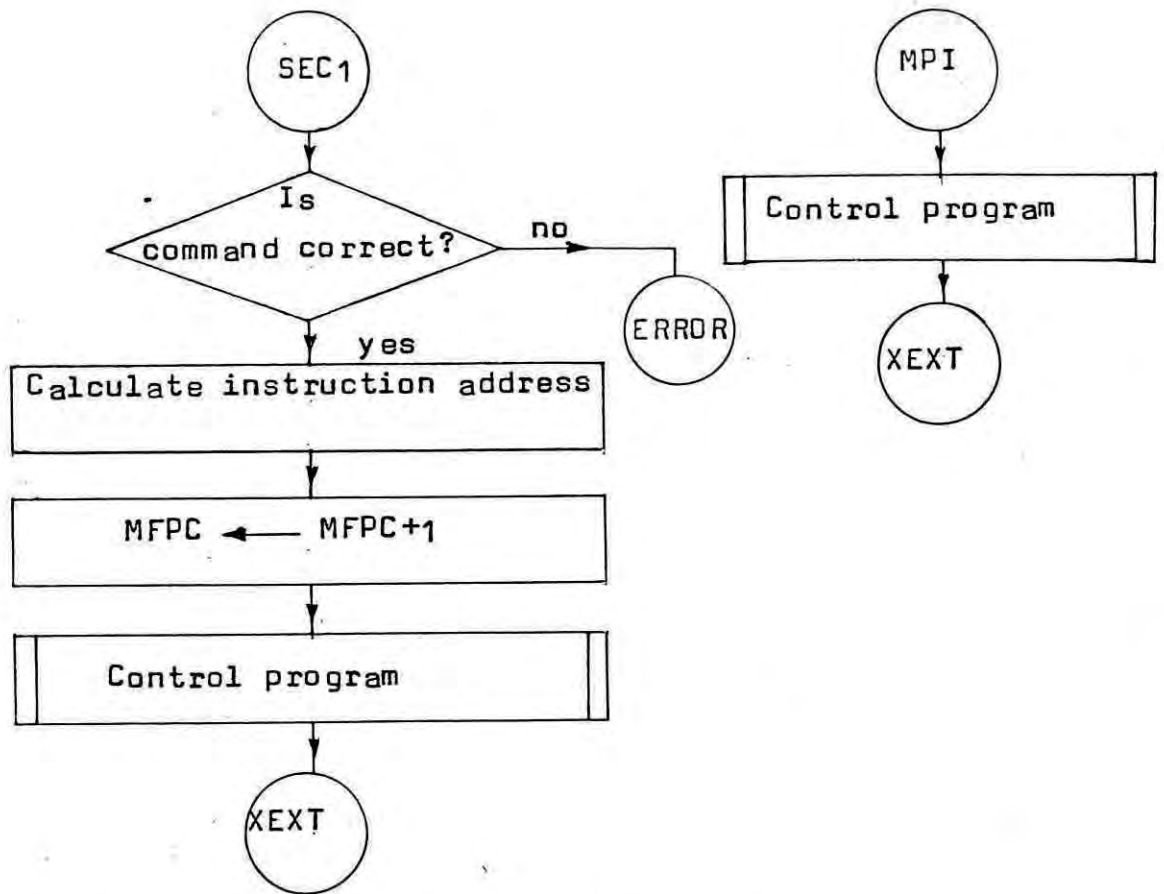


Fig. B-15 Flowchart to start simulation

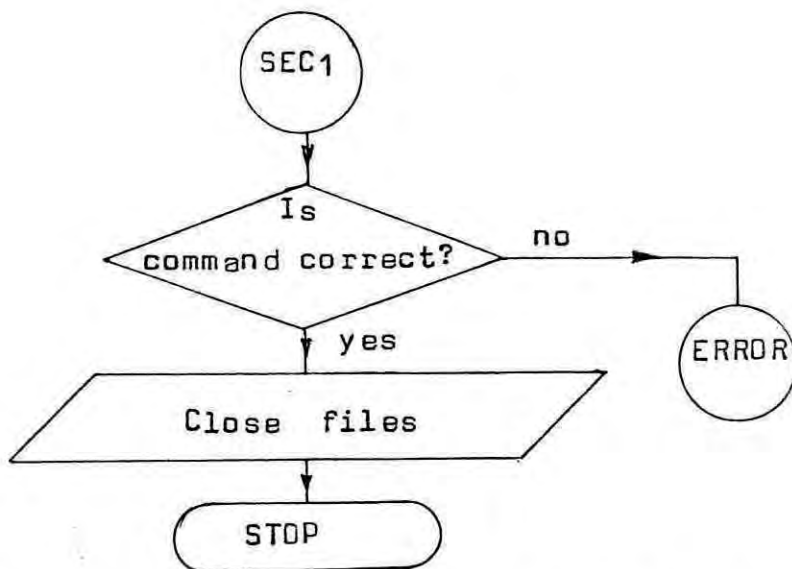


Fig. B-16 Flowchart to stop simulation.

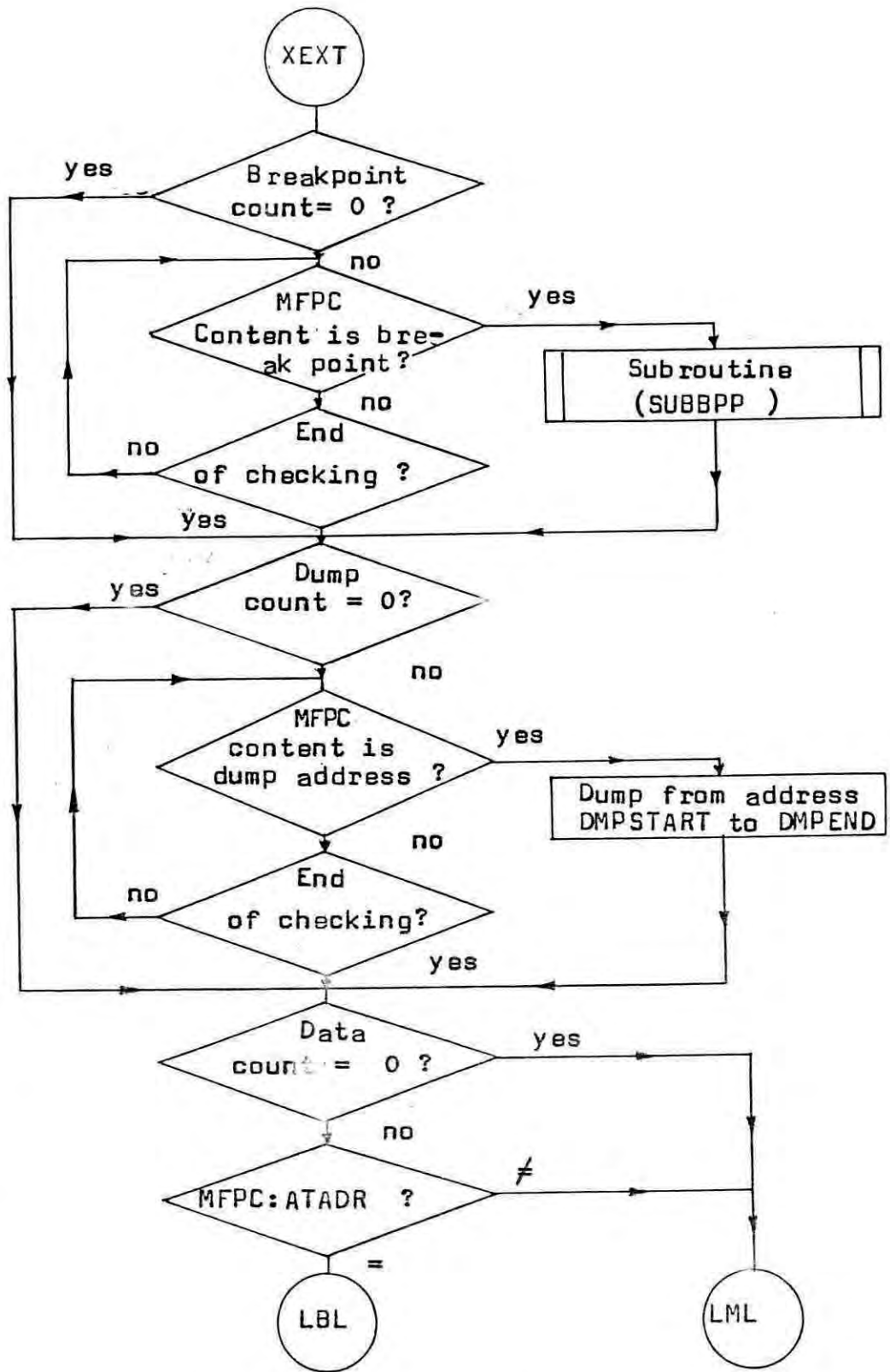


Fig. B 17 Flowchart of Debugger

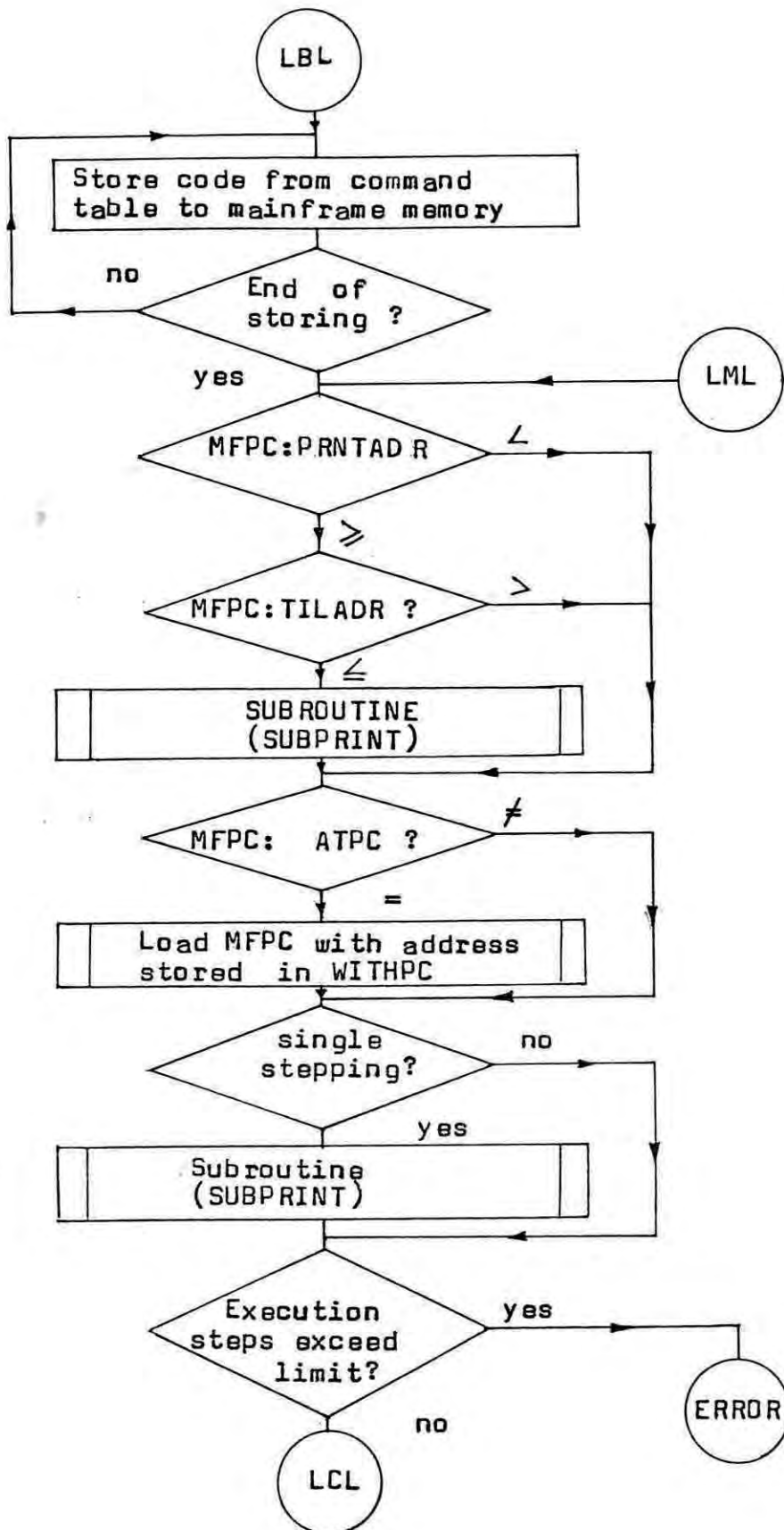


Fig. B-17 cont.

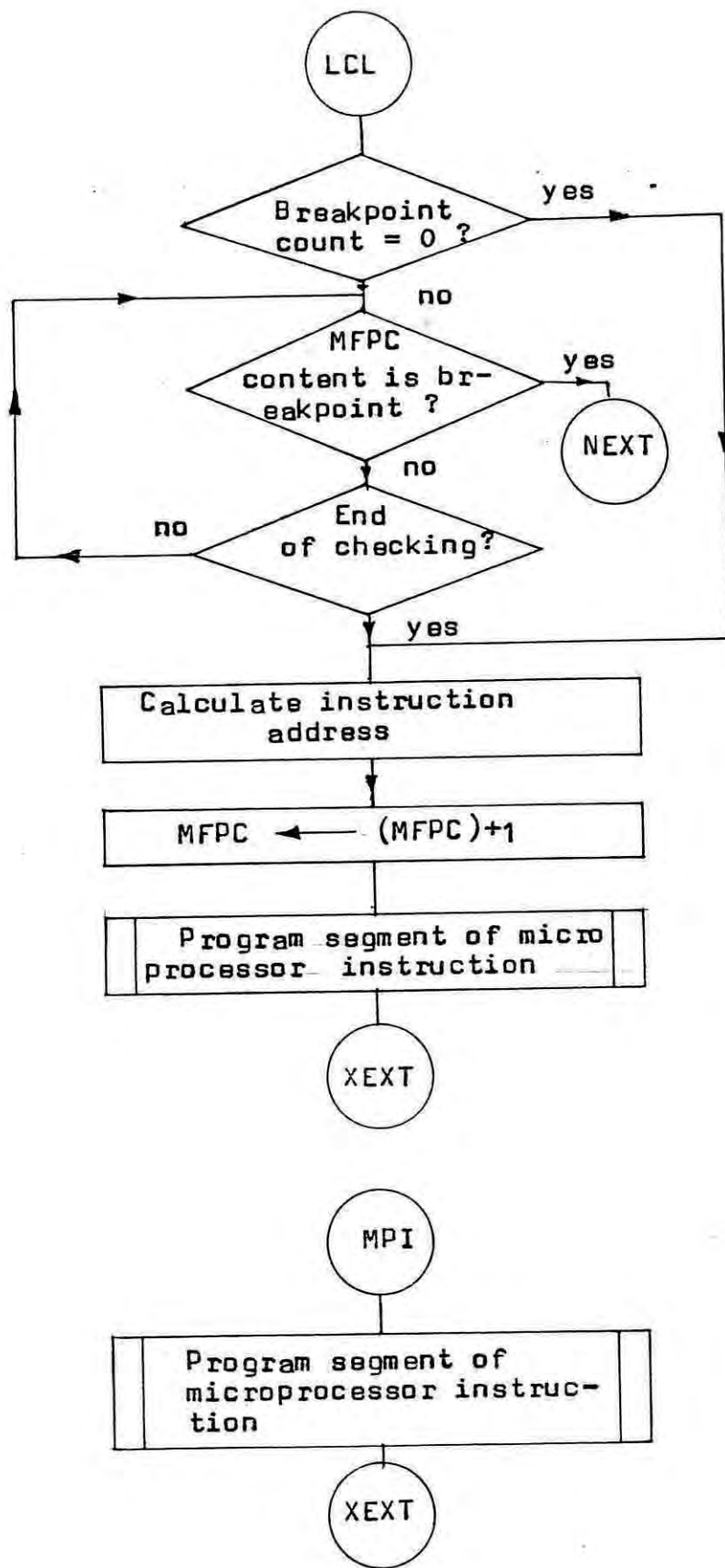


Fig. B-17 cont.

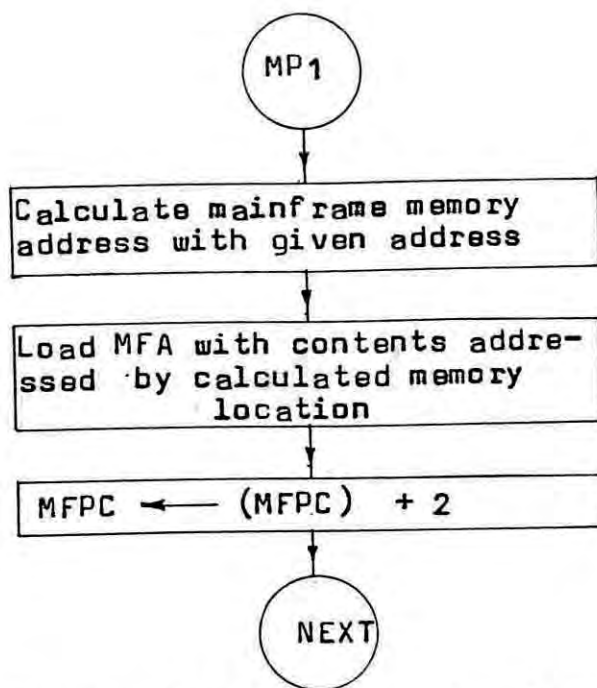


Fig. 18 LDA-Load contents of memory location into accumulator using direct addressing

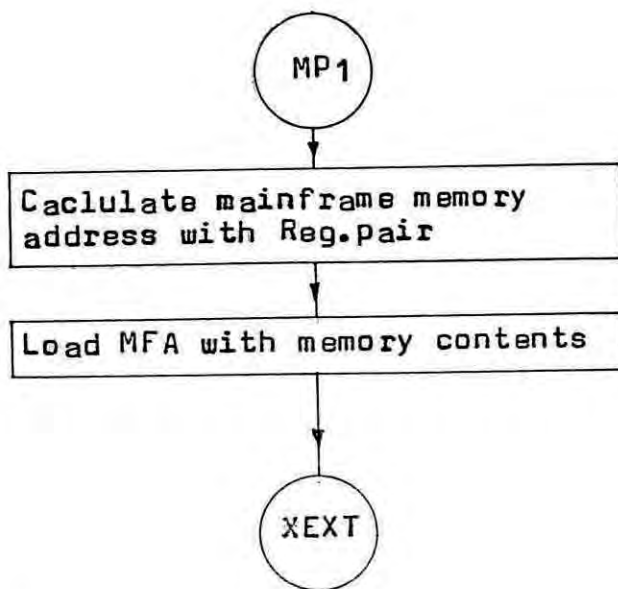


Fig. 19 LDAX-Load accumulator from memory location addressed by register pair.



Appendix- B-21

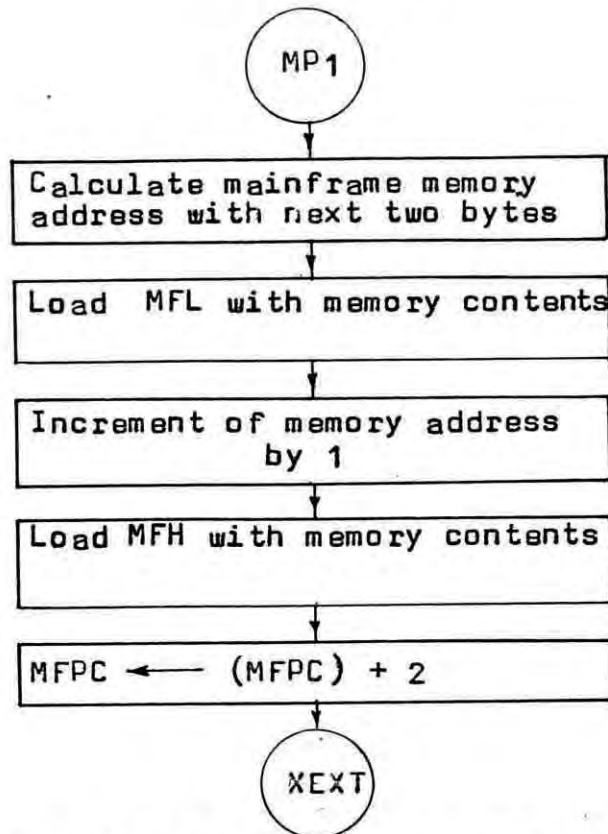


Fig. B-20 LHL D-Load H and L register direct

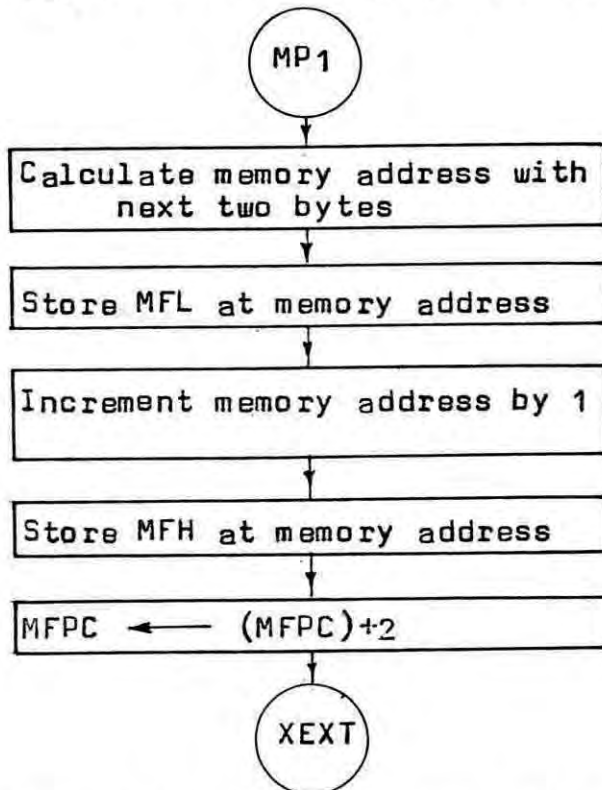


Fig. B-21 SHLD-Store contents of H and L using direct addressing.

Appendix- B-22

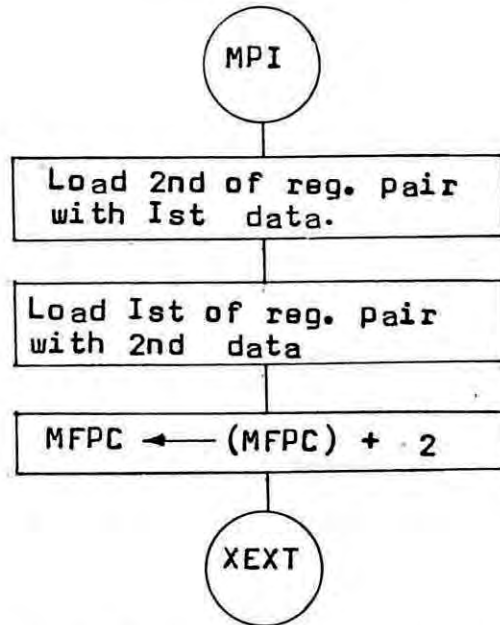


Fig. B-22 LXI-Load register pair with immediate data

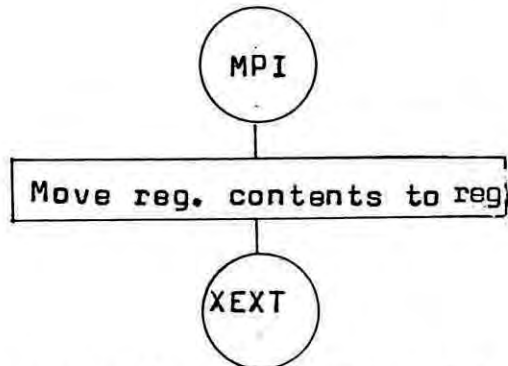


Fig. B-23. MOV (reg,reg)-Move data between registers

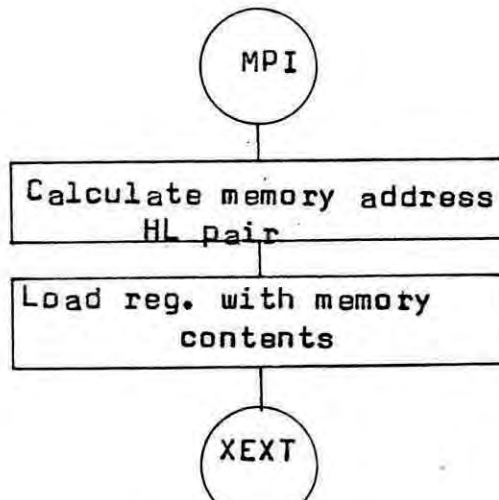


Fig. B-24. MOV (reg,M)-Move data from memory to register

Appendix- B- 23

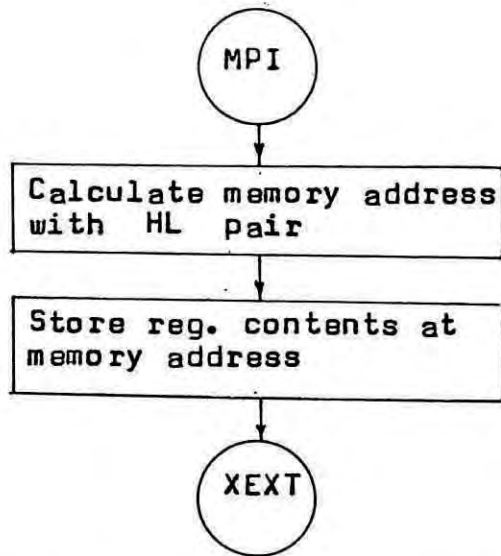


Fig. B-25. MOV(M, reg)-Move data from register to memory

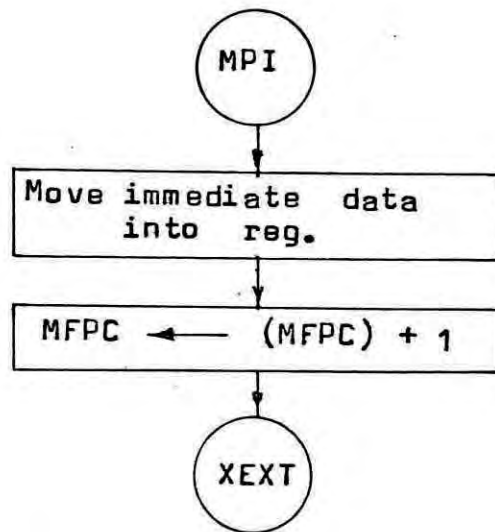


Fig. B-26. MVI reg-Move immediate data to register

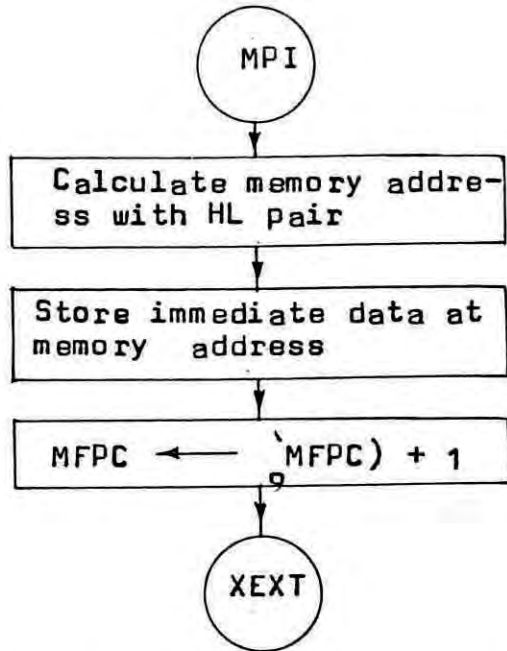


Fig. B-27 MVI M-Move immediate data to memory

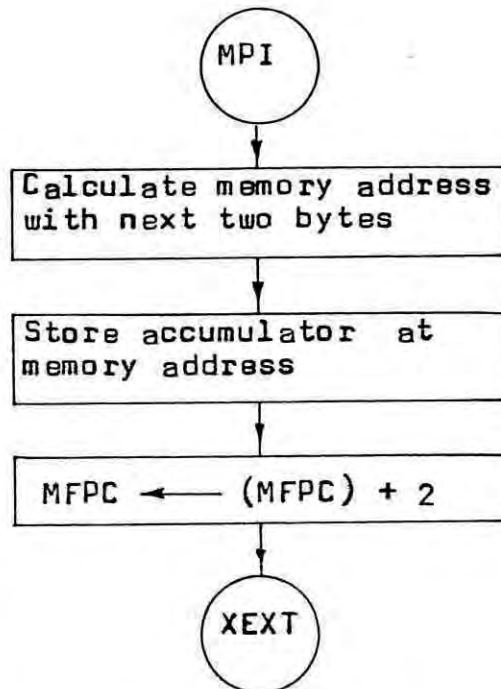


Fig. B-28 STA-Store contents of accumulator into memory

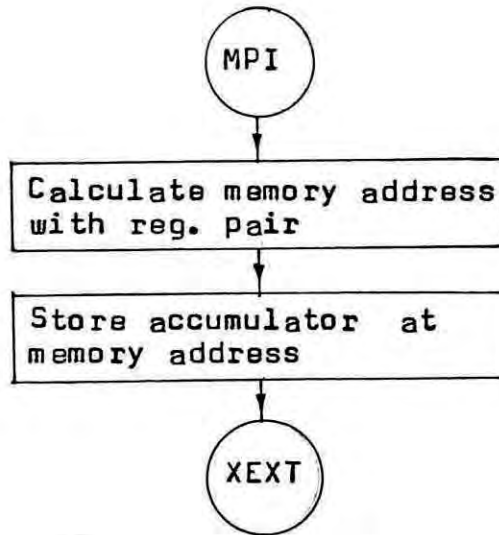


Fig. B-29 STAX- Store accumulator contents memory location addressed by register pair

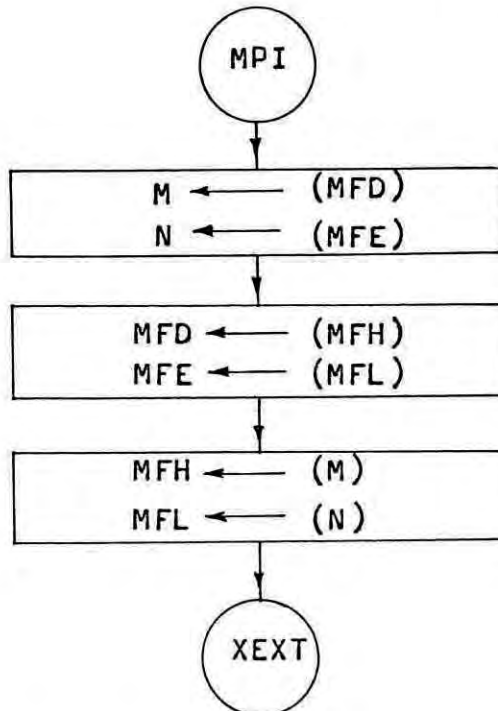


Fig. B-30 XCHG-Exchange MFD,MFE and MFH,MFL contents

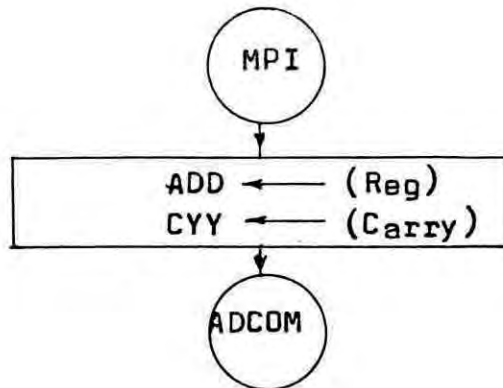


Fig. B-31 ADC reg-Add register with carry

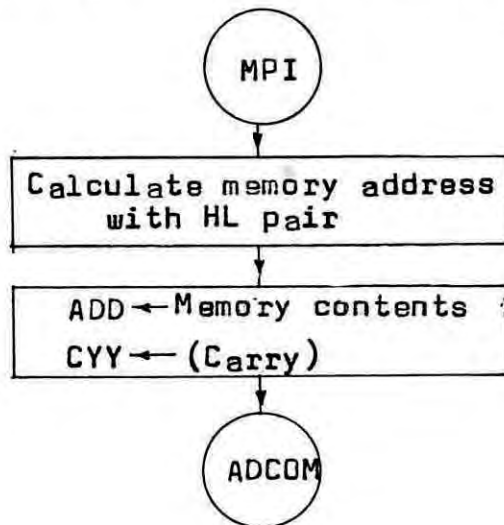


Fig. B-32 ADC M- Add memory contents with carry

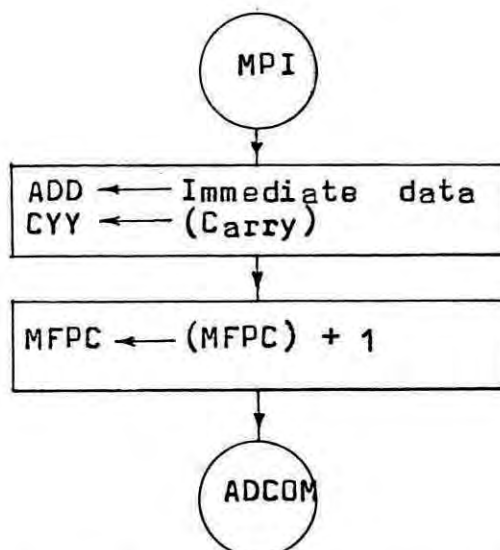


Fig. 33. ACI- Add accumulator and immediate data with carry.

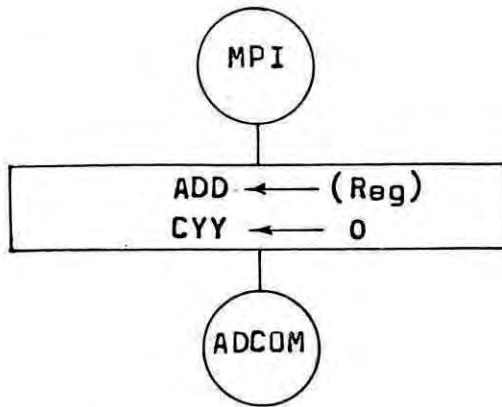


Fig. B- 34 ADD reg-Add. register to contents of accumulator

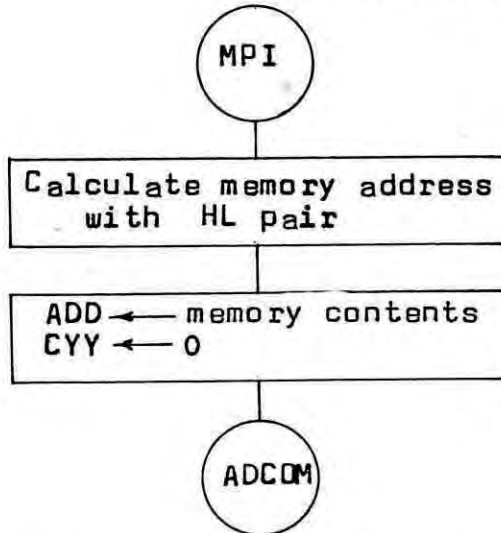


Fig. B-35 ADD M- Add memory contents

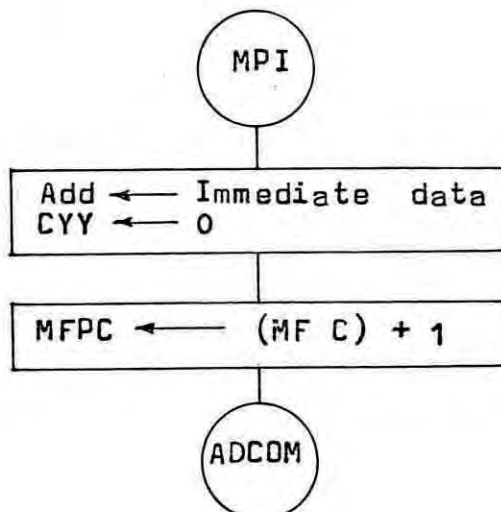


Fig. B-36 ADI-Add immediate data

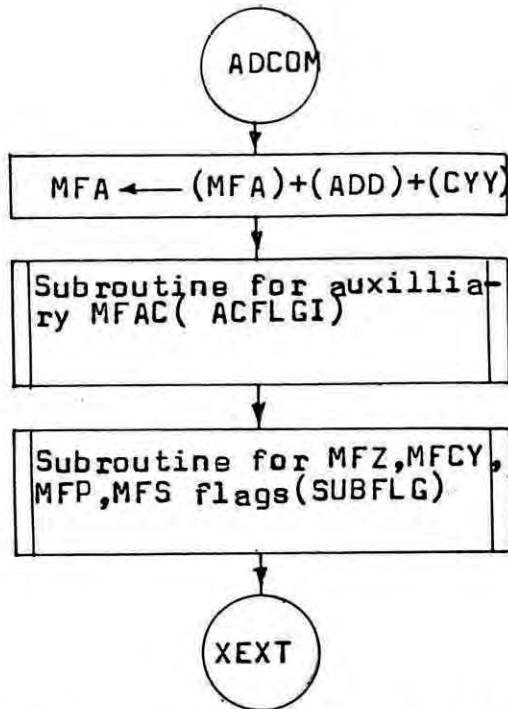


Fig. B-37 Continuation of Fig. B-31 to B-36

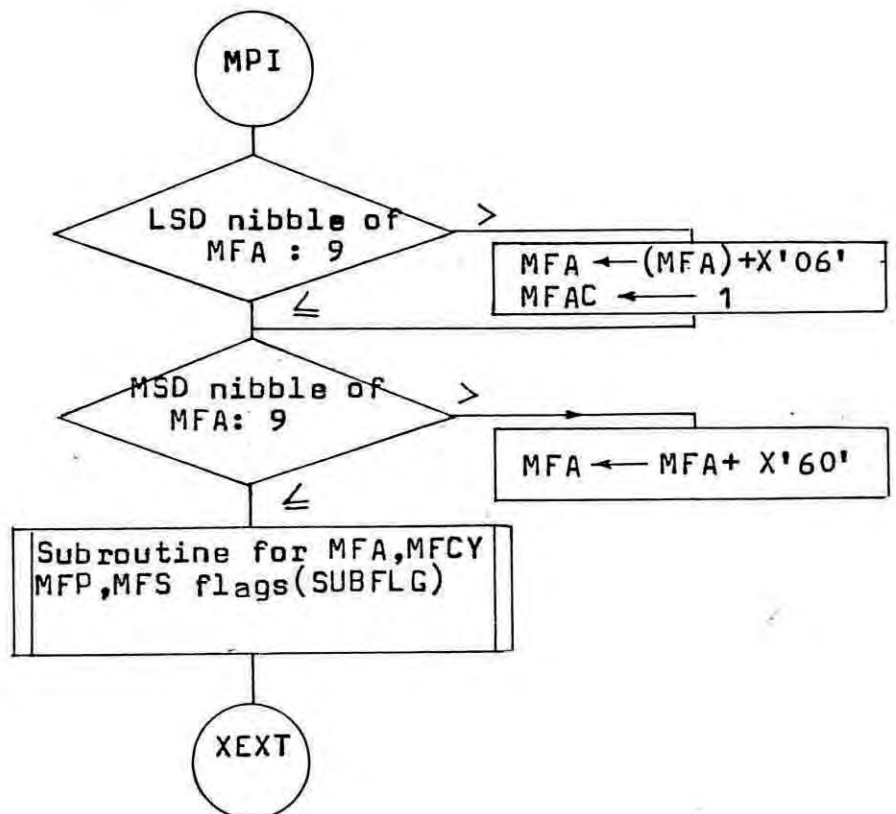


Fig. B-38 DAA-Decimal adjust accumulator



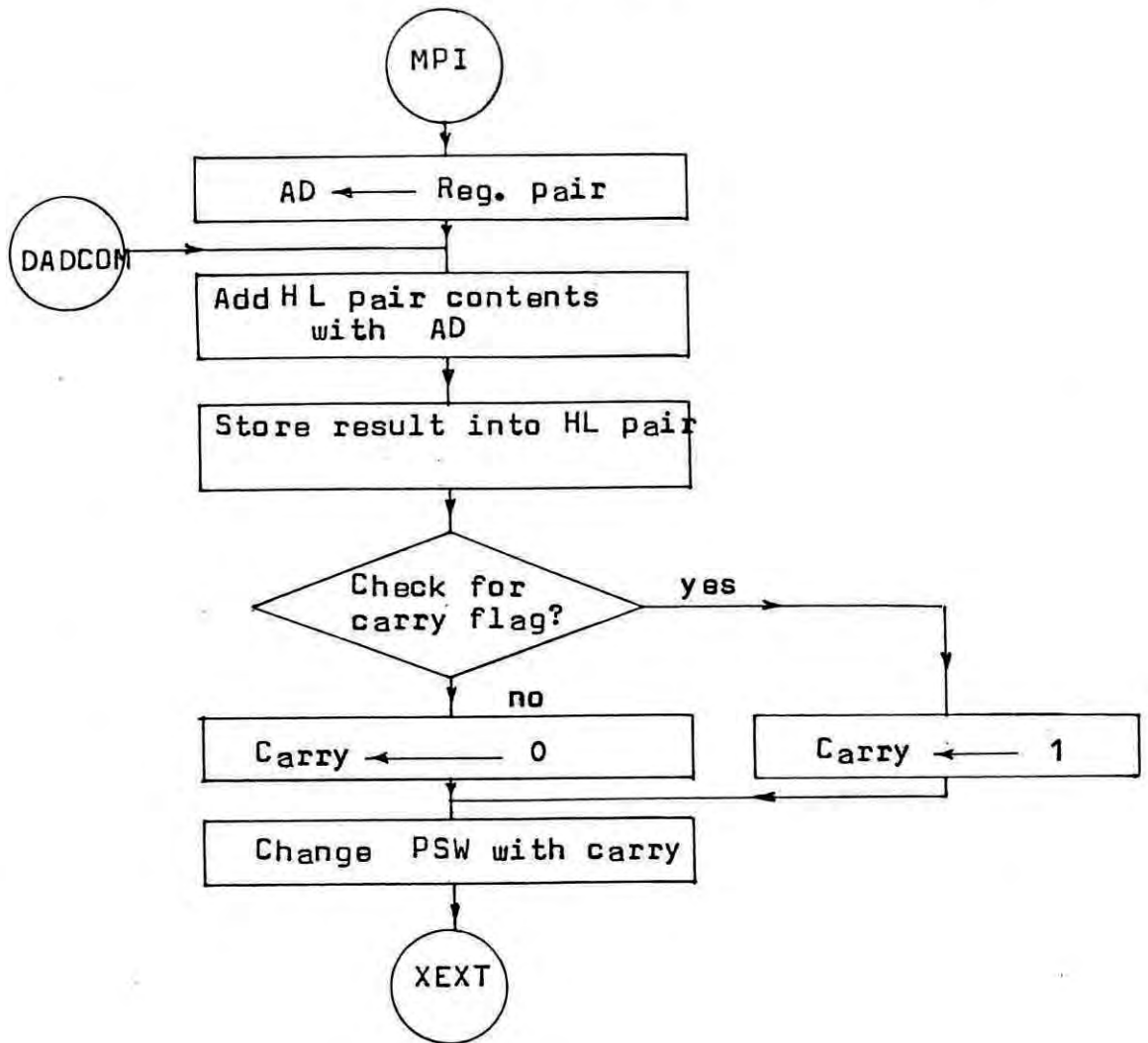


Fig. B-39 DAD-Double add a register pair to Hand L

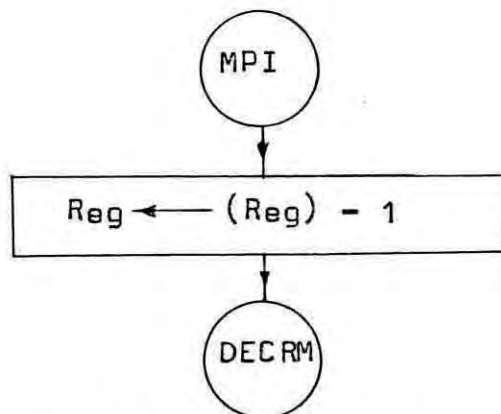


Fig. B-40 DCR reg-Decrement specified register contents

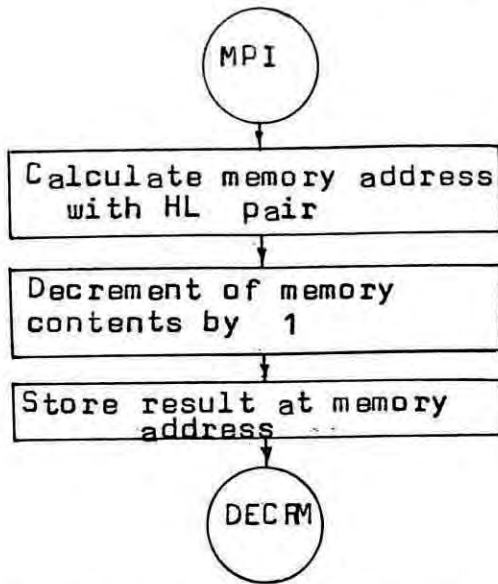


Fig. B-41 D RCM-Decrement of memory contents.

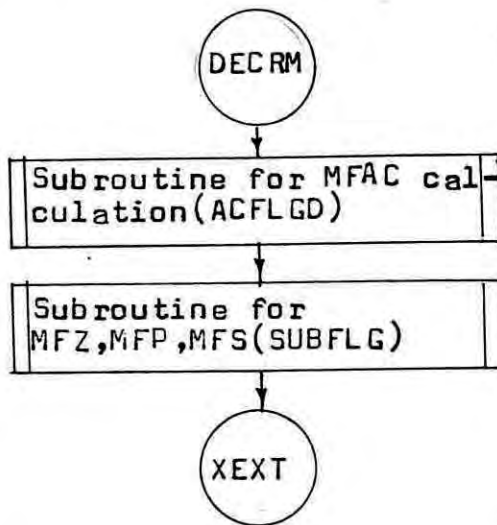


Fig. B-42 continuation of Fig. B-40 to B-41

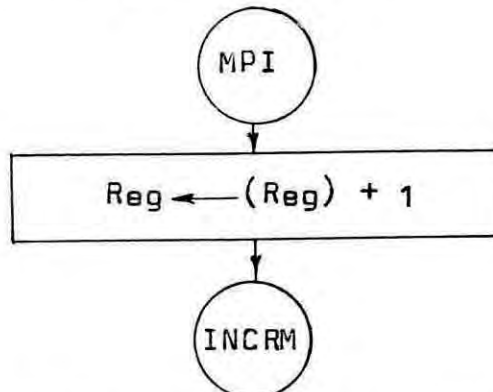


Fig. B-43 INR reg-Increment specified register

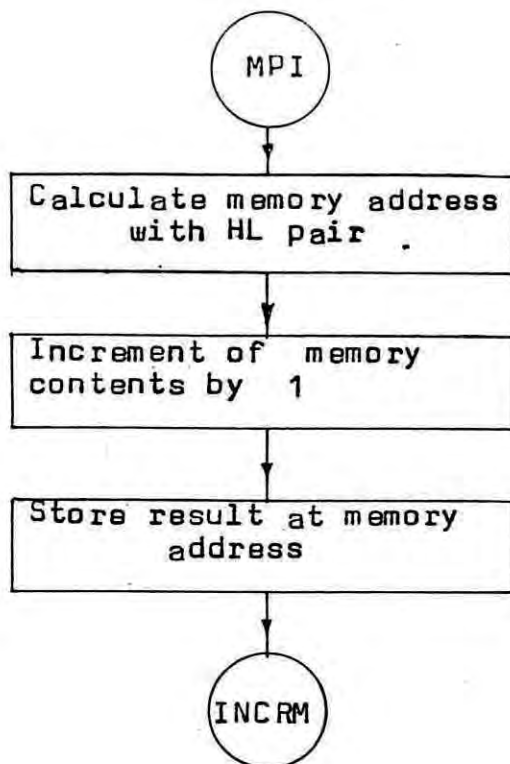


Fig. B-44 INR M-Increment specified memory contents

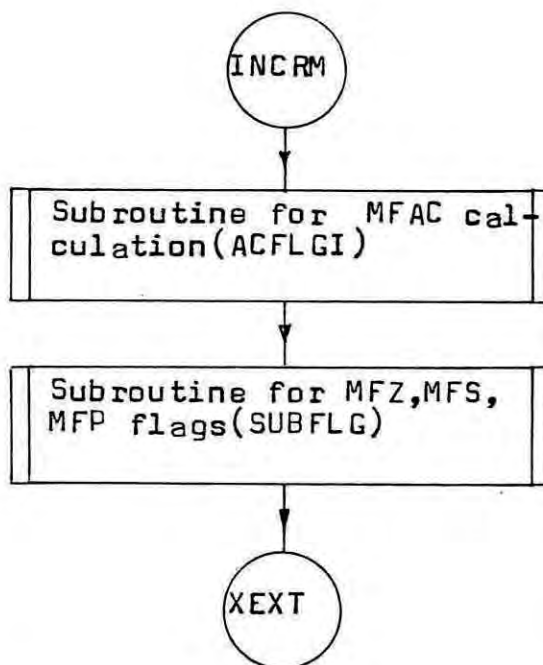


Fig. B-45 Continuation of Fig. B-43 and B-44

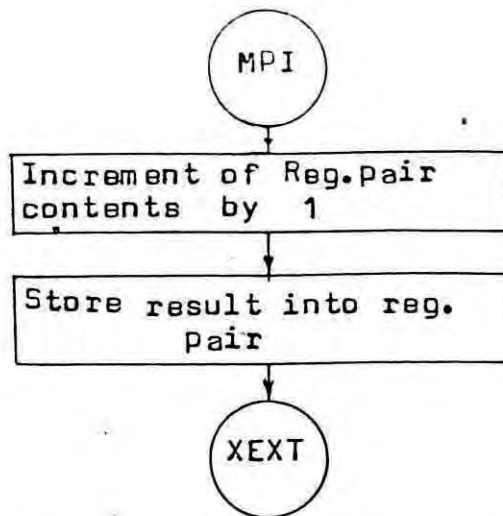


Fig. B-46 INX - Increment contents of a reg. pair

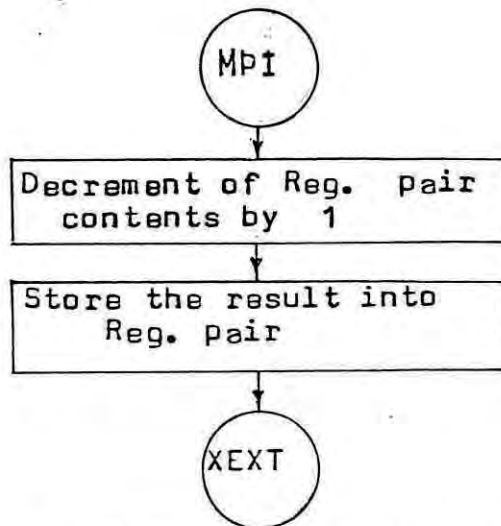


Fig. B-47 DCX - Decrement contents of a reg. pair

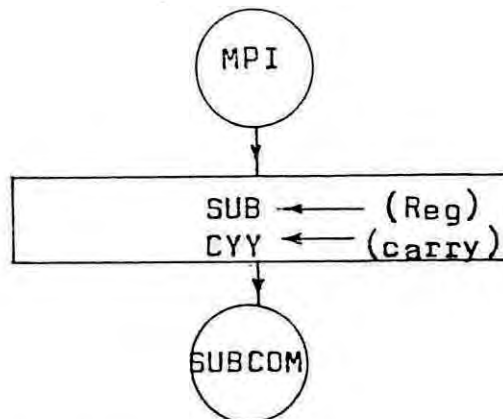


Fig. B-48 SBB reg-subtract contents of a register from accumulator with borrow

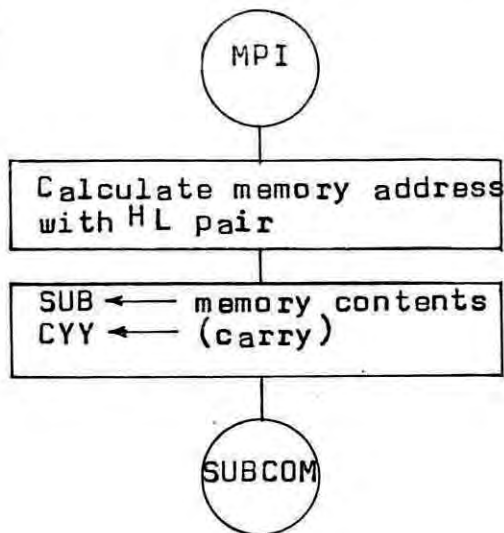


Fig. B-49 SBB M-Subtract memory contents from accumulator with borrow.

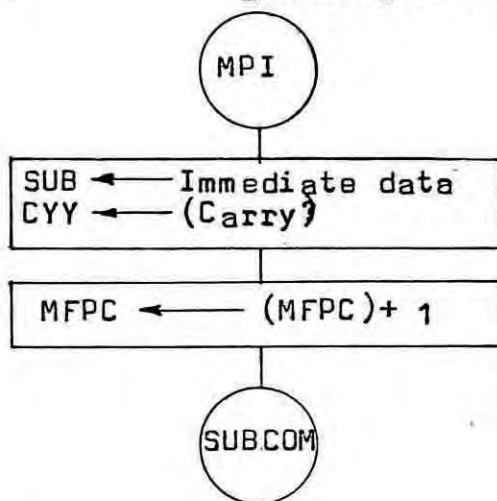


Fig. B-50 SBI-Subtract immediate data from accumulator with borrow

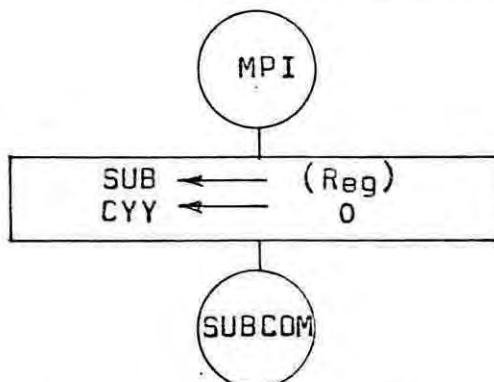


Fig. B-51 SUB reg-Subtract register contents from accumulator.

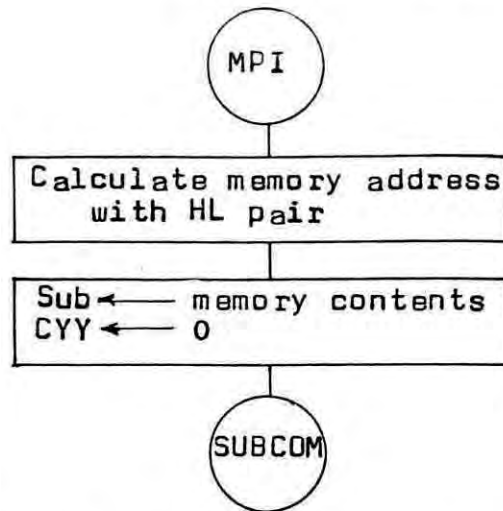


Fig. B- 52 SUB M-Subtract memory contents from accumulator

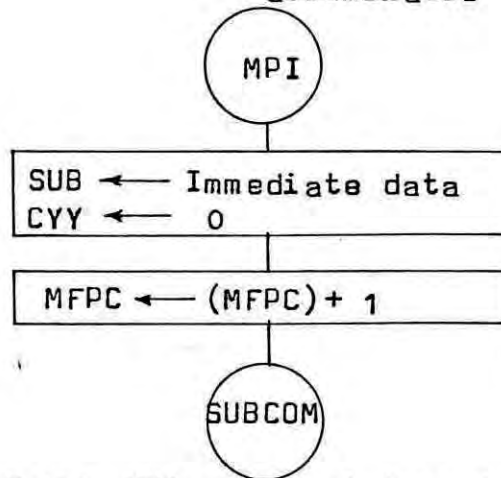


Fig. B-53 SUI-subtract immediate data from accumulator

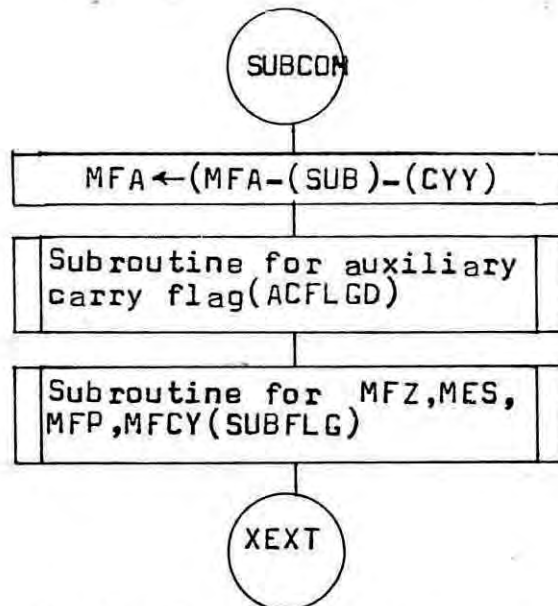


Fig. B-54 Continuation from Fig. B-48 to B-53.

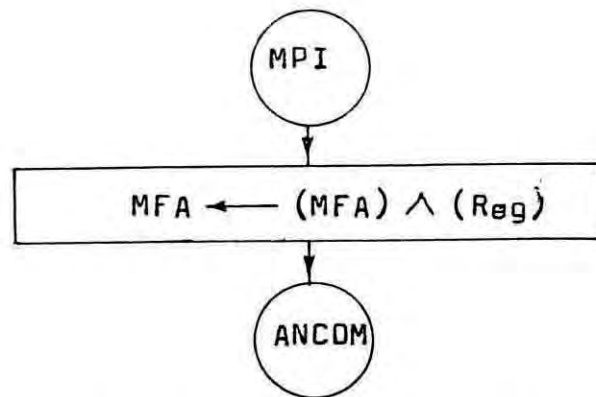


Fig. B-55 ANA reg-Logical AND register with accumulator.

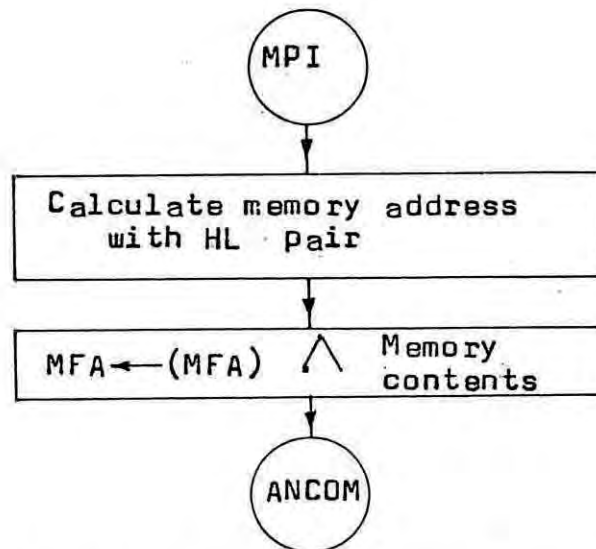


Fig. B-56 ANA M-Logical AND memory contents with accumulator

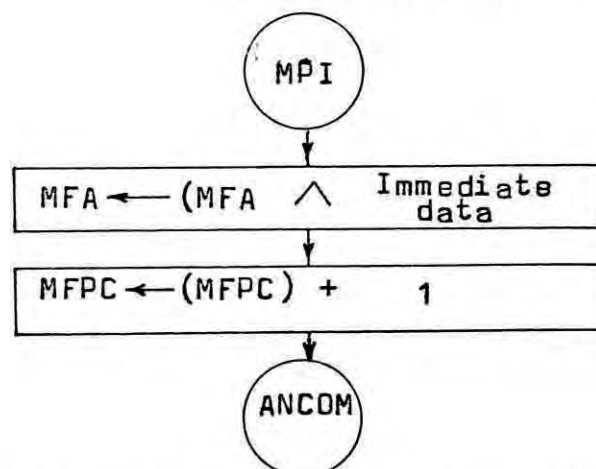


Fig. B-57 ANI-Logical AND immediate data with accumulator

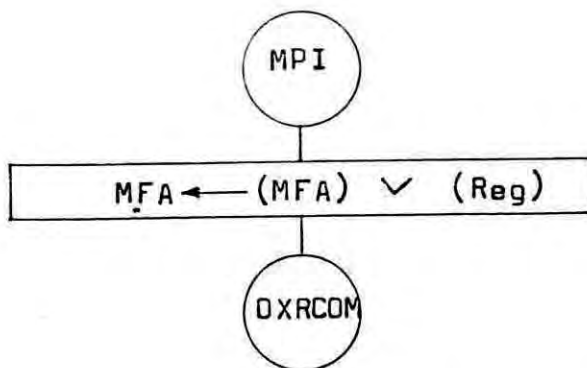


Fig. B-58 DRA reg-logical OR the register contents with accumulator.

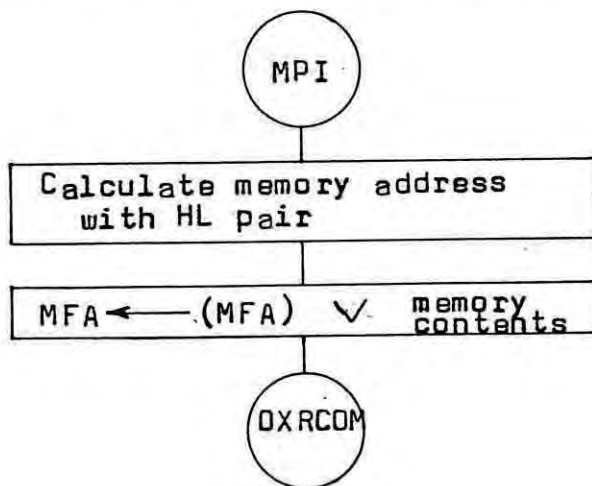


Fig. B-59 DRA M-Logical OR the accumulator with register contents.

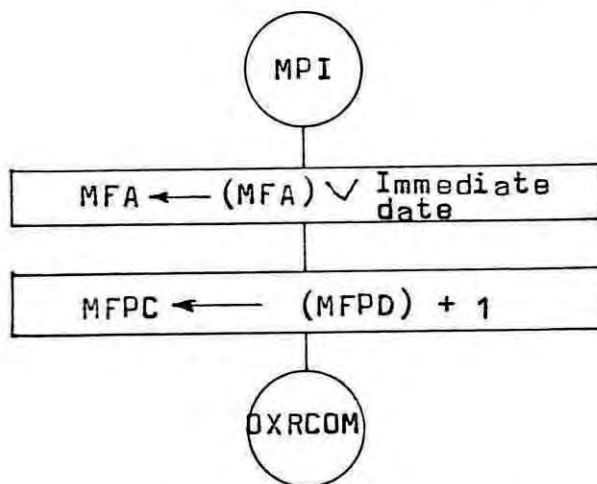


Fig. B-60 ORI-Logical OR the accumulator with immediate data.



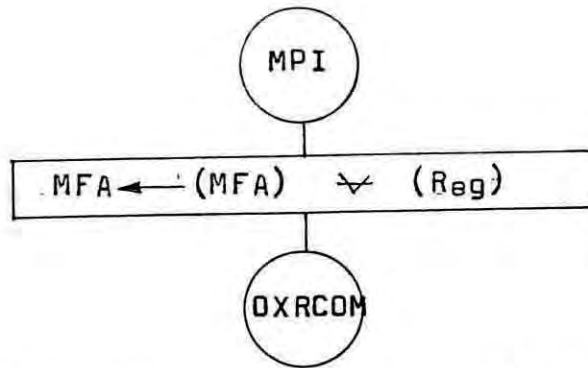


Fig. B-61 XRA reg-Exclusive OR accumulator with register contents.

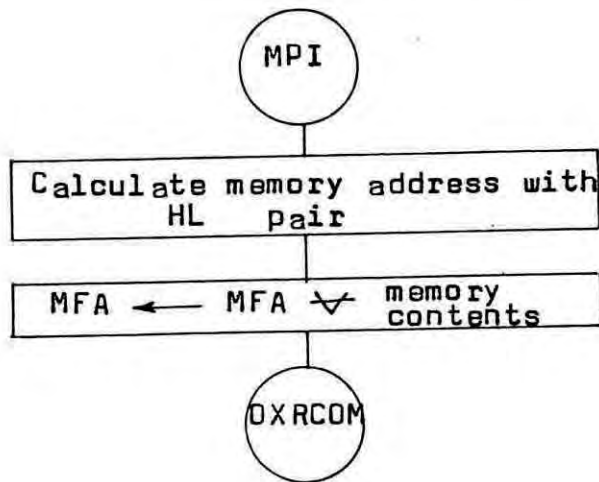


Fig. B-62 XRA M-Exclusive OR accumulator with memory contents.

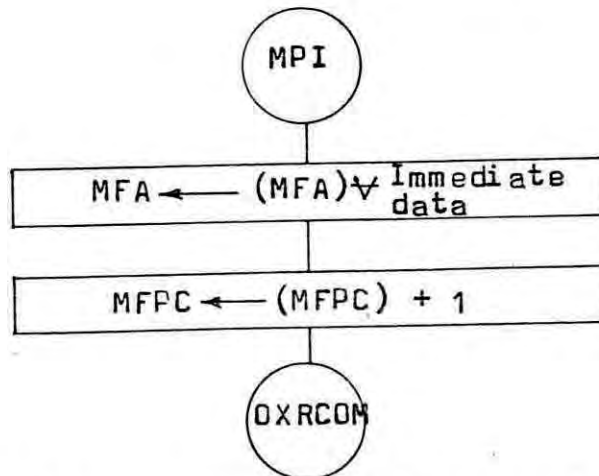


Fig. B-63 XRI-Exclusive OR accumulator contents with immediate data.

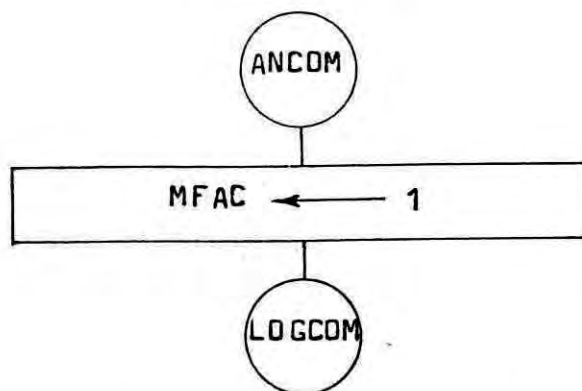


Fig. B-64 Continuation from Fig. B-55 to B-57

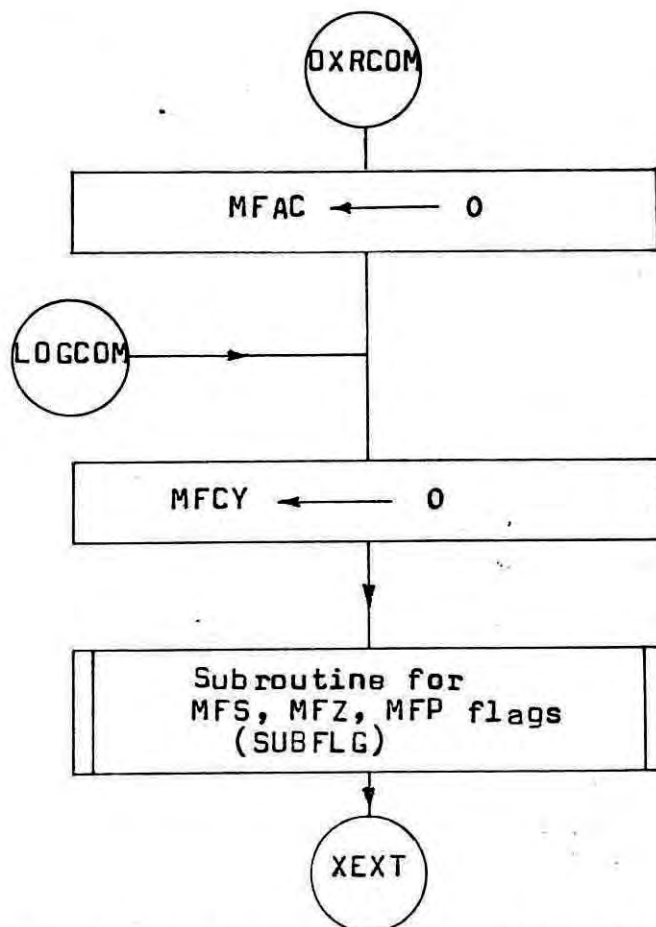


Fig. B-65 Continuation from Fig. B-58 to B-63 and B-64.

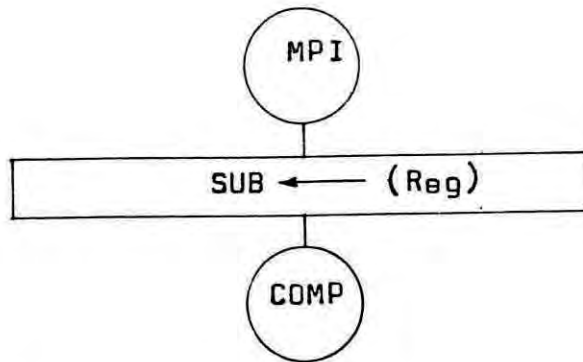


Fig. B-66 COMP reg-Compare register contents with accumulator

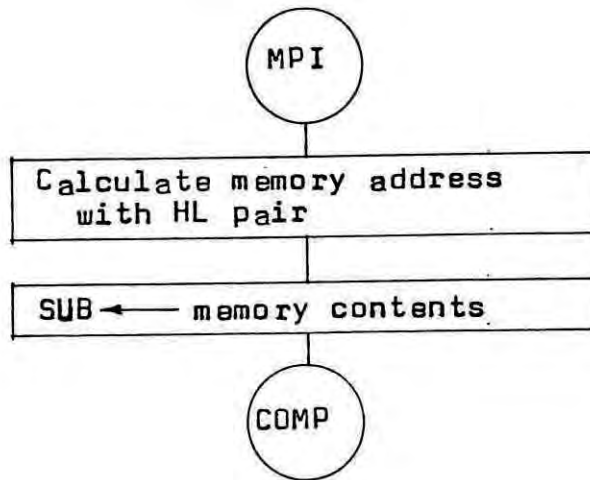


Fig. B-67 COMP M-Compare memory contents with accumulator

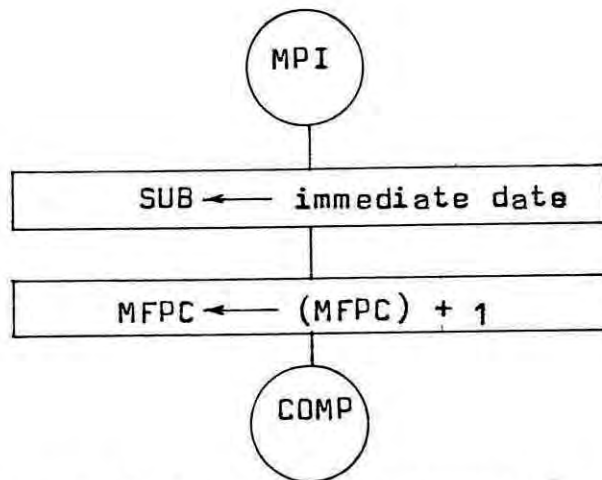


Fig. B-69 CPI-Compare immediate data with accumulator

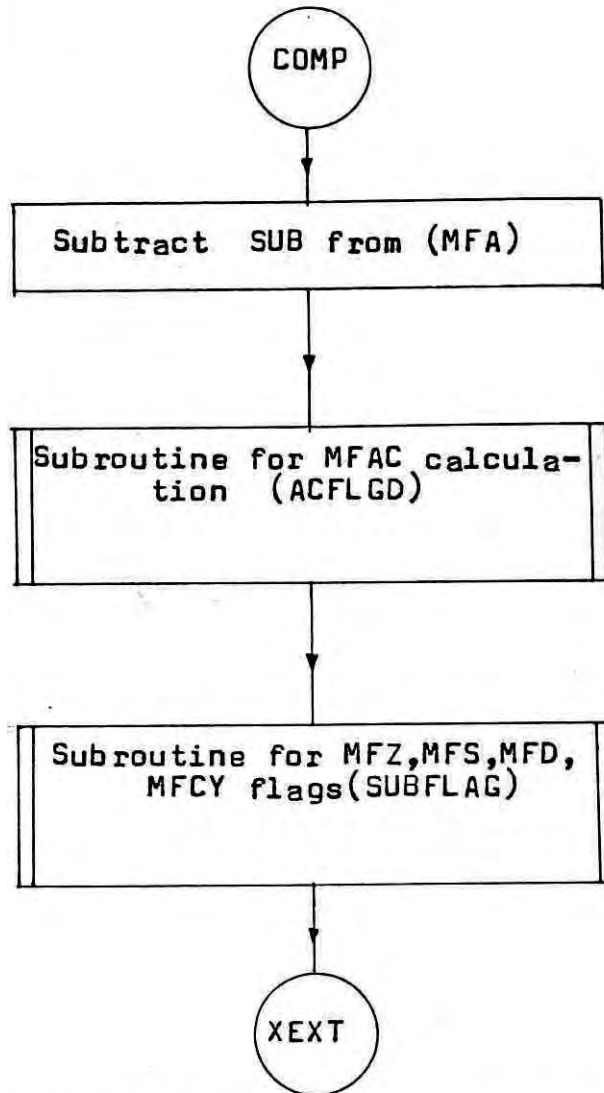


Fig. B-69 Continuation from Fig. B-66 to B-68.

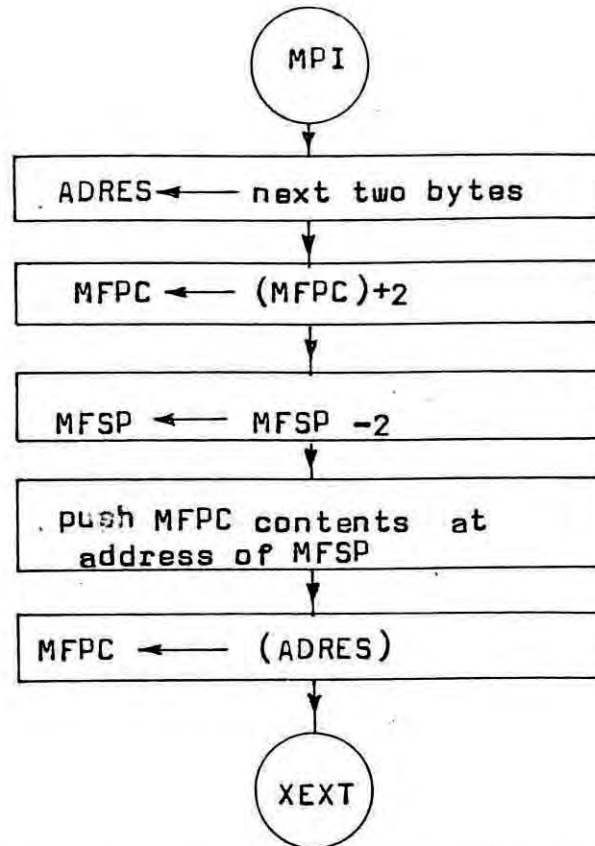


Fig. B-70 CALL-call subroutine defined in operand

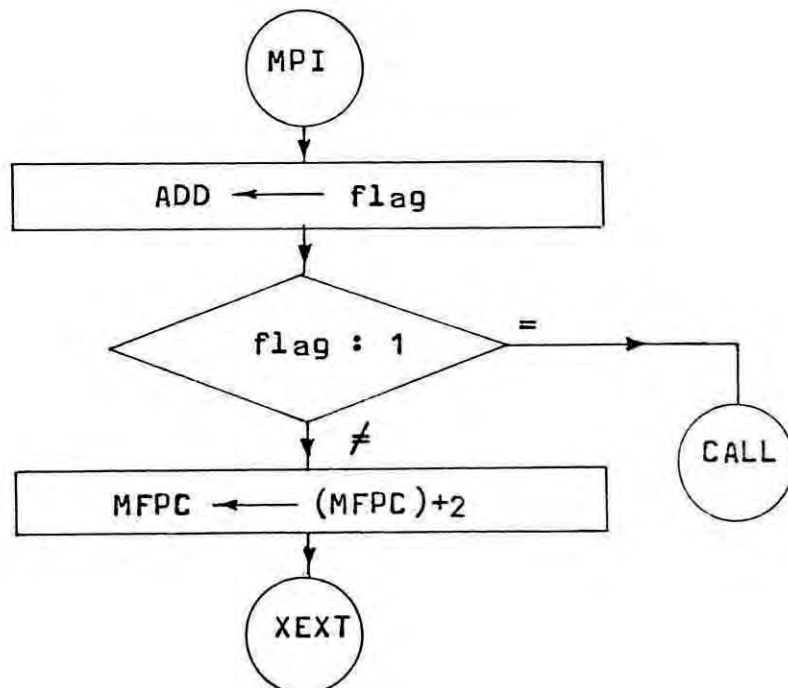


Fig. B-71 Call if flag is 1

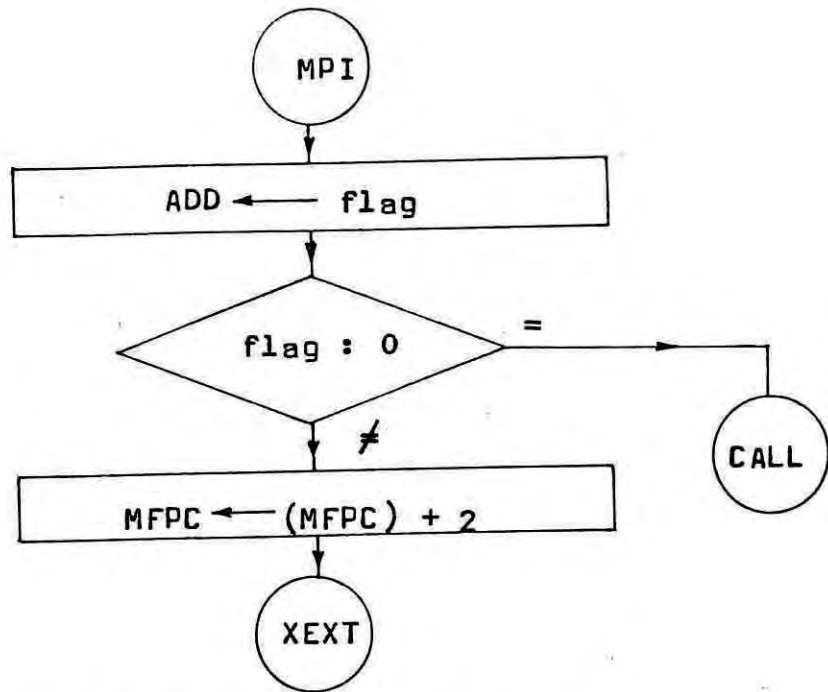


Fig. B-72 Call if flag is 0

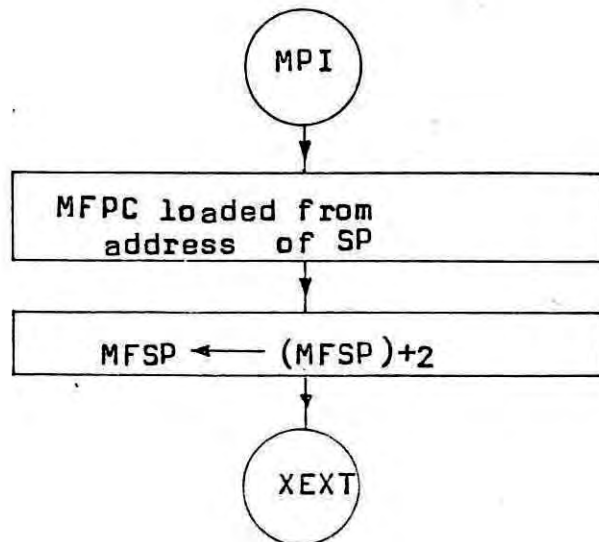


Fig. B-73 RET-Return from subroutine

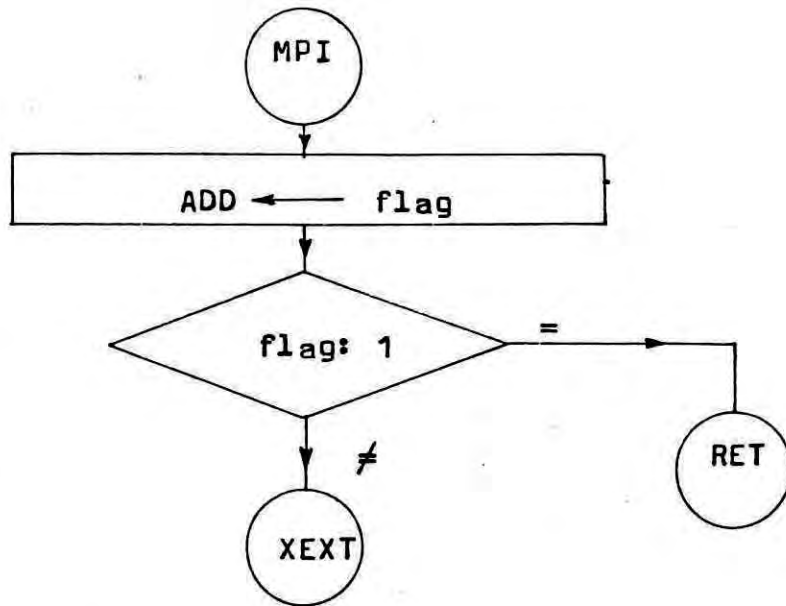


Fig. B-74 Return from subroutine if flag is 1

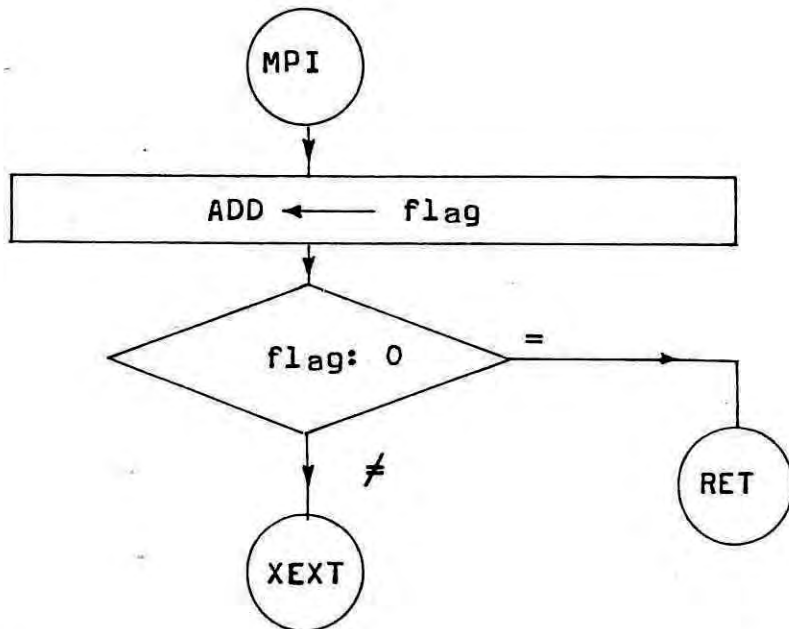


Fig. B-75 Return from subroutine if flag is 0.

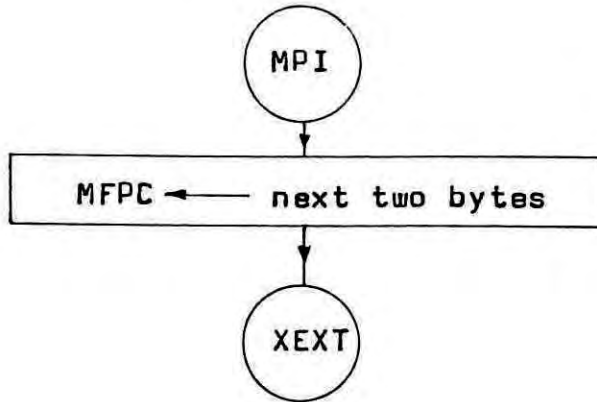


Fig. B-76 JMP= jump to the location addressed by 2nd and 3rd instruction byte.

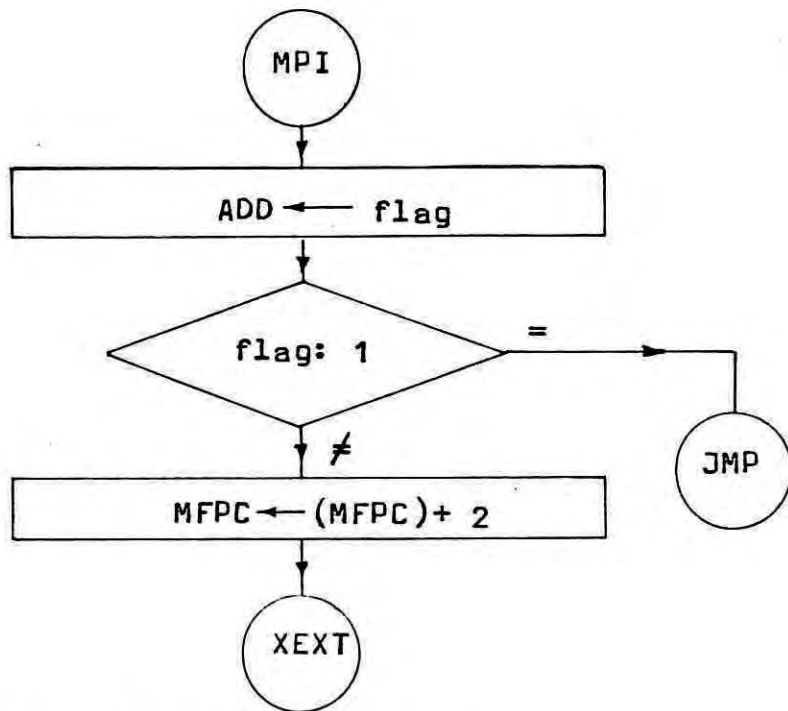


Fig. B-77 Jump if flag is 1



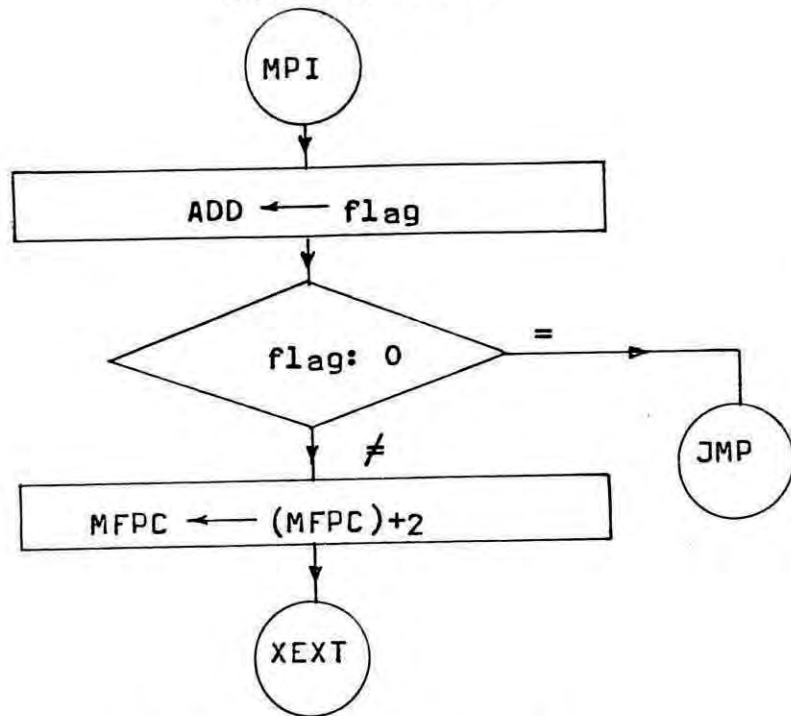


Fig. B-78 jump if flag is 0.

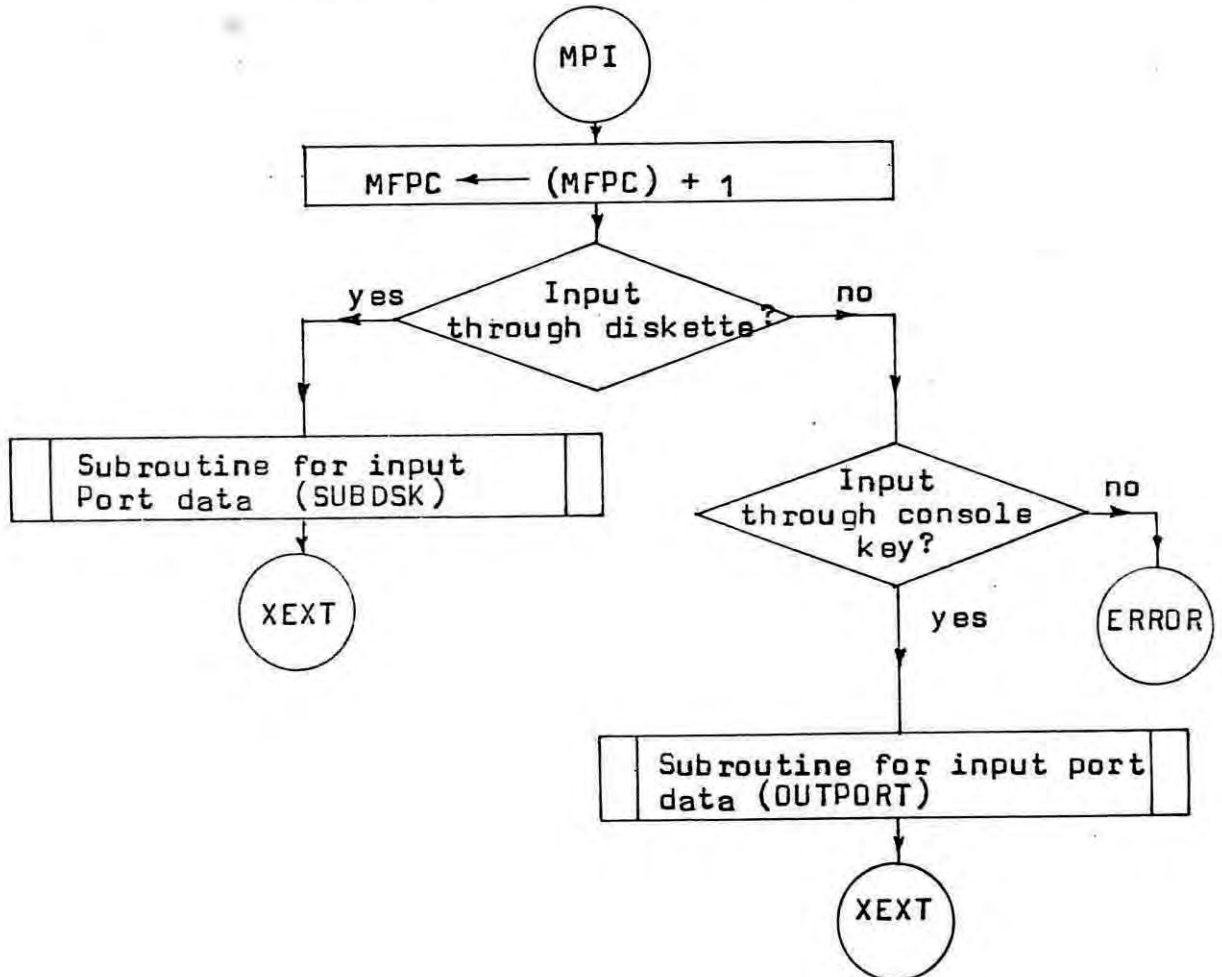


Fig. B-79 IN-Input into Accumulate from simulated port (diskette/console key board)

Appendix-B-46

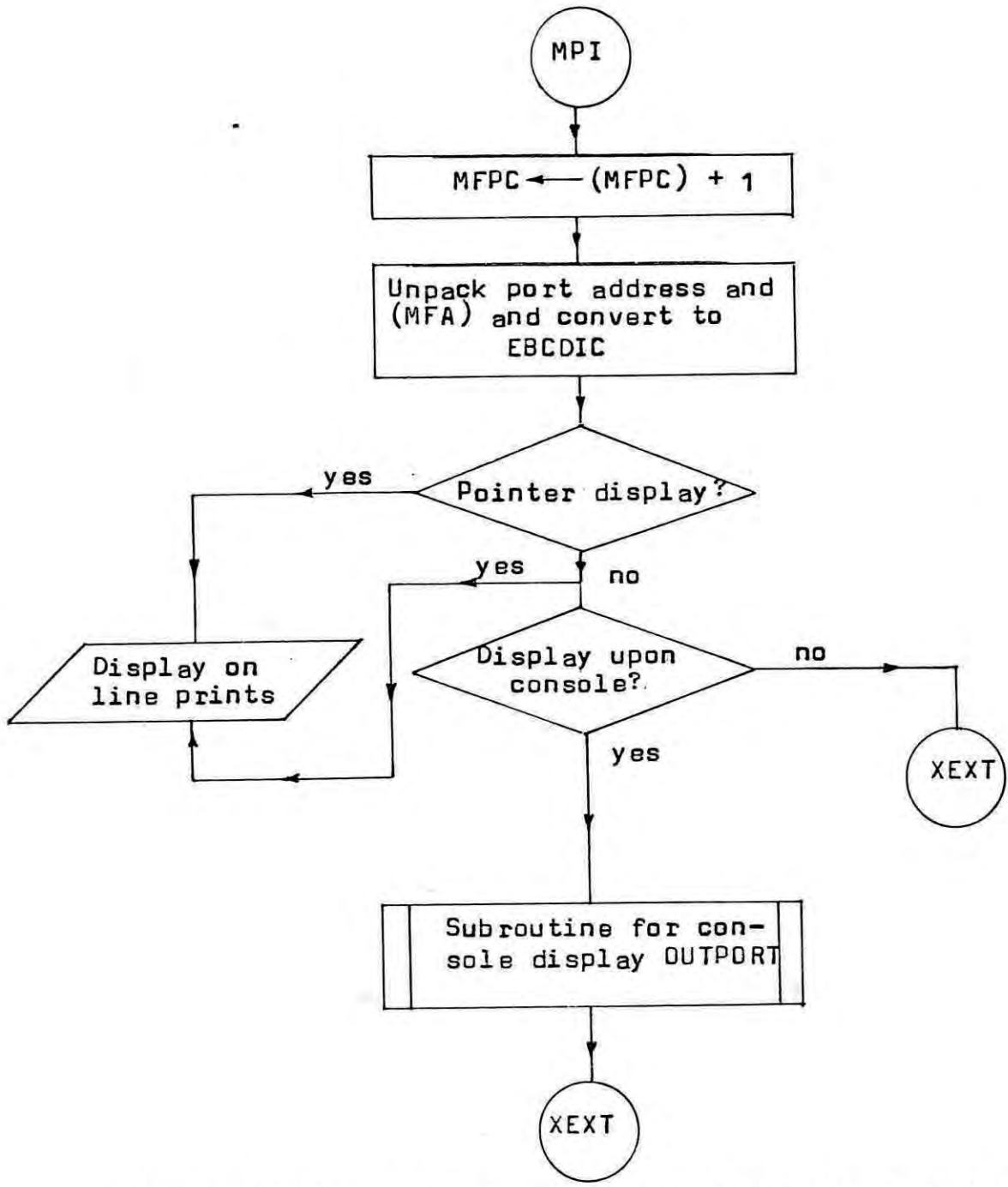


Fig. 80 OUT-Output contents of Accumulator on simulated port(lime printer/console).

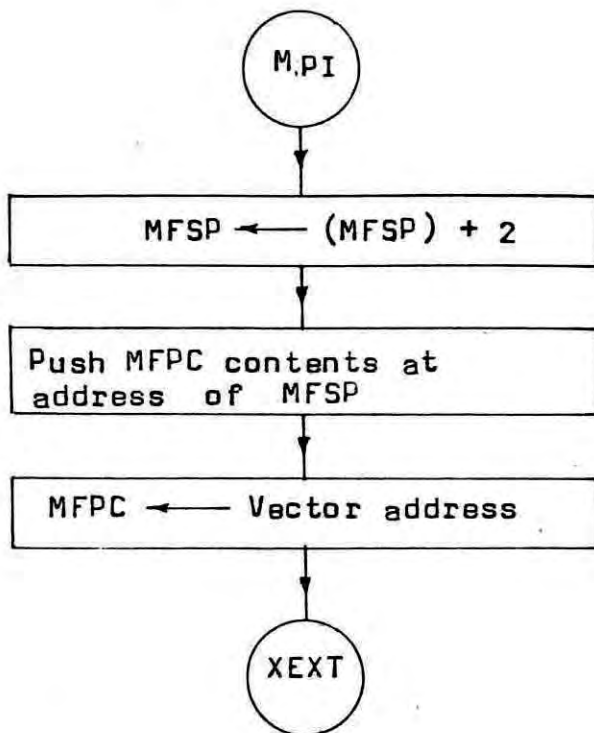


Fig. B-81 RST-Restart

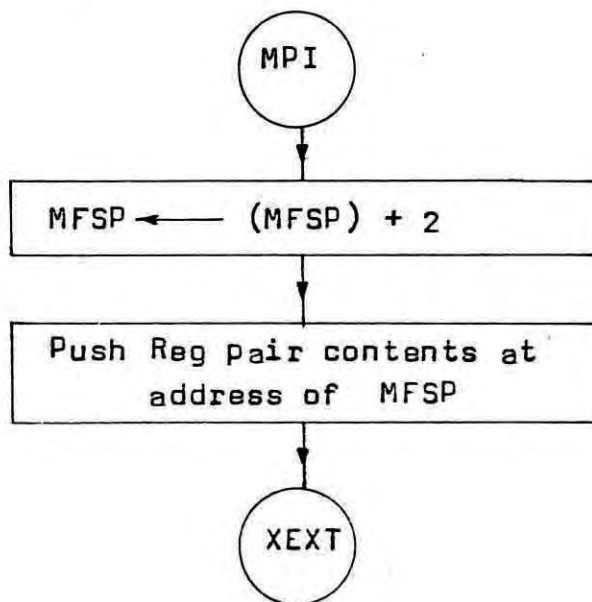


Fig. B-82 PUSH-Write contents of specified register pair into top of stack.

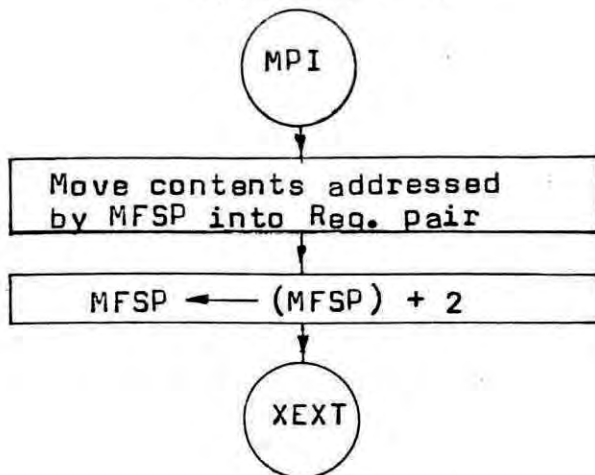


Fig. B-83 POP-Read top two stack bytes into specified reg. pair.

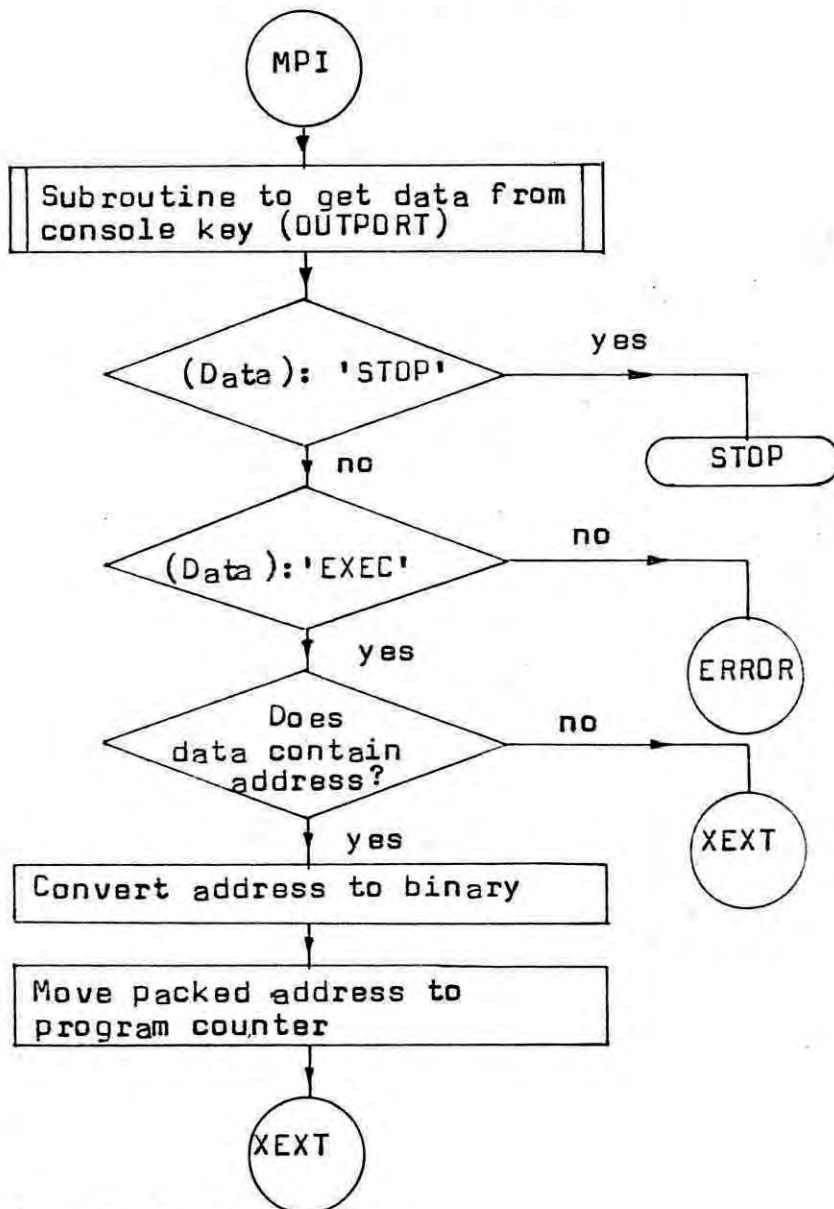


Fig. B-84 HLT - Halt.  
F

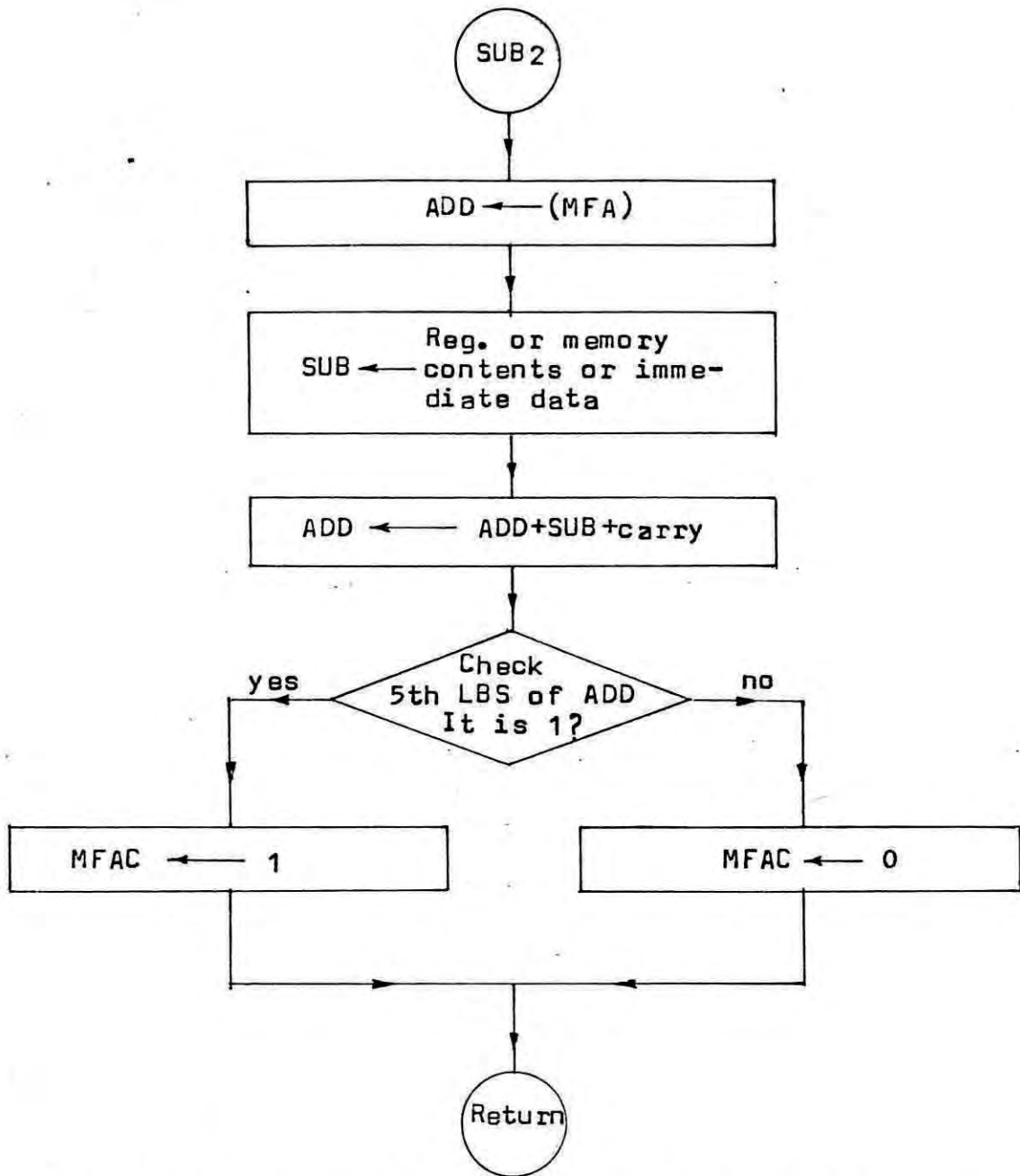


Fig. B-85 Flowchart of a subroutine to set the auxiliary carry flag (for addition and increment operation)

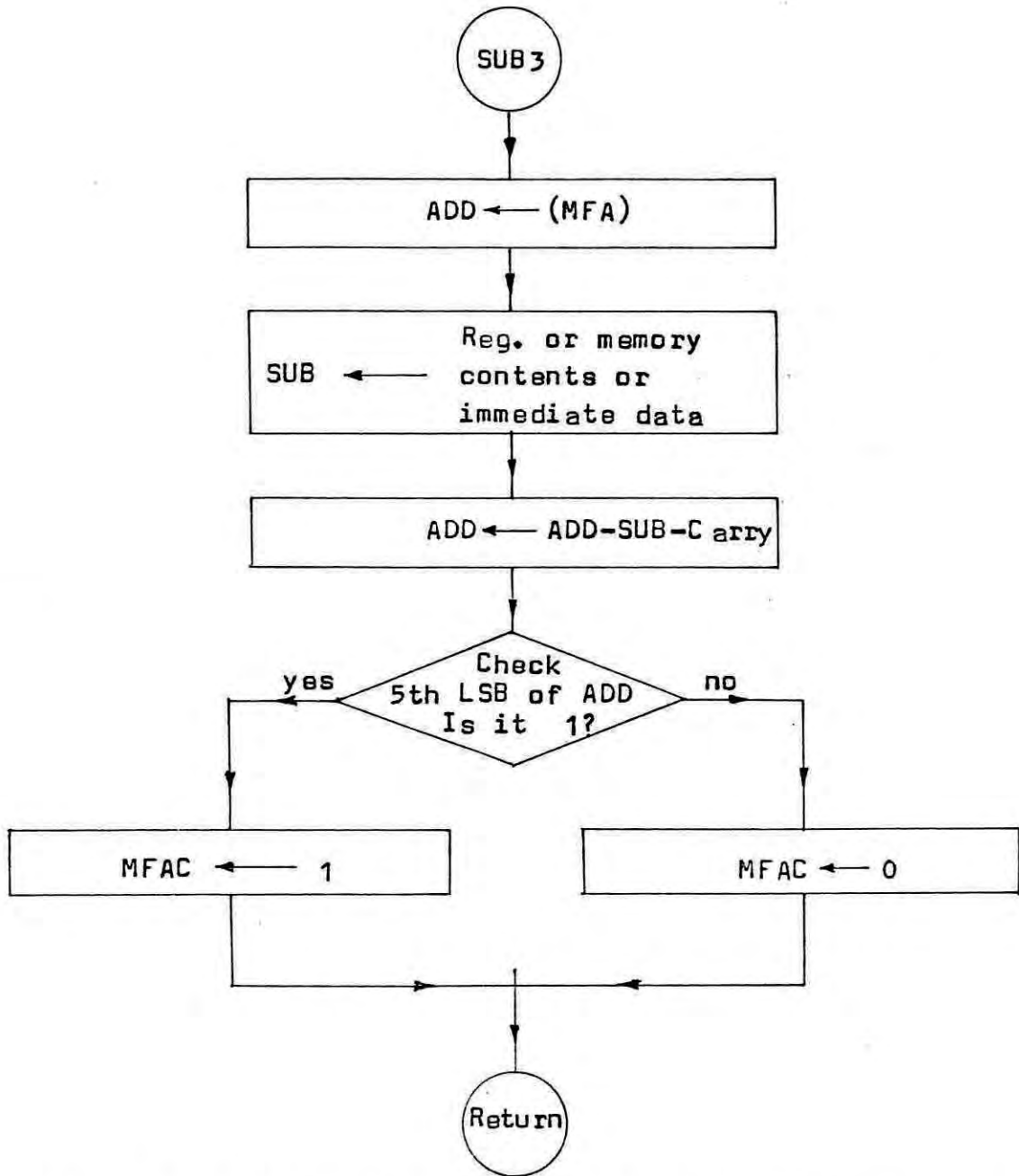


Fig. B-86 Subroutine flowchart to set auxiliary carry flag (for subtraction and decrement operation)

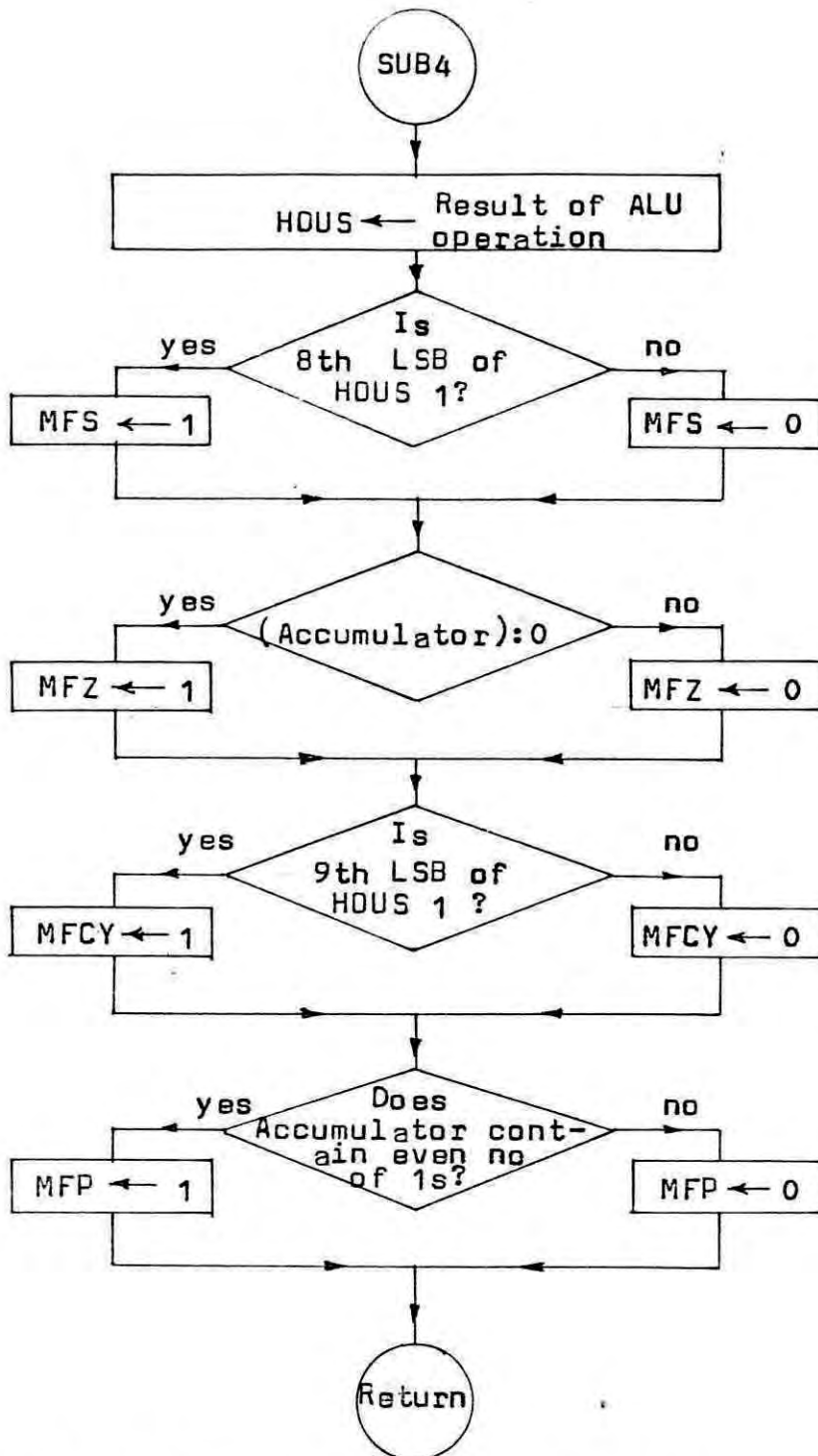


Fig. B-87 Subroutine to set sign, zero, carry and parity flags.

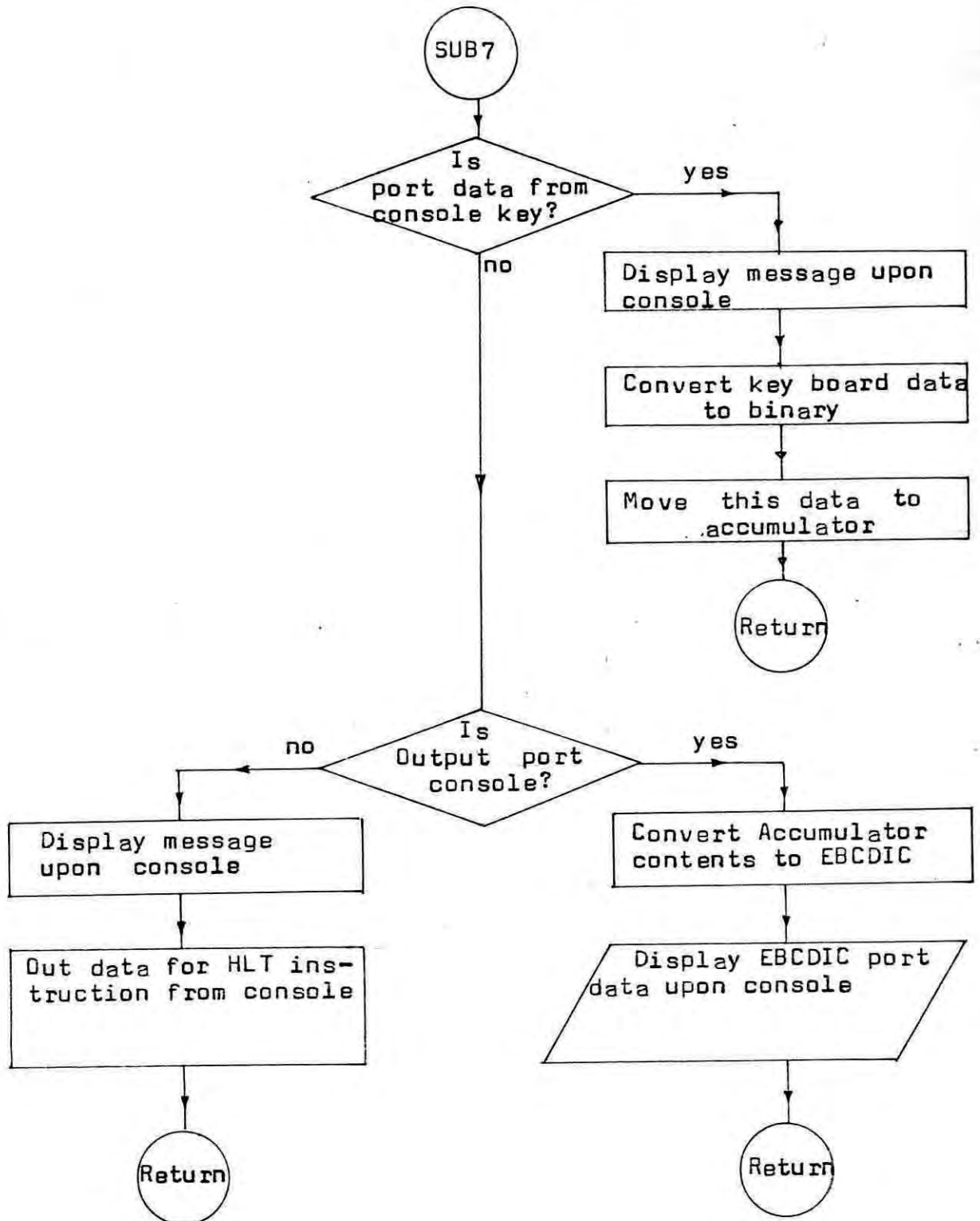


Fig. B-88 : Subroutine for console operations.



APPENDIX - C

INTEL 8085 INSTRUCTION SET

## DATA TRANSFER GROUP

Move	Move (cont)	Move Immediate
MOV { A,A 7F A,B 78 A,C 79 A,D 7A A,E 7B A,H 7C A,L 7D A,M 7E	MOV { E,A 5F E,B 58 E,C 59 E,D 5A E,E 5B E,H 5C E,L 5D E,M 5E	MVI { A, byte 3E B, byte 06 C, byte 0E D, byte 16 E, byte 1E H, byte 26 L, byte 2E M, byte 36
MOV { B,A 47 B,B 40 B,C 41 B,D 42 B,E 43 B,H 44 B,L 45 B,M 46	MOV { H,A 67 H,B 60 H,C 61 H,D 62 H,E 63 H,H 64 H,L 65 H,M 66	LXI { B, dble 01 D, dble 11 H, dble 21 SP, dble 31
MOV { C,A 4F C,B 48 C,C 49 C,D 4A C,E 4B C,H 4C C,L 4D C,M 4E	MOV { L,A 6F L,B 68 L,C 69 L,D 6A L,E 6B L,H 6C L,L 6D L,M 6E	Load/Store LDAX B 0A LDAX D 1A LHLD adr 2A LDA adr 3A STAX B 02 STAX D 12 SHLD adr 22 STA adr 32
MOV { D,A 57 D,B 50 D,C 51 D,D 52 D,E 53 D,H 54 D,L 55 D,M 56	MOV { M,A 77 M,B 70 M,C 71 M,D 72 M,E 73 M,H 74 M,L 75	
	XCHG EB	

- byte = constant, or logical/arithmetic expression that evaluates to an 8-bit data quantity. (Second byte of 2-byte instructions).
- dble = constant, or logical/arithmetic expression that evaluates to a 16-bit data quantity. (Second and Third bytes of 3-byte instructions)
- adr = 16-bit address (Second and Third bytes of 3-byte instructions).
- \* = all flags (C, Z, S, P, AC) affected.
- \*\* = all flags except CARRY affected; (exception: INX and DCX affect no flags).
- † = only CARRY affected.

All mnemonics copyright ©Intel Corporation 1976.

## ARITHMETIC AND LOGICAL GROUP

Add*	Incrementer	Logical*
ADD { A 87 B 80 C 81 D 82 E 83 H 84 L 85 M 86	INR { A 3C B 04 C 0C D 14 E 1C H 24 L 2C M 34	ANA { A A7 B A0 C A1 D A2 E A3 H A4 L A5 M A6
ADC { A 8F B 88 C 89 D 8A E 8B H 8C L 8D M 8E	INX { B 03 D 13 H 23 SP 33	XRA { A AF B A8 C A9 D AA E AB H AC L AD M AE
	Decrement**	
	DCR { A 3D B 05 C 0D D 15 E 1D H 25 L 2D M 35	ORA { A B7 B B0 C B1 D B2 E B3 H B4 L B5 M B6
	DCX { B 0B D 1B H 2B SP 3B	CMP { A BF B B8 C B9 D BA E BB H BC L BD M BE
	Specials	Arith & Logical Immediate
	DAA* 27 CMA 2F STC† 37 CMC† 3F	ADI byte C6 ACI byte CE SUI byte D6 SBI byte DE ANI byte E6 XRI byte EE ORI byte F6 CPI byte FE
	Double Add †	Rotate †
DAD { B 09 D 19 H 29 SP 39	RLC 07 RRC 0F RAL 17 RAR 1F	

### BRANCH CONTROL GROUP

Jump	
JMP adr	C3
JNZ adr	C2
JZ adr	CA
JNC adr	D2
JC adr	DA
JPO adr	E2
JPE adr	EA
JP adr	F2
JM adr	FA
PCHL	E9

#### Call

CALL adr	CD
CNZ adr	C4
CZ adr	CC
CNC adr	D4
CC adr	DC
CPO adr	E4
CPE adr	EC
CP adr	F4
CM adr	FC

#### Return

RET	C9
RNZ	C0
RZ	C8
RNC	D0
RC	D8
RPO	E0
RPE	E8
RP	F0
RM	F8

#### Restart

RST	0	C7
	1	CF
	2	D7
	3	DF
	4	E7
	5	EF
	6	F7
7	FF	

### I/O AND MACHINE CONTROL

#### Stack Ops

PUSH	B	C5
	D	D5
	H	E5
	PSW	F5
POP	B	C1
	D	D1
	H	E1
	PSW*	F1
XTHL	E3	
SPHL	F9	

#### Input/Output

OUT byte	D3
IN byte	DB

#### Control

DI	F3
EI	FB
NOP	00
HLT	76

#### New Instructions (8085 Only)

RIM	20
SIM	30

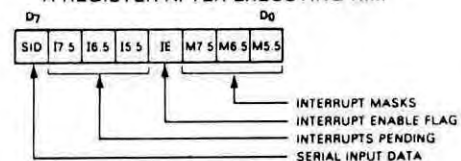
### RESTART TABLE

Name	Code	Restart Address
RST 0	C7	0000 <sub>16</sub>
RST 1	CF	0008 <sub>16</sub>
RST 2	D7	0010 <sub>16</sub>
RST 3	DF	0018 <sub>16</sub>
RST 4	E7	0020 <sub>16</sub>
TRAP	Hardware*	0024 <sub>16</sub>
	Function	
RST 5	EF	0028 <sub>16</sub>
RST 5 5	Hardware*	002C <sub>16</sub>
	Function	
RST 6	F7	0030 <sub>16</sub>
RST 6 5	Hardware*	0034 <sub>16</sub>
	Function	
RST 7	FF	0038 <sub>16</sub>
RST 7 5	Hardware*	003C <sub>16</sub>
	Function	

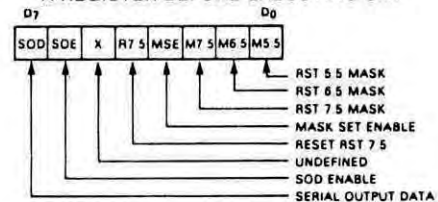
\*NOTE The hardware functions refer to the on-chip Interrupt feature of the 8085 only.

### USE OF THE A REGISTER BY RIM AND SIM INSTRUCTIONS (8085 ONLY)

#### A REGISTER AFTER EXECUTING RIM



#### A REGISTER BEFORE EXECUTING SIM



00	NOP	2B	DCX	H	56	MOV	D,M	81	ADD	C	AC	XRA	H	D7	RST	2	
01	LXI	B,dbble	2C	INR	L	57	MOV	D,A	82	ADD	D	AD	XRA	L	D8	RC	
02	STAX	B	2D	DCR	L	58	MOV	E,B	83	ADD	E	AE	XRA	M	D9	---	
03	INX	B	2E	MVI	L,byte	59	MOV	E,C	84	ADD	H	AF	XRA	A	DA	JC	adr
04	INR	B	2F	CMA		5A	MOV	E,D	85	ADD	L	B0	ORA	B	DB	IN	byte
05	DCR	B	30	SIM*		5B	MOV	E,E	86	ADD	M	B1	ORA	C	DC	CC	adr
06	MVI	B,byte	31	LXI	SP,dbble	5C	MOV	E,H	87	ADD	A	B2	ORA	D	DD	---	
07	RLC		32	STA	adr	5D	MOV	E,L	88	ADC	B	B3	ORA	E	DE	SBI	byte
08	---		33	INX	SP	5E	MOV	E,M	89	ADC	C	B4	ORA	H	DF	RST	3
09	DAD	B	34	INR	M	5F	MOV	E,A	8A	ADC	D	B5	ORA	L	E0	RPO	
0A	LDAX	B	35	DCR	M	60	MOV	H,B	8B	ADC	E	B6	ORA	M	E1	POP	H
0B	DCX	B	36	MVI	M,byte	61	MOV	H,C	8C	ADC	H	B7	ORA	A	E2	JPO	adr
0C	INR	C	37	STC		62	MOV	H,D	8D	ADC	L	B8	CMP	B	E3	XTHL	
0D	DCR	C	38	---		63	MOV	H,E	8E	ADC	M	B9	CMP	C	E4	CPO	adr
0E	MVI	C,byte	39	DAD	SP	64	MOV	H,H	8F	ADC	A	BA	CMP	D	E5	PUSH	H
0F	RRC		3A	LDA	adr	65	MOV	H,L	90	SUB	B	BB	CMP	E	E6	ANI	byte
10	---		3B	DCX	SP	66	MOV	H,M	91	SUB	C	BC	CMP	H	E7	RST	4
11	LXI	D,dbble	3C	INR	A	67	MOV	H,A	92	SUB	D	BD	CMP	L	E8	RPE	
12	STAX	D	3D	DCR	A	68	MOV	L,B	93	SUB	E	BE	CMP	M	E9	PCHL	
13	INX	D	3E	MVI	A,byte	69	MOV	L,C	94	SUB	H	BF	CMP	A	EA	JPE	adr
14	INR	D	3F	CMC		6A	MOV	L,D	95	SUB	L	C0	RNZ		EB	XCHG	
15	DCR	D	40	MOV	B,B	6B	MOV	L,E	96	SUB	M	C1	POP	B	EC	CPE	adr
16	MVI	D,byte	41	MOV	B,C	6C	MOV	L,H	97	SUB	A	C2	JNZ	adr	ED	---	
17	RAL		42	MOV	B,D	6D	MOV	L,L	98	SBB	B	C3	JMP	adr	EE	XRI	byte
18	---		43	MOV	B,E	6E	MOV	L,M	99	SBB	C	C4	CNZ	adr	EF	RST	5
19	DAD	D	44	MOV	B,H	6F	MOV	L,A	9A	SBB	D	C5	PUSH	B	F0	RP	
1A	LDAX	D	45	MOV	B,L	70	MOV	M,B	9B	SBB	E	C6	ADI	byte	F1	POP	PSW
1B	DCX	D	46	MOV	B,M	71	MOV	M,C	9C	SBB	H	C7	RST	0	F2	JP	adr
1C	INR	E	47	MOV	B,A	72	MOV	M,D	9D	SBB	L	C8	RZ		F3	DI	
1D	DCR	E	48	MOV	C,B	73	MOV	M,E	9E	SBB	M	C9	RET		F4	CP	adr
1E	MVI	E,byte	49	MOV	C,C	74	MOV	M,H	9F	SBB	A	CA	JZ	adr	F5	PUSH	PSW
1F	RAR		4A	MOV	C,D	75	MOV	M,L	A0	ANA	B	CB	---		F6	ORI	byte
20	RIM*		4B	MOV	C,E	76	HLT		A1	ANA	C	CC	CZ	adr	F7	RST	6
21	LXI	H,dbble	4C	MOV	C,H	77	MOV	M,A	A2	ANA	D	CD	CALL	adr	F8	RM	
22	SHLD	adr	4D	MOV	C,L	78	MOV	A,B	A3	ANA	E	CE	ACI	byte	F9	SPHL	
23	INX	H	4E	MOV	C,M	79	MOV	A,C	A4	ANA	H	CF	RST	1	FA	JM	adr
24	INR	H	4F	MOV	C,A	7A	MOV	A,D	A5	ANA	L	D0	RNC		FB	EI	
25	DCR	H	50	MOV	D,B	7B	MOV	A,E	A6	ANA	M	D1	POP	D	FC	CM	adr
26	MVI	H,byte	51	MOV	D,C	7C	MOV	A,H	A7	ANA	A	D2	JNC	adr	FD	---	
27	DAA		52	MOV	D,D	7D	MOV	A,L	A8	XRA	B	D3	OUT	byte	FE	CPI	byte
28	---		53	MOV	D,E	7E	MOV	A,M	A9	XRA	C	D4	CNC	adr	FF	RST	7
29	DAD	H	54	MOV	D,H	7F	MOV	A,A	AA	XRA	D	D5	PUSH	D			
2A	LHLD	adr	55	MOV	D,L	80	ADD	B	AB	XRA	E	D6	SUI	byte			

\*8085 Only.

## REFERENCE

1. A. ALAM B. PRITSKER & ROBERTE. YOUNG, "Simulation with GAP-PL/1". A Wiley.
2. ADAM OSBORNE " An introduction to Micro-Computers, Volume A, 2nd Edition, OSBORNE/McGraw-Hill, Berkely, California.
3. BARBARA J. BURIAN "A simplified approach to S/370 Assembly language programing" printice Hall.
4. JOHN L. HILBURN PAUL M. JULICH" Microcomputers/Micro-processors: hardware, software and applications, Prentice-Hall Series.
5. JOHN McLEOD, P.E. CONSULTANT "SIMULATION- The dynamic modeling of ideas and systems with Computers McGRAW-HILL BOOK COMPANY-Copy right(C) 1968 McGRAW HILL.
6. JON M. SMITH"Mathematical modeling and digital simulation for Engineers & Scientists- A Wiley-Interscience pub.
7. MANESH J. SHAH "Enquiry simulation using Small Scientific Computers, Prentice Hall Series.
8. MANUAL IBM system/370 principles of operation file No. S/370-01.
9. MANUAL IBM DOS/VS Supervisor and I/O Macros Release 34 file No. S370-30.
10. MANUAL IBM DOS/VS system Control Statements.
11. MANUAL DOS/VS Messages Release 34, File No. S37-40.
12. MOHAMED RAFIQUZZAMAN " Microcomputer theory and application with the Inlet SDK-85.
13. NED CHADIN, Ph.D. "370/370 programming in Assembly language", McGraw-Hill Book Company.
14. RUGER L. TOKHEM, M.S. "Schaums outline of Theory and problems of microprocessor fundamentals" McGraw Hill.
15. WALTER G. RUDD "Assembly language preparing and the IBM 360 and 370 Computer Prentice Hall, Inc.