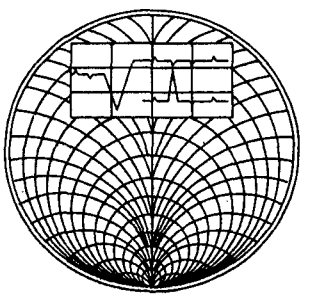
MICROPOWER-i

User Manual

Version - 20

Technical Clarification / Suggestion :
✍ / ☎
Technical Support Division,
Vi Microsystems Pvt. Ltd.,
Chennai - 600 096.
Ph: 044-2496 3142, 044-5201 5899



Vi Microsystems Pvt. Ltd.,
Chennai - 96

PREFACE

The "Mpi (85) User Manual/Workbook" has been devised for INTEL 8085A CPU on Micropower-i. It shall serve as an indispensable adid in the programming aspects based in software, hardware and system oriented functions.

The Manual has been formulated with an eye to the requirement of the beginner and hence has a whole complement of easily comprehendable examples which are supplemented by challenging exercises for you to enhance your ingenuity.

Chapter 1 provides an introduction to the Addresing Modes of8085.

Chapter 2 is a discussion on the Software programming features with a glimpse into all the arithmetic, logical primitives offered by the 8085.

Chapter 3 is a window into the field of interfacing or hardware programming with many of the peripherals of the INTEL family. Here the focus is on the programming aspect of the peripherals.

Chapter 4 is the further step towards transforming you into a seasoned programmer. Herein you find typical application examples which employ more than a single peripheral.

Chapter 5 is the final contribution and this introduces you to the facilities offered by the trainer for system programming.

The Appendices provide the 8085 Instruction Set, ASCII and Hex Conversion Tables.

We are rightly confident that working through this manual in all sincerity you will become well versed wioth 8085. In case of any difficulty, you are requested to consult our Manual "CAT #MPI-85/80-001".

Suggestions are welcome for further improvement on this Manual.

Write to :

The Customer-Support Division,
Vi Microsystems Pvt. Ltd.,
Plot No.75, Electronics Estate,
Perungudi, Chennai - 600 096.
Phone: (044) 496 1842, 496 1852.
Fax: (044) 496 1536.
E-mail: vimicro@vsnl.com

CONTENTS

CHAPTER

CHAPTER 1

8085 - A PROGRAMMING OVERVIEW

The 8085 incorporates a powerful array of instruction. The figure 1.1 shows the internal register set of 8085A. This section provides a general overview of the instruction set.

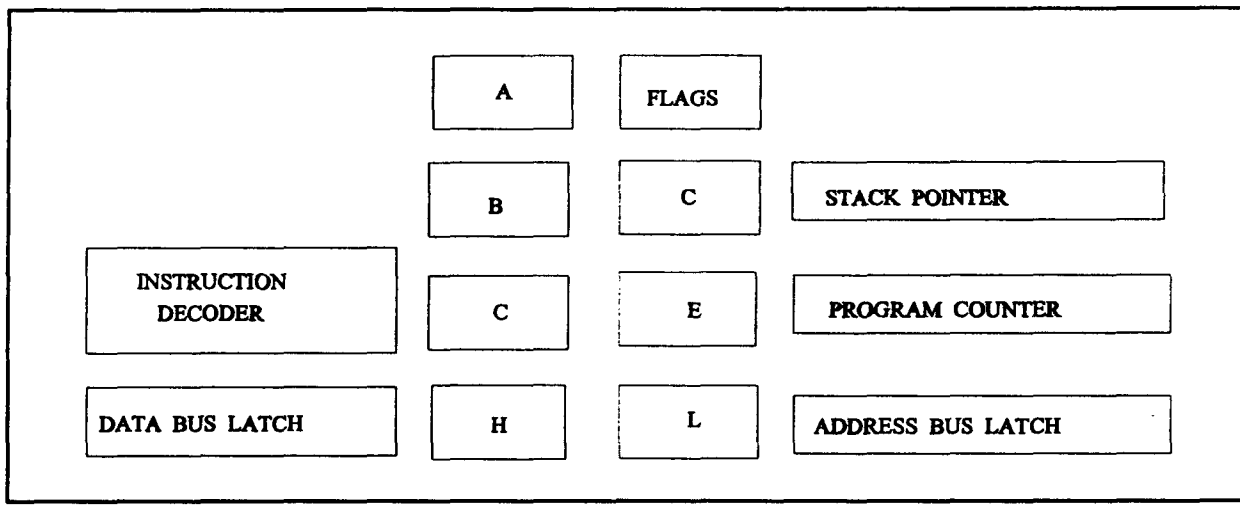


Figure 1.1

1.1 ADDRESSING MODES

Instruction can be categorized according to their method of addressing the hardware registers and/or memory.

IMPLIED ADDRESSING

The addressing mode of certain instruction is implied by the instruction's function. For example, the STC (Set Carry Flag) instruction deals only with the carry flag; the DAA (Decimal Adjust Accumulator) instruction deals with the accumulator.

REGISTER ADDRESSING

Quite a large set of instructions call for register addressing. With these instructions, you must specify one of the registers. A through E, H or L as well as accumulator is implied as a second operand. For example, the instruction CMP E may be interpreted as compare the contents of the E register with the contents of the accumulator.

Most of the instructions that use register addressing deal with 8-bit values . However, a few of there instructions deal with 16-bit register pairs. For example, the PCHL instruction exchanges the contents of the program counter with the contents of the H and L registers.

IMMEDIATE ADDRESSING

Instructions that use immediate addressing have data assembled as a part of the instruction itself. For example, the instruction CPI 'C' may be interpreted as "compare the contents of the accumulator with the letter C". when this instruction is executed, the processor fetches the first instruction byte and determines that it must fetch one more byte. the processor fetches the next byte into one of its internal registers and then performs the compare operation.

Notice that the names of the immediate instructions indicate that they use immediate data. thus, the name of an add instruction is ADD; the name of an add immediate instruction is ADI

All but two of the immediate instructions use the accumulator as an implied operand, as in the CPI instruction shown previously. The MVI (move immediate) instruction can move its immediate data to any of the working registers, including the accumulator, or to memory, Thus, the instruction MVI D, FF moves the hexadecimal value FF to the D register.

The LXI instruction (Load register pair immediate) is even more unusual in that its immediate data is a 16-bit value. This instruction is commonly used to load addresses into a register pair. As mentioned previously, your program must initialize the stack pointer; LXI is the instruction most commonly used for this purpose. For example, the instruction LXI SP, 30FF loads the stack pointer with the hexadecimal value 30FF.

DIRECT ADDRESSING

Jump instructions include a 16-bit address as part of the instruction. For example, the instruction JMP 1000 causes a jump to the hexadecimal address 1000 by replacing the current contents of the program counter with the new value 1000.

Instructions that include a direct address require three bytes of storage: one for the instruction code and two for the 16-bit address.

REGISTER INDIRECT ADDRESSING

Register indirect instructions reference memory via a register pair. Thus, the instruction MOV M,C moves the contents of C register into the memory address stored in the H and L register pair. The instruction LDAXB loads the accumulator with the byte of data specified by the address in the B and C register pair.

COMBINED ADDRESSING MODES

Some instructions use a combination of addressing modes. A CALL instruction, for example, combines direct addressing and register indirect addressing. The direct address in a CALL instruction specifies the address of the desired subroutines; the register indirect address is the stack pointer. The CALL instruction pushes the current contents of the program counter into the memory location specified by the stack pointer.

CHAPTER 2

SOFTWARE EXAMPLES

2.1 INTRODUCTION

In this Chapter on software experiments, you will encounter experiments that help you to enhance your ability as a software programmer. Only the basic instructions of the 8085 are used in this chapter, which will be easy for you to follow. However, if possible, they can be replaced by some other equivalent instruction.

When writing a program on your own, try to make it SMALL AND EFFICIENT:

SMALL in the sense, it should occupy less memory space ie., the opcodes must be limited; EFFICIENT in the sense, the task desired to do by the program must be done in the most effective way.

Try to execute all the programs you write using the trainer. There is no other way to make sure that your program is correct.

This Chapter consists of an assorted pack of 19 Experimental sections, each inturn containing an EXAMPLE PROGRAM to understand and distinct EXERCISES to practise.

You are requested to:

i. Key in the EXAMPLE PROGRAM and check for mentioned results.

ii. Change data in the above and check for REQUIRED results.

iii Alight into the EXERCISES to test your understanding and work on them to strengthen your comprehension.

This will help you through the next chapter which is on Hardware programming experiments, on the peripherals available on the kit Micropower-i with 8085.

NOTE

All addresses and data are in hex form unless otherwise specified. The notation (4100) means the contents of the memory location 4100.

2.2 USING LINE ASSEMBLER

In the following examples, all the programs are written in 8085 mnemonics which is understandable by you, decoded to its own binary equivalent which the microprocessor can understand and execute. These binary equivalents for mnemonics are called object codes or simply opcodes. The opcodes can be found out by referring to the instruction set of 8085A given in the Appendix of this manual.

In Micropower-I trainer, you have two options for entering the programs.

i. You can either key in the opcodes directly in to the RAM using substiute command or.

ii. You can make use of the built in line assembler. The usage of line assembler is detailed below with a sample program.

Sample Program

```
MV1     A, 45
MVI     B, 54
ADD     B
STA     5100
HLT
```

Folowing is the procedure to enter the mnemonics in Micropower-I.

In the command prompt, type a followed by enter key, to invoke hte line assembler

```
Micropower - i
#A<CR>
```

<CR> Stands for Enter Key in the IBC PC keyboard

It will Prompt you as,

```
4200 ■
```

You specify the orgin, from where your program should start. You can use RAM address starting from 5000H.

```
4200 ORG 5000 <CR>
```

The reasons will be,

```
5000 ■
```

Now you start entering the mnemonics. The delimiter between mnemonics and operand is space. You have to type in only the mnemonics and operand field. You need not type the address, as it is generated by the line assembler for you.

```
5000        MV1  A, 45      <CR>
5002        MV1  B, 54      <CR>
5004        ADD  B          <CR>
5005        STA  @5100      <CR>
5008        HLT             <CR>
5009        END             <CR>
5009        .               <CR>
```

```
Micropower-i
  # ■
```

If you make any typing mistake while entering the program, use backspace keyand edit that line. The editor is a line editor. So you can edit only one line at a time. After entering the last instruction of our program, type '.' (dot) followed by carriage return to come to command prompt.

NOTE

'.' (dot) followed by enter key is the command terminator.

Now the program has been assembled. You can check it by using the SU command (substitute memory).

Type SU 5000 <CR>,

You will notice the following data, the corresponding opcodes of the mnemonics entered by you.

```
Micropower-i
#SU 5000<CR>
```

Edit Memory

```
5000  3E   -   □<CR>
5001  45   -   □<CR>
5002  06   -   □
5003  54   -   □
5004  80   -   □
5005  32   -   □
5006  00   -   □
5007  51   -   □
5008  76   -   □
5009  XX   -   □    .<CR>
```

```
Micropower-i
#
```

You can view your program with the help of unassembler.

Type "U" followed by enter.

```
Micropower-i
#U<CR>
```

IT will prompt you for the starting address as,

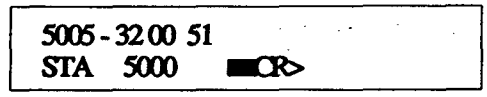
```
ORG? 
DISASSEMBLER
```

Give the orgin as 5000

```
ORG? 5000 <CR>
DISASSEMBLER
```

It will display both the opcodes and mnemonics.

```
5000 - 3E 45
MVI A, 45 CR>
```



2.3 EXPERIMENT 1 - ONE'S COMPLEMENT

OBJECTIVE: To find the One's Complement of the data in memory location 4150 and store the result at 4151.

THEORY : In the One's complement of a binary number the ones are changed to zeros and the zeros to ones. Usually this one's complement is used to represent negative numbers. In this representation, the positive numbers are just binary numbers and hence they start with a "0" with a " 1" on the left. For example, the one's complement for 56 = 01010110 is A9 = 10101001

EXAMPLE : The example given is to find the One's complement of 8F and store it in memory location 4151. The number 8F is at memory location 4150.

Data : (4150) = 1000 1111 (binary) = 8F
Result : (4151) = 0111 0000 (binary) =70

PROGRAM :

```
MVI A, 8F     ; Get data
CMA           ; Compliment data
STA 4150      ; Store One's complemented result
HLT
```

OBJECT CODES: Now this program written in 8085 mnemonics is understandable by you, but not by the microprocessor. Each mnemonic must be decoded to its own binary equivalent which the microprocessor can understand and execute. Those binary equivalents for mnemonics are called object codes or simply opcodes. The opcodes can be found out by referring to the instruction set of 8085 given at the back of this manual.

The object codes for the above program is

Memory Address	object codes	Mnemonics
4100	3E	MVI A, 8F
4101	8F	
4102	2F	CMA
4103	32	STA 4150
4104	50	
4105	41	
4106	76	HLT

PROCEDURE

1. Enter the above object codes into RAM memory from 4100 using the SUB Command.

2. Enter data to be complimented at 4150 which is 8F using the SUB command.

3. Press Go key and enter 4100 . It is the address from where execution will start. Now press EXEC key.

4. Now reset the kit using RESET key.

5. Check at location 4151, whether the program has been run correctly, using the SUB command.

6. Try changing data at 4150 and execute the program each time, and record the data and results.

DISCUSSION: In Microprocessors, subtraction is done using two's complement method. This two's complement method. This two's complement is got by adding a logical one to the one's complement number. Try to write a program for finding the two's complement of a number. Usually the CPL is used to transfer data to or from a peripheral device that uses negative logic (e.g. a display which is turned on by a zero, off by a one).

EXERCISE: (i) Find the two's complement of a number at memory location 4150 and store the result at 4151.

(Hint: Two's complement is one's complement plus one)

sample problem:

Data : (4150) = 3E

Result : (4151) =C2

(ii) Subtract two numbers using Two's complement and store the result at 4152.

(Hint : Find two's complement of one number and add up with the other)

Sample problem:

Data : (4150) =64
(4151) =32
Result : (4152) =32

2.4 EXPERIMENT 2 - MASK OFF MOST SIGNIFICANT FOUR BITS

OBJECTIVE: To place the least significant four bits (lower nibble) of the contents of memory location 4150 in memory location 4151. Clear the most significant four bits (higher nibble) of the number.

THEORY: To mask off the most significant four bits of 4150 (i.e. to clear the higher nibble), logically AND the contents of memory location 4150 with 0000 1111 binary. The ANDing will clear the higher nibble which is called Masking off the most significant four bits. So, to clear any bit, AND with it a number with ones but zero at that particular bit. This will clear that bit alone.

EXAMPLE: The contents of 4150 is B8. The result will be stored at 4151.

Data : (4150) = B8

Result: (4151) = 08

PROGRAM:

```
LDA   4150          ;   Get data.
ANI   0000  1111 B  ;   Mask of higher nibble.
STA   4151          ;   Store result.
HLT
```

OBJECT CODES

Memory Address	Object codes	Mnemonics
4100	3A	LDA 4150
4101	50	
4102	41	
4103	E6	ANI 0F
4104	0F	
4105	32	STA 4151
4106	51	
4107	41	
4108	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: The AND instruction is generally used to clear off bits that are not in use. Try to mask off the least significant four bits.

EXERCISE: (i) Mask off the least significant four bits of memory location 4150 and place the result at 4151.

Sample problem :

Data : (4150) = C4

Result : (4150) = C0

ii. Mask off the 7th and 3rd bits of register which contains b9. Store the result at 4150 (use AND instruction).

Sample Problem:

Data : (Register H) = 139

Result : (4150) = 88

2.5 EXPERIMENT 3 - SET INDIVIDUAL BITS

OBJECTIVE : To set the 0th and 5th bits of Accumulator A and store the result at memory location 4150.

THEORY: Bits can be set individually by ORing with data that has a one in that particular bit. For example to set the 3rd bit of a register, OR that register with 0000 1000b. This will set the third bit of that register.

EXAMPLE: Accumulator A contains 00 initially. Set bits 0 and 5 of Accumulator A and store result at 4150.

Data : A = 00.
Result : (4150) =21 (Bits 0 and 5 are set).

PROGRAM

```
MVI     A,00        ;   Initial value of A = 0.
ORI     0010 0001 B ;   Set bits 0 and 5.
STA     4150        ;   Store result..
HLT
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	3 E	MVI A, 00
4101	00	
4102	F6	ORI 21
4103	21	
4104	32	STA 4150
4105	50	
4106	41	
4107	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: The OR and the AND instructions are used generally to set and to mask bits respectively. Try changing data for the above program and record results.

EXERCISES: i. Set bits 7 and 1 of Register C using the OR instruction where C = 94 and store result at 4150.

Sample Problem :

Data : C =94

Result : (4150) =96.

ii) The contents of location 4160 = 00. Set the bits 4 and 5 of 4160 using OR instruction

Set the 1st and 0^{th} bits of contents of 4160 using OR instruction and store result at 4161.

Sample Problem :

Data : (4160) = 44
Result : (4141) = 47

2.6 EXPERIMENT 4 - PACKED TO UNPACKED:

OBJECTIVE: Divide the contents of 4150 into two 4-bit sections and store the higher nibble at 4151 and lower nibble at 4152.

THEORY: First get the data and rotate left four times, through carry, mask off its higher nibble and store it. Again get the same data, mask off its higher nibble and store it.

EXAMPLE: The contents of 4150 is 3E. The result stored at 4151 and 4152 are 03 and 0E.

Data : (4150) =3E

Result: (4151) =03
(4152) =0E

PROGRAM

```
LXI   H , 4150      ;     point data
MOV   A , M         ;     Get data
MOV   B,A
RRC                 ;     shift right four times
RRC
RRC
RRC
ANI   0000 1111 B   ;     Mask off higher nibble
INX   H
MOV   M,A           ;     Store higher nibble alone
MOV   A,B           ;     Get data again
ANI   0000 1111 B   ;     Mask off higher nibble
INX   H
MOV   M,A           ;     Store lower nibble
HLT
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H ,4150
4101	50	
4102	41	
4103	7E	MOV A, M
4104	47	MOV B,A
4105	0F	RRC
4106	0F	RRC
4107	0F	RRC
4108	0F	RRC
4109	E6	ANI 0F
410A	0F	
410B	23	INX H
410C	77	MOV M,A
410D	78	MOV A,B

410E	E6	ANI 0F
410F	0F	
4110	23	INX H
4111	77	MOV M, A
4112	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: Instructions using register M occupy only one word of program memory. But H&L must be initialised before using M. RRC shifts Accumulator right one bit circularly with the least significant bit going to the most significant bit and carry. Shifting Accumulator right four times require four RRC's.

EXERCISE: i. Write a program for unpacked to packed. Unpack the contents of memory location 4150 and 4151 and store the packed result at 4152.

Sample problem:

Data : (4150) = 06
(4151) =0A
Result : (4152)=6A

2.7 EXPERIMENT 5 - LOGICAL OPERATIONS

OBJECTIVE: To learn about logical instructions available with 8085 . The available logical operations are AND, OR, XOR, NOT.

THEORY: Logical operations, as you all know, form the major part of the instruction of any microprocessor. They are very useful for a varied number o f applications. As seen, from the previous examples, the AND, NOT (Complement) and OR are already explained. The XOR is given here as an example. The truth tables for the four logical operations are as follows:

AND			OR			XOR			NOT	
X	Y	Z	X	Y	Z	X	Y	Z	X	Z
0	0	0	0	0	0	0	0	0	0	1
0	1	0	1	0	1	0	1	1	1	0
1	0	0	0	1	1	1	0	1		
1	1	1	1	1	1	1	1	0		

Where X, Y are two inputs, Z is the output.

EXAMPLE: The two numbers to be XORed are at memory locations 4150 and 4151 and the XORed result must be stored at 4152.

Data : (4150) = 43
(4151) = 64

Result : 0100 0011
0110 0100 (XOR)

0010 0111

(4152) = 27.

PROGRAM:

```
LDA  4150  ;  Get first data.
MOV  B, A
LDA  4151  ;  Get second data.
XRA  B     ;  XOR two data.
STA  4152  ;  store result.
HLT
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	3A	LDA 4150
4101	50	
4102	41	
4103	47	MOV B,A
4104	3A	LDA 4151
4105	51	
4106	41	
4107	A8	XRAB
4108	32	STA 4152
4109	52	
410A	41	
410B	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: Try to write programs to logically AND, OR, NOT numbers and store them as explained here for XOR.

EXERCISES: i) OR two numbers 26 and 5A and store result at 4150.

ii) AND two numbers at 4150 and4151 and store result at 4152.

Sample problem:

Data : (4150) = 65
(4151) = 28
Result : 0110 0101
0010 1000

0010 0000
(4152) = 20

iii) NOT the contents of Register B which is 5F and store result at 4150.

iv) Find the two's complement of the contents of 4150 and store result at 4151.

Sample problem:

Data : (4150) = 55
Result : (4151) = AB

2.8 EXERCISE 6 - ADDITION OF TWO HEX NUMBERS

OBJECTIVE: To add two hexadecimal numbers residing in memory and store the result in memory. The two numbers are actually two 8-bit binary numbers represented in hex form.

THEORY: In this experiment, the two numbers to be added reside in memory. So, first they should be brought to Accumulator A and to any one of the other registers, say B. For this the absolute memory addressing of 8085 is used.

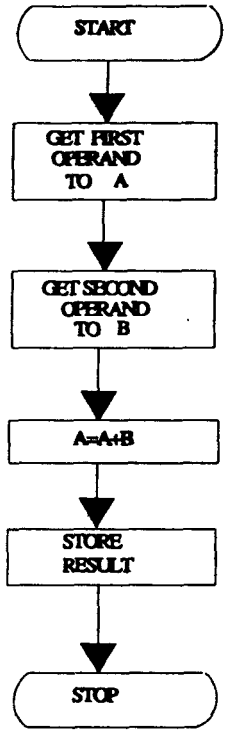
After moving the data to be added to accumulator A and register B, add those two using the AD instruction. After adding, the result is in accumulator A which is again stored into memory using absolute addressing mode.

EXAMPLE: Let us assume that the two data are at memory locations 4150 and 4151 and the result is going to be stored in 4152. so the example treats the locations to have the contents as

(4150) = 23; (4151) = 35.

These two data must be entered separately using the SUB command by selecting the address 4150. After executing the program, the contents of 4152 will be 23+35 =58, which is stored at location 4152, which can be verified again using SUB command.

FLOWCHART FOR ADDITION



PROGRAM:

```
LDA  4150    ;   Load Accumulator A with contents of 4150.
MOV  B,A     ;   Get first operand to B register.
LDA  4151    ;   Get second operand to Accumulator A  Now A
                 =35 and  B =23.
ADD  B       ;   Accumulator A = Accumulator A +  Register
                 B.
STA  4152    ;   Store the result  58 to location 4152 through
                  absolute addressing  mode.
HLT          ;   Stop execution.
```

OBJECT CODES:

Memory Address	Object Codes	Mnemonics
4100	3A	LDA 4150
4101	50	
4102	41	
4103	47	MOV B,A
4104	3A	LDA 4151
4105	51	
4106	41	
4107	80	ADD B
4108	32	STA 4152
4109	52	
410A	41	
410B	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: Now that you know the basic instructions like LDA , ADD , STA try to make use of them and write simple programs like Add with carry, Add using immediate data and so on.

EXERCISES: i. Add the data 4A and 23 using Immediate addressing mode and store result at 4150.

ii) Do decimal addition of contents of 4150 & 4151 and store result at 4152 & 4153. [Hint : Use DAA]

sample problem:

Data : (4150) = 99
(4151) = 99

Result : 99+99 = 198
(4152) = 98
(4153) = 01

2.9 EXPERIMENT 7 - SUBSTRACTION OF TWO HEX NUMBERS

OBJECTIVE: To subtract two hex numbers residing in memory and to store the result also in memory. The access to memory is using HL register pair.

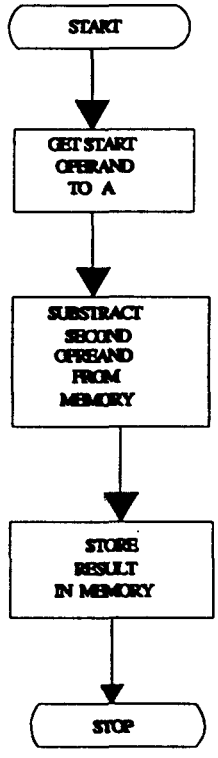
THEORY: In this experiment, the HL register pair is first initialised to the start address of memory at which the data is stored. Then data is brought to accumulator A and the other one is subtracted from memory itself. The result from A is then stored into memory again using the HL register.

EXAMPLE: Let the data to be subtracted be 24 from 49. Let these data be entered into memory at 4150 and 4151. The result of this subtraction, which will be 49-24=25 be stored at 4152. So, the data to be entered before execution are,

Data : (4150) =49
(4151)=24

Result : (4152)=25

FLOW CHART FOR SUBTRACTION



PROGRAM

```
LXI   H,  4150    ;  Initiatialise HL to start address of data.
MOV   A,  M       ;  Get first operand to Accumulator A. A=49.
INX   H           ;  Point to next data.
SUB   M           ;  Subtract M from Accumulator A which is 49-24.
INX   H           ;  point to next location.
MOV   M,A         ;  Store result at 4152.
HLT               ;  Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H, 4150
4101	50	
4102	41	
4103	7E	MOV A,M
4104	23	INX H
4105	96	SUB M
4106	23	INX H
4107	77	MOV M,A
4108	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: The above program for subtraction can be further reduced by using immediate instructions for load and subtract.

EXERCISES:

i. Subtract 38 from 69 using immediate mode of addressing for load, subtract; store result at memory location 4150.

ii. Subtract two decimal numbers at 4150 and 4151 and store result at 4152.

Sample problems:

a) Data : (4150) = 29 (4152) = 17
Result: (4151) = 12

b) Data : (4150) =85 (4152) = 26
Result : (4151)=59

2.10 EXPERIMENT 8 8 - BIT BY 8-BIT MULTIPLICATION

OBJECTIVE: To multiply two numbers residing in memory and to store the result in memory.

THEORY: As far as the two previous experiments are concerned , they are very simple in the sense that 8085 offers instructions to add and subtract two numbers. Since there is no such instruction to multiply or to divide. some logic has to be applied to do those tasks. Multiplication can be done by repeated addition whereas division by repeated substraction.

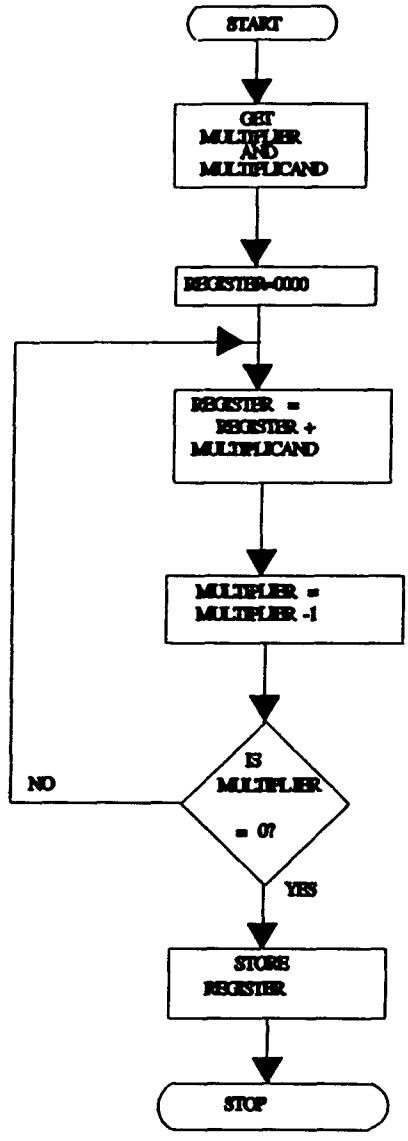
In this experiment, the multiplicand and the multiplier stored at memory locations are taken up by register pair DE and reg B. Using the DAD instruction, 16- b it addition is done repeatedly by the multiplicant and until the multiplier becomes zero. The result is again stored in a memory location.

EXAMPLE: Let the multiplicand and the multiplier be at locations 4150 and 4152. The result will be stored at location 4154. The data at locations are:

Data : (4150) =93 (4152) =23
(4151) =00 (4153) =00

Result (4154)=19
(4155)=14
93x23=1419.

FLOW CHART FOR MULTIPLICATION



PROGRAM:

```
        LDA   4152   ;     Load Multiplier
        MOV   B,A    ;     Get the multiplier to B
        LXI   D,0000       ;
        LHILD 4150   ;
        XCHG               ;     Load multiplicand in DE
LOOP:   DAD   D      ;
        DCR   B
        JNZ   LOOP   ;     If not loop again.
        SHIL  4154   ;     Else store result.
        HLT
```

OBJECT CODES

Memory Address	Object codes	Mnemonics
4100	3A	LDA 4152
4101	52	
4102	41	
4103	47	MOV B,A
4104	11	LXI D,0000
4105	00	
4106	00	
4107	2A	LHLD 4150
4108	50	
4109	41	
410A	EB	XCHG
410B	19	LOOP:DAD D
410C	05	DCR B
410D	C2	JNZ LOOP
410	0B	
410F	41	
4110	22	SHLD 4154
4111	54	
4112	41	
4113	76	HLT

PROCEDURE: Follow a similar procedure as stated i n the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION: This method is the generally used one, when no instruction for multiplication is provided. Try writing programs to multiply two numbers using the RLC instruction. Even though limitations are present for multiplication by rotation, it could prove upto some point.

EXERCISES: Multiply the 16-bit unsigned numbers at 4150 and 4151 by the 8-bit unsigned number in memory location 4152. Store result in memory locations from 4153 onwards.

Sample problems:

a) Data : (4150) = 03 (LSB) (4152) = 05
(4151) = 00 (MSB)

Result : (4153) = 0F (LSB)
(4154) = 00
(4155) = 00 (MSB)

3 × 5 = 15 (0F hex)

b) Data : (4150) = 6F (LSB) 4152 =61
(4151) = 72 (MSB)

Result : (4153) = 0F (LSB)
(4154) = 5C
(4155) = 2B (MSB)
726F×61 = 2B5C0F

ii. Multiply two numbers 26 and 08 using the instruction "Rotate Left" and store result at 4150.

[Hint : Rotating left once is equivalent to multiplying by 2]

2.11 EXPERIMENT 9 - 8 BIT BY 8-BIT DIVISION

OBJECTIVE: To divide two numbers in memory and to store the result in memory again.

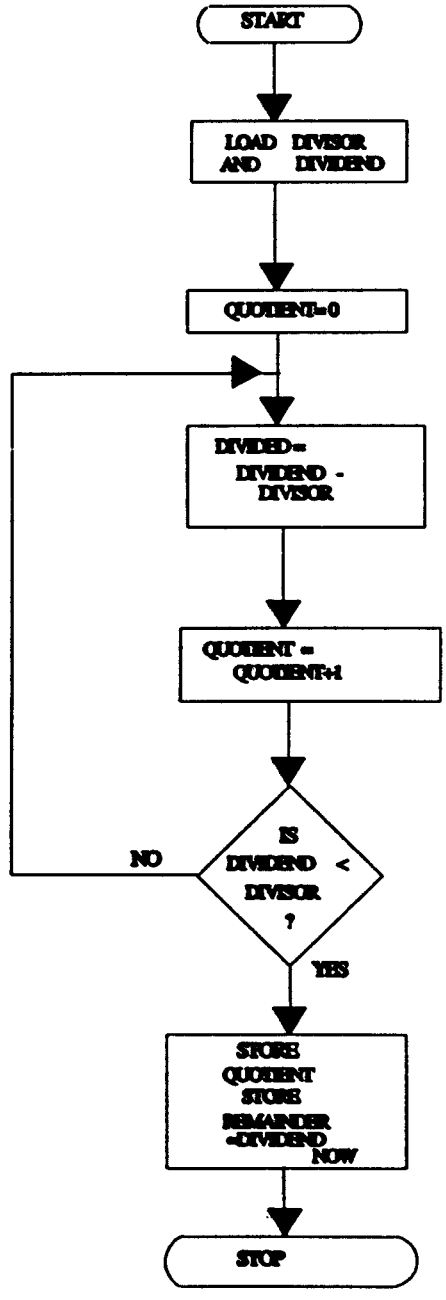
THEORY: Division is done here using the method of repeated subtraction. The dividend is loaded to register A and divisor to register B. Subtract divisor from dividend until the dividend is less than the divisor. 'count number of times of subtraction. That count is the quotient and the dividend now becomes the remainder.

EXAMPLE: The divisor and the dividend are at locations 4150 & 4151. The quotient & remainder will be stored at locations 4152 &4153.

Data : (4150) = 08 - Divisor
(41451)=42 - Dividend.

Result : (4152) =02 - Remainder
(4151) =08 - Quotient.

FLOW CHART FOR DIVISION



PROGRAM :

```
        LDA   4150      ;     Load divisor.
        MOV   B,A       ;     to B.
        LDA   4151      ;     Load dividend to A.
        MVI   C,00      ;     Initialise quotient = 0.
LOOP :  SUB   B         ;     Dividend    -Divisor.
        INR   C         ;     Quotient = Quotient +1.
        CMP   B         ;     Is divided < Divisor.
        JNC   LOOP      ;     If not repeat again.
        STA   4152      ;     yes, then store remainder.
        MOV   A,C
        STA   4153      ;     Store Quotient.
        HLT             ;     Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	3A	LDA 4150
4101	50	
4102	41	
4103	47	MOV B,A
4104	3A	LDA 4151
4105	51	
4106	41	
4107	0E	MVI C,00
4108	00	
4109	90	LOOP: SUB B
410A	0C	INR C
410B	B8	CMP B
410C	D2	JNC LOOP

Memory Address	Object Codes	Mnemonics
410D	09	
410E	41	
410F	32	STA 4152
4110	52	
4111	41	
4112	79	MOV A,C
4113	32	STA 4153
4114	53	
4115	41	
4116	76	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

EXERCISES: i Divide the 16-bit unsigned number at memory location 4150 and 4151 by the 8-bit unsigned number at memory location 4152 and store quotient and remainder starting from 4153. [Hint: Rotating right once is equivalent to dividing once by 2] .

Sample problem:

Data : (4150) = 63 (4152) =21
(4151) = 52
Result : (4153) = 7F Quotient
(4154) = 02
(4155) = 04 Remainder

ii. Divide 14 by 04 using Rotate Right instruction and store result at 4150.

2.12 EXPERIMENT 10 - MULTIPRECISION ADDITION (24-BIT)

OBJECTIVE : To add two numbers that are more than 8-bit in length, the length being given as the number of 8-bits in that number.

THEORY: The multiprecision addition program adds two numbers having length greater than 8-bits, the length given at the start of the program calculated by the number of 8-bits of data in that number. The addition is done actually in 8-bits alone. But it is added with the next set of 8-bit numbers with carry. This type of addition is done till the count is zero which gives the result of the addition of two numbers of length greater than 8- bits.

EXAMPLE: The length of the two numbers being the same, the length is loaded at location 4150. The data is followed then from 4151. The second number starts from 4160.

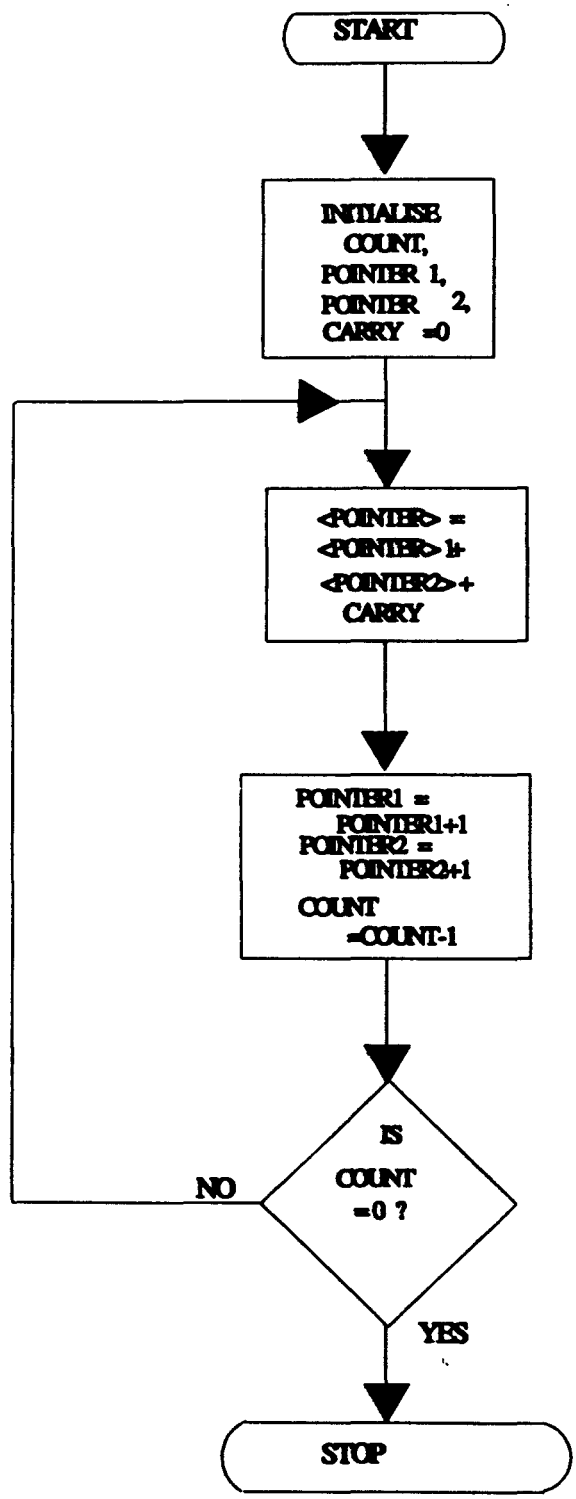
Data :

(4150) = 03 -Count
(4151) = 29 - First number
(4152) = A4
(4153) = 50
(4160) = FB - Second number
(4161) = 37
(4162) = 28

```
50 A4 29
28 37 FB (+)
------------
78 DC 24
------------
```

Result : (4151) =24
(4152)=DC
(4153)=78

FLOW CHART FOR MULTI PRECESSION ADDITION



PROGRAM :

```
        XRA  A        ;  Accumulator =  0 ;  CF = 0
        LXI  H ,4150  ;  Initialise start address of data
        MOV  B, M     ;  Get count to B; B = 03
        LXI  D,4160   ;  Start address of second data
        INX  H        ;  Point to first data
LOOP:   LDAX D        ;  Get 8-bits of second number
        ADC  M        ;  Add with corresponding 8-bits of first number with
                          carry
        MOV  M,A      ;  Store corresponding result
        INX  H        ;  Point to next subsequent
        INX  D        ;  data
        DCR  B        ;  Check length
        JNZ  LOOP     ;  If not equal to zero, repeat.
        HLT           ;  Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	AF	XRA A
4101	21	LXI H, 4150
4102	50	
4103	41	
4104	46	MOV B,M
4105	11	LXI D,4160
4106	60	
4107	41	
4108	23	INX H
4109	1A	LOOP: DAX D
410A	8E	ADC M
410B	77	MOV M,A
410C	23	INX H
410D	13	INX D
410E	05	DCR B

410F	C2	JNZ LOOP
4110	09	
4111	41	
4112	7 6	HLT

PROCEDURE: Follow a similar procedure as stated in the first experiment to enter and execute the program. Try the program for different data and check results.

DISCUSSION : This program for multi-precision addition is a very useful way to learn more on addition programs. Try multi-precision subtraction, multi-precision addition with 32, 48 bits and so on.

EXERCISE: i. Subtract two multiple - word numbers. The length of the numbers is at memory location 4150, the numbers themselves start from 4151 and 4160 respectively and the result replaces the number starting in memory location 4151. Subtract the number string in 4160 from the one starting at 4151.

Sample problem:

Data : (4150) = 04 - Count
(4151) = C3 - First number.
(4152) = A7
(4153) = 5B
(4154) = 2F
(4160) = B8 - Second number.
(4161) = 35
(4162) = DF
(4163) = 14

Result : (4151) = 0B
(4152) = 72
(4153) = 7C
(4154) = 1A

2F 5B A7 C3
14 DF 35 B8 (-)

1A 7C 72 0B
---

ii. Add the four numbers CA, 43, 56 and 19 and stire the result at memory location 4150 and 4151.

2.13 EXPERIMENT 11 - BIGGEST NUMBER IN AN ARRAY

OBJECTIVE: To find the biggest number in an array of 8-bit unsigned numbers, the length of the block being given and residing in memory along with the array.

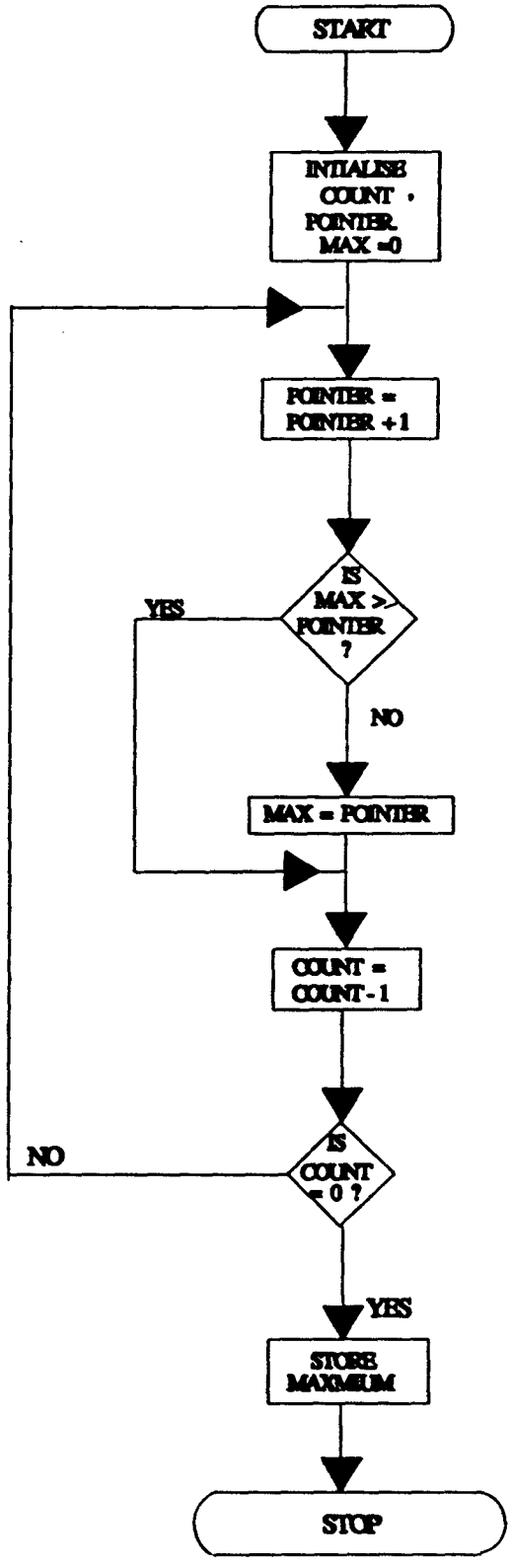
THEORY: To find the Biggest number in an given array, the contents of the array must be compared with an arbitrary Biggest number. In this experiment, since all numbers are said to be unsigned 8-bit numbers, let register A have the biggest number i.e., Zero. Now compare the first number with register A. Further comparison is with this biggest number and this comparison is done till the end of the array. Now the biggest number in Register A is put in memory.

EXAMPLE: The length of the array is at memory location 4150 while the array itself starts from 4151. Let the number of elements in the array be 05.

Data : (4150) = 05 - No.of elements.
(4151) = 67 -Array starts.
(4152) = 79
(4153) = 15
(4154) = E3
(4155) = 72

Result: (4156) = E3.

FLOW CHART FOR BIGGEST NUMBER IN ARRAY



```
PROGRAM:    LXI   H, 4150   ;   point to data start.
            MOV   B,M       ;   Get count to B.
            XRA   A         ;   Maximum = Minimum
                                possible value (Zero).
LOOP:       INX   H
            CMP   M         ;   Is next number > Maximum?
            JNC   LOOP1     ;
            MOV   A,M       ;   Yes, replace maximum.
LOOP1:      DCR   B         ;   Check length of array.
            JNZ   LOOP      ;   If B ≠0 then repeat.
            INX   H         ;
            MOV   M,A       ;   Save naximum number
            HLT             ;   Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H, 4150
4101	50	
4102	41	
4103	46	MOV B,M
4104	AF	XRA A
4105	23	LOOP: INX H
4106	BE	CMP M
4107	D2	JNZ LOOP 1
4108	0B	
4109	41	
410A	7E	MOV A,M
410B	05	LOOP:DCR B
410C	C2	JNZ LOOP
410D	05	
410E	41	
410F	23	INX H

4110	77	MOV M,A
4111	76	HLT

PROCEDURE: The procedure is similar as explained before to enter the program and execute it. check results again using SUB Command.

DISCUSSION: This program can be modified to find the minimum number of the array with some minor changes. Try changing datawith both programs with different array lengths also and record results.

EXERCISES

i. Find the smallest of two numbers 39 and 65 and store result at 4150.

ii. Find the smallest number in a array of a four elements residing from 4150 and store result at 4160.

Sample problem:

Data : (4150) = 04 - count
(4151) = 56
(4152) = 73
(4153) = 13
(4153) = E5

Result : (4160) = 13

iii. Find the biggest and smallest number of an array in a single program itself ; the length of the array is at memory location 4150 and the array itself starts at 4151 and store 4151 and store the maximum number at 4160 and minimum number at 4161.

Sample problem:

Data : (4150) = 05 - Length of array.
(4151) = 85 -Array starts.
(4152) = 19
(4153) = 70
(4154) = 36
(4155) = 59

Result : (4160) = 85 - Maximum number.
(4161) = 19 - Minimum number.

2.14 EXPERIMENT 12- ARRANGE IN DESCENDING ORDER

OBJECTIVE: To arrange an array of 8-bit unsigned numbers in descending order.

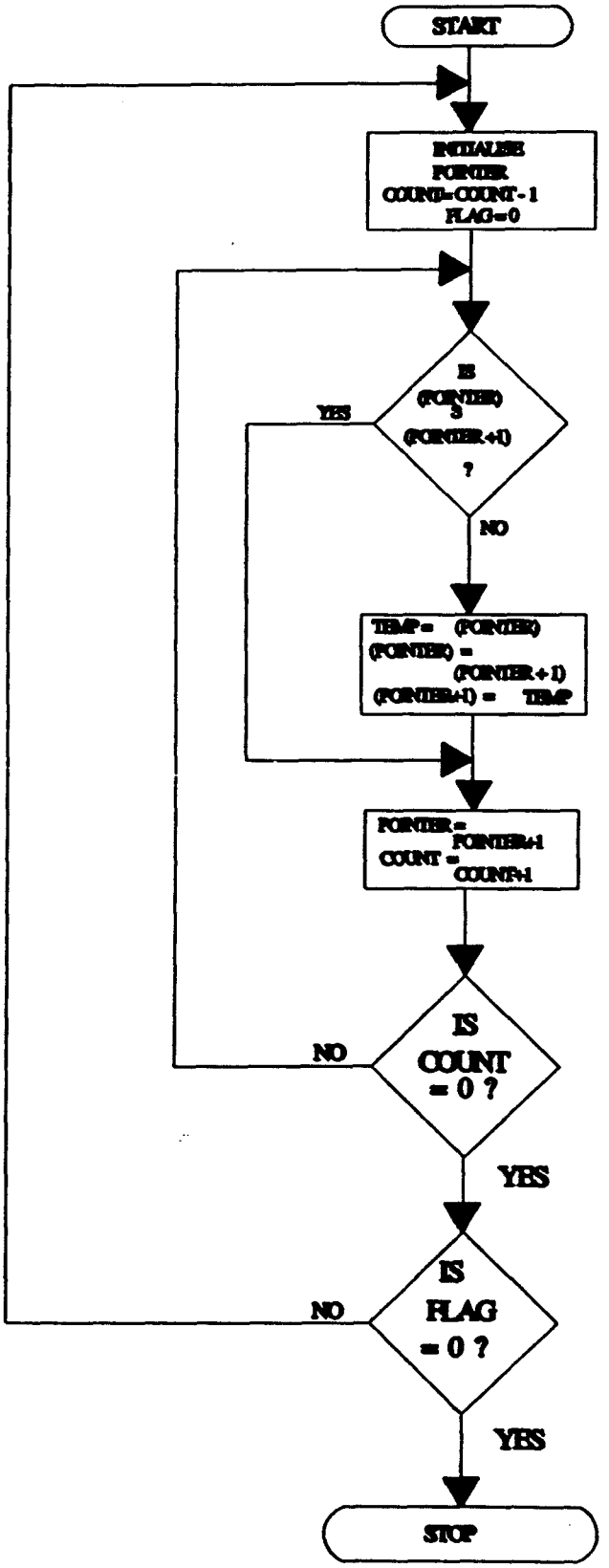
THEORY: The sorting technique used here is relatively simple. First clear a flag named, say XX. Secondly examine successively each pair of numbers in the array. If any pair are out of order, interchange them and set flag XX.
Thirdly, if XX ≠0 repeat from the first step. Therefore XX will be set to 1 if any pair of numbers is out of order. If XX =0 the array is in proper order.

EXAMPLE: The length of the array is at memory location 4150 which is assumed to be 06. The array starts from 4151.

Data :
(4150) = 06 - No.of elements.
(4151) = 2A - Array starts.
(4152) = B5
(4153) = 60
(4154) = 3F
(4155) = D1
(4156) = 19

Result :
(4151) = D1
(4152) = B5
(4153) = 60
(4154) = 3F
(4155) = 2A
(4156) = 19

FLOW CHART FOR DESCENDING ORDER



PROGRAM

```
START :  MVI  B, 00H    ;   Flag  = 0
         LXI  H,4150    ;   Count = Length of array
         MOV  C,M       ;
         DCR  C         ;   Number of pairs = Count -1
         INX  H         ;   Point to start of array
LOOP :   MOV  A,M       ;   Get kth element
         INX  H         ;
         CMP  M         ;   Compare to (K+1) th element
         JNC  LOOP1     ;   No interchange
                            if kth > = (K+1)th
         MOV  D,M       ;   Interchange if out of order
         MOV  M,A       ;
         DCX  H         ;
         MOV  M,D       ;
         INX  H         ;
         MVI  B,01      ;   Flag =1
LOOP1:   DCR  C         ;   Count down
         JNZ  LOOP      ;
         DCR  B         ;   IS Flag =1 ?
         JZ   START     ;   Do another sort, if yes
         HLT            ;   If FLAG = 0, stop execution
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	06	START: MVI B, 00H
4101	00	
4102	21	LXI H, 4150
4103	50	
4104	41	
4105	4E	MOV C,M
4106	0D	DCR C
4107	23	INX H
4108	7E	LOOP: MOV A, M
4109	23	INX H

410A	BE	CMP M
410B	D2	JNC LOOP1
410C	15	
410D	41	
410E	56	MOV D,M
410F	77	MOV M , A
4110	2B	DCX H
4111	72	MOV M, D
4112	23	INX H
4113	06	MVI B,01H
4114	01	
4115	0D	LOOP: DCR C
4116	C2	JNZ LOOP
4117	08	
4118	41	
4119	05	DCR B
411A	CA	JZ START
411B	00	
411C	41	
411D	76	HLT

PROCEDURE: The procedure to enter the program and execute the same is similar as explained before. They changing new data and record information.

DISCUSSION: Try to write remarks for the following:

i. What if two numbers are equal? Whether an interchange would occur or not.

ii. If the JNC, LOOP1 statement of this program is changed to JC LOOP1, will the program do the sort in ascending order? change data and execute program each time and record informations concerned.

EXERCISE: i. Arrange the given array of numbers in ascending order; the length of the array is at memory location 4150,while the array itself starts from 4151.

Sample problem:

Data : (4150) = 07 - No . of elements.
(4151) = 06 - Array starts.
(4152) = 68
(4153) = F2
(4154) = 87
(4155) = 30
(4156) = 59
(4157) = 2A

Result : (4151) = 06
(4152) = 2A
(4153) = 30
(4154) = 59
(4155) = 68
(4156) = 87
(4157) = F2

ii. Do a 16-bit sort of an array of unsigned numbers in descending order. The length of the array is at 4150 and the array starts from 4151.

Sample problem :

Data : (4150) = 0 3 - Length of array
(4151) = 2A
(4152) = B5
(4153) = 60
(4154) = 3F
(4155) = D1
(4156) = C6

The numbers are B52A, 3F60 and C6D1.

Result : (4151) = D1
(4152) = C6
(4153) = 2A
(4154) = B5
(4155) = 60
(4156) = 3F

2.15 EXPERIMENT 13 - BCD TO HEX CONVERSION

OBJECTIVE: To convert two BCD numbers in memory to a hex number.

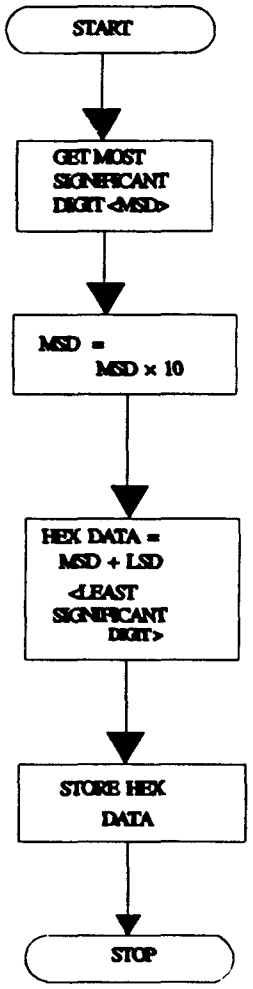
THEORY: Out of the two BCD digits at 4150 and 4151, the one at 4150 is the most significant digit. The program multiplies the most significant digit by ten using repeated addition. Then the least significant digit will be added to it.

EXAMPLE: The BCD digits are at locations 4150 and 4151 and result will be stored at 4152.

Data : (4150) = 02 (MSB)
(4151)= 09 (LSB)

Result : (4152) = 1D =29 (decimal).

FLOW CHART FOR BCD TO HEX



PROGRAM

```
LXI   H, 4150   ;
MOV   A,M       ;   Get most significant digit (MSD)
ADD   A         ;   MSD × 2.
MOV   B,A       ;   Save MSD × 2.
ADD   A         ;   MSD  × 4.
ADD   A         ;   MSD  × 8.
ADD   B         ;   MSD  × 10.
INX   H         ;   Point LSD
ADD   M         ;   Add to form hex equivalent.
INX   H         ;
MOV   M, A      ;   Store result.
HLT             ;   Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H, 4150
4101	50	
4102	41	
4103	7E	MOV A, M
4104	87	ADD A
4105	47	MOV B,A
4106	87	ADD A
4107	87	ADD A
4108	80	ADD B
4109	23	INX H
410A	86	ADD M
410B	23	INX H
410C	77	MOV M, A
410D	78	HLT

PROCEDURE: Follow similar procedures as explained previously to enter and execute programs. Also try change data and record results.

DISCUSSION: Try to write a program for hex to BCD conversion, which will help you to practise code conversions.

EXERCISES: i. Convert the hex data in memory location 4150 to its equivalent BCD number at 4151 and 4152. The hex is unsigned and less than 100.

Sample problem :

Data : (4150) = 47

Result : (4151)= 07 (MSB)
(4152)=01 47 = 71 (Decimal)

2.16 EXPERIMENT 14 - HEX TO DECIMAL CONVERSION

OBJECTIVE: To Convert the hex number in memory to its equivalent decimal number in memory.

THEORY: In this experiment, the hex number is converted to its equivalent decimal number in the following way. First count the number of hundreds, the number of tens and units present in that hex number.Using these three, add up to get the equivalent decimal number. For example, take the hex number A9 and converting it as:

```
  A9
- 64
  ---     Hundreds = 01
  45
  ---
```

Since 64 (100 decimal) cannot be subtracted from 45, number of hundreds = 01. Now count tens.

```
  45
- 0A     Tens = 01
  ---
  3b
- 0A     Tens =02
  ---
  31
- 0A     Tens =03
  ---
  27
- 0A     Tens = 04
```

```
  ---
   1d
 - 0A      Tens = 05
  ---
   13
  -0A      Tens= 06
  ---
   09
  ---
```

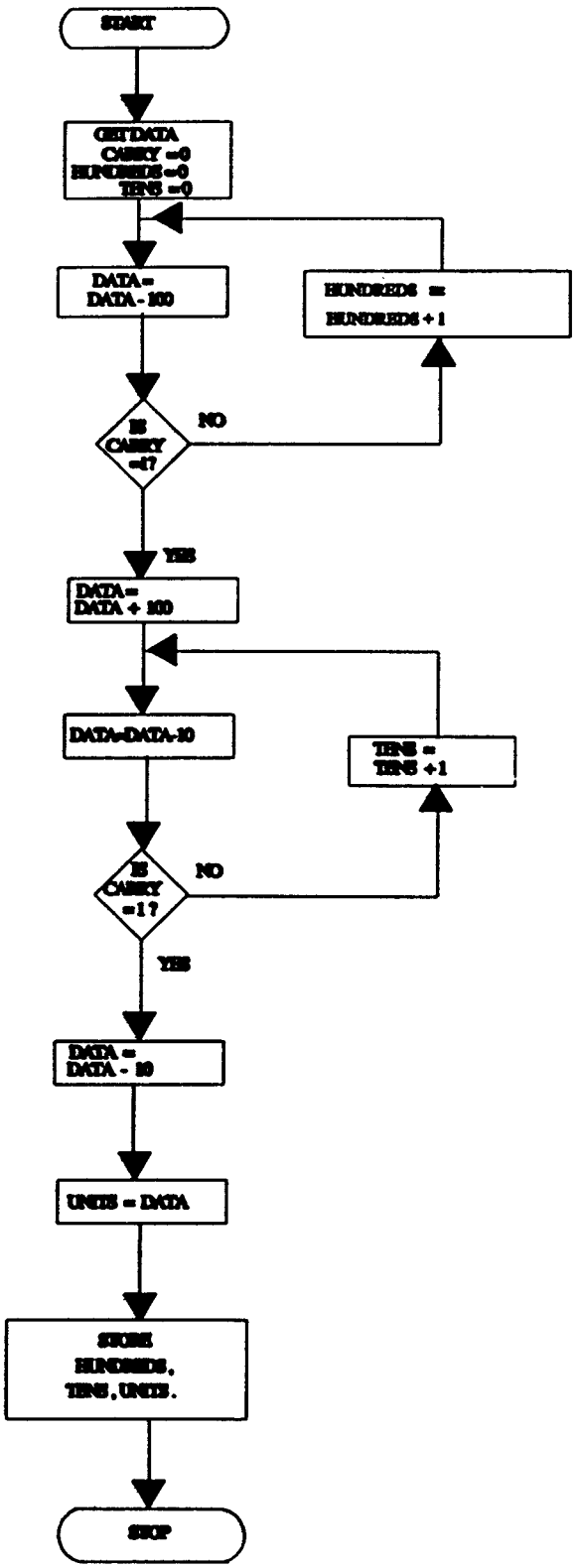
Now from 09, 0A cannot be subtracted. Hence tens = 06. Therefore the decimal equivalent of A9 is 169.

EXAMPLE: Let us work with this program with the example A9 itself. So at 4150, it is A9.

Data : (4150) =A9

Result : (4151) = 01
(4152) =69
$A9_{16} = 169_{10}$

FLOW CHART FOR HEX TO DECIMAL



PROGRAM:

```
        LXI   H, 4150     ;   Point to data.
        LXI   B, 0000     ;   Initialise Hundreds = 0,
                              Tens    = 0.
        MOV   A,M         ;   Get hex data to A.
LOOP :  SUI   64
        JC    LOOP1
        INR   B           ;   Hundreds = Hundreds +1
        JMP   LOOP
LOOP1:  ADI   64          ;   If subtracted extra, add it clear carry flag.
LOOP2:  SUI   0A
        JC    LOOP3
        INR   C           ;   Tens  =  Tens + 1.
        JMP   LOOP2       ;
LOOP3:  ADI   0A          ;   If subtracted extra, add it again.
        INX   H           ;   A =     Units.
        MOV   M,B         ;   Store Hundreds.
        MOV   B,A         ;   Combine Tens in C and
        MOV   A,C         ;   units in A  to form a
        RLC               ;   single 8-bit number.
        RLC
        RLC
        RLC
        ADD   B
        INX   H
        MOV   M,A         ;   Store Tens and units.
        HLT
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H, 4150
4101	50	
4102	41	
4103	01	LXI B, 0000
4104	00	

Memory Address	Object Codes	Mnemonics
4105	00	
4106	7E	MOV A,M
4107	D6	LOOP : SUI 64
4108	64	
4109	DA	JC LOOP1
410A	10	
410B	41	
410C	04	INR B
410D	C3	JMP LOOP
410	07	
410F	41	
4110	C6	LOOP1:-ADI 64
4111	64	
4112	D6	LOOP2: SUI 0A
4113	0A	
4114	DA	JC LOOP3
4115	1B	
4116	41	
4117	0C	INR C
4118	C3	JMP LOOP 2
4119	12	
411A	41	
411B	C6	LOOP : ADI 0A
411C	0A	
411D	23	INX H
411E	70	MOV M. B

411F	47	MOV B,A
4120	79	MOV A,C
4121	07	RLC
4122	07	RLC
4123	07	RLC
4124	07	RLC
4125	80	ADD B
4126	23	INX H
4127	77	MOV M,A
4128	76	HLT

PROCEDURE: A similar procedure for entering data and executing programs must be followed which is already explained in previous experiments.

DISCUSSION: Try to write the program for decimal to hex conversion. Don't try to follow similar logics given in these experiments. If you want to become an expert programmer always think of new ideas of doing things.

EXERCISE: i Convert the decimal at memory location 4150 and 4151 to its equivalent hex number to at 4152.

Sample Problem:

Data : (4150) = 50
(4151) = 01

Result : (4152) = 96
$150_{10} = 96_{16}$

2.17 EXPERIMENT 15 - ASCII TO DECIMAL CONVERSION

OBJECTIVE: To convert the ASCII number in memory to its equivalent decimal number.

THEORY: Conversion of an ASCII number to decimal is very simple because all the decimal numbers form a sequence in ASCII. Any ASCII number can be converted to decimal just by subtracting 30 from it.

ASCII Number (Hex)	Decimal Equivalent
30	00
31	01
32	02
33	03
34	04
35	05
36	06
37	07
38	08
39	09

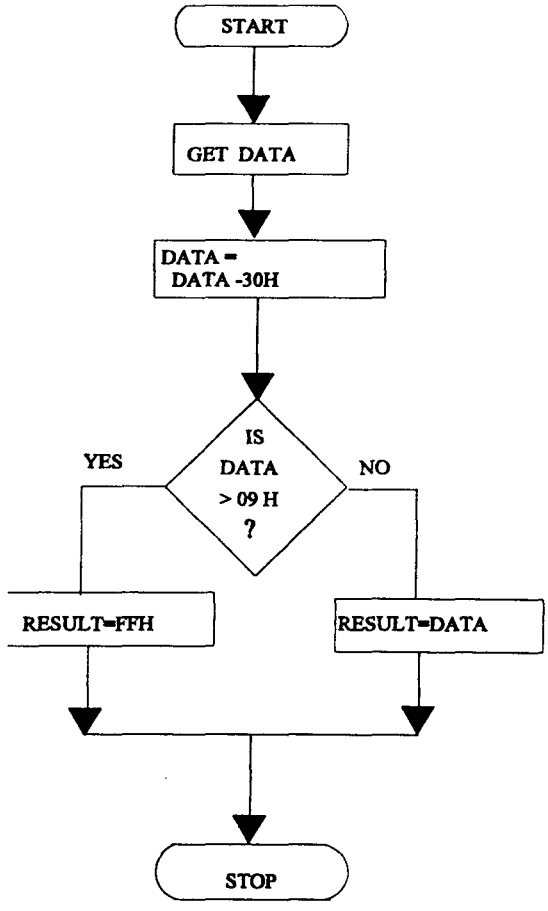
EXAMPLE: Let the ASCII number be at location 4150 and the result be stored at 4151.

Data : (4150) = 35
Result : (4151) = 05.

Data : (4150) = 3A
Result : (4151) = FF.

So if the decimal number got is not a valid one. FF will be stored as the result to indicate 4 the error.

FLOW CHART FOR ASCII TO DECIMAL



PROGRAM :

```
        LXI   H,    4150  ;   Point to data.
        MOV   A,M         ;   Get operand.
        SUI   30          ;   Convert to decimal.
        CPI   0A          ;   Check whether it is a valid decimal
                              number.
        JC    LOOP        ;   Yes. store result.
        MVI   A,FF        ;    No. make result = FF.
LOOP :  INX   H           ;
        MOV   M,A         ;
        HLT               ;   Stop execution.
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H, 4150
4101	50	
4102	41	
4103	7E	MOV A, M
4104	DE	SUI 30
4105	30	
4106	FE	CPI 0A
4107	0A	
4108	DA	JC LOOP
4109	0D	
410A	41	
410B	3E	MVI A, FF
410C	FF	
410D	23	LOOP: INX H
410E	77	MOV M,A
410F	76	HLT

PROCEDURE: The procedure to enter data and execute programs is as explained in the previous experiments. Record results for the different data.

DISCUSSION: Similar to this way, a lot of programs can be done based on ASCII code conversions like ASCII to decimal, ASCII to hex, hex to ASCII etc. Try them and be sure you are familiar with all code conversion techniques.

EXERCISES :

i. Convert the decimal number at 4150 to its equivalent ASCII at 4151.

Sample problems :

(a) Data : (4150) = 03
Result : (4151) = 33

(b) Data : (4150) = 10
Result : (4151) = FF - (Error)

ii. Convert the hex number at 4150 tp its equivalent ASCII at 4152.

Sample problems:

(a) Data : (4150) = 0F
Result : (4151) = 46

(b) Data : (4150) = 09
Result : (4151) = 39

(c) Data : (4150) = 10
Result : (4151) = FF - (Error)

iii. Also try ASCII to hex conversion.

2.18 EXPERIMENT 16 - DELAY LOOPS

OBJECTIVE: To write a program that generates a delay of 1 ms with the 8085 at a clock rate of 3.07 2MHZ.

THEORY: We can produce timing intervals in several ways:

i. In hardware, using monostable multivibrator, which produce a single pulse of fixed duration in response to a trigger input.

ii. In a combination of hardware and software with a flexiable programmable time such as 8253 .

iii. In software using delay routines. These routines use the processor as a counter with a stable clock as reference. Even though, these delay routines require no external hardware, they consume processor time.

A simple delay routine works as follows

Load a register with a specified value. Decrement register contents. If register ≠0 then goto previous step. The above routine is strictly for timing purposes, it serves no other function. The amount of time taken depends upon the execution time of the instructions used.

EXAMPLE: In the example given below, the 1 ms delay required is determined by the count given to the register. The calculation of the delay is given following the program.

PROGRAM:

```
        MVI  B,Count    ;   Load data count to B register
                            used for delay.
LOOP :  DCR  B          ;   Check if B = 0.
        JNZ  LOOP       ;   If B ≠ 0, loop back again.
        HLT
```

DELAY CALCULATION

The above program is part of a program where delay routines are used. The delay time is the actually the calculation of the total execution time for the program. Since the delay time is taken to be 1 ms, the count should be varied in order to get that delay. Now, write down the T-states taken for each instruction.

```
        MVI  B,Count    7    T-States.
LOOP :  DCR  B          4    T-States
        JNZ  LOOP       10   T-States.
```

The above program takes,

7+ ([4+10] * COUNT) T - states to execute.

For a delay time 1 ms, calculation is as follows:

i) Clock rate = 3.072 MHZ.

ii) Clock time = $\frac{1}{3.072\ \text{MHZ}}$ = 0.325 micro seconds.

iii) Total time = 7 * 0.325 micro sec.
+ ([4+10)* 0.325 micro sec.] *count)

iv) 1 ms = 2.275 micro sec. + (4.55 micro sec. * count)

v) 1 ms -2.275 micro sec. = count = 219 = DB (hex)

4.55 micro sec.

So, if the count = DB, then this routine will give a delay of 1 ms. Do not execute this program alone. you can learn nothing just by executing it. Use this as a subroutine and execute it along with additional loops so that you will see the 1 ms delay, the routine provides.

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	06	MVI B, DB
4101	DB	
4102	05	LOOP : DCR B
4103	C2	JNZ LOOP
4104	02	
4105	41	
4106	76	HLT

PROCEDURE: The procedure to enter data and execute programs is as explained in the previous experiments. Record results for the different data.

DISCUSION: The above routine provides a delay of 1 ms. If the delay time should be increased, then the count in register B should be increased. But that is limited by the fact that the B register can store only upto a maximum of FF. But still now, delay can be increased by using 16- bit registers. However, using 8-bit registers, the entire routine can be placed inside a similar routine which uses another register and so on.

EXERCISE: i. Write a delay routine that produces a delay of one second. Use either 16-bit or 8-bit registers.

2.19 EXPERIMENT 17 - SUM OF 'N' ELEMENTS

OBJECTIVE: To compute the 16-bit sum of N elements of a table. The starting address of table is 4150 whose first entry is the number of elements N. The result is stored at 4140 and 4141. If the result should exceed 16-bits, then only the least 16-bits will be stored.

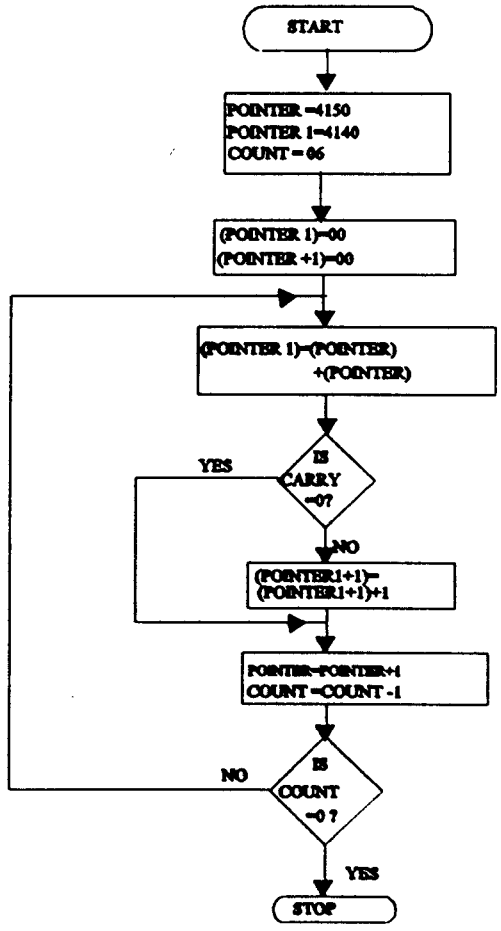
THEORY: The data to be added are held at memory locations whose starting address is held by the HL register. The data is added in 8- bits and the result is stored at 4140 . For each and every carry during this addition, a counter is incremented. This counter forms the Most significant digit of the addition which is stored at location 4141.

EXAMPLE: The table starts at 4150 . The result will be stored at 4140 and 4141.

Data : (4150) = 06 - Count
(4151) = 54 - Entries of table.
(4152) = 63
(4153) = 7A
(4154) = 2B
(4155) = 8F
(4156) = FF

Result : (4140) = EA
(4141) = 02

FLOW CHART FOR SUM OF "N" ELEMENTS



PROGRAM:

```
        LXI   H,    4150   ;    point to table base.
        MOV   B,    M      ;    Get length of counter.
        INX   H            ;    point to first entry.
        MVI   E,    00H    ;    Intialise e register.
        XRA   A            ;    Clear  accumulator
LOOP1   ADD   M
        JNC   LOOP2        ;    Check carry
        INR   E            ;
LOOP2   INX   H
        DCR   B            ;    Decrement counter
        JNZ   LOOP1        ;    Is counter      = 0  ?
        STA   4140
        MOV   A, E
        STA   4141
        HLT
```

OBJECT CODES

Memory Address	Object Codes	Mnemonics
4100	21	LXI H , 4150
4101	50	
4102	41	
4103	46	MOV B , M
4104	23	INX H
4105	1E	
4106	00	MVIE, 00H
4107	AF	XRA A
4108	86	LOOP1 ADDM
4109	D2	JNC LOOP2
410A	0D	
410B	41	
410C	1C	INR E
410D	23	LOOP2 INX H

410E	05	DCR B
410F	C2	JNZ LOOP 1
4110	08	
4111	41	
4112	32	STA 4140
4113	40	
4114	41	
4115	7B	MOV A , E
4116	32	STA 4141
4117	41	
4118	41	
4119	76	HLT

PROCEDURE: The procedure to enter data and execute programs is as explained in the previous experiments. Record results for the different data.

DISCUSSION: This is not the only method of doing sum of 'N' elements. Without using index registers, the same program can be re-written which you can try. This program is only for 16-bit sum.

EXERCISE:

i. Compute the 24-bit sum of N elements in table and store the result.

Sample problem:

Data : (4150) = 06 - Count.
(4151) = FF - Elements of array
(4152) = FF
(4153) = FF
(4154) = FF
(4155) = FF
(4156) = FF

Result : (4140) = FA
(4141) = 05
(4142) = 00

2.20 EXPERIMENT-18: SUBROUTINES

A subroutine is a group of instructions that performs a subtask (eg. time delay or arithmetical operations) of repeated occurrence. The subroutine is written as a separate unit, apart from the main program, and the Microprocessor transfers program execution from the main program to the subroutine whenever it is called to perform the task. After completion of the subroutine task, the Microprocessor returns to the main program.

This subroutine technique eliminates the need to write a sub task repeatedly; thus, it uses memory more efficiently. Most microprocessors have special instructions for transferring control to subroutines and restoring control to main program. The 8085 Microprocessor refers this special instruction as CALL .The following is the sequence of the CALL instruction.

i) Upon the execution of a CALL instruction the contents of the program counter are saved on stack..

ii) program execution is transferee to the subroutine address.

iii) When a RET instruction is encountered in the subroutine, the contents of stack are placed onto PC so that execution starts at the main program.

The stack pointer mentioned here is a 16-bit register which holds the top address of the group of memory locations assigned for temporary storage of data during program execution.

Usually the stack is defined at the high end of the memory map or at locations which will not collide with user programs. The main reason for this is, the stack actually grows down when data is stored. For example, if you store data in stack, the stack pointer gets decremented and if your read data from stack it is incremented.

The size of the stack is limited only by the available memory. Study the instructions that affect the stack and stack pointer and write the similarities and dissimilarities between them.

The following points are worth nothing while using subroutines.

i) Do not affect the stack in the subroutine, for it contains the return address to the main program.

ii) If the stack is to be used in the subroutine, save it before changing and reload before returning to the main program.

The 8085 provides both conditional and unconditional CALL and RET.

First, let us take a small program with subroutine and check stack, PC, registers at each and every point.

EXAMPLE-1:

To illustrate the change of contents between stack and PC during subroutine CALL.

LINE NO.	ADDRESS	MNEMONICS	COMMENTS
1	4100	LXI SP, 5FFF	Stack pointer Register =5FFF
2	4120	CALL 4200	(5FFE)= 23 (5FFD)= 41 Stack pointer Register =5FFD
3	4170	HLT	(PC) = 4200
4	4200	MVI A,36	
5	4223	RET	Stack pointer Register=5FFF (PC) = 4123

1. The stack pointer register is initialised to 5FFF.

2. The CALL Instruction saves the address of the next instruction following the CALL in the stack, i.e. at address 5FFE and 5FFD. The stack pointer register is now decremented by two. The subroutine address 4200 is taken by PC.

3. Main program ends.

4. Subroutine starts.

5. The RET instruction takes two bytes from top of stack and saves them in PC. The stack pointer register is incremented by two.

NOTE :

Understand the similarity and differences between CALL and PUSH, RET and POP.

EXERCISES :

1. For a given decimal number 23, find the equivalent Hex, ASCII and BCD Numbers in a single program. Use Separate subroutines for each code conversion.

DATA :	(4500)	=	17	(Hex)
	(4511)	=	32	(ASCII LSB)
	(4512)	=	33	(ASCII MSB)
	(4513)	=	02	(BCD LSB)
	(4514)	=	03	(BCD MSB)

2. In the Micropower-i trainer, let the stack be initialised to 0FFF.

i) What will happen if a CALL is made now?

ii) Will the control be transferred to the main program after the RET instruction in the subroutine is executed ?

3. Why should the stack be saved and restored back whil changing it in a subroutine?

4. Initialise stack pointer to 40FF.

Push the contents of B, D, H and PSW. Write down the contents of B,D,H,PSW, SP and the contents of the memory from 40F0 to 40FF. Observe the changes in these registers and memory.

2.21 TESTING FLAGS

OBJECTIVE: To manipulte the carry (C) ; Zero (Z) , Sign (S) and parity (P) flags and to determine the conditions that set and reset these flags.

THEORY: Conditional jump instructions give the computer the power to make decisions based upon data or computed results. The flags always monitor the accumulator and registers and signal the presence of a specific condition. A profound knowledge of the role of flag register, the instructions that affect them, are necessary to enhance your ability as a programmer. The flag register of 8085 is as follows.

where,

S	Z		AC		P		C
7	6	5	4	3	2	1	0

S - Sign Flag : Acquires the value of the most significant bit of the result following the execution of any BCD or Boolean instruction.

Z - Zero Flag : It is set to 1 whenever any arithmetic or Boolen operation generates 0 result. Reset if vice versa.

AC - Auxiliary carry flag : Holds any carry from bit 3 to 4 resulting from and arithmetic operation.

P - Parity Flag : Set if the result of an arithmetic or Boolean operation produces a result which containas an even number of 1 bits. Cleared if there is odd number of 1 bits.

C - Carry Flag : Holds carry out of the most significant bit in any arithmetic operation.

In the example given below, you will be testing the status of S,Z, C and P flags which lead to conditional jumps, calls and returns, This program is a "do nothing " program. Each instruction sets and resets flags which can be monitored separately using the single step command.

PROGRAM :

```
NOP
MVI  A,05
ADI  FF
ORI  14
ADI  90
RAL
XRA  A
HLT
```

OBJECT CODES

The object codes for the above program are as folows.

Memory Address (Hex)	Object Codes	Mnemonics
4100	00	NOP
4101	3E	MVI A , 05
4102	05	
4103	C6	ADI FF
4104	FF	
4105	E6	ANI F4
4106	F4	
4107	F6	ORI 14
4108	14	
4109	C6	ADI 90

410A	90	
410B	17	RAL
410C	AF	XRA A
410D	76	HLT

PROCEDURE

i) Key in the opcodes in the trainer from the address 4100.

ii) Only by single-stepping through the program, you can view the Flag status corresponding to individual instructions.

iii) The first instruction is a "NOP". Its primary function is to make you see the first instruction before it is executed (Remember that only after executing the first instruction, the STEP command will wait).

iv) Press the STEP key when prompted for entry.

v) Enter the address 4100 and press NEXT.

vi) Now, the NOP would have been executed and the display shows the address from where the program actually starts. Record this address in the table.

vii) Press EXEC to display command prompt.

viii) Press REG key.

ix) Press 0 for Flag Register key.

x) Note down the value in hex as its equivalent binary number in the table given below.

xi) Press EXEC to come back to prompt.

xii) Press STEP key followed by NEXT.

xiii) Now the next instruction is executed and the flags are updated accordingly. Press EXEC to come to prompt. Record the contnts of F in the table given below.

xiv) Repeat the same from step (vii) and record the values of Flag register after each operation.

xv) Repeat until the address 410D is displayed when STEP key is pressed.

Step No.	Mnemonics Executed	Flag Register								Program counter contents
		S	Z	X	AC	X	P	X	C	
1	NOP									
2	MVI A,05									
3	ADI FF									
4	ORI 14									
5	ADI 90									
6	RAL									
7	XRA A									
8	HLT									

DISCUSSION: In the above example, you stepped through a simple "Do-NOTHING" nprogram that manipulated the Flags of 8085. From the above table, the following conclusion can be made.

i) The carry flagis set to 1, if a carry/borrow was generated as a result of an arithmetic operation or due to rotation. (Step 3).

ii) The parity Flag is set if the number of ones in a number is even. (Step 1, 5, 8).

iii) The Auxilary Carry Flag is set if there is a carry out of bit 3 to bit 4. (Step 3, 4).

iv) The zero Flag is set if the result of an arithmetic or logical operation results in a zero. (Step 8).

v) The sign Flag is set if the most significant bit of the Accumulator is set. (Step 6.7).

EXERCISES

i) what are the instructions that always set the Auxiliary carry Flag to 1 ?

ii) what does the instruction CMC do ? Will Carry Flag be set to 1 or 0 after the execution of the instruction CMC ?

iii) Will the rotate instruction affect both Carry and Auxiliary Carry Flag ?

iv) Will the parity Flag be set if the number of zeroes is even in a number ?

v) Which Increment and Decrement instructions will not affect Flags ?.

vi) Does all Logical operations reset Carry Flag to Zero ?.

vii) Name the instructions that clear the Auxiliary Carry Flag to Zero ?

EPILOGUE

After finishing all the above experiments, there is no doubt that you will become a good programmer in arithmetic, logical, code conversion operations. To become still more experienced with the hardware based software aspect, try the experiments in the next chapter which deals on Hardware programming examples on the peripherals available on Micropower-i with 8085 CPU.

CHAPTER 3

HARDWARE EXAMPLES

3.1 INTRODUCTION

This chapter emphasizes more on the INTERFACE PROGRAMMING ASPECTS of the various peripherals available on Micropower-i.

As in the earlier Chapter, you will have to try and exercise the given EXAMPLES AND EXERCISES to strengthen your technical muscle, in understanding the operational capabilities of the peripheral devices used. Chapter 2&3 should no doubt set a milestone in shaping you be a proficient system programmer.

Ample detail pertaining to the programming aspect of the peripherals has been provided in the succeeding sections. But for further details that concern the internal architecture, timing diagrams and hardware organisation of the chips the users are requested to consult the Manufacturer Data Sheets of either Intel or Nation Semiconductor as regards the case.

NOTE

The signals specified with a "*" to its right indicate active low signals.

3.2 PROGRAMMABLE INTERVAL TIMER (8253): (U4)

3.2.1 COMPONENT

The Intel 8253 is used in this design as a general purpose timer and also to provide baud rates for the USARTS used in this design.

3.2.2 COMPONENT DESCRIPTION

The unique features of the Intel 8253 are as follows:

3 Independent 16-bit counters
Input clock from DC to 2 MHZ
Programmable Counter Modes
Count Binary or BCD

NOTE

On the kit, the CHANNEL 0 and 2 are used for the two RS232C Communication baud rate generation for the two 8251. If the 8251 is not going to be used, the timer can be used for different applications. **Channel 1 is reserved for single step and as such the user cannot use this channel for his applications.**

The pins with which the processor communication to 8253 are RD*, WR*, A0, A1 Data bus, D0-D7. The CS* selects the device.

The timer's basic operation can be briefly given as,

CS*	RD*	WR*	A1	A0	FUNCTION
0	1	0	0	0	Load Counter 0
0	1	0	0	1	Load Counter 1
0	1	0	1	0	Load Counter 2
0	1	0	1	1	Write control word
0	0	1	0	0	Read counter 0
0	0	1	0	1	Read counter 1
0	0	1	1	0	Read counter 2
0	0	1	1	1	No-operation 3-state
1	x	x	x	x	Disabled 3-state
0	1	1	x	x	No-operation 3-state

3.2.3 PROGRAMMING DESCRIPTION

CONTROL WORD REGISTER

It stores the operational mode of each counter. This register can only be written into, no read operation of its contents are possible. Each counter of 8253 can be individually programmed by writing corresponding control words by simple I/O operations.

CONTROLWORD FORMAT

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RL1	RL0	M2	M1	M0	BCD

DEFINITION

SC-Select Counter

SC1	SC0	FUNCTION
0	0	Select Counter 0
0	0	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

RL-Read / Load

RL1	RL0	FUNCTION
0	0	Counter latch
0	1	Read / Load LSB only
1	0	Read / Load MSB only
1	1	Read / Load LSB first then MSB

NOTE

Any counter can be read even when it is under operation.

Mode Selection

M2	M1	MO	MODE
0	0	0	Mode 0
0	0	1	Mode 1
0	1	0	Mode 2
0	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

*BCD
0 - Binary Counter 16 - Bits.
1 - BCD Counter (4 Decades)

Depending on the requirement and referring the definitions explained above, we have to find the mode word for initialising 8253. As you know, SC0 and SC1 are used to select any one of three counters present in 8253. RL0 and RL1 selects one of the four types of loading the counter value. Then comes the mode select. In the mode select we have six different types of modes to select. They are,

MODE 0: INTERRUPT ON TERMINAL COUNT

The output will be initially low after mode set operation. After loading the counter, the output will remain low while counting and on terminal count, the output will become high, until reloaded again.

MODE 1: PROGRAMMABLE ONE-SHOT

After loading the counter, the output will remain low following the rising edge of the gate input. The output will go high on the terminal counter. It is re- triggerable any rising edge of the gate input.

MODE 2: RATE GENERATOR

It is a simple divide by N counter. The output will be low for the one period of the input clock. The period from one output pulse to the next equals the number of input counts in the count register. If the count register is reloaded between output pulses the present period will not be affected, but the subsequent period will reflect the new value.

MODE 3: SQUARE WAVE GENERATOR

It is similar to Mode 2 except that the output will remain high until one half of count and go low for the other half for even number count. If the count is odd, the output will be high for (count+1)/2 counts and low for (count-1)/2 counts. This mode is used for generating baud rate for 8251 USART:

MODE 4: SOFTWARE TRIGGERED STROBE

The output is high after mode is set and also during counting. On terminal count, the output will go low for one clock period and becomes high again. This mode can be used for interrupt generation.

MODE 5: HARDWARE TRIGGERED STROBE

Counter starts counting after rising edge of trigger input and output goes low for one clock period on terminal count. The counter is retriggerable.Then finally, we have one more option of selecting a counter as BCD or Binary using the "BCD" bit of the control word.

The three counters are identical in operation. Each counter consists of a single, 16-bit, pre-settable, DOWN counter. The counter may operate in either binary or BCD and its input, gate and output are configured by the selection of MODES stored in the Control Register.

The counters are fully independent and each can have separate Mode configuration and counting operation. The reading of the contents of each counter is available to the programmer with simple READ operations for event counting applications.

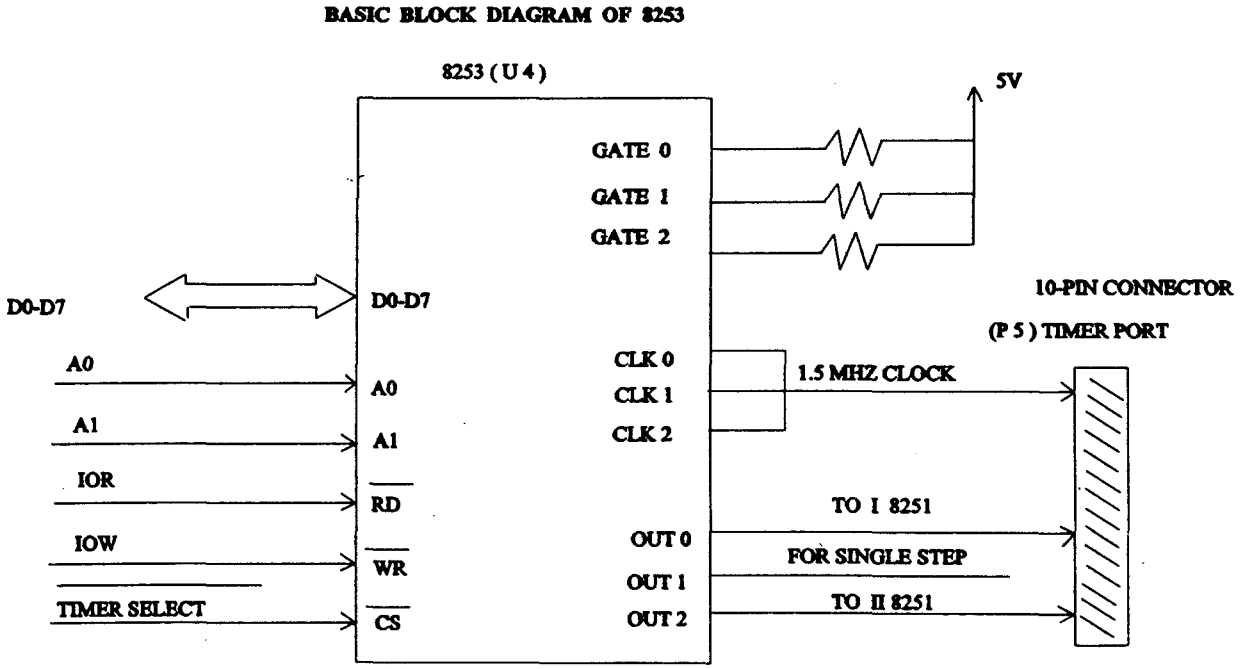
For instance, if we want to initialise 8253 for the following parameters, say "Channel 2 - Square Wave - at 15 Khz". Then,

SC1, SC0	=	1,0	--->	Select Channel 2
RL1, RL0	=	1,1	--->	Select LSB first then MSB
M2,M1,M0	=	0,1,1	--->	Select square wave mode
BCD	=	0	--->	Select a Binary Counter
Control word	=	B6 (hex)		

Using the OUT instruction of 8085 output the control word to the control register and the count to Channel 2 so that you get an output frequency of 15 MHZ whose input clock is nearly 1.5MHz.

3.2.4 CIRCUIT IMPLEMENTATION

The following figure 3.1 gives a clear idea about the interface between the 8253 and the 8085 microprocessor.



$\overline{IOR}$ = I/O READ FROM DECODING LOGIC
IOW = I/O WRITE FROM DECODING LOGIC
D0-D7 = DATA BUS FROM MICROPROCESSOR
TIMER SELECT = CHIP SELECT FOR TIMER FROM ADDRESS DECODING LOGIC. IT SELECTS FOR I/O ADDRESSES 04 TO 07

As is clear from the above block diagram, the Data bus D_0-D_7 is connected to microprocessor data bus. The IOR* and IOW* are connected to the RD* and WR* of 8253 respectively. The TIMERSELECT which is got of the address decoding logic provides the Chip Select to the 8253. This Chip Select is low for I/O addresses from 04 to 07. The A0 and A1 lines are connected to the and A1 address lines of the microprocessor respectively.

The I/O addresses for the timer are as follows.

IC No.	Function	Address
U4	Control register	07
U4	Channel 0	04
U4	Channel 1	05
U4	Channel 2	06

Having learnt about the hardware organisation, functional description of 8253, let us write a simple program to generate a square wave of 150KHz from Channel 2 of 8253.

3.2.5 EXAMPLE 1 - SQUARE WAVE GENERATION USING 8253

OBJECTIVE: To generate a square wave of frequency 150KHZ from Channel 2 of 88253.

THEORY : To get a square wave out of any channel, the channel must first be initialised to the mode in which a square wave can be obtained. The mode for which a square Wave Generator. As explained before, the output frequency of clock from Channel 2 will depend upon the count initialised to Channel 2. To get an output clock frequency of 150KHZ, where the input clock frequency is 1500KHz, the count to be initialised must be 1500/150 = 0A (hex). So, if Channel 2 is initialised to Mode 3 and count 0A, the output clock frequency will be 150KHZ (square wave).

The control word that must first be written into the control register will be 10110110 b, which initialises the control register to Select Channel 2, Read / Load LSB first, then MSB, Mode3, Binary counter 16-bits.

PROGRAM:

This program is to generate a clock of frequency 150 KHz at Channel 2, 8253 whose input clock is 1.5 MHz.

```
                              TMRCNT EQU  07
                              TMRCH2 EQU  06

4100    3E B6     START:      MVI  A,B6
4102    D3 07                 OUT  TMRCNT
4104    3E 0A                 MVI  A,0AH
4106    D3 06                 OUT  TMRCH2
4108    AF                    XRA  A
4109    D3 06                 OUT  TMRCH2
410B    76                    HLT
```

VERIFICATION:

Enter the program given above as shown from the address and then execute the program. Using an oscilloscope, verify at OUT 2 or 8253 (Pin no.17 of 8253) whether you get a square wave of frequency of 150 KHz.

3.2.6 EXERCISES:

i) What is the output of 8253 at Channel 2, if Channel 2 is initialised for Square wave generator mode and the count being 0003.

ii) Write a program to generate a pulse at Channel 2 of 8253 at a time delay of one second.

CAUTION:

As the Channel 1 of the 8253 is Used for the Single Stepping Purpose as already Stated, avoid Writing Programs for the Channel 1. If You Write Programs, then the Processor Will Get Interrupted and the Task You Desired to Perform Will Not Be Done.

3.3 USART (8251): (U16 & U27)

3.3.1 COMPONENT:

The RS232 interface of Micropower-i comprises of the UNIVERSAL SYNCHRONOUS ASYNCHRONOUS RECEIVER / TRANSMITTER 8251 (USART).

3.3.2 COMPONENT DESCRIPTION

The 8251 is used here as a peripheral device for serial communications and is programmed by the CPU to technique. The USART accepts data characters them into a continuous serial data stream for transmission. Simultaneously, it can receive serial data streams and convert them into parallel data characters for the CPU. The CPU can read the complete status of the USART at any time. These include data transmission errors and control signals. The main features of 8251 are,

i) Both synchronous and Asynchronous operation,
ii) False Start Bit Detection,
iii) Automatic Break Detect and Handling,
iv) Clock rate - 1, 16 or 64 times Baud rate,
v) Error Detection - Parity, Overrun and Framing
vi) Break Character Generation.

Micropower-i consists of two such USARTs providing COM1 and COM2 serial ports.

The control pins with which the 8251 communicates with the CPU are the RESET, CLK, WR*,RD*, CS*, C/D*, D0-D7.

The basic operation of 8251 can be briefed as,

C/D*	RD*	WR*	CS*	FUNCTION
0	0	1	0	8251 DATA=>DATA BUS
0	1	0	0	DATA BUS=>8251 DATA
1	0	1	0	STATUS=> DATA BUS
1	1	0	0	DATA BUS=> CONTROL
X	1	1	0	DATA BUS=> 3 STATE
X	X	X	1	DATA BUS=> 3 STATE

The other control pins available for serial interface are DSR (Data set Ready), DTR (Data Terminal Ready), RTS* (Request to Send), CTS* (Clear to Send), RxD (Receive Data) and TxD (Transmit Data).

The signals by which the CPU controls the Transmission and Reception of 8251 are,

TxRDY --- Indicates to CPU, controls that 8251 is ready to accept a character for transmitting serially.

TxEMPTY --- Indicates that the 8251 has no characters to transmit.

TxC* --- Clock that controls transmitting speed. In synchronous transmission Baud rate equals this clock. But in Asynchronous transmission, Baud rate is a fraction of the clock which can be selected when writing control words.

RxRDY --- Indicates the CPU that the 8251 has received a character ready to be inputted to the CPU.

RxC* --- It is similar to TxC* with all the same features for synchronous and asynchronous operation except that it is used for reception.

SYNDET/BD --- This pin is used in synchronous mode as synchronous detect & can be used as either input or output and is programmable by the control word.

RS232C SERIAL DEFINITION

The RS232C is a standard developed by the EIA (Electronic Industries Association) for data transmission in a serial fashion between the transmitter and receiver. The RS232C standard operates at a different voltage level which is as follows. A logic high is at a voltage of -12 V and a logic low is at a voltage of 12 V.

The standard way to interface between RS232C and TTL levels is with MAX 232. Of the 25 handshake signals provided by the RS232C standard, we will discuss only four signals which are used in our design. They are the RTS*, CTS*, Rxd and TxD signals. When connected with another system with serial communication, these signals can be explained as:

i. CTS*: Clear to Send - Enables 8251 transmit serial data, if Tx Enables bit in the command byte is set to one.

ii. RTS*: Request to send - When low, indicates that 8251 can receive serial data. It can be made low by setting the RTS bit in the command byte.

iii. TxD: Transmit serial data.

iv. RxD: Receive serial data.

3.3.3 PROGRAMMING DESCRIPTION

Prior to starting data transmission or reception, the 8251 must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251 and must immediately follow a Reset operation (internal or external).

These control words are split into two formats

1. Mode Instruction Word
2. Command Instruction Word

MODE INSTRUCTION DEFINITION

This format defines the general operational characteristics of the 8251. It must follow immediately a Reset operation. This format defines the Baud rate, Character length, Parity and Stop bits required to work with Asynchronous data communication. By selecting the baud factor to Sync mode, the 8251 can be made to operate in the Synchronous mode.

Figure 3.2. gives the Mode Instruction Format for the Asynchronous Mode.

COMMAND INSTRUCTION DEFINITION

This format defines a status word that is used to control the actual operation of the 8251. All control words written into the 8251 after the Mode Instruction will load the Command Instruction. This command Instructions can be written into the 8251 at any time in the data block during the operation of the 8251. To return to the Mode instruction format, a Master reset is required. The command Instruction format controls the actual operation of the selected format. Figure 3.3. gives the complete information of command instruction format.

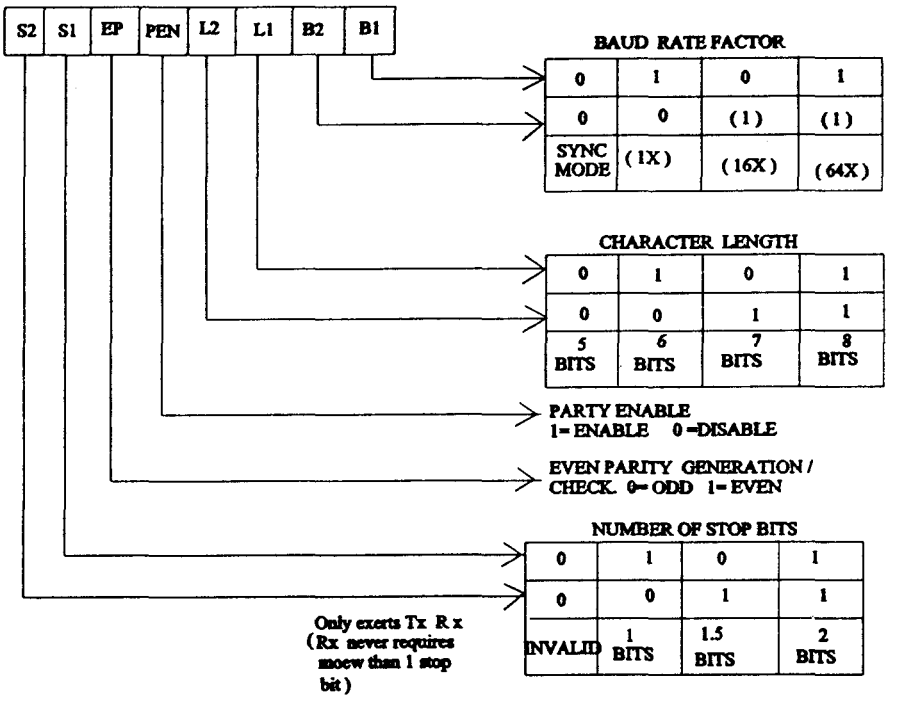
The 8251 has facilities that allow the programmer to read the status of the device at any time during the functional operation. Figure 3.4. gives the information regarding the status register format. Using these information the user can write a good software using protocols for transmitting and receiving.

Then initialising 8251 using the Mode instruction to the following

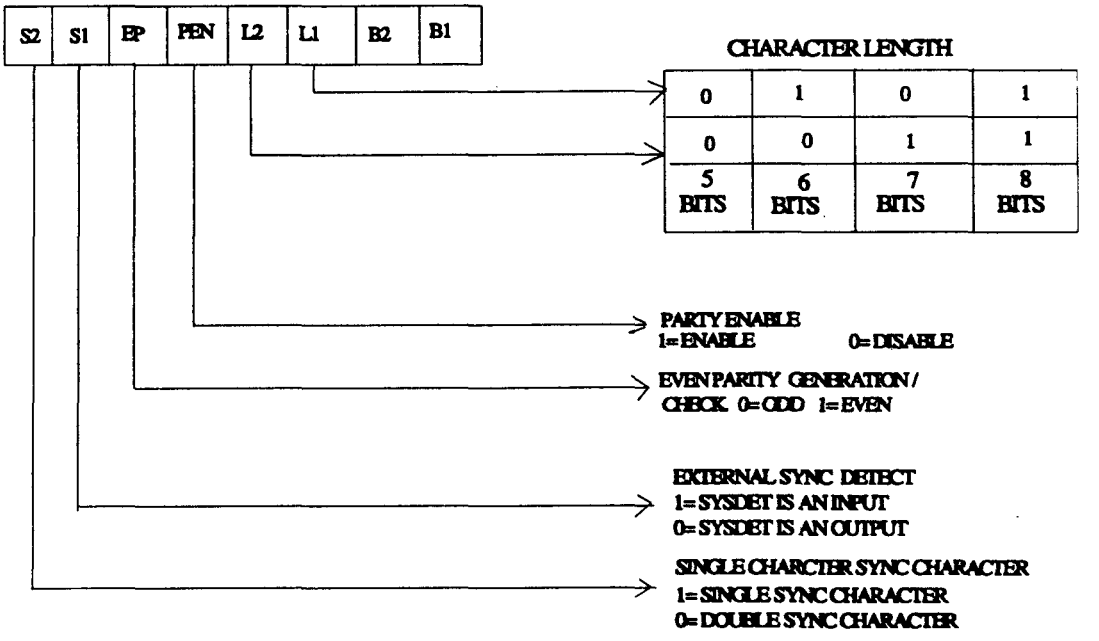
8 bit data
No parity
Baud rate factor (16x)
1 stop bit
gives a Mode command word of 4 E.

MODE INSTRUCTION FORMAT

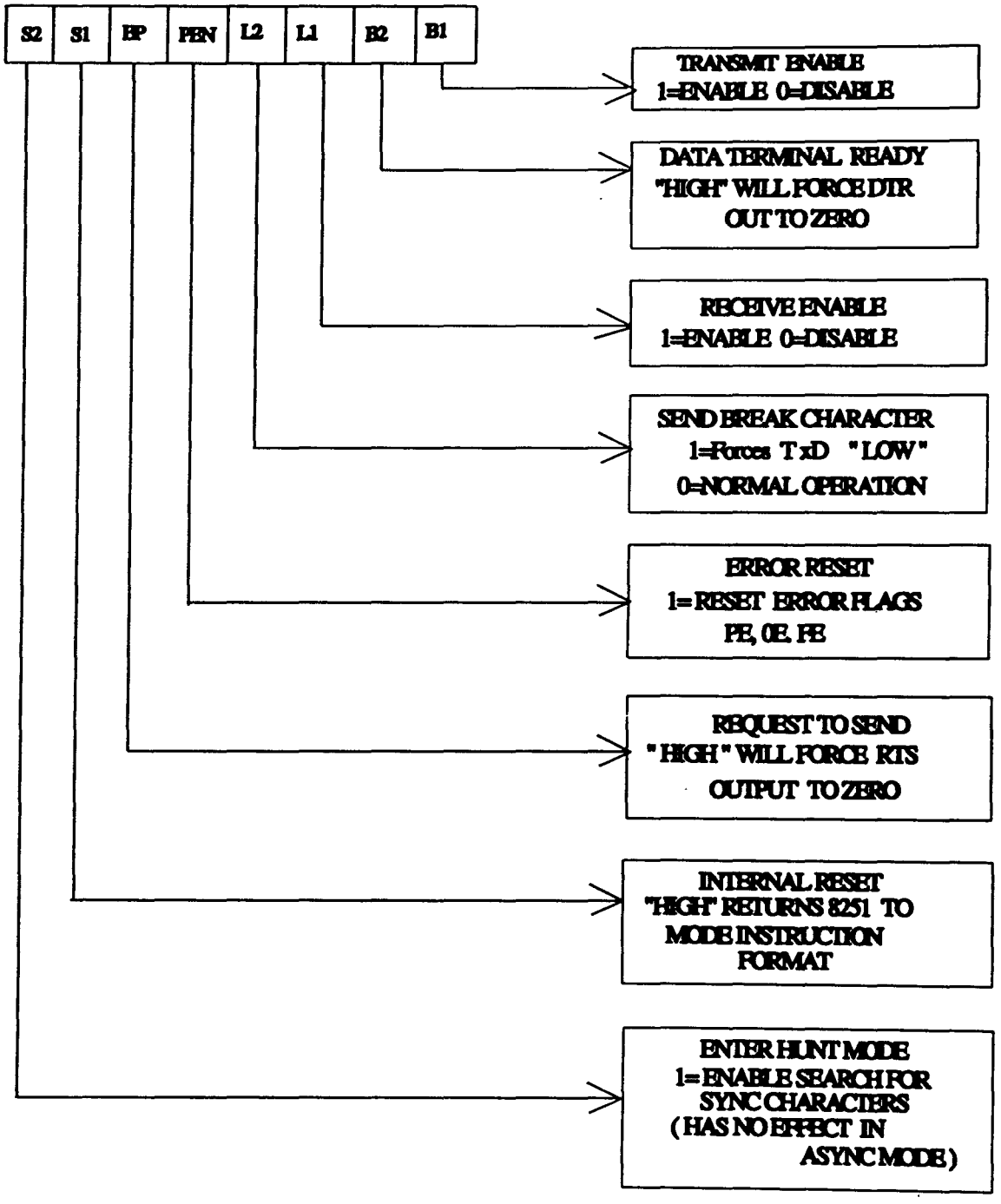
ASYNCHRONOUS MODE 8251



SYNCHRONOUS MODE 8251



COMMAND INSTRUCTION FORMAT 8251



NOTE

Error reset must be performed whenever RX enable and enter hunt are programmed

STATUS READ FORMAT 8251

D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYNDET	FE	OE	PE	TXEMPTY	RXRDY	TXDY

TXRDY Terdy status bit Has different Meanings from the Txrdy output pin. The former is not conditioned by Cts & txen the latter is conditioned by both cts & Txen (i.e) txrdy status bit = db buffer empty txrdy pin Out = Db buffer empty. (Cts-0) (txen-1)

RXDY Receiver ready this bit indicates that the 8251 Contains a character that is ready to Be input to the cpu.

TXEMPTY Transmitter empty when the 8251 Has no no character to transmit this bit will go high.

PE Parity error the PE flag is set when Parity Error Is detected. It is reset by the ER bit of the command instruction. Pe does not Inhibit operation of the 8251

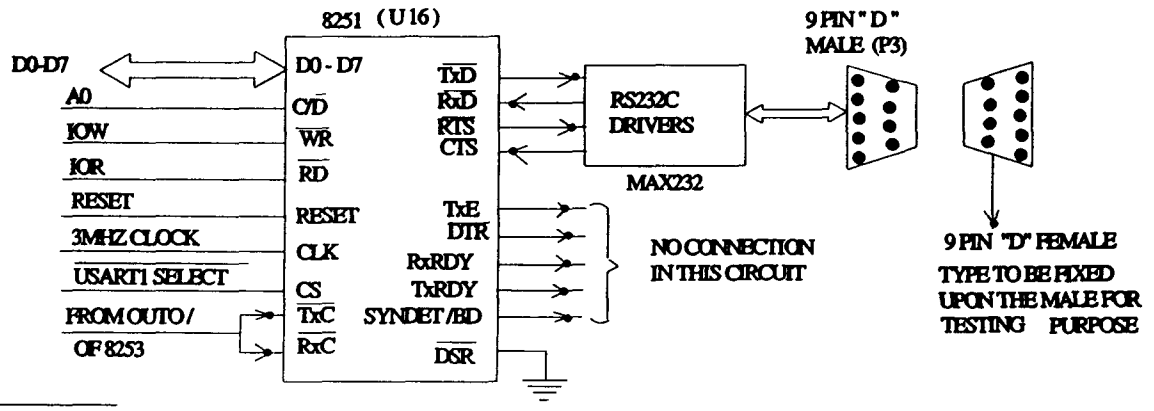
OE Overrun error. The OE flag is set when the CPU does not read a character before the next one becomes available. OT is reset by the ER bit of the command instruction. OE does not inhibit the operation of 8251 however the previously overrun character is lost.

FE Framing error (async only). The fe flag is set when a valid stop bit is not detected at end of every Character. It is reset by the er bitf the command instruction. FE does not inhibit the operation of 8251

SYNDET Sync detect this pin is used in synchronous mode for Syndet and is used in asynchronous mode For break detect

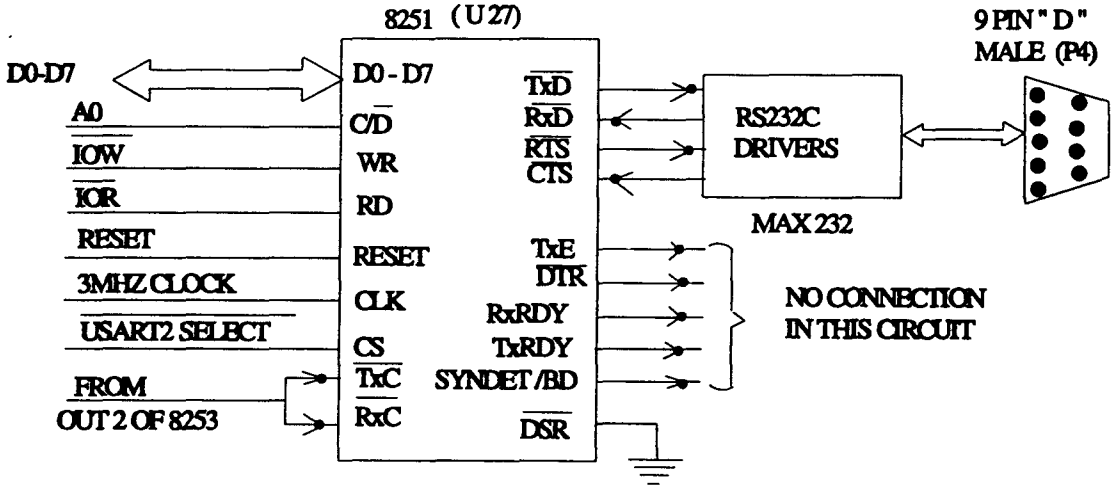
DSR Data set ready, indicates that the dsr is at zero level.

BASIC BLOCK DIAGRAM OF 8251 INTERFACE



USART 1 SELECT IS THE CS TO 8251 (U16) WHICH SELECTS FOR I/O ADDRESS 08&09

BASIC BLOCK DIAGRAM OF 8251 INTERFACE



USART2 SELECT IS THE CS TO 8251 (U27) WHICH SELECTS FOR I/O ADDRESS 1C & 1D

One important factor to be noted is that the Intel corporation is not giving guarantee for a good reception in case of a 1 Baud factor.

Then the Command instruction gives rise to a command word of 37 when initialised to

Reset Error flags,
Enable transmission and reception
Make RTS and DTR active low.

After initialising, the 8251 can be used for serial transmission or reception. It has an inbuilt buffer for both transmission and reception. The Status Register in the 8251 allows the user to check both buffers for empty by reading its contents.

3.3.4 CIRCUIT IMPLEMENTATION

The simple block diagram to describe the interface between the USART and the microprocessor is given in Figure 3.5.

As seen from the block diagram the D0-D7 lines are connected to the Processor data bus. The RD and WR are connected to the IOR and LOW respectively. A 3 MHz clock is given to the CLK input. The active high RESET is given to the Reset input of the 8251. The A0 is connected to C/D of 8251. So, when A0 is high, then it is Control and if it low, it is Data. The CS is given the USARTSELECT which goes low for addresses 08 and 09 for the 8251 (U16) and for addresses IC for the 8251 (U27), (U16).

The outputs of 8251 are brought out through Max 232 which are RS232C driver chips. The outputs of the driver ships are brought out at connector P3 and P4 for COM1 and COM2 respectively.

The timer 8253 provides the Baud clock required for 8251 to transmit and receive data serially. Hence before initialising 8251, of 8253 has to be done to ensure proper serial communication.

The clock input to 8253 is 1.5MHz. To get a 9600 Baud rate on 8251, the clock to 8251 should be 150 16x is selected for 8251. Hence to initialise 8253 ti give a clock frequency of 150 Khz, the counter of 8253 is to be loaded with a count which is derived as,

	Clock input to 8253	=	1500 Khz	
	Output clock required	=	150 Khz.	
Then,				
	Count	=	1500/150	= 10 or 0Ah

The example program here must be checked using a 9-pin D Female connector which must be placed on the 9-pin D Male connector already available on the Micropower-i, which is the serial port connector. Now, do the following connections in the 9-pin D Female connector, also called loop back connector. This loop back connector is to loop back the serial data transmitted to the receive buffer of the 8251, so that the data transmitted can be read back without having another system to check the seal port. The connections for the loop back connector are,

1	--->	NC	6	--->	NC
2	--->	TXD	7	--->	
3	--->	RXD	8	--->	
4	--->	NC	9	--->	NC
5	--->	NC			

These connections will short RTS with CTS and TxD with RxD which will enable the receive buffer at the instant of transmission and the data transmitted will be received by the same 8251 and status will be set accordingly so that CPU can read the data from the 8251.

The I/O address for the two USART's are as follows:

IC NO.	FUNCTION	ADDRESS
U16	Control / Status	09
U16	Data	08
U27	Control / Status	0B
U27	Data	0A

3.3.5 EXAMPLE 2 - TRANSMITTING AND RECEIVING A CHARACTER

OBJECTIVE: To initialise 8253 and 8251 and to check the transmission and reception of a character.

THEORY: The program first initialises the 8253 to give an output clock frequency of 150KHZ of Channel 0 which will give a 9600 baud rate of 8251. Then the 8251 is also initialised as said above with data 4E and 37. The program, after initialising, will read the status register and check for TxEMPTY. If the transmitter buffer is empty then it will send 41 to the serial port and then check for a character in the receive buffer. If some character is present then, it is received and stored at location 4200. If the serial port is not proper the program will be in a constant loop either in the transmission mode or in the reception mode.

PROGRAM:

```
          09 00          UATCNT   EQU      09
          08 00          UATDAT   EQU      08
          07 00          TMRCNT   EQU      07
          04 00          TMRCH0   EQU      04
                         ;Initialisation of 8253
4100      3E 36                   MVI   A,36H
4102      D3 07                   OUT   TMRCNT
4104      3E 0A                   MVI   A,0AH
4106      D3 04                   OUT   TRMRCH0
4108      AF                      XRA   A
4109      D3 04                   OUT   TMRCH0
                         ;Initialisation of 8251
410B      3E 4E                         MVI   A,4EH
410D      D3 09                   OUT   TMRCH0
410F      3E 37                   MVI   A,37H
4111      D3 09                   OUT   UATCNT
                         ;Check 8251s TxEMPTY
                         ;and then send the data
                         ;41 to 8251
4113                        LOOP:
4113      DB 09                   IN    UATCNT
```

```
4115    E6 04           ANI   04H
4117    CA 13 41        JZ    LOOP
411A    3E 41           MVI   A,41H
411C    D3 08           OUT   UATDAT
                LOOP1:
411E    DB 09           IN    UATCNT    ;Check 8251s RxRDY and
4120    E6 02           ANI   02H       ;then get the data and
4122    CA 1E 41        JZ    LOOP1     ;and store at 4200
4125    DB 08           IN    UATDAT
4127    32 00 42        STA   4200
412A    76              HLT
```

VERIFICATION :

Write the above program in location 4100H. Short the pins of 2 and 3 (RXD0 and TXD0), 7 and 8 (RTS0 and CTS0) in 9 pin 'D' male connecter 'P3' and execute the program.

After executing the program, check at location 4200H using substitute command. If the contents of this location is 41 then the serial port is OK. And verify RTS, CTS, TxD and RxD signals are using as oscilloscope.

3.3.6 EXERCISES

i. Write a program to transmit data 00 to FF and receive them back and store at address starting from 4500. So, after executing the program, check at locations starting from 4500 and check whether it contains 00 to FF continuously.

ii. Change Baud rate to 4800 and repeat (i).

iii. Describe the various signals used in RS232C communications.

3.4 PROGRAMMABLE PERIPHERAL INTERFACE (8255): (U5 & U6)

3.4.1 COMPONENT

The Intel 8255 is the one used here to provide the parallel interface for the Micropower-i. Two such numbers are available providing a total of 48 I/O lines of user interface.

3.4.2 COMPONENT DESCRIPTION

The 24 I/O lines of each 8255 can be programmed in two groups of 12 and used in three major modes of operation. Its unique features may be as,

24 programmable I/O pins
Improved timing characteristics
Direct bit set/reset capability
Reduces system package count

The control pins with which the CPU communicates to 8255 are the RESET, CS*, RD*, WR*, A0, A1, D0-D7. Its basic operation can be given as

A1	A0	RD*	WR*	CS*	FUCTION
0	0	0	1	0	INPUT OPERATION (READ) PORT A=> DATA BUS
0	1	0	1	0	PORT B=> DATA BUS
1	0	0	1	0	PORT C=> DATA BUS
0	0	1	0	0	OUTPUT OPERATION (WRITE) DATA BUS=> PORT A
0	1	1	0	0	DATA BUS=> PORT B
1	0	1	0	0	DATA BUS=> PROT C
1	1	1	0	0	DATA BUS=> CONTROL
X	X	X	X	1	DISABLE FUCNTION DATA BUS=> 3 - STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS=> 3 - STATE

3.4.3 PROGRAMMING DESCRIPTION

The 8255 offers three modes of operation, that can be selected by writing control words appropriately. The three modes are,

MODE 0 - Basic Input / Output
MODE 1 - Strobed Input / output
MODE 2 - Bi-Directional Bus

The 24 I/O lines of the 8255 are organised as three ports A, B and C each having eight I/O lines. The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions.

The Figure 3.6, illustrates the operation of 8255 in all the three modes. From the figure it is understood that in Mode 0 all the ports can be configured either as input or output port. Hence this mode can be used for almost all applications.

The Figure 3.7. shows the mode definition format for 8255. Using this the mode control word is configured according to the requirement and can be written into the control register.

For instance to have,
APORT as Input Port
BPORT as Output Port
CPORT higher nibble as Input Port
CPORT lower nibble as Output Port,

then the mode control word will be **9C**

SINGLE BIT SET / RESET FEATURE

Any of the eight bits of Port C can be set or reset using a single OUTput instruction, This feature reduces software requirements in Control-based applications. The Bit Set / Reset Format is given in Figure 3.8. When Port C is being used as status / Control for Port A or Port B, these bits can be set or reset using the Bit Set / Reset operation just as if they were data output ports.

3.4.4 CIRCUIT IMPLEMENTATION

The Figure 3.9 gives a clear idea of the interface between the 8255 and the 8085 microprocessor.

As seen from the block diagram the D0-D7 lines are connected to the processor data lines. The RESET, IOR and IOW lines of the circuit respectively. The CS is got from 8255 SELECT which goes low for I/O addresses 0C to 0F for 8255 (U5) and 10 to 13 for 8255 (U6).

The following are the I/O addresses for 8255

IC NO.	FUNCTION	ADDRESS
U5	Control Register	0F
U5	Port A	0C
U5	Port B	0D
U5	Port C	0E
U6	Control Register	13
U6	Port A	10
U6	Port B	11
U6	Port C	12

BASIC MODE DEFINITION 8255

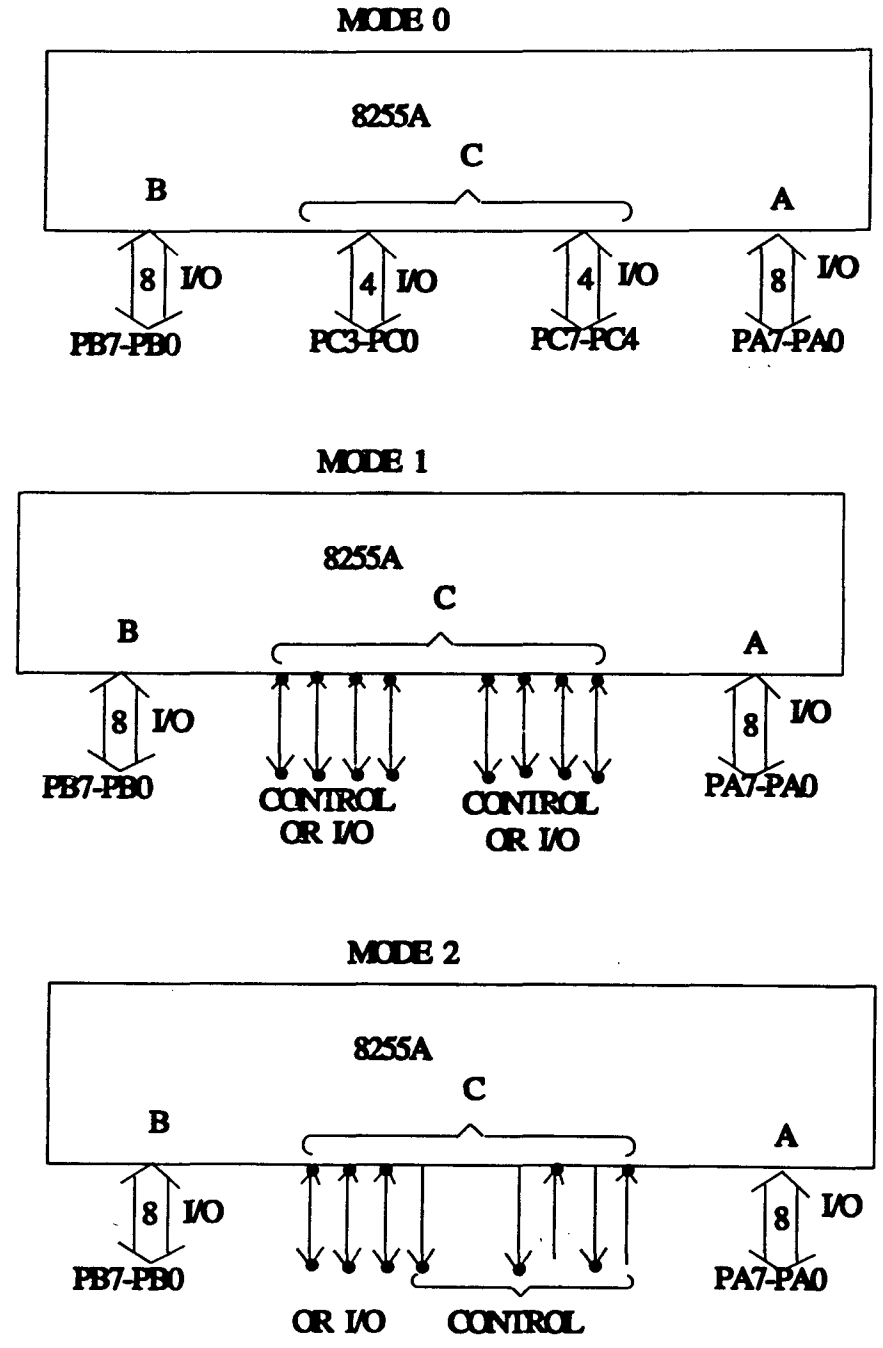


Figure - 3.6

MODE DEFINITION FORMAT 8255

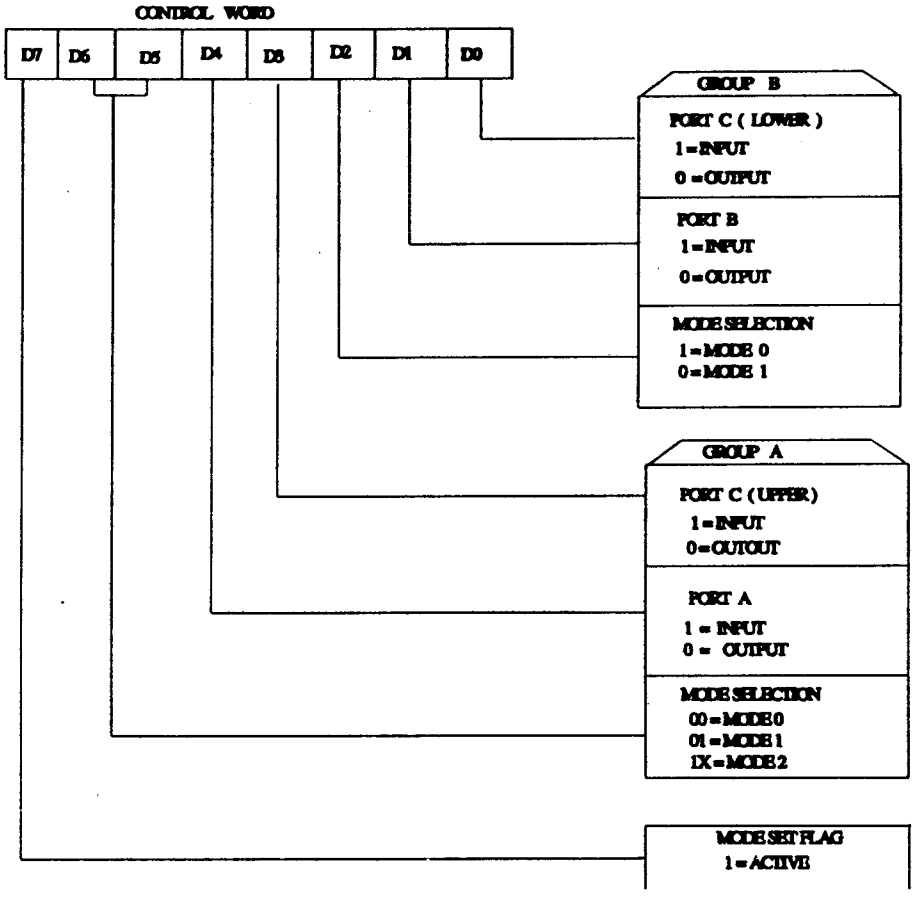


Figure 3.7

BIT SET/RESET FORMAT 8255

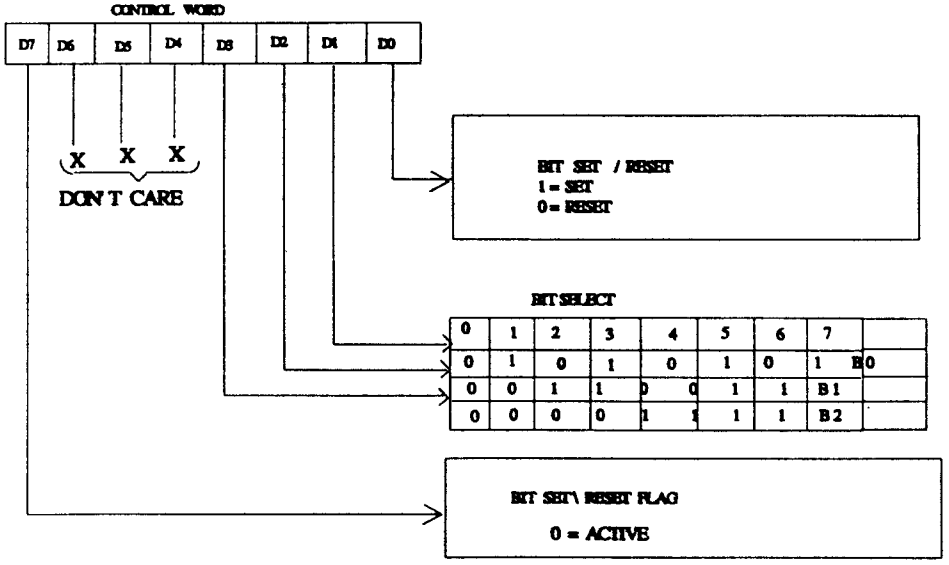
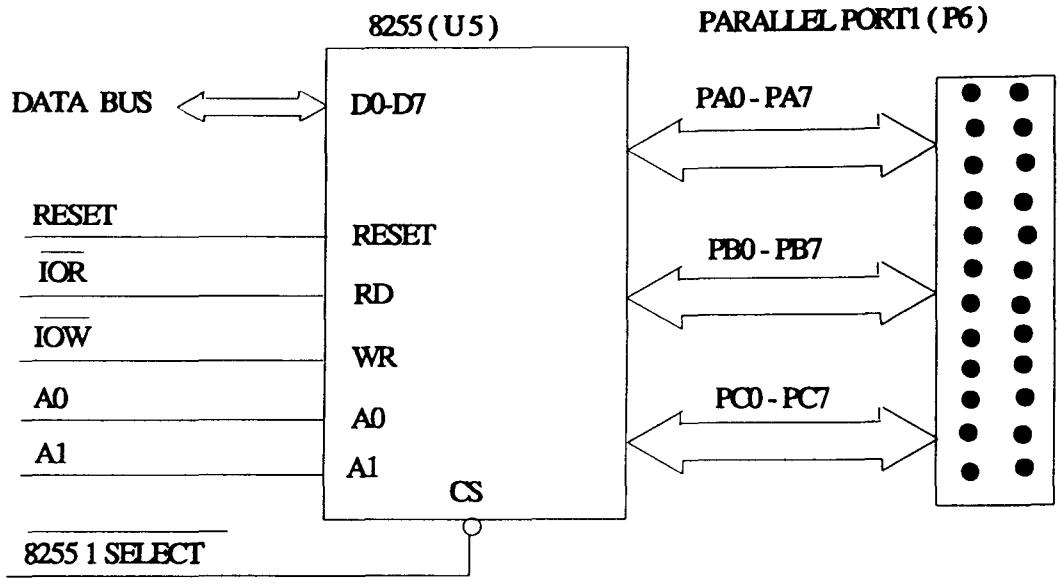
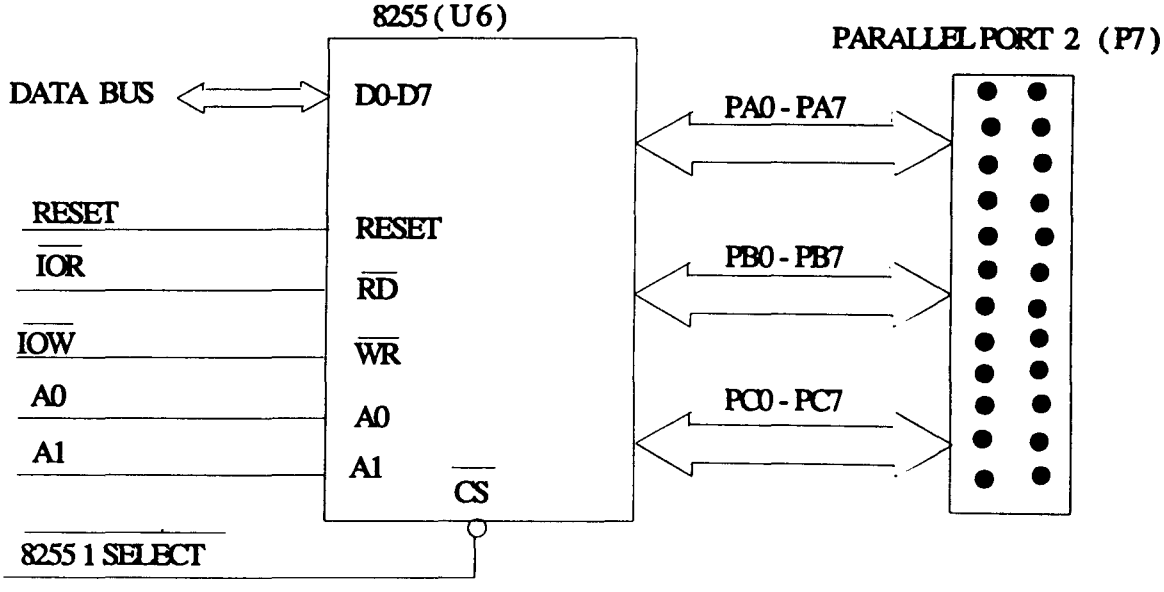


Figure 3.8

BASIC BLOCK DIAGRAM OF 8255 INTERFACE



8255 1 SELECT IS THE CS TO 8255 (U5) WHICH
SELECT FOR I/O ADDRESS OC TO OF



8255 1 SELECT IS THE CS TO 8255 (U6) WHICH
SELECT FOR I/O ADDRESS 10 TO 13

3.4.5 EXERCISE 3 - SQUARE WAVE GENERATION (U5)

OBJECTIVE: To generate a square wave at all I/O lines of 8255. The frequency of the square wave depends solely on the time delay routine used.

THEORY: To generate a square wave of 8255 first initialise all the ports of the 8255 as output ports by writing the corresponding control word. Then make the 24 I/O lines high and low with a certain delay and you will get a Square wave out of 8255. If you don't give any delay then you cannot see any square wave because of the speed at which the 8255 output lines change states from high to low and low to high. Increase the delay to reduce the frequency and reduce the delay to increase the frequency.

PROGRAM:

```
          0F 00       CT8255    EQU     0F
          0C 00       PA8255    EQU     0C
          0D 00       PB8255    EQU     0D
          0E 00       PC8255    EQU     0E
4100      3E 80                 MVI     A,80H
4102      D3 0F                 OUT     CT8255
4104      3E FF                 MVI     A,FFH
4106      D3 0C       LOOP:     OUT     PA8255
4108      D3 0D                 OUT     PB8255
410A      D3 0E                 OUT     PC8255
410C      CD 13 41              CALL    DELAY
410F      2F                    CMA
4110      C3 06 41              JMP     LOOP
4113      F5          DELAY:    PUSH    AF
4114      21 FF FF              LXI     H,FFFFH
4117      00          DELAY1:   NOP
4118      2B                    DCX     H
4119      7D                    MOV     A,L
411A      B4                    ORA     H
411B      C2 17 41              JNZ     DELAY1
411E      F1                    POP     AF
411F      C9                    RET
```

VERIFICATION: After entering the program given above and executing it, verify for the square wave at the output lines of the 8255. Calculate the frequency for the delay used. Change delay and verify whether the frequency changes or not.

EXERCISES:

i. Configure 8255-U5 as output and 8255-U6 as input ports. Connect a 26-core flat cable between the two connectors as shown in the block diagram. Output data to 8255-U5 and check them by inputting through 8255-U6.

ii. Using our PLC add-on board, write a program that will generate a sequential logic display on PLC through 8255-U6.

1. Output 01 to Port A, give a delay of 2 seconds.
2. Output 02 to Port A, give a delay of 3 seconds.
3. Output 04 to Port A, give a delay of 5 seconds.
4. Output 08 to Port A, give a delay of 4 seconds.
5. Repeat again from Step 1.

3.5 PROGRAMMABLE INTERRUPT CONTROLLER (8259): (U52)

3.5.1 COMPONENT

The Interrupt Interface of Micropower-i comprises of the Intel Programmable Interrupt controller 8259.

3.5.2 COMPONENT DESCRIPTION

The 8259 can handle upto eight vectored priority interrupts and is also cascadable upto 64 vectored priority interrupts without additional circuitry using 8 No. of 8259.

Its distinctive characteristics may be briefed as,

i) Eight-level Priority controller
if) Expandable to 64 levels
iii) Programmable Interrupt Modes
iv) Individual Request Mask capability
v) Software Programmable capability of all Modes of Interrupt Servicing.

The 8259 is designed to minimize the software and real time overhead in handling multi-level priority interrupts. It has several modes, permitting optimization for a variety of system requirements. It also allows priority specification and rotating priority.

The 8259 interacts with the CPU by D0-D7, CS*, RD*, WR*, A0, INT, INTA. The other pins which control its interrupt handling are,

CAS0-CAS2 - CASCADE Lines
Private 8259 bus to control a multiple 8259 structure. These are outputs for a master and inputs for a slave 8259.

SO / EN* - INTERRUPT REQUESTS.

IR0 - IR7 - An interrupt request is executed by raising an IR input high. It can be either Edge or Level Triggered Mode.

3.5.3 PROGRAMMING DESCRIPTION

The powerful features of 8259A in a microcomputer system are its programmability and the interrupt routine addressing capability. The latter allows direct or indirect jumping to the specific interrupt routine requested without any polling of the interrupting devices. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an MCS-80/85 system:

If one or more of the Interrupt Requests are raised high then the 8259A sends an Interrupt to the CPU.

The CPU acknowledges the interrupt of an INTA*.

Now the highest priority interrupt is set and the corresponding IR bit reset. It also releases a CALL instruction code (11001101) onto the data bus.

This will initiate two more INTA pulses from the CPU.

The 8259 will now release its preprogrammed subroutine addresses on the data bus with the LSB first followed by the MSB.

This completes the 3-byte CALL instruction release by the 8259.

After initialisation the 8259A will enter into "FULLY NESTED MODE" unless another mode is programmed. The interrupt requests are ordered in priority for 0 through 7 (0-highest). When an interrupt is acknowledged, the highest priority request is determined and its vector placed on the bus. Additionally, a bit of the Interrupt Service Register (ISO-7) is set. This bit remains set until the Microprocessor issues an End Of Interrupt (EOI) command immediately before returning from the service routine, or if AEOI (Automatic End of Interrupt) bit is set until the trailing edge of the last INTA.

While the IS bit is set all further interrupts of same or lower priority are inhibited. Higher priority interrupts will generate and interrupt and receive an Acknowledge if the Processor's Interrupt Enable Flip Flop is enabled.

The 8259 accepts two types of command words generated by the CPU.

i) INITIALISATION COMMAND WORD (ICW)

Whenever a command is issued with A0=0 and D4=1 it is interpreted as ICW1. Before normal operation can begin each 8259 in the system must be initialised using 2 to 4 bytes of ICWs.

ii) OPERATION COMMAND WORD (OCW)

These are command words which command the 8259 to operate in various interrupt modes. These modes are,

a) Fully-Nested Mode
b) Rotating-priority Mode
c) Special Mask Mode
d) Polled Mode.

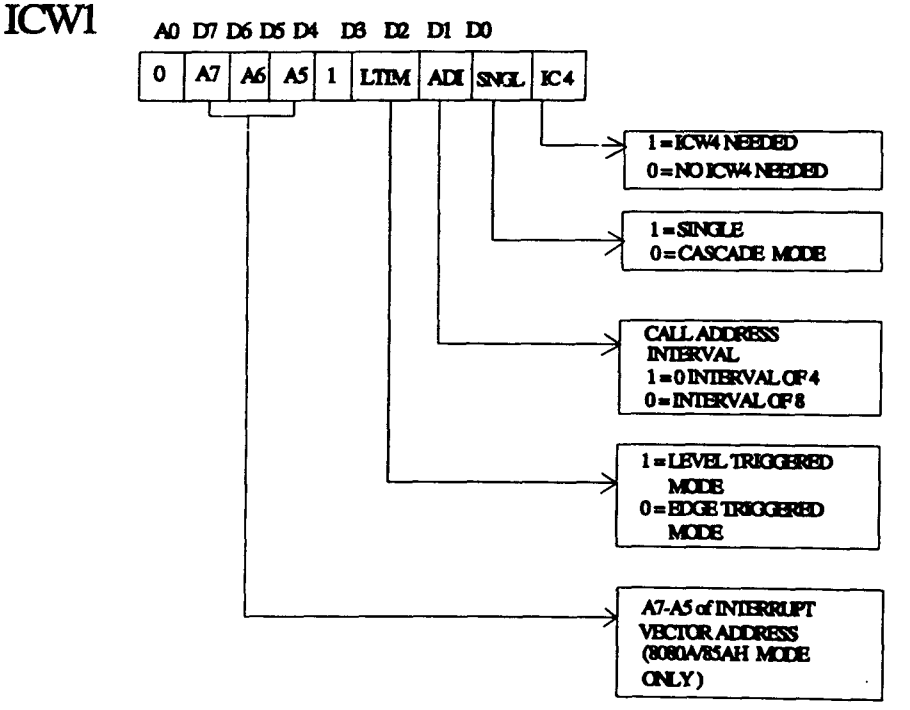
The OCWs can be written into the 8259 any time after initialisation.

INITIALISATION

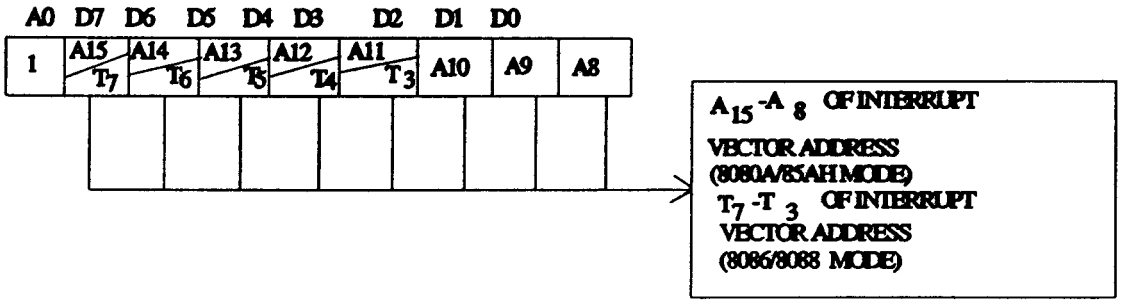
The key for using the PIC is to perform the proper initialisation sequence for the desired operating mode.

Figure 3.10 defines the Initialisation Command word format.

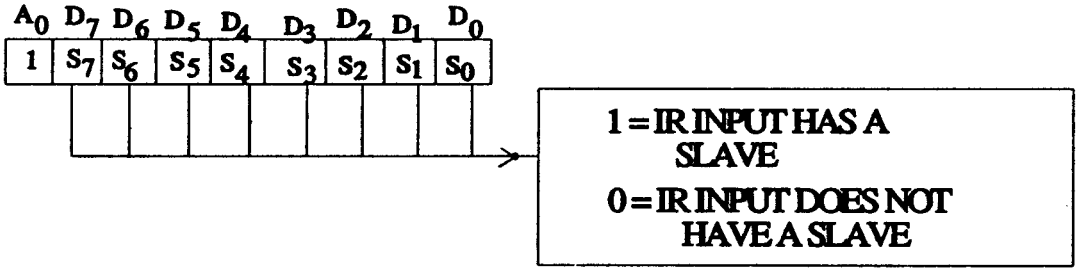
INITIALIZATION COMMAND WORD FORMAT (8259A)



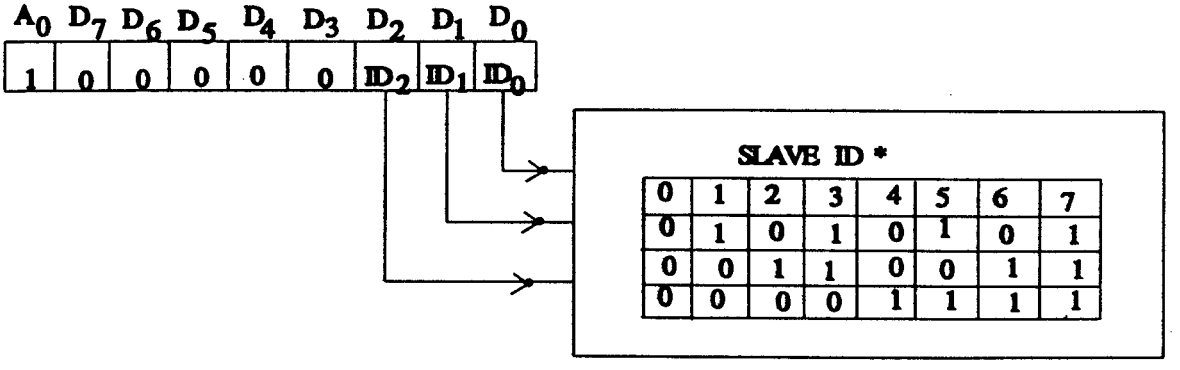
ICW2



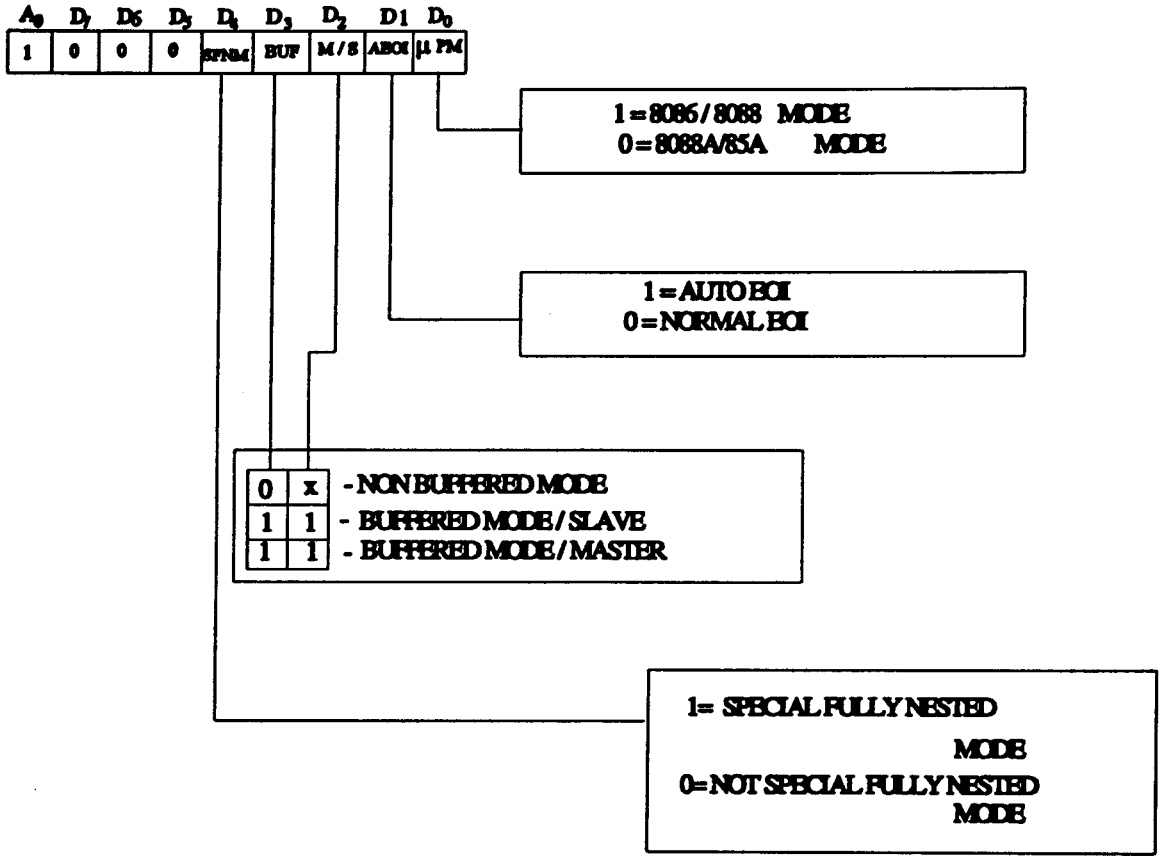
ICW3 (MASTER DEVICE)



ICW3 (SLAVE DEVICE)



ICW4



* - SLAVE ID = CORRESPONDING MASTER IR INPUT

PROGRAMMING THE INITIALISATION COMMAND REGISTERS

ICW1

This word is written to port 18. Bits D7 - D5 are made zero here and D2=1. D3=0 for Edge Triggering. If Level Triggering Mode is programmed, then it must be removed after reception of acknowledge pulses from the CPU to avoid multiple interrupt requests. D1=1 for any one 8259 is used; D0 can be either 0 or 1. Let it be = 1.

Therefore, IC1=0001 0111B (Binary) = 17

ICW2

This is written to port 19. This specifies the A15-A8 of interrupt Vector address. The address here is 1F and hence,

ICW2 = 0001 1111 B (Binary) = 1F

ICW3

This word is used only in cascaded 8259 systems. This word is accepted only if SNGL = 0 in ICW1.

ICWR

This word is also output to port 19. Let D0=0, D1=1 for Automatic End of Interrupt (AEOI). D2 & D3 specify buffered mode. D4=0 for Fully Nested Mode.

Therefore, ICW4 = 0000 0010 B (Binary) = 02

PROGRAMMING THE OPERATION CONTROL REGISTERS

After the three (four) initialisation control words have been written, the PIC is ready to receive interrupts on IRO-IR7 and will operate in the fully nested mode. To specify the rotating priority modes, the special mask mode, the polled mode, the interrupt mask, or the EOI commands requires that the operation control registers (OCWs) be programmed.

Figure F3.11 defines the Operation Command Word format.

OCW1

This word is written to or read form port 19. It is used to mask the Interrupts.

Therefore, OCW1 = 0000 0000 B = 00

OCW2

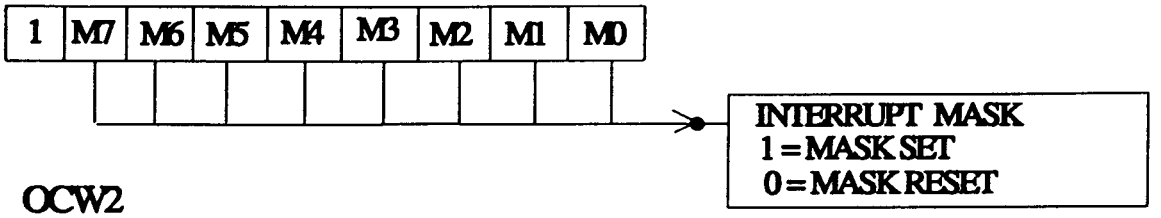
This is written toport 18 and is used to specify the EOI command to the PIC. This word is not used as "AEOI" is selected, in ICW4.

OCW3

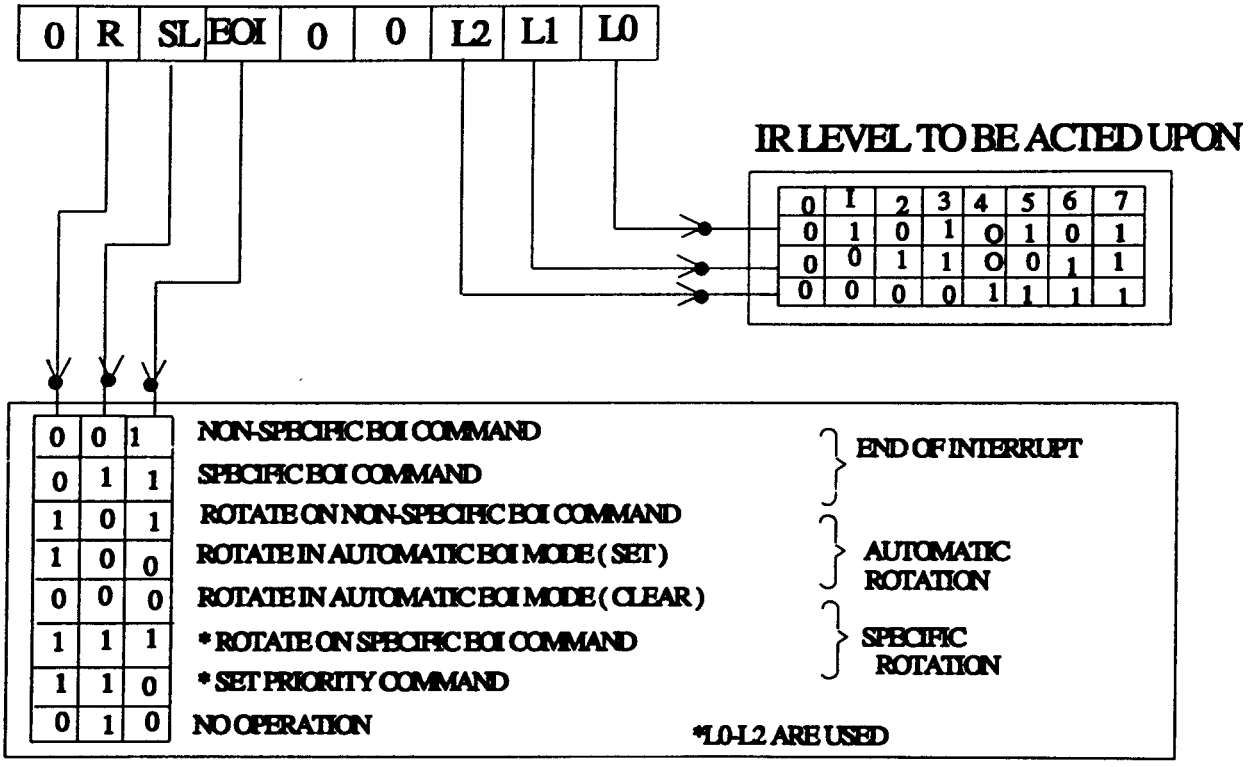
This is written to port 18. Here D3=1 and D4=0 to distinguish from ICW1 and ICW2. D5 & D6 allow special mask mode to be programmed. D2=1 for Polled Mode. This word is not used.

OPERATION COMMAND WORD FORMAT (8259)

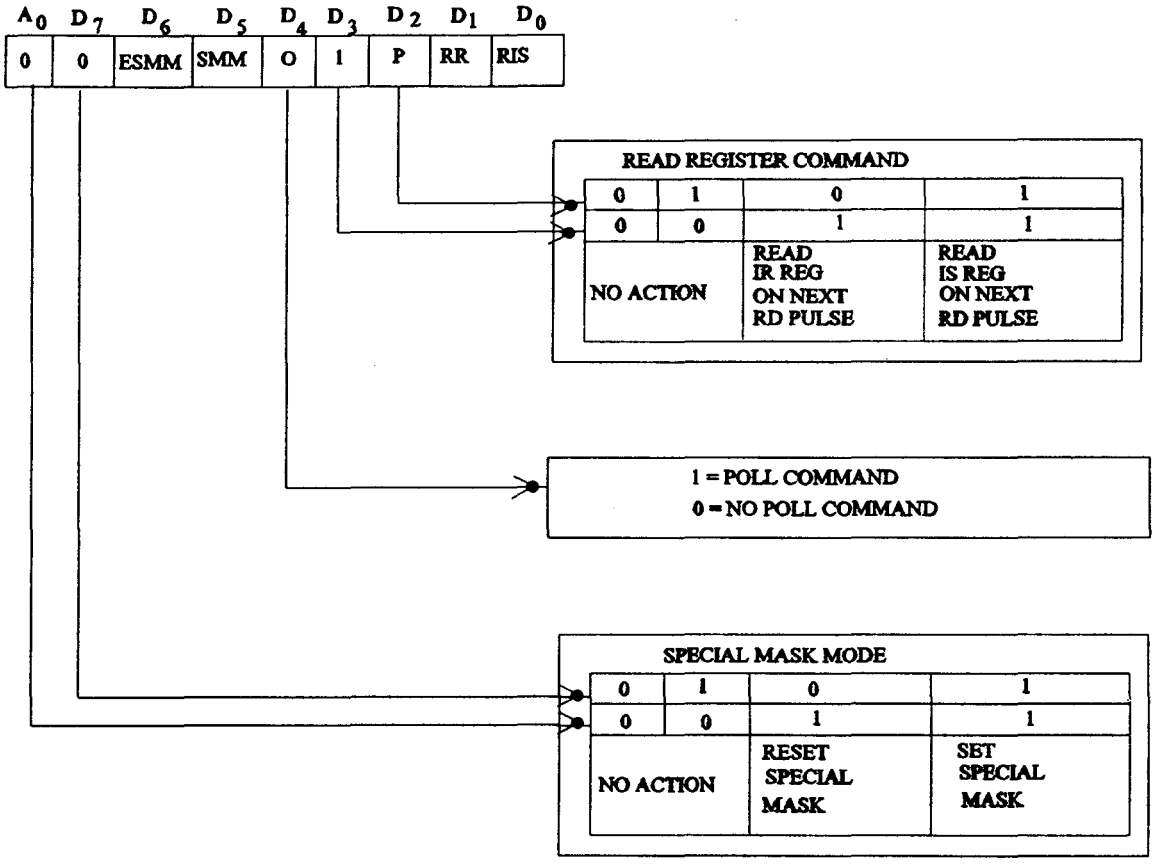
OCW 1



OCW2



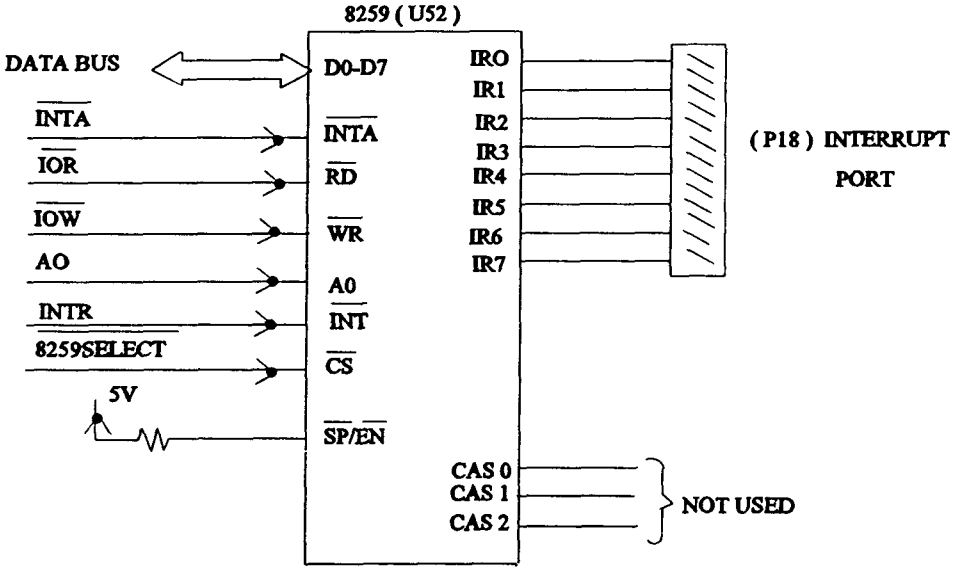
OCW3


A0 D7 D6 D5 D4 D3 D2 D1 D0
0 0 ESMM SMM O 1 P RR RIS
READ REGISTER COMMAND
0 1 0 1
0 0 1 1
NO ACTION
READ IR REG ON NEXT RD PULSE
READ IS REG ON NEXT RD PULSE
1 = POLL COMMAND
0 = NO POLL COMMAND
SPECIAL MASK MODE
0 1 0 1
0 0 1 1
NO ACTION
RESET SPECIAL MASK
SET SPECIAL MASK

3.5.4 CIRCUIT IMPLEMENTATION

This Figure 3.12 given below gives a clear idea of the interface between the 8259 and the microprocessor.

BASIC BLOCK DIAGRAM OF 8259 INTERFACE



8259 SELECT IS THE CHIP SELECT TO 8259 WHICH SELECTS FOR I/O ADDRESS 18 & 19

As shown in the block diagram, the D0-D7 lines are connected to microprocessor data bus. The RD,WR,A0, INT, INTA lines of 8259 are connected to the IOR, IOW, A0, INT, INTA lines of the circuit respectively. The IR0-IR7 interrupt request levels are given to a 10-pin connector for user interface. The CAS0-CAS2 lines are not used for this circuit consists of only a single 8259.

The I/O address for the 8259 are as follows

IC	FUNCTION	ADDRESS
U52	ICW1	18
U52	ICW2	19
U52	ICW3	19
U52	ICW4	19
U52	ICW1	19
U52	ICW2	18
U52	ICW3	18

As you have learned the organisation and programming of the Programable Interrupt Controller, let us write a sample program to verify it.

3.5.5 EXAMPLE - 4 - INTERRUPT GENERATION

OBJECTIVE:

To write a sample program to initialise the Programmable Interrupt Controller 8259 and write its interrupt service routine.

PROGRAM:

;Program to initialise 8259

```
4100  3E 17       MVI  A,17H   ;ICW1
4102  D3 18       OUT  18H
4104  3E 42       MVI  A,42H   ;ICW2
4106  D3 19       OUT  19H
4108  3E 00       MVI  A,00H   ;ICW4
410A  D3 19       OUT  19H
410C  3E 00       MVI  A,00H
410E  D3 19       OUT  19H
4110  00          NOP
4111  76

4200  C3 00 45    JMP  4500H
```

;8-bit Addition sub- routine

```
4500  3E 05       MVI  A,05H
4502  06 05       MVI  B,05H
4504  80          ADD  B
4505  32 00 50    STA  @5000H
4508  CF          RST  1
```

VERIFICATION:

After executing the program at 4100, interrupt the 8259 using interrupt port pins (P18) connector. Now addition sub- routine will execute and result '0A' data is stored in 5000H location.

3.5.6 EXERCISES

i. Write a program to set the 8259 in the rotating priority mode and mask of the interrupts below level 5. Check what happens when Level 4 is requesting service.

ii. What is the difference between edge triggered and level triggered interrupts?

3.6 LIQUID CRYSTAL ALPHANUMERIC DISPLAY INTERFACE

3.6.1 COMPONENT

In Micro power (i), Liquid crystal alphanumeric display module arranged as 2 lines of 16 characters or lines of I/O characteristics are used as the output device.

3.6.2 COMPONENT DESCRIPTION

The display module includes a CMOS VLSI Microprocessor controller which supplies character memory RAM, character generator RAM and ROM, and all refresh, control and timing signals.

3.6.3 BASIC SYSTEM INTERFACE

Interfacing the LCD module with a microprocessor bus requires minimum hardware. We have to provide only +5VDC and parallel Ttl level ASCII in order to popular microprocessor with a minimum of additional hardware and allows such control functions as cursor home, cursor addressing, dislay shift and programmable characters to be easily implemented.

The control signals with which the LCD module communications with the CPU are the data bus D0 - D7, RS, R/W and E.

The following table shows the LCD interface pin connections.

INTERFACE PIN CONNECTIONS

PIN NO.	SYMBOL	FUNCTION
1	Vcs	Power Supply - ground
2	V_{DD}	Power supply +5VDC ± 0.25V
3	Vo	LCD Driving voltage (viewing angle adjustement)
4	RS	Set high to READ and WRITE alphanumeric data and character generate data. Set low to READ status and address information and to WRITE control words.
5	R/$\overline{W}$	READ/$\overline{WRITE}$ set low to WRITE to he module. Set high to READ from hte display.
6	E	Enable pulse. Data is clocked into the DAYSTAR module on the trailing edge of this pulse. Valid busy flag data appears on data bit 7, a maximum of 320 ns after the leading edge of this pulse and is valid for 20 ns after the leading edge after the trailing edge.
7 8 9 10	DB0 DB1 DB2 DB3	Lower order 4 lines of bi-directional tri-state data bus. Used for data transfer between the MPU and the display. DB7 can be used as a BUSY flag.
11 12 13 14	DB4 DB5 DB6 DB7	Higher order 4 lines of bi-directional tri-state data bus. Used for data transfer between the MPU and the display. DB7 can be used as a BUSY flag.

3.6.4 PROGRAMMING DESCRIPTION

The following table briefs all the commands and its format accepted by LCD.

DISPLAY CONTROL INSTRUCTIONS

INSTRUCTION	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	DESCRIPTION
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears display and returns the cursor to the home position. (Address 0)
Return Home	0	0	0	0	0	0	0	0	0	1	Returns the cursor to the home position (Address 0). Also returns shifted display to home poistion. DD RAM contents remain unchanged
Entry Mode Set	0	0	0	0	0	0	0	0	1	I/D S	Sets the cursor movement direction and specifies wheather or not to shift the display. These operations are performed during data write and read.
Dispaly ON/OFF Control	0	0	0	0	0	0	0	1	D	C B	Sets display ON/OFF (D) cursor ON/OFF (C) and blink of character at cursor position (B).
Cursor or Display Shift	0	0	0	0	0	0	1	S/C	R/L	* *	Sets data bus length (DL), number of display lines (N) and character font (f).
Function Set	0	0	0	0	0	1	DL	N	F	* *	Sets data bus length (DL), number of display lines (N) and character font (F).
Set CG RAM Address	0	0	0	1	|-----A_{CS}----------|						Sets the CG RAM address. CG RAM data is sent and received only following this command.
Set DD RAM	0	0	1	|---------A_{DD}----------|							Sets the DD RAM address. DD RAM data is sent and received only following this command
Read Busy Flag Address	0	1	BF	|--------AC----------|							Reads Busy flag (BF) indicating that internal operation is being performed; reads address counter (AC) content
Write data CG or DD RAM	1	0	|----------Write----------| Data								Wries data into DD RAM or CG RAM and increments (or decrements) AC.
Read Data from CG or DD RAM	1	1	|----------Read----------| Data								Reads data from DD Ram or CG RAM and increments (or decrements) AC.

DETAILED DESCRITPTION OF COMMANDS

i. Clear Display

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	0	1

S

Writes ASCII "space" (20H) into all of the DD RAM. The cursor returns to address 0 (A_{DD} = "00") and the display, if it has been shifted, returns to the original to the original position.

ii. Return Home

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	1	*

* (Don't care)

Returns the cursor to address 0 (A_{DD} = "00") and display, if it has been shifted, to the original position. The DD RAM contents remain unchanged.

iii. Entry mode set

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	0	1	I/D	S

I/D: This mode automatically increments (I/D = 1) or decrements (I/D = 0) THE DD RAM address by one every time a character is written to or read from the DD RAM. (The cursor moves to the right when increment.) The same applies to writing abd reading of GG RAM.

S : Automatically shift the entire display to either the right or the left after each character is written, if S = 1; either to the left (when I/D = 1), or to the right (when I/D = 0). Therefore, the cursor looks as if it stood still and only the display moved. The display is not shifted when reading from the DD RAM, or if S = 0.

iv. Display ON/OFF Control
(Display Blanking)

	RS	R/W	DB_7						DB_0
Code	0	0	0	0	0	0	1	D	C	B

D : The display is turned ON when D = 1 and OFF when D = 0. When the display is turned OFF by D = 0, the display data remains in the DD RAM and it can be displayed immediately by setting D = 1.

C : The cursor is displayed as an undebar when C = 1 and not displayed when C = 0. Even

if the cursor is not displayed, the function of I/D, etc. Does not change during display data write. The cursor is displayed using 5 dots in either the 8th line when the 5x7 dot character font is selected or in 11th line when 5x10 dot character font is selected.

B : The character at the cursor position blinks when B = 1. The blink is done character at 0.4 second intervals. The cursor underbar and the blink can be set concurrently.

v. Cursor or Display shift

	RS	R/W	DB 7							DB 0
Code	0	0	0	0	0	1	S/L	R/L	*	*

* (Don't Care)

Shift the cursor position or display to the right and the left without writting or reading the display data. This function is used for correction or search of display.

S/C	R/L	
0	0	Shifts the cursor position to the left. (AC is decremented by one.)
0	1	Shifts the cursor position to the right. (AC is incremented by one.)
1	0	Shifts the entire display to the left. The cursor follows the display shift.
1	1	Shifts the entire display to eh right. The cursor follows the display shift.

vi. Function set

	RS	R/W	DB 7							DB 0
Code	0	0	0	0	1	DL	N	F	*	*

* (Don't Care)

DL : Sets interface data length. Data is sent or received in 8-bit length (DB7 through DB0) when DL = 1 and 4-bit length (DB7 through DB4) when DL = 0. When 4-bit length is selected, data must be written or read twice for each 8-bit byte of data.

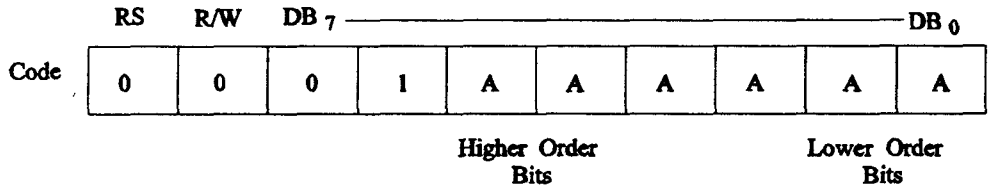
N : Sets number of display lines.

F : Sets character font.

N	F	No.of Display Lines	Character Font	Duty Factor	Remarks
0	0	1	5×7 dots	1/8	
0	1	1	5×10 dots	1/11	
1	*	2	5×7 dots	1/16	Cannot dispaly 2 lines with 5×10 dot character font.

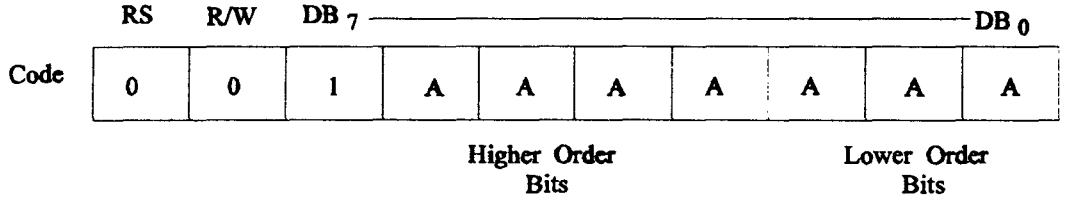
* (Don't care)

vii. Set CG RAM Address



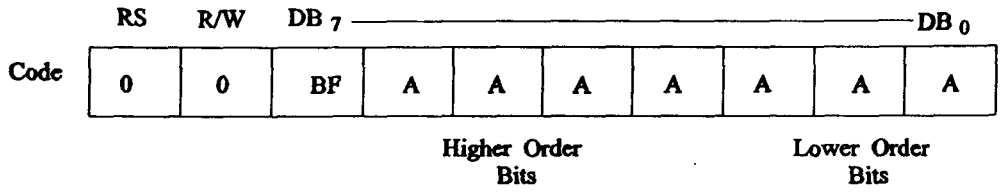
Sets the DD RAM address as a binary number "AAAAAAA" in the address counter. Data pertaining to the DD RAM is written or read from when N=0 (1 - line display), the value of "AAAAAAA" is "00" through "4F" (Hex). When N = 1(2 - line display), the value of "AAAAAAA" is "00" through "27" (Hex) for the first line, and "40" through "67" (Hex) for the second line.

viii. Set DD RAM Address



Sets the DD RAM address as a binary number "AAAAAAA" in the address counter. Data pertaining to the DD RAM is written or read from the host system after this command is executed. "AAAAAAA" is "00" through "4F" (Hex). When N = 1 (2 - line diaplay), the value of "AAAAAAA" is "00" through "27" (Hex) for the first line, and "40" through "67" (Hex) for the second line.

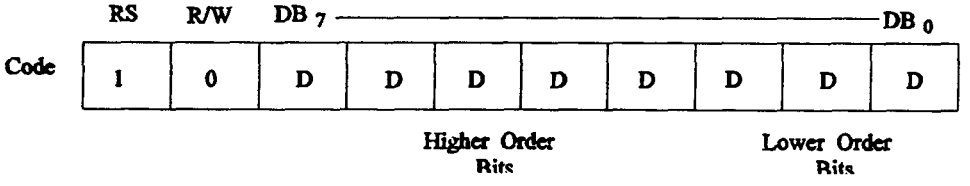
ix. Read Busy Flag and Address



Reads the busy Flag (BF), which indicates that the system is internally operating on an instruction received previously. When BF = 1, it indicates that internal operation is going on and the next instruction is not accepted until BF returns to "0". Check the BF status before the next write operation.

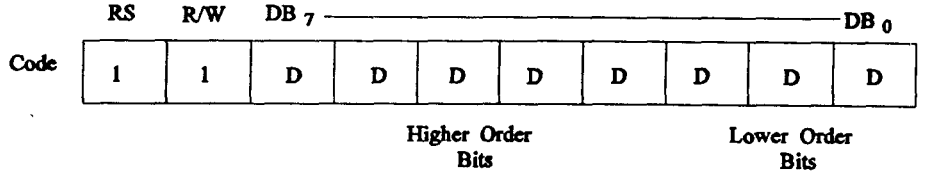
At the same time, this value of the address counter is expressed in a binary number "AAAAAAA". The address counter is usd for both CG an DD RAM address, and its value is determied by the previous instruction. Address contents are the same as items (7) and (8).

x. **Write Data to CG or DD RAM**



Writes binary 8 bit data "DDDDDDDD" to the CG or the DD RAM. Wheather the CG or the DD RAM is to be written is determined by the previous command. (CG RAM address setting or DD RAM address setting). After writing, the address is automitically incremented or decremented by one according to entry mode. Display shift also follows the entry mde.

xi. **Read Data from CG or DD RAM**



Reads binary 8-bit data "DDDDDDDD" from the CG or the DD RAM. Wheather the CG RAM or the DD RAM is to read is determined by the previous istruction. PRIOR to inputting read instructions, either the CG RAM address set instruction or the DD RAM address set instruction must be executed.

After reading data, the addressis automatically incremented or decremented by one according o the entry mde. However, display shift is not performed regardless ofentry mode setting.

CHARACTER GENERATOR ROM (CG ROM)

The character generator ROM generator ROM generates character patterns of 5x7 dots or 5x10 dots from 8-bit character codes. It can generate 160 5x7 dot character pattens and 32 5x10 dot character patterns. APPENDIX E shows the correspondence between codes and character codes and character patterns.

USER PROGRAMMABLE CHARACTER GENERATOR RAM (CG RAM)

The character generator RAM is the RAM to which hte user can write programmable character patterns. If the display uses 5x7 dots, 8 character patterns can be written; if it uses 5x10 dots, 4 may be programmed. Write the character coes listed in the left edges of APPENDIX E to display character patterns stored in CG RAM.

APPENDIX E shows relation between CG RAM address and data and display patterns. As shown in APPENDIX E, area that is not used for display can be used as general data RAM.

BUSY FLAG

When data or commands are written to the display module it requires a certain period to place the data in the correct location or to execute the desired command. The time required to complete this process and prepare for a new character or command is the appropriate execution time. The character load execution ime determnes the maximum speed in which characters can be input (the character loading rate). No data should be sent to the module until it has processed a previously entered character. To obtain the maximum character loading rate, the busy flag (Data Bit 7) should be read to determine when the module is ready to accept the next character.

Character loading can also be accomplished while ignoring rate. In this method, characters may be entered at a rate determined by the user to be less than the display module,s character loading rate.

PROGRAMMING SEQUENCE

From the above discussion it should be clear that the sequence involved in displaying characters in the LCD module requires proper loading of the following commands.

i. Function Set

ii. Clear Display

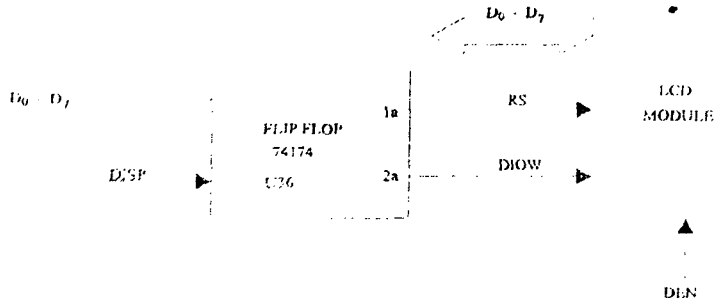
iii. Entry Mode Set

iv. Cursor or Display Shift

v. Set DD RAM Address

vi. Write Data to DD RAM.

3.6.5 CIRCUIT IMPLEMENTATION



DEN is the Enable pulse to LCD which selects for I/O address 30H. Read/Write (DIOW) and register select (RS) control lines of LCD are provided by the Flipflop (U_{36}) whose I/O address is 34H.

3.6.6 EXAMPLE 5 - DISPLAY A STRING IN LCD

OBJECTIVE: To initialize LCD and to display the string "VI MICROSYSTEMS" in the first row of the display.

THEORY: The LCD module be sent data to set the desired functions, before sending the characters for command from / to the module the BUSY flag must be read to determine when the module is ready to accept the next character.

PROGRAM:

```
                    ORG   5000
5000  CD 4C 50      CALL  @504C     ;Bsychk
5003  3E 00         MVI   A, 00     ;Make RS and DIOE low
5005  D3 3D         OUT   3D
5007  3E 38         MVI   A, 38     ;Function set
5009  D3 3C         OUT   3C
500B  CD 4C 50      CALL  @504C
500E  3E 00         MVI   A, 00
5010  D3 3D         OUT   3D
5012  3E 01         MVI   A, 01     ;Clear display
5014  D3 3D         OUT   3D
5016  CD 4C 50      CALL  @504C
5019  3E 00         MVI   A, 00
501B  D3 3D         OUT   3D
501D  3E 06         MVI   A, 06     ;Entry mode set
501F  D3 3C         OUT   3C
5021  CD 4C 50      CALL  @504C
5024  3E 00         MVI   A, 00
```

```
5026  D3 3D          OUT   3D
5028  3E 0C          MVI   A, 0C        ;Display Control
502A  D3 3C          OUT   3C
502C  CD 4C 50       CALL  @504C
502F  3E 00          MVI   A, 00
5031  D3 3D          OUT   3D
5033  3E 80          MVI   80           ;Set DDRAM address
5035  D3 3C          OUT   3C
5037  21 58 50       LXI   H, @5058     ;Lookup table for message
503A  0E 0F          MVI   C, 0F
503C  CD 4C 50       CALL  @504C
503F  3E 01          MVI   A, 01        ;Make RS  1, DIOW  0
5041  D3 3D          OUT   3D
5043  7E             MOV   A,M          ;Write data to DDRAM
5044  D3 3C          OUT   3C
5046  23             INX   H
5047  0D             DCR   C
5048  C2 3C 50       JNZ   @503C        ;Rept
504B  76             HLT
              BSYCHK:
504C  3E 02          MVI   A, 02        ;Make RS  0, DIOW  1
504E  D3 3D          OUT   3D
              BUSY:
5050  DB 3C          IN  3C             ;Check Busy flag
5052  E6 80          ANI   80
5054  C2 50 50       JNZ   @5050        ;BUSY
5057  C9             RET
                     END
```

Enter the following lookup table (ASCII data for the message VI MICROSYSTEMS) starting from location 5058 using SU command.

Lookup table 5058: 56, 69, 20, 4D, 69, 63, 72, 6F, 73, 74, 65, 6D, 73.

VERIFICATION

Enter the program and the lookup table. Execute the program and see that the execute the message "VI MICROSYSTEMS" is displayed.

3.6.7 EXERCISES

i. Display a rolling message "You and I are friends" using display shift option.

ii. Blink the message "Hello".

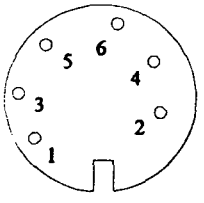
iii. Display the symbol " "
(Hint: Use character generator RAM).

3.7 IBM PC KEYBOARD INTERFACE

The IBM PC keyboard Interface is centered around a 5-pin DIN connector used for connecting the 84-keys IBM PC keyboard or 101-keys IBM PC AT/XT keyboard. The keyboard is operated in XT mode.

3.7.1 BASIC OPERATION

The configuration of 6-pin P32 connector provided in the IBM PC keyboards is given in figure F3.21.KVcc and GND are usually connected to system Vcc and Gnd. The RST pin is not connected to any of the system signals.



1 - NC
2 - KB DATA
3 - KBVcc
4 - GND
5 -Nc
6-KBCHL

Fig F 3.21 IBM PC keyboard Connector Configuration

The clock is given by the keyboard. And the data provided by the keyboard is synchronous with the clock. The relationship between the two is given in figure.

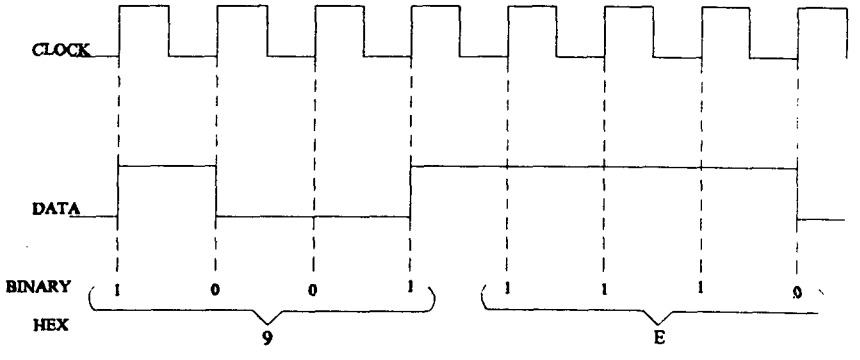
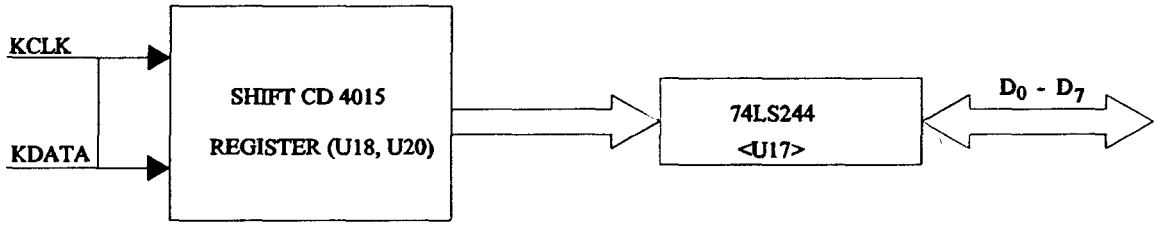


Fig - 3.22 Clock and Data Synchronization

As seen from the Figure every bit is synchronized at the raising edge of the clock. So using a form and this also generates an interrupt when 8-bits have been shifted. Now this 8-bit data is latched and can be read for further provessing.

3.7.2 CIRCUIT IMPLEMENTATION

The simplified block diagram of keyboard interface is given in figure F3.23



Refer circuit diagram for more details. IC 4015(U18, U20) is configued as serial to parallel shift register with tristate outputs. The KBCLK signal from the keyboard is delayed and inverted using two flipflops in CD4015. The PCLK signal is the clock input to CD4015.

The KBDATA signal from keyboard is connected to D1 input of the shift register.

The shift register scans the input D1 (pin 17) for each clock input (pin 11). The bit 0 or 1 at this time is put inside the shift register in the Q_A position. When next clock comes, the new data bit is put in Q_A position and the old data bit is moved from Q_A to Q_B. As each new data bit is received, the already received bits are shifted right.

To read the latch 74LS244, the scan code is F04, then next data is valid for key press.

TABLE

IC No	Function	Address
U18, U20, U17	Keyboard data buffer	00

PROGRAMMING DESCRIPTION

When a key is pressed in the IBM PC keyboard, it sends the data serially in synchronization with the clock signal. The keyboard interface provided in the trainer shifts the data from the keyboard and forms 8 bit byte. When the parallel data is ready, it sets the KBINT signal high. The data can then be read from the keyboard buffer.

3.7.3 PROGRAMMING SEQUENCE

To read a key from keyboard, do the following.

i. Clear the keyboard interrupt if any.

ii. Poll the KBINT line.

iii. If KBINT is high, then read the scan code and store it in memory.

3.7.4 EXERCISE

i. Write a program to input keys from the keyboard and display the corresponding characters in the LCD.

3.8 PRINTER INTERFACE: (U11, U19, U22)

3.8.1 COMPONENT

The implementation of the printer logic is done here using a 74LS273 as the 8-bit Output latch, a 74LS174 as the 4-bit control latch and a 74LS244 as the 5-bit status buffer. The printer port of Micropower-i is specifically designed for a Centronics Parallel Printer Port interface, but it can also be used as a general input/output port for any application that matches its input/output capabilities. It has 12 TTL buffer output points, which are latched. The interface also has five steady state input points.

3.8.2 COMPONENT DESCRIPTION

When the printer port is attached to a printer to take printouts, printer commands can be given using the PRINT key, available in the keypad. When the printer command is activated, data are loaded into the 8-bit latched output port and the strobe line is activated, writing data to the printer. One output buffer transfers 8-bit data from the Data Bust to the 8-bit latch, whose outputs are available on the 25 pin D-shell connector. The next output instruction transfers the 3 control signals and the strobe signal from the Data Bus to the 4-bit latch. The correspondance between the Data Bus and the outputs of the 4-bit latch are given as,

4-BIT CONTROL LATCH (PCNT) 30			
D0 (DATA BUS)	D1 (DATA BUS)	D2 (DATA BUS)	D3 (DATA BUS)
INIT* (Printer)	SLCT IN* (Printer)	STROBE* (Printer)	AUTO FEED (Printer)

Out of the four control signals available only the STROBE* is used to latch data onto printer. The other signals are used for printer control operation for which the software must be modified.

The input instruction allow the processor to read the 5 status lines of the printer through the status buffer. The correspondance between the data bus and the input lines of the 5-Bit Status Buffer are given as:

5-BIT STATUS BUFFER (PSTAT) 2C				
D0 (DATA BUS)	D1 (DATA BUS)	D2 (DATA BUS)	D3 (DATA BUS)	D4 (DATA BUS)
SELECT (Printer)	P_END (Printer)	ERROR* (Printer)	ACK* (Printer)	BUSY (Printer)

SINGNAL TIMING

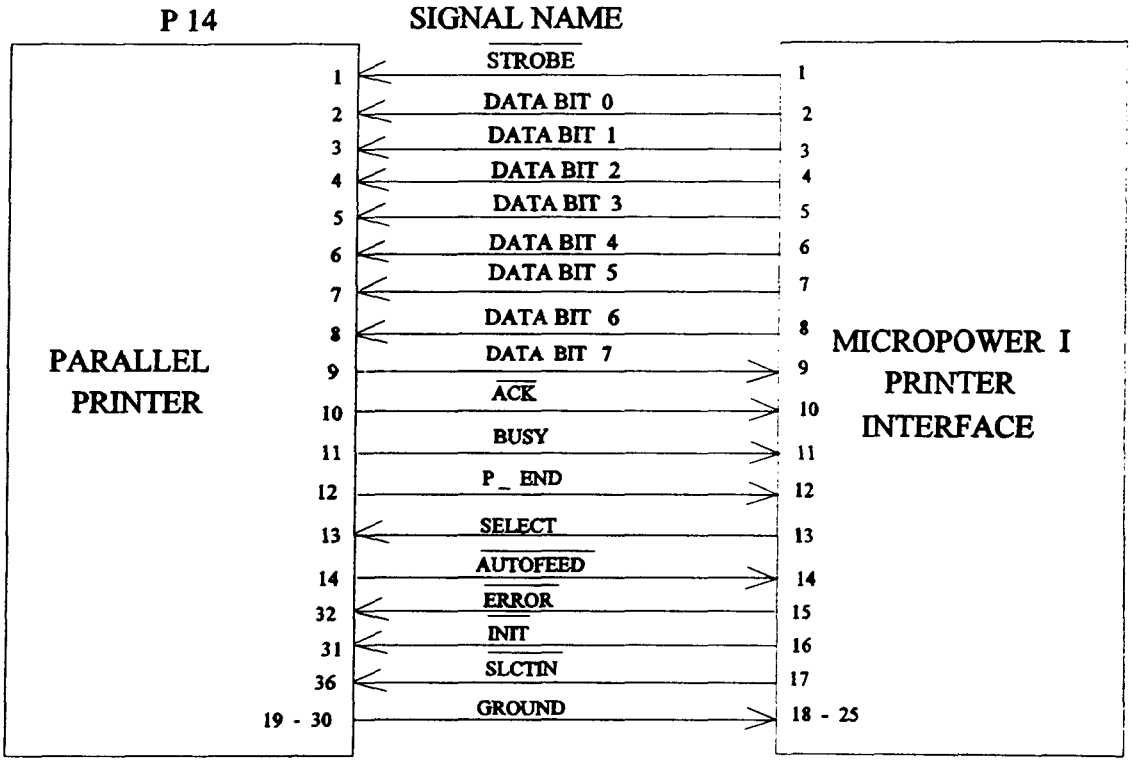
Each character transmitted to the printer must be clocked by a STROBE* signal. When a STROBE* is received it sets the BUSY signal to inform the host that the printer is no longer able to receive further data. When the character has been processed, an ACK pulse is sent and BUSY is dropped to inform the host that the printer can receive further data.

When the printer is powered-ON, a HIGH level BUSY signal is output through the interface until the printer is put in the READY state to accept data. At the end of the initialisation an ACK pulse is sent to the host before dropping the BUSY signal. The printer timing diagram is given in Figure 3.15 which clearly indicates the timing of STROBE*, data latch etc.

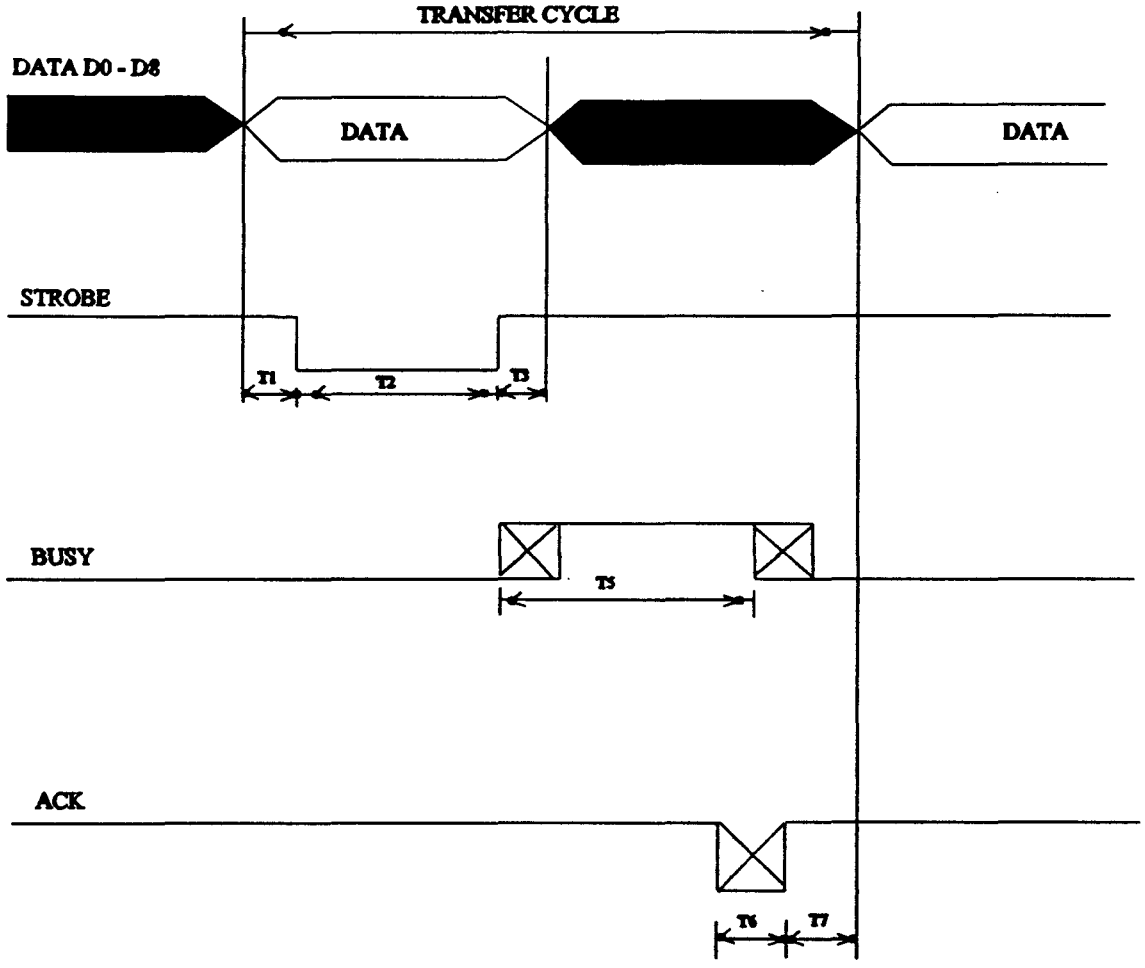
3.8.3 PROGRAMMING DESCRIPTION

In this experiment only BUSY line & the ERROR line is used to check whether the printer is busy or ready for receiving more data.

First, the BUSY signal is checked and if printer is not busy, then 8-bit data is latched onto printer clocked by a STROBE** signal. For the printer to print that character a carriage return must be sent. So after sending the Carriage Return it will be seen that the character is printed. The connections between the printer interface and the Printer is given in Figure 3.14.



PRINTER SIGNALS TIMING

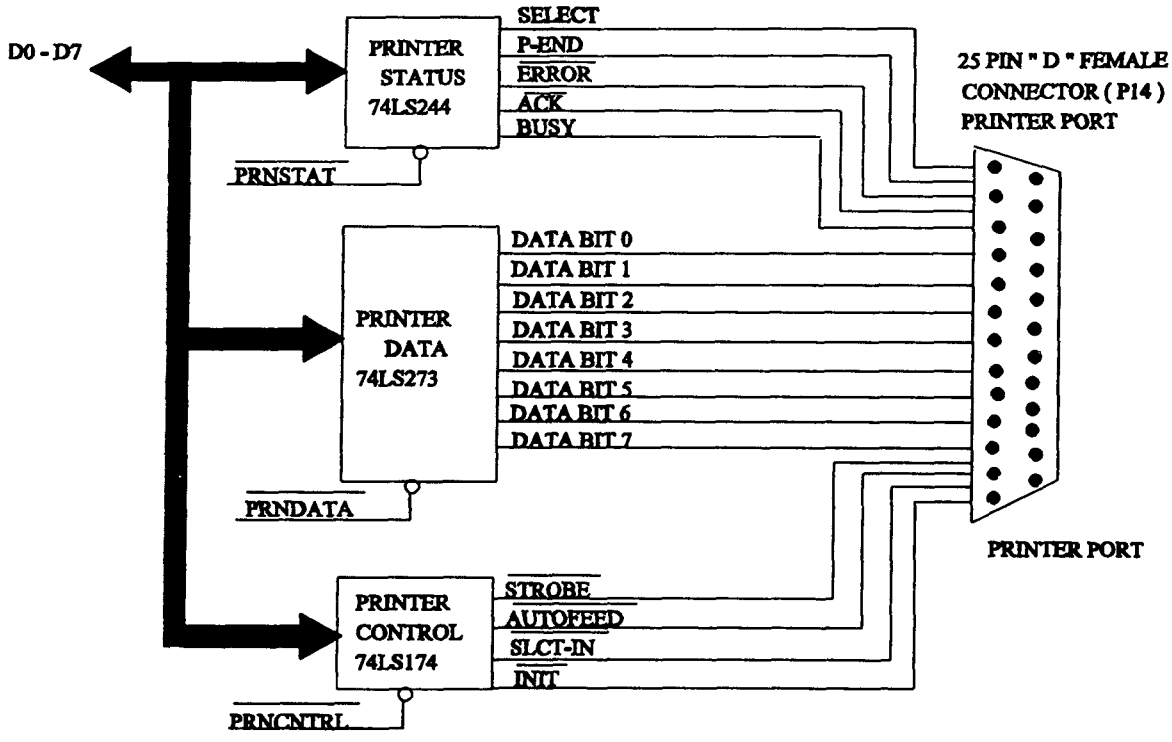


T1 = T3 = 1 SEC
T2 = 1-500 µSEC
T4 = 10 µ SEC
T5 = 20 µSEC
T6 = 2.5-6.6 µSEC
T7 = HOST RESPONSE TIME

3.8.4 CIRCUIT IMPLEMENTATION

The figure 3.16 given below gives a clear idea of the Printer interface.

BASIC BLOCK DIAGRAM OF PRINTER INTERFACE



PRNSTAT SELECT FOR I/O ADDRESS 2C
PRNDATA SELECTS FOR I/O ADDRESS 34
PRNCNTRL SELECTS FOR I/O ADDRESS 30

As seen from the block diagram, the D0-D7 data lines are used by the two latches and one buffer for the status, control and data lines respectively. The outputs of the two latches and one buffer are given to the 25-pin D Female connector as shown in the figure.

The I/O addresses for the Printer are as follows:

IC	FUNCTION	ADDRESS
U_{11} 74174	Printer Control	30
U_{19} 74273	Printer Data	34
U_{22} 74244	Printr Status	2C

3.8.5 EXAMPLE 6 - PRINT A SINGLE CHARACTER

OBJECTIVE: To write a sample program that will print a single character "A" onto the printer. Checking for the printer Busy and Error signals should also be done.

THEORY: First of all, the printer must be initialised which is by writing a 05 to the control register which makes Strobe high and Select low. Then check for the busy and Error signals sent out by the printer by reading its status. After the printer is ready to receive data output the ASCII code character "A", which is 41 hex, to the Printer data. Also followed by this character is the Carriage Return (0D hex), because the Centronics compatible printers will print the characters received only after receiving a Carriage Return.

In the CHECK routine, the printer status is again read and the printer is checked for the again read and the printer is checked for the Paper error signals. It will be in a continuous loop if the No-Paper error signal is detected. If no errors are encountered, then the data will be outputted while making the strobe low and high after 1 microsecond so that the data will be latched. Now the data can be seen printed.

PROGRAM:

```
PDATA   EQU  34H
PCNTRL  EQU  30H
PSTAT   EQU  2CH

5000                ORG  5000H
5000  31 00 55      LXI  SP, @5500H
5003  3E 05         MVI  A,05        ; Strobe High
5005  D3 30         OUT  PCNTRL      ; Select low
5007  DB 2C         IN   PSTAT
5009  E6 07         ANI  07
500B  FE 05         CPI  05          ; Check for error
500D  C2 3A 50      JNZ  @503A
5010  3E 41         MVI  A, 41H      ; ASCII code for "A"
5012  CD 1B 50      CALL @501B
5015  3E 0D         MVI  A, 0DH      ; ASCII Code for Carriage
5017  CD 1B 50      CALL @501B
501A  76            HLT
                PRINTDAT:
501B  5F            MOV  E,A
501C  CD 2E 50      CALL @502E
501F  7B            MOV  A, E
5020  D3 34         OUT  PDATA       ; Output data
5022  3E 01         MVI  A, 01
5024  D3 30         OUT  PCNTRL      ; Strobe low
5026  AF            XRA  A
5027  AF            XRA  A
5028  3E 05         MVI  A, 05       ; Strobe high
502A  D3 30         OUT  PCNTRL
502C  00            NOP
502D  C9            RET
                LOOP:
502E  DB 2C         IN   PSTAT
5030  E6 1F         ANI  1FH
5032  FE 0D         CPI  0DH         ; No paper check
5034  C8            RZ
5035  FE 02         CPI  02          ; Error check
5037  C2 2E 50      JNZ  @502E
                ERROR:
503A  3E 06         MVI  A,06H
503C  06 45         MVI  B,45H
503E  CD 05 00      CALL @0005
5041  76            HLT
```

VERIFICATION: Enter the above program from the address as stated and execute it. But before execution, be sure that you have connected the Printer with paper to the Centronics Parallel Printer Port on Micropower-i. If you have done so, and you have executed the program, you can see the character "A" being printed on the printer.

3.8.6 EXERCISES

i) Print the message "Hello! I am now aware of printer interface.

3.9 AUDIO CASSETTE INTERFACE (U1, U33)

The Audio Cassette Interface of Micropower-i allows the user to store their programs on to a audio cassette and retrieve them back also. The data is stored and retrieve as named files.

The MIC and EAR sockets provided on the kit are for connecting to the MIC and EAR sockets of the tape recorder respectively.

The basic steps to be followed for writing and reading from the tape is as given below:

I. TAPE WRITE COMMAND

1. Connect the Microphone of the recorder to the MIC jack of the Micropower-i through the cable provided.

2. Keep the Volume, Bass & Treble controls of the Tape Recorder at the Maximum level.

3. Be sure to keep the recorder ready to record data. If the recorder is not ready and if you press the Final NEXT key then data will be sent out but will not be recorded on the tape.

4. Note down the counter number from where recording starts.

5. Release the RECORD key of the tape recorder after the storing of the data is done.

II. TAPE READ COMMAND

1. Rewind the tape to the start of the file by checking with the count you noted earlier.

2. Connect the Ear phone of the recorder to the EAR jack of the kit through the cable & keep the volume at the maximum level.

With a prompt in the address and data fields, the system searches for a file. When a file is read the filename is displayed. If that is the required one then data is transferred form the tape into the memory with the message "LOAD" in address field. The starting and ending address of the memory block are read from the tape itself. A checksum which is also recorded during a tape write operation will be read back and compared with the check sum of the actual data transferred. If they do not match or if the data is incorrect, the system displays "Err" then control returns to the monitor. In such cases, check for any loose connections and re-execute the command.

DATA FORMAT

The method of storing data onto tape is somewhat simple. Each and every bit is converted to a combination of 1KHz to 2KHz pulses. The combination of the pulses differ for that of a bit 0 and bit 1. Information is recorded on the tape in the following formats.

1. BIT FORMAT

Each data byte consists of 8-bits. Each and every bit is converted to a combination of 1KHz and 2KHz pulses. The bit can be either a 1 or a 0. A "1" is combination of 8 cycles of 2KHz pulses followed by 2 cycles of 1KHz pulese. Similarly, a "0" is a combination of 4 cycles of 2KHs pulses and 4 cycles of 1KHz pulses.

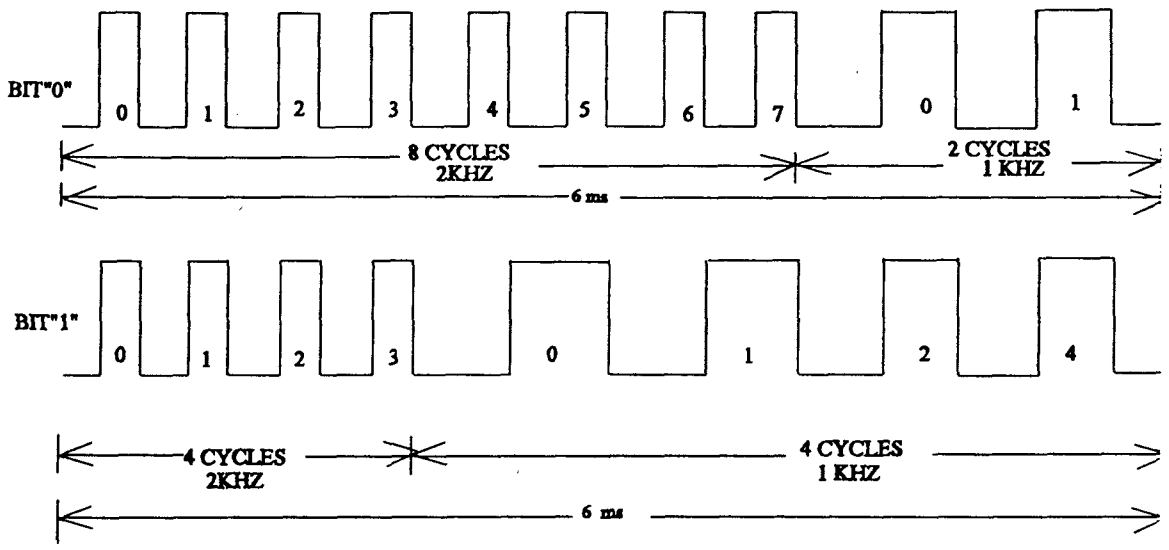
2. BYTE FORMAT

The byte to be stored is taken and is tested bit by bit and the signals are sent correspondingly. First a start bit is sent to indicate a start of byte which is a "0". Next the 8 data bits are sent with the LSB taking the leading position. Each and even bit is tested and the corresponding signals are sent out in a serial fashion. After the end of the data bis, a stop bit is sent to indicate the end of a byte, which is a "1".

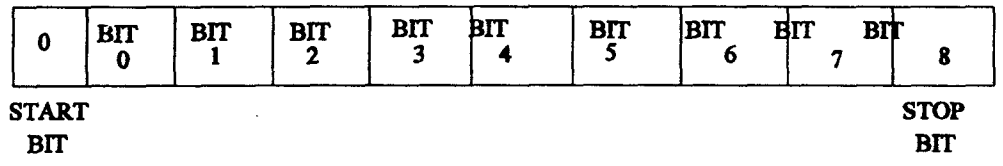
3. FILE FORMAT

The recording of a file includes a lot of synchronisation signals as shown in the Figure 3.17. The starting of a file is indicated by a lead syn c of 1KHz for four seconds. It is followed by the file name given during the Tape Write command. The start and the end address of the file are then given followed by a checksum of all the data in the specified block. Then a mid-sync pulse of 2KHz for a seconds is given. This is followed by the data itself. The file ends with a tail sync of 2KHz pulses for 2 seconds.

(1) BIT FORMAT



(11) BYTE FORMAT



(111) FILE FORMAT

LEAD SYNC	FILE NAME	START ADDR	END ADDR	CHECK SUM	MID SYNC	SS DATA SS	TAIL SYNC
1KHZ 4sec	1 BYTE	2 BYTES	2 BYTES	1 BYTE	2KHZ 2sec	VARIABLE LENGTH	2KHZ 2sec

3.10 ADC INTERFACE (ADC0809): (U51)

3.10.1 COMPONENT

The ADC interface of Micropower-i is based upon National Semiconductor's ADC0809.

3.10.2 COMPONENT DESCRIPTION

The main features are as explained below:

i. 8-bits resolution.

ii. 100 micro sec. conversion time.

iii. 8- channel multiplexer with latched control logic.

iv. Operates ratiomatically or with 5V dc or analog span adjusted voltage reference.

v. No need for external zero and full scale adjustments.

vi. Low power consumption - 15 mW.

The basic working of an ADC is as explained below:

BASIC WORKING OF AN ADC

The function of an A/D converter is to produce a digital word which represents the magnitude of some analog voltage or current. The resolution of an A/D converter refers to the number of bits in the output binary word. An 8-bit converter, for example, has a resolution of 1 PART IN 256. Another specification for an A/D converts is its conversion time. This is simply the time it takes to produce a valid output binary code for an applied input voltage. When we refer to a converter as high-speed we mean it has a short conversion time.

There are many different ways to do an A/D conversion, which represent a wide variety of conversion times. Some types may be Parallel comparator A/D converter, Dual-Slope A/D converter, successive-approximation A/D converter. Here only the Successive approximation is discussed, for ADC0809 uses only this technique for its conversion.

SUCCESSIVE APPROXIMATION ADC

As the name implies, a Successive Approximation ADC finds the digital equivalent of an analog voltage by comparing it with a sequence of known analog reference voltages where the reference chosen for (n+1)th comparison depend on the result of the nth comparison. Typically , the sequence of analog reference Voltages in obtained by feeding proper digital data words to a DAC. A k-bit ADC using this technique will generally have k-bit ADC using this technique will generally have a k-bit DAC inside where the step size of the ADC's quantization is equal to the change in the analog output of the DAC when its input changes by one LSB.

Figure 3.18 shows the block diagram of such an ADC. In order to convert a given analog voltage held by a sample and Hold Circuitry, the user issues a Start-of-Conversion (SOC) pulse to the ADC. The SAR/Logic/Control unit then takes over and using the conversion algorithm described obtains the digital equivalent of the analog signal. When the conversion is complete, the ADC returns an End-of-conversion (EOC) pulse to the user indicating that its k-bit digital output is now valid and can be read.

For the ADC shown in Figure 3.18, the Successive Approximation Algorithm can be summarised as follows:

i. Add an analog voltage equal to 1/2 LSB of DAC to the analog input voltage V_A to be converted: (see note below)

$V_A = V_A + V_{1/2}$ LSB (DAC)

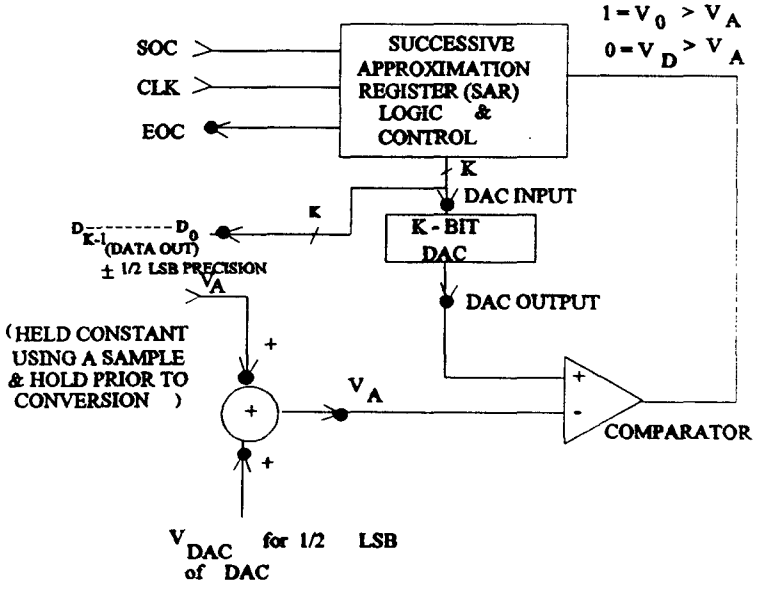
ii. Reset all SAR bits (k-bit register)

(SAR) = 000...00

iii. Initialise N = k.

iv. Set bit (N-1) of SAR and output SAR contents to the k-bit DAC.

BLOCK DIAGRAM OF K- BIT SUCCESSIVE APPROXIMATE ADC (+_ 1/2 LSB PRECISION)



v. Compare V_D with V_A. If $V_A < V_A$ go to step (vii); else continue.

vi. Reset bit (N-1) of SAR.

vii. N = N-1.

viii. If N▼ 0, goto setp (iv); else continue.

ix. Output SAR contents as the conversion result.fs

x. Generate EOC pulse.

NOTE

Step (i) of the above algorithm is only needed to ensure the accuracy of the result to ± 1/2 LSB. If this step is omitted, the accuracy of the result will only be ± LSB. Also note that if step (i) us omitted, there is no need to check the SAR bit 0 position. Instead, one can arbitrarily set or reset that bit without affecting accuracy. Morever step (viii) can be modified as:

"Step (viii). If N▼ 1, go to step (iv); else continue" in which case bit 0 of the output is always 0 and is never tested.

It is a monolithic CMOS device with an 8-bit analog-to-digital converter, 8 - channel multiplexer and microprocessor compatible control logic. The 8-bit A/D converter uses Successive Approximation technique which is explained above.

As seen from the pin configuration of the ADC, given at the back of the manual, the IN0 to IN7 are 8-Analog Channel inputs, OUT0 to OUT7 are the 8-bits for outputting the digital data.

SOC	-	Start of Conversion.
EOC	-	End of Conversion.
ALE	-	Address Latch Enable
ADDA, ADDB & ADDC	-	Channel Selection Logic.

The simple block diagram of how the ADC is interfaced to the 8085 Microprocessor is given in Figure 3.19.

As seen from the block diagram, the 8 Channel inputs are given to a connector for external connection.

The eight data outputs are connected to D0-D7 through a 8-bit latch. The clock 1.5 Mhz is divided by two to be given to the ADC0809. The SOC is given though a latch using D0. The EOC is also read from another latch throug D5. The ADDA, ADDB, ADDC and ALE are also given through a latch using D0-D3.

BASIC BLOCK DIAGRAM OF ADC INTERFACE

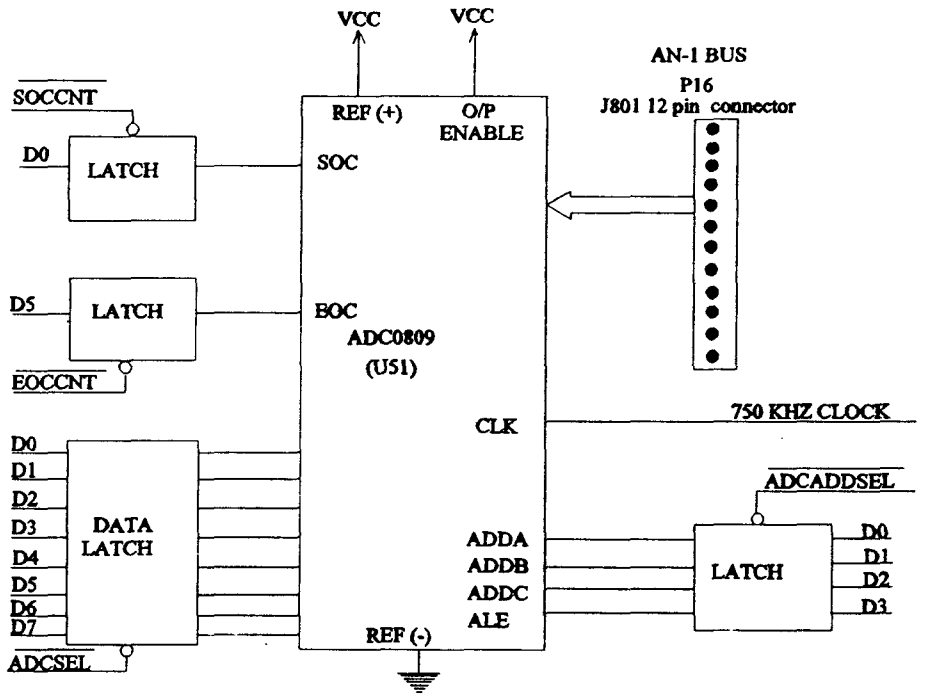


Figure - 3.19

IC NO.	FUNCTION	ADDRESS
U48	ADC address	38
U47	ADC data	24
U50, U49	ADC SOC	26
U22	ADC EOC	2C
U49	ADC CLK	-

3.10.3 PROGRAMMING DESCRIPTION

The ADC0809 requires some data be given, before giving the digital output data. Those data include Channel Selection, SOC. First give the Channel number that must be read from and converted. Next give a pulse to ALE to latch the channel data. Now, you have selected the Channel. Now give a pulse to SOC so that conversion can begin. Now, go on checking for EOC. Once you have received the EOC, you can read the digital data, for the output Enable always remain enabled in this design.

The following is a simple program for ADC0809 to read and write data.

3.10.4 EXAMPLE 7 - A/D CONVERSION PROGRAM

OBJECTIVE: To write a sample program that will initialise the channel and read the digital data from ADC0809.

THEORY: The channel 0 is connected to a trimpot in the circuit itself as shown in block diagram. So, by varying the trimpot the input voltage to the Channel 0 ADC0809 is controlled, thereby the digital data from ADC is changed. Initialise Channel 0 by giving a pulse at ALE. Give SOC. Check for EOC and read data when ready.

PROGRAM

```
4100  3E 00          MVI  A,00      ; Select Channel 0
4102  D3 38          OUT  38H
4104  3E 08          MVI  A,08H     ; ALE high
4106  D3 38          OUT  38H
4108  3E 00          MVI  A,00H     ; ALE Low
410A  D3 38          OUT  38H
410C  3E 01          MVI  A,01H     ; SOC high
410E  D3 26          OUT  26H
4110  00            NOP
4111  3E 00          MVI  A,00H     ; SOC Low
4113  D3 26          OUT  26H
4115  DB 2C CHKEOC : IN   2C        ; Check for EOC
4117  E6 20          ANI  20
4119  CA 15 41       JZ   CHKEOC
411C  DB 24          IN   24
411E  32 00 42       STA  4200H
4121  76             HLT
```

VERIFICATION: Now vary the trimpot value and execute the program once again. Now the value stored at 4200 should change according to the input analog voltage. It is not necessary to use the trimpot and Channel 0. Using the 16-pin header AN-1 Bus, connect external analog input to any of the channels and read the equivalent digital data.

3.10.5 EXERCISES

1. Write a program to read from Channel 1 of ADC0809, given an external voltage at Channel 1, and display the digital data obtained in the seven-segment display in the data field.

3.11 DAC INTERFACE (DAC0800): (U40)

3.11.1 COMPONENT

The DAC0800 of National Semiconductor's forms the DAC interface the Micropower-i.

3.11.2 COMPONENT DESCRIPTION

It is a monolithic high-speed current output Digital-to-Analog Converter. Its unique features are:

i. Typical settling time of 100 ms.

ii. Complementary current outputs.

iii. Differentiable output voltages of 20 V peak-to-peak by simply changing load resistors.

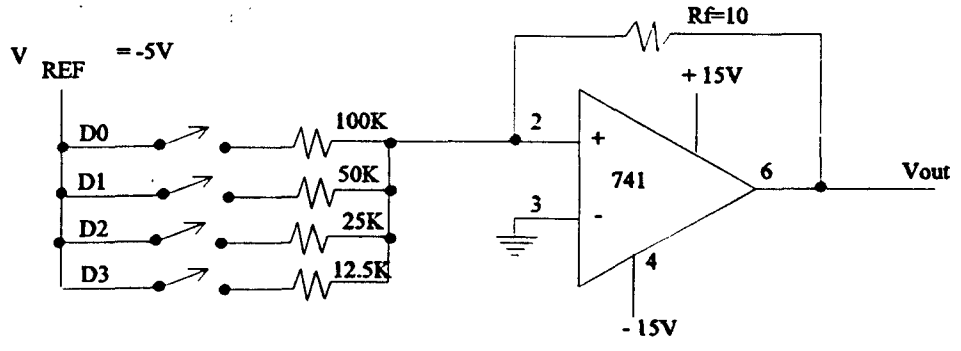
iv. 2 quadrant wide range multiplying capability.

BASIC OPERATION OF A DAC

The purpose of a digital-to-analog converter is to convert a binary word to a proportional current or voltage. To see how this is done let's look at the simple op-amp circuit in figure 3.20. This circuit functions as an adder.

When one of the switches is closed a current will flow from the -5V (V_{REF}) through that resistor to the summing point. The op -amp will pull the current on through the feed-back resistor to produce a proportional output 0.05 mA will flow into the summing point. In order to pull this current through the feedback resistor, the Op-amp must put a voltage of 0.05 mA * 10 K ohms or 0.5 V on its output. If you also close switch D1, it will send another 0.1 mA into the summing point. In order to pull the sum of the currents through the feedback resistor, the op -amp has to output a voltage of 0.15ma* 10K ohm or 1.5V.

SIMPLE 4 - BIT DIGITAL TO ANALOG CONVERTER



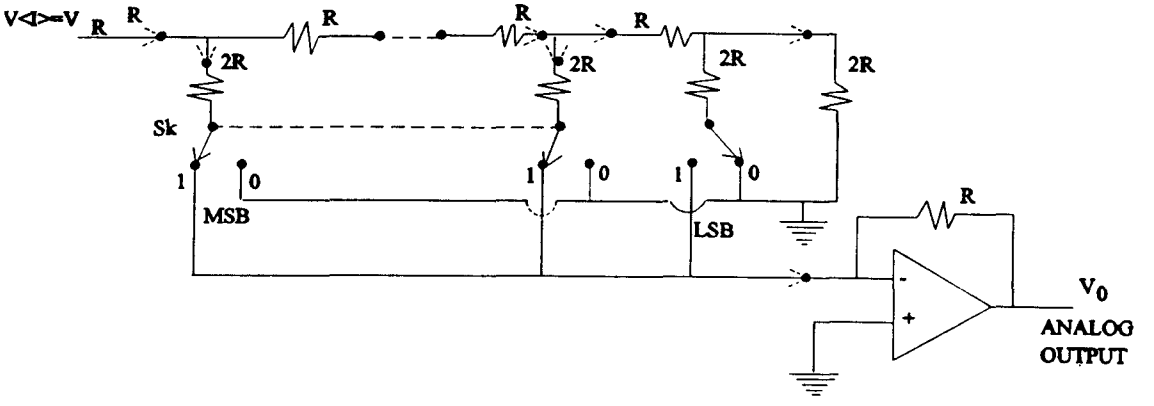
The point here is that the binary-weighted resistors produce binary-weighted currents which are summed by the Op-amp to produce a proportional output voltage. The binary word applied to the switches produces a proportional output voltage. Technically the output voltage is "digital" display on a digital voltmeter can. However, the output simulates an analog signal, so we refer to it as analog. Switch D3 in the above figure represents the most significant bit, because closing it produces the largest current. Note that since VREF is negative, the output will go positive as switches are closed.

The precision obtainable in a Digital to Analog Conversion using the Weighted Resistor DAC is a function of th various resistances. Obtaining equal precision of the various resistors over such a wide range of values (i.e. 1 to 128) Weighted Resistor DAC can be solved by an alternate design, the R-2R Ladder Network DAC, where the only resistance values needed are R and 2R. The circuit of a k-bit R-2R Ladder Network DAC is shown in Figure 3.21.

Switch S_k connects to the 1-point if DK = 1 and to the 0 point if $D_k = 0$. We have shown V(1) = V_R and V(0) = 0 in figure so that the DAC shown is a Unipolar DAC for straight unsigned binary inputs. Modifications for other binary codes - offset binary, 2's compliment, etc. can be made. Finally, it should be noted that in the following figure, if a bit input D_i is 0, the corresponding input to the Op--amp's non-inverting input is left open.

This is not a problem if the Op--amp were indeed an ideal Op--amp. In an actual circuit, though, an open input is not desirable since it might cause 'pick up' due to unavoidable couplings. It is therefore necessary to provide some means of minimizing pick-ups at such at such open inputs.

R-2R LADDER NETWORK DAC <K-BIT, UNIPOLAR>



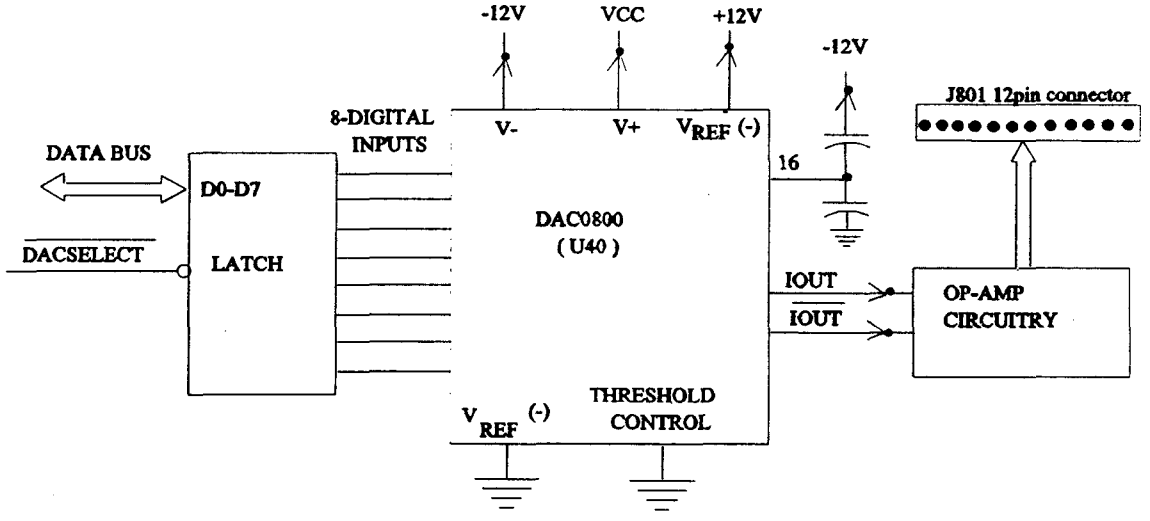
As explained above, the DAC0800 uses the R-2R ladder network for the D to A conversion.

3.11.3 CIRCUIT IMPLEMENTATION

The basic block diagram of a DAC is given in Figure 3.22.

As seen from the block diagram, the 8-data lines of the DAC are connected to the data bus through an 8-bit latch. The current outputs are passed onto an Op-amp circuit which gives the voltage output. This voltage output is differential and a maximum of 20V peak-to-peak can be obtained by changing the load resistor value of the Op-amp.

BASIC BLOCK DIAGRAM OF DAC INTERFACE



$\overline{\text{DACSELECT}}$ IS THE CHIPSELECT TO DAC0800 WHICH SELECTS FOR I/O ADDRESS 20

Whatever digital data is outputted to DAC, the equivalent voltage will be go out of the Op-amp. So, programs can be written to produce square wave, triangular wave, sawtooth wave, sine wave etc., at the output of DAC. Those can be obtained by outputting the corresponding data. One is done here for example. The others, you have to try and get it.

3.11.4 EXAMPLE 8-SAW-TOOTH WAVE GENERATION

OBJECTIVE: To write a sample program for the DAC that generates a saw-tooth wave at the output of the DAC.

THEORY: To obtain a saw-tooth wave, do the following:

i. First output 00 to DAC.

ii. Increment by one and output to DAC until the count has reached FF.

iii. Again start from the begining i.e., first step onwards.

PROGRAM

```
4100  2E 01      START:   MVI  L,01     ;Initialise counter
4102  7D          LOOP:    MOV  A,L      ;Output data
4103  D3 20                OUT  20H
4105  2C                   INR  L        ;Counter = Counter +1
4106  FE 00                CPI  0CH      ;If Counter = 0,
4108  C2 02 41             CNZ  LOOP     ;then start again
410B  C3 00 41             JMP  START    ;From begining
```

VERIFICATION: Enter the above program and execute it. Check using an oscilloscope whether you get a saw-tooth wave at DAC output.

3.11.5 EXERCISES

1. Calculate the frequency got out of the above example.
2. Write a program to generate a square wave at the output of DAC.
3. Write a program to generate a sine wave at the output of DAC.

3.12 EPROM PROGRAMMER : (U45, U46, U41, U42, U43)

3.12.1 COMPONENT

The implementation of the onboard EPROM Programmer is done using to obtain the appropriate voltages. The buffers are driven by an octal latch 74LS273. The address and data are provided through an 8255 operating in output and input mode and the programmer socket is itself a 28 pin ZIF (Zero Insertion Force) socket.

3.12.2 COMPONENT DESCRIPTION

To use the onboard programmer the user, is required to type EP followed by ENTER key. The user can 2764A, 27128A, 27256 and 27256A. The programming voltages, the pin at which the voltages appears, the capacity of each EPROM and hence the number of address lines is as tabulated below.

TABLE

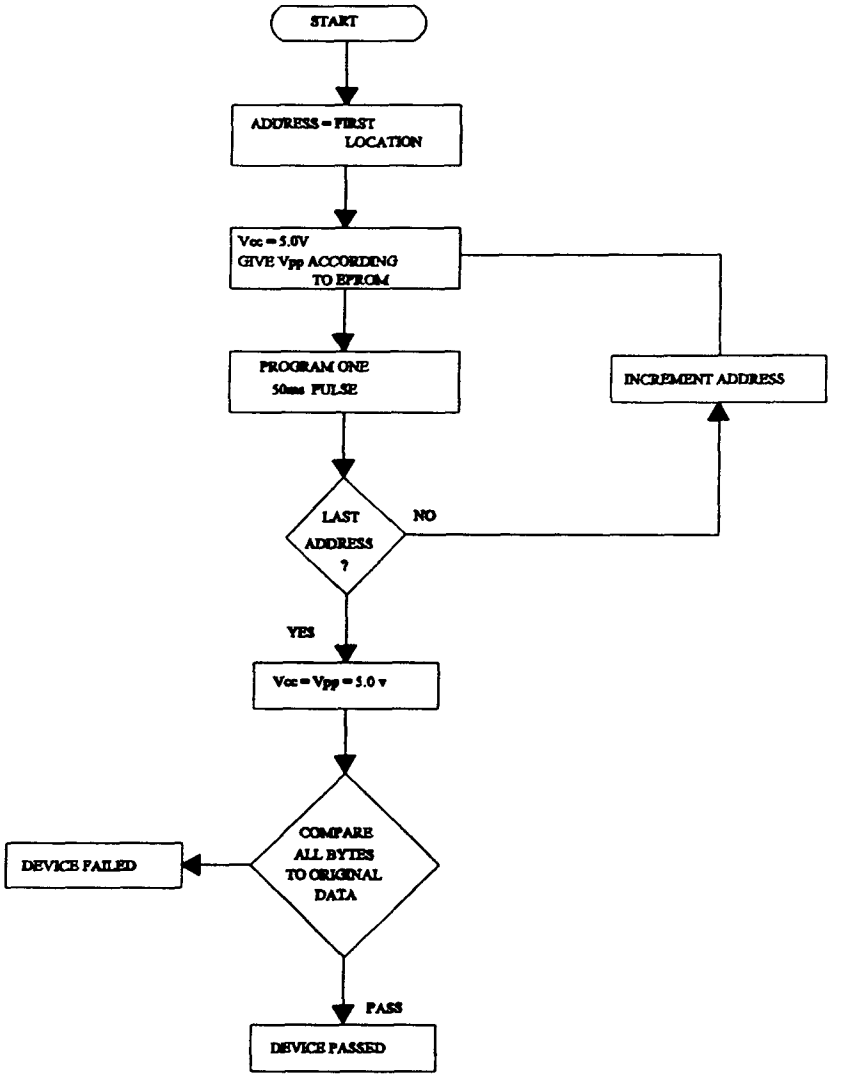
EPROM	Programming Voltage	Pin NO.in ZIF	Capacity	No . of address lines
2716	25V	23	2K	11
2732	25V	22	4K	12
2732A	21.5V	22	4K	12
2764	21.5V	1	8K	13
2764A	12.5V	1	8K	13
27128	21.5V	1	16K	14
27128A	12.5V	1	16K	14
27256	21.5V	1	32K	15
27256A	12.5V	1	32K	15

PROGRAMMING MODES AND PROGRAM VERIFY

Programming an EPROM can be accompished in any of three ways namely the standard programming Alogorithm, the quick pulse programming Algorithm or the Intelligent programming algorithm. The EPROM is programmed using the algorithm which it supports. Either standard or Quick pulse is used.

STANDARD PROGRAMMING ALGORITHM

The flowchart below shows the method followed in programming an EPROM using standard Algorithm.

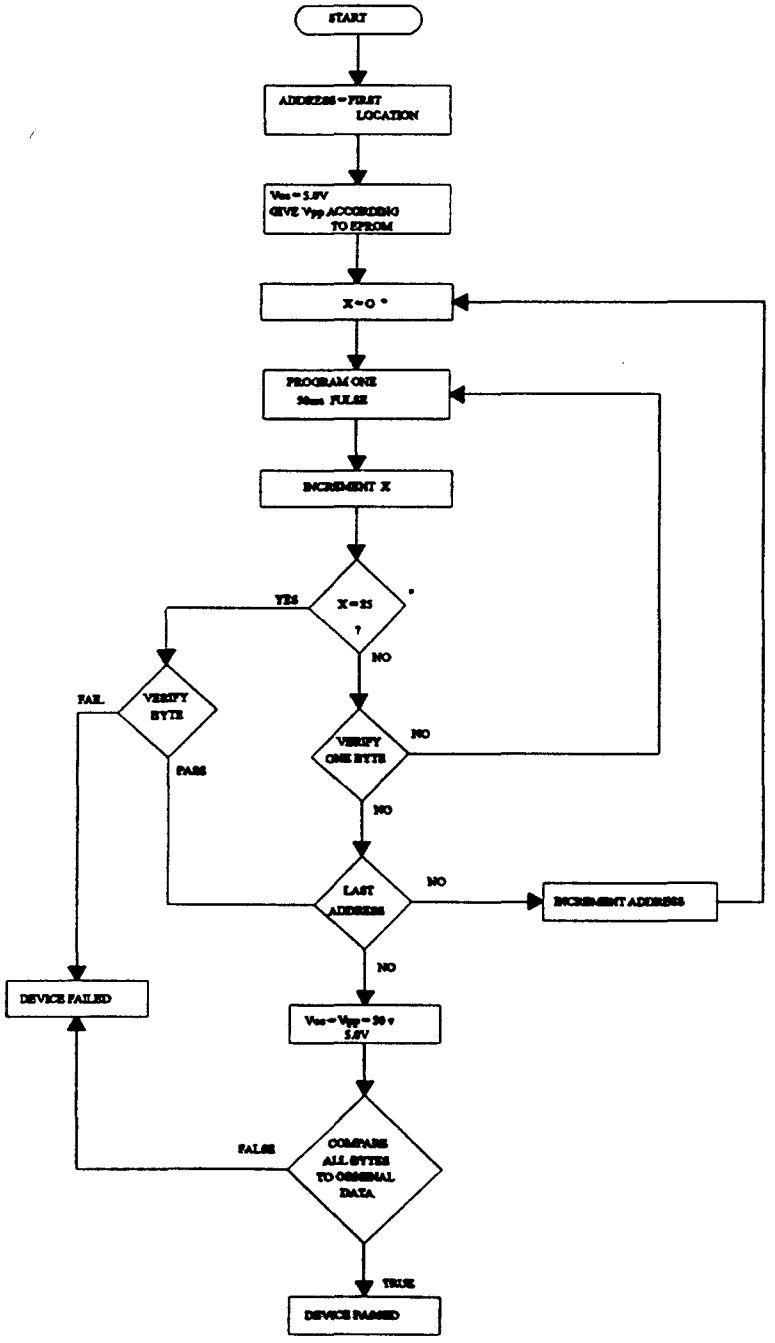


The device is in the programming voltage. The data to be programmed is applied 8 bits in parallel to the data output pins. The levels required for address and data are TTL (Logic high is 2.0V and low is 0.8V).

When the address and data are stable, a 20 microseconds active low, TTL program pulse is applied to the CE* input. A program pulse is applied at each address location to be programmed, the pulse having a maximum width of 55 microseconds, either sequentially, individually or at random.

QUICK PULSE ALGORITHM

This algorithm substantially reduces the throughput time in the programming environment. It uses initial pulses of 100 microseconds followed by a byte verification to determine when the address byte has been successfully programmed. Upto 25-100 microseconds pulses per byte are provided before a failure is recognized. The flow chart for quick - pulse programming Algorithm is as shown in figure F3.



For this algorithm, the entire sequence of programming pulse and byte verifications is performed at Vcc = 6.25V and Vpp at 12.75V. When programming of the EPROM is completed, all bytes should be compared to the original data with Vcc = Vpp = 5.0V

VOLTAGE GENERATION

Hence the circuit must be capable of sending the appropriate voltages and these voltages are derived as below. From the +30V supply, through zener networks 12.5V, 22V and 27V are derived. From the +12V supply a 5.6V and 6.3V supply are obtained.

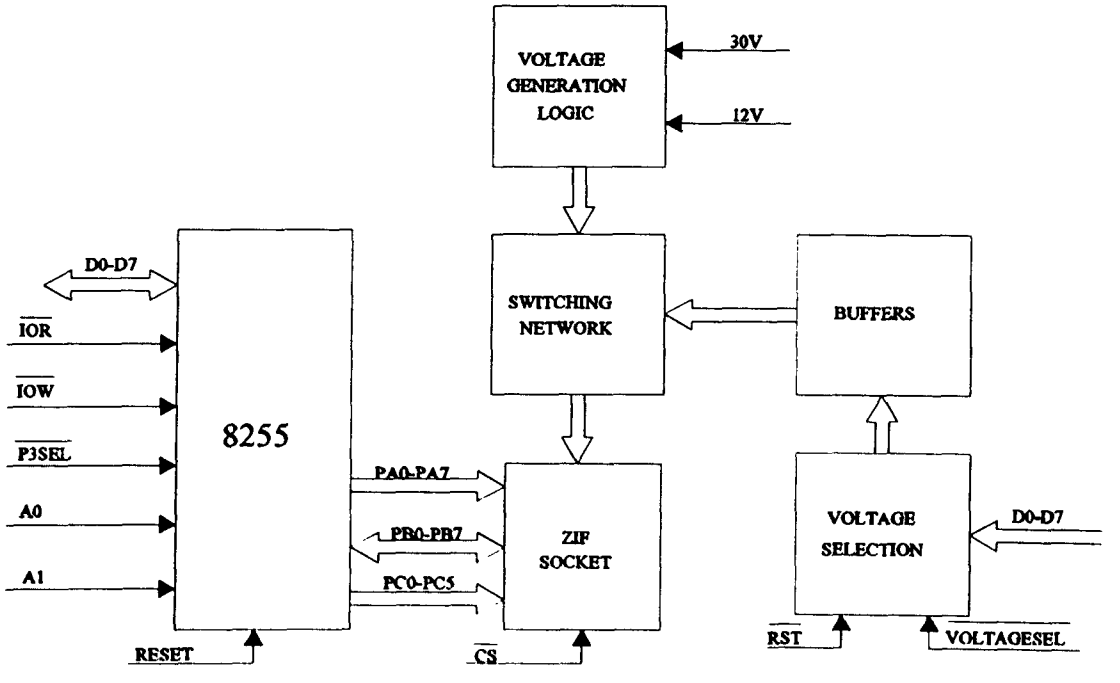
The programming voltages are derived from the emitters of switching transistors. The base of a switching transistors is connected to the output of a buffer, of 74LS07. The buffer outputs are pulled high and the buffer driving it should get a TTL high input. This is done by making the appropriate data line high.

ADDRESS AND DATA LINES

The address and data lines are the port outputs of the 8255 (U28). The port A port C are configured in the output mode. The lower order address lines are derived from port A and the other address lines from port C. The data lines are from port B of the 8255 (U28). When the data is input to the EPROM as in programming, then port B acts as output port. When the function selected is either reading or blank checking then port B is configured in input mode.

3.12.3 CIRCUIT IMPLEMENTAION

The following Figure is the simple block diagram of the EPROM programming interface in the Micropower-i.



$\overline{P3SEL}$	-	SELECTS FOR ADDRESSES 14 TO 17
$\overline{CS}$	-	SELECTS FOR ADDRESSES 1C TO 1 F
$\overline{EPROMSEL}$	-	SELECTS FOR ADDRESSES 18
$\overline{IOR}$	-	I/O READ FROM CONTROL BUS
$\overline{IOW}$	-	I/O WRITE FROM CONTROL BUS
$\overline{D0\text{-}D7}$	-	DATA BUS FROM PROCESSOR

USING THE PROGRAMMER

Pressing EP followed by Enter key, the user enters the programmer mode.

After going in to the programmer mode the user can select any of the EPROMs from 2716 to 27256A. The selection can be done by pressing 'Y' in the responce field against the particular IC name.

Then the function to be performed, (i.e) blank checking, reading or writting onto the EPROM can be selected by pressing 'Y' in the appropriate option in the menu. All these details are indicated in the technical reference.

The usage of EPROM programmer is deteiled in the technical reference manual.

The following example shows you hoe to blankck\heck, program and verify the programmed contents for a 2716 EPROM.

3.12.4 EXAMPLE 9 - EPROM PROGRAMMER

OBJECTIVE : To blank check, program and verify the contents in a 2716 EPROM.

THEORY : Reset the system. Press EP followed by ENTER key. It will display 2716'. Press Y followed by Enter key to select 2716. Now insert the EPROM into the ZIF socket talking care to use the lower 24 pins of the ZIF.

Then you have to select the function. (i.e) blank check. Enter 'Y' and Enter key against the blank check option. Then enter the start address (0000) and end address (07FF) followed by carriage return (ENTER key). Upon getting these addresses, the trainer reads all the locations and checks wheather they all contain FF. If all the locations are blank, the trainer displays "SUCCESS" and comes back to the EPROM selection menu. If the EPROM is not blank, then the message blamk check "FAIL" is displayed and the trainer returns to the EPROM select menu.

Now let us copy the locations 0000 to 07FF of the monitor EPROM to the 2716 in the ZIF.

Select 2716 then select program option.

Then enter the start and end address of the EPROM followed by the destination address (from where you want to copy).

(i.e)	0000	07FF	0000
	EPROM Start Address	EPROM End Address	Destination Address (Monitor ROM)

Giving all the three addresses and pressing Enter key causes the programmer to write into the blank locations. A databyte is written by switching to "OFF" state or "0" state the flip-flop or memory cell in the EPROM. The selection of the addresses is done sequentially and zeros are selectively programmed. While programming, the address of the EPROM currently being programmed. After programming all the desired address, the trainer displays "SUCCESS".

Now we should verify if the EPROM has the correct data. Hence let us read the contents of the EPROM into RAM to check with the monitor contents. This is achieved by selecting the 'READ' option after selecting the 2716 EPROM option. Then give the start and end address of the EROM following by destination address, (i.e) RAM address.

0000	07FF	5000
EPROM Start Address	EPROM End Address	RAM Start Address

After reading, it comes to the EPROM selection menu. Now press '.' (dot) followed by Enter key. to come to the command prompt. Now compare the contents of source with that of data read from EPROM. Use BC (Block compare) command to compare from 0000 to 07FF with 5000. The trainer comes back to the command prompt to indicate that the EPROM has been pogrammed properly.

3.12.5 EXERCISES

i. Obtain the following EPROMs from your stores. 2732, 2732A, 2764, 2764A, 27128, 27128A, 27256, 27256A. Blank check all the EPROMs.

ii. a. Write a program to implement a digital clock. Configure addresses from 6000.

b. Copy your program in a 2764 EPROM and insert it into the memory socket U9.

c. Check if your digital clock is working.

CHAPTER 4

APPLICATION EXAMPLES

4.1 INTRODUCTION

The following are some sample experiments that uses more than one peripheral i.e., the program written, uses a combination of the peripherals available on the Micropower-i. With these experiments, you can learn a lot about interfacing two different peripherals which will help to enhance your knowledge in the field of programming based on hardware.

EXAMPLE 1 - DIFFERENTIATE RAM & EPROM

In the standard Micropower-i with 8085 processor, the addressed form 0000 to 1FFF are EPROM locations, while 4100 to 5FFF are User RAM locations. What is the difference between these two?

An EPROM is an Erasable Programmable Read Only Memory. It cannot be written into using a kit. To write into the EPROM, separate EPROM programmers must be used.

A RAM is a Random Access Memory which can be either written into or read form. But the data written into can be retained only if the power is retained. If the power is turned-off, then the data in the RAM is lost. But the EPROM can retain the data even when the power is turned-off.

Let us examine these two memory types.

(i) Using SUB command enter 45 from 4100 to 410F.

(ii) Examine the following memory locations using SUB command and write down the contents next to each address. Use the first column of data here.

ADDRESS	DATA	DATA	ADDRESS	DATA	DATA
0000	---	---	4100	---	---
0001	---	---	4101	---	---
0002	---	---	4102	---	---
0003	---	---	4103	---	---
0004	---	---	4104	---	---
0005	---	---	4105	---	---
0006	---	---	4106		
0007	---	---	4107	---	---
0008	---	---	4108	---	---

0009	---	---	4109		
000A	---	---	410A	---	---
000B	---	---	410B	---	---
000C	---	---	410C	---	---
000D	---	---	410D	---	---
000E	---	---	410E	---	---
000F	---	---	410F	---	---

(iii) Now turn the power to the trainer off. Wait for around 10 to 15 seconds and then turn-on the trainer now.

(iv) Examine the memory locations as stated in Step (ii) and write the data read in the second column provided. By comparing these two sets of data, it will be clear that the data at 4100 to 410F has changed but the data at 0000 to 000F remains the same.

(v) Therefore, the addresses from 4100 is RAM while 0000 is EPROM. Data is lost from RAM as the power is turned -off. When power is given again, a random data appears.

(vi) Enter a new data at 0000. What happens? "Err." is displayed, which indicates that 0000 cannot be written into.

EXERCISE

i) Write a program to differentiate between a RAM and an EPROM. Given a memory address, you should find whether it can be written into or not by your program.

EXAMPLE 2 - INTERFACING ADC0809 AND DAC0800:

OBJECTIVE: Feed in a sign wave to the ADC Channel 1 using a function generator, read the equivalent digital voltage and out it to the DAC0800 and get back the same sine wave out of it.

THEORY: The problem is that ADC being unipolar, DAC is a bipolar one. The ranges for both ADC and DAC are from 00 to FFH, but a 00H from ADC will correspond to 0V while form a DAC it will correspond to -5V. Only 80H will correspond to 0V in DAC. So a conversion is required before the sample's hex equivalent from ADC is given to DAC. The conversion can be got by the following the sample the following technique. After getting the hex equivalent of the sample the following is performed, hex equivalent + (FF - hex equivalent) / 2. If there is any remainder while dividing then we add one to the quotient. Output this quotient to the DAC. Take samples at a particular time interval. Do not go above 90Hz on the function generator. It will be a problem if you go above 90Hz.

PROGRAM:

```
                                ; Program to read from the channel 1
                                ; of ADC0809 and output the digital
                                ; data to DAC0800 and get the same
                                ; analog voltage which was fed to
                                ; channel 1 of ADC0809
                                ;
      28    00                  ADSOC     EQU   26H
      20    00                  DAC       EQU   20H
      38    00                  ADCNTRL   EQU   38H
      2C    00                  ADSTAT    EQU   2CH
      24    00                  DATA      EQU   24H
4100  3E    01                  MVI   A, 01H
4102  D3    38                  OUT   ADCNTRL
4104  C6    08                  ADI   08H
4106  D3    38                  OUT   ADCNTRL
4108  D6    08                  SUI   08H
410A  D3    38                  OUT   ADCNTRL
410C  3E    01                  MVI   A, 01H
410E  D3    26                  OUT   ADSOC
4110  AF                        XRA   A
4111  DB    2C          REC:    IN    ADSTAT
4113  E6    20                  ANI   20H
4115  CA    11 41               JZ    REC
4118  DB    24                  IN    DATA
411A  57                        MOV   D,A

411B  3E    01                  MVI   A, 01H
411D  D3    26                  OUT   ADSOC
411F  AF                        XRA   A
4120  D3    26                  OUT   ADSOC
4122  3E    FF                  MVI   A, FFH
4124  92                        SUB   D
4125  1E    00
4127  D6    02          L01:    SUI   02H
4129  DA    30 41               JC    L02
412C  1C                        INR   E
412D  C3    27 41               JMP   L01
4130  C6    02          L02:    ADI   02H
4132  32    62 41               STA   4162H
4135  7A                        MOV   A,D
4136  83                        ADD   E
4137  57                        MOV   D,A
4138  3A    62 41               LDA   4162H
```

```
413B  FE   00          CPI   00H
413D  CA   41 41       JZ    L03
4140  14               INR   D
4141  7A        L03:   MOV   A, D
4142  D3   20         OUT   DAC
4144  CD   4A 41      CALL  DELAY
4147  C3   11 41      JMP   REC
414A  06   FF   DELAY: MVI   B,FFH
414C  00         LZ:   NOP
414D  00               NOP
414E  00               NOP
414F  00               NOP
4150  00               NOP
4151  05               DCR   B
4152  C2   4C 41      JNZ   LZ
4155  C9               RET
```

VERIFICATION: After entering the above program, execute it. Us a two channel oscilloscope and view both the function generator output and the DAC output. They will be similar.

CHAPTER 5

SYSTEM ORIENTED FUNCTIONS

5.1 INTRODUCTION

You have come a long way with 8085 now. You have worked out all the software and hardware examples as well as the exercises. Hence now you are definitely a very good programmer. To further enhance your proficiency as a programmer, you need to learn the facilities which are provided by the system namely the system calls. Use of system calls shall shorten your code for any program and transform it into small and efficient, as was said in Chapter 2.

In addition you will learn about the software interrupts of 8085 and also about a communication package DATACOM which will enable you to transfer data to and from a system.

5.2 SYSTEM CALLS

The monitor of Micropower-i with 8085 processor, offers several user-callable useful routines. These system calls make the program more efficient by its powerful subroutines. The details of the routines are given in Micropower-i Technical Reference for 8085 and Z80 Piggy Back. Here let us write small programs that use these system calls. Please go through the Monitor system calls in the Technical Reference before studying this material.

5.2.1 EXAMPLE 1

OBJECTIVE : To clear the LCD using system call function 02.

THEORY : First, initialise A to 02 and then CALL the routine 0005.

PROGRAM

```
                        ORG  5000
5000 :   3E 02          MVI  A, 02
5002 :   CD 05 00       CALL @0005
5005 :   76             HLT
                        END
```

VERIFICATION : Enter the above program and execute it. The LCD will be cleared. Actually, the system call will not be returned back to 5005 as it should be in the case a CALL instruction.

5.2.2 EXAMPLE 2

OBJECTIVE : To write a program that reads a key pressed from keyboard and displays the same in the LCD.

THEORY : To read a key from the keyboard, and to display the same in second row of LCD initialise A to 08, load 'B' register with 40H then call 0005. After a key pressed, the routine reads the keycode, converts it into ASCII and displays it in the LCD at desired location (as specified by B register).

PROGRAM

```
                    ORG  5000
5000 : 3E 08        MVI  A, 08
5002 : 06 40              MVI  B, 40
5004 : CD 05 00     CALL @0005
5007 : CF           RST  1
                    END
```

VERIFICATION : Enter the above program and execute it. Now press a key from the keyboard. The corresponding character will be displayed at the 2nd row of LCD. The above program displays all the keys pressed continuously until you press Enter key. If you press Enter, the function call returns to your main program where you have the RST 1 instruction. After executing RST 1, it will come to command prompt.

More details about RST 1 can be has from next chapter.

5.3 INTERRUPT

An Interrupt is a signal sent to the microprocessor, which may request service at any time and is asynchronous to the program. Whenever a program branches to a subroutine, such branching is synchronous to program execution i.e., scheduled by the program. An interrupt, however, may occur at any time, and will generally suspend the execution of the current program (with out the program knowing it). Because it may happen at any time relative to program execution, it is called asynchronous.

Why use interrupts? Interrupts allow events such as alarms, power failure, the passage of a certain amount of time and peripherals having data or being ready to accept data, to get the attention of the CPU.

The advantages of interrupts are obvious, but there are also some disadvantages. These include:

(i) Interrupt systems may require a large amount of extra hardware.

(ii) Interrupt still require data transfers under program control through the CPU.

(iii) Interrupt are random inputs which make debugging and testing difficult. Errors may occur often and therefore may be very hard to find.

8085 INTERRUPTS

Out of the different ways of getting data to or from the microprocessor, the most used ones are the

i. Programmed I/O transfer
ii. Interrupt driver I/O transfer
iii. DMA (Direct Memory Access)

Under this programmed I/O comes the software interrupts of 8085. (RESTART instructions)

Under the interrupt-driven I/O comes the five hardware interrupt lines of 8085 (TRAP, RST 7.5, RST 6.5, RST 5.5 and INTR).

The DMA is not discussed here.

1. First, let us have look on the RESTART instructions of 8085. The 8085 allows 8 RST instructions to be used, each with a single byte opcode. These restart instructions represent efficient ways to call frequently used subroutines. The eight restart instructions are RST 0, RST 1, RST 2, RST 3, RST 4, RST 5, RST 6 and RST 7. WHAT AN RST DOES?

Referring to Table 5-1, the effect of a RST instruction is basically a Call. For instance, the execution of RST 0, will begin as:

i. Pushing the contents of the Program Counter onto stack.
ii. Program branches to address 0000.
iii. The Subroutine located between 0000 and 0007 is carried out with the RET returning the processing to the main program.

This restart is a special kind of call because it branches to a predetermined address. As already mentioned, it is an one byte instruction as against to three bytes to a CALL instruction. Therefore, this RST is an efficient way to call frequency used subroutines.

TABLE 5-1.

Instruction	Effect	Opcode	Vector location
RST 0	CALL 0000	C7	0000
RST 1	CALL 0008	CF	0008
RST 2	CALL 0010	D7	0010
RST 3	CALL 00018	DF	0018
RST 4	CALL 0020	E7	0020
RST 5	CALL 0028	EF	0028
RST 6	CALL 0030	F7	0030
RST 7	CALL 0038	FF	0038

VECTORED CALLS

The word Vector implies direction. The RST instuction are like vectors because they point to specific locations in memory. The starting address of each restart subroutine in called a VECTOR LOCATION. RST 0 points to vector location 0000, RST 1 points to vector location 0008 and so on.

Since only 8 bytes are available at the vector location, mostly these vector locations are used for the starting address of the longer subroutine that is needed.

2. Now, we come to the section on the hardware restarts of the 8085. This interrupt-driven I/O is much more efficient than the previous one for CPU time is not wasted i.e., the CPU need not wait for an interrupt.

   The hardware restarts are also vectored like the software and jump to predetermined locations. Such hardware interrupts VECTORED INTERRUPTS, because the program branches to a vector location where the starting address of a service subroutine is stored.

   The vector locations for the hardware restarts are given in Table 5-2. As seen they are located exactly halfway between the software restart locations.

TABLE 5-2

Interrupt	Priority	Vector Location
TRAP	1	0024H
RST 7.5	2	003CH
RST 6.5	3	0034H
RST 5.5	4	002CH
INTR	5	None

INTERRUPT PRIORITIES

The priorities of these interrupts are also given in this table. If two or more interrupts are active at same time, the 8085 takes in the order of the priority level: TRAP is serviced first, then RST 7.5 and so forth.

To get a better understanding of how interrupt work, look at Figure 5.1. The hardware interrupts are given here in detail.

TRAP: This interrupt is activated by a positive edge and a sustained high level as shown in Figure 4.1. This avoids false triggering caused by noise and transients.Again refer to Figure 5.1. The positive edge of TRAP signal will set the D flip-flop. Because of the AND gate, however, the final TRAP also depends on a sustained high level TRAP input. This is why the TRAP is both edge and level sensitive.

The TRAP flip-flop can be cleared either by a low RESET IN or a high TRAP ACKNOWLEDGE that the 8085 sends to clear the flip-flop after the recognition of a TRAP for further use of it.

RST 7.5: This interrupt is activated by edge -triggering alone which allows false triggering to happen.

Upon sensing this interrupt, the 8085 sets the D Flip-Flop. The output of this flip-flop, named I7.5, is one input to an AND gate.

This RST 7.5 can be cleared is with a high RST 7.5 bit using the software instruction EI.

RST 6.5 & RST 7.5 : These two inputs are directly connected to AND gates. Therefore, a sustained high level is required to enable the two interrupts.

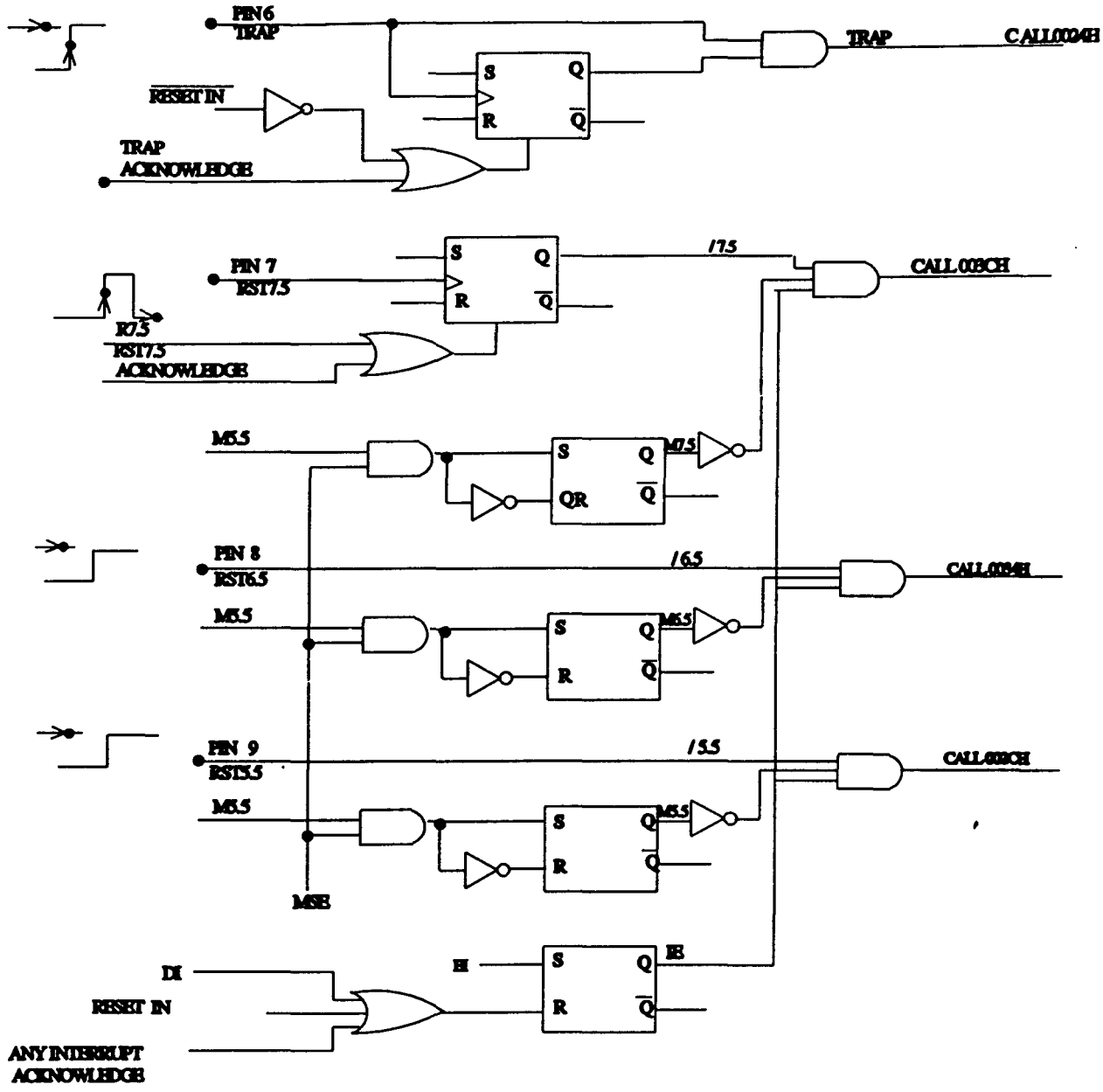

PIN6
TRAP
TRAP
CALL0024H
RESET IN
TRAP
ACKNOWLEDGE
S
Q
R
Q
PIN 7
RST7.5
I7.5
RST7.5
ACKNOWLEDGE
I7.5
CALL003CH
M5.5
M7.5
QR
PIN 8
RST6.5
I6.5
CALL0034H
M5.5
M6.5
PIN 9
RST5.5
I5.5
CALL002CH
M5.5
M5.5
MSE
DI
RESET IN
ANY INTERRUPT
ACKNOWLEDGE
IE

Figure- 5.1

MASKABLE INTERRUPTS

The signals I7.5, I6.5 and I5.5 are called pending interrupts. The signal IE (bottom flip-flop) is called the interrupt enable flag; it must be high to activate the AND gates. Also notice the M7.5, M6.5 and M5.5 signals; they must be low to enable the AND gates.

To activate the RST 7.5 interrupt, I7.5 must be high. M7.5 must be low, IE must be high. Similarly, to get a high RST 6.5, I6.5 must be high, M6.5 low and IE high. For the RST 5.5 interrupt to be active, I5.5 must be high, M5.5 low, and IE high.

The M7.5, M6.5 and M5.5 signals are called interrupt masks because they can disable a pending interrupt. For example, if M7.5 is high, it disables the AND gate it drives; this prevents a pending I7.5 interrupt form reaching the final output.

The RST 7.5, RST 6.5 and RST 5.5 interrupts are maskable, this means that they can be disabled by applying high M7.5, M6.5 and M5.5 signals. The TRAP is non-maskable; once it goes high and stays high, a TRAP interrupts appears at the final output.

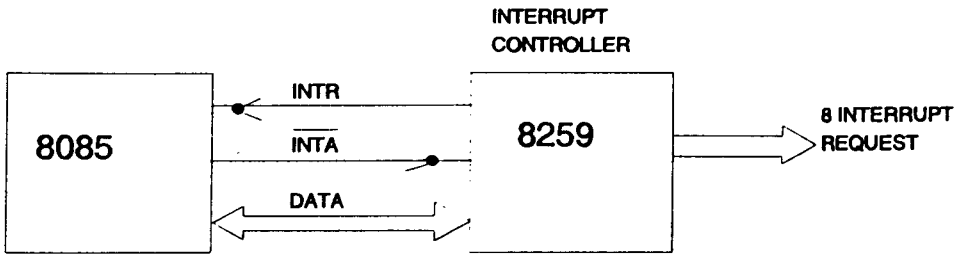
EXTENDING THE INTERRUPT SYSTEM

The TRAP, RST 7.5, RST 6.5 and RST 5.5 give us four interrupts-request lines. Sometimes, more than four interrupt-requests are needed. Then we can go for Polled Mode of interrupt in which more than one device is connected to a line. And the status of each device is polled the device requesting service is determined. This is a very boring task.

So, by making use of the 8259 and the INTR line the interrupt system can be extended.

The 8259, known as the interrupt controller, can receive upto eight interrupt requests from eight peripheral devices. The 8259 is to be programmed with the starting addresses for the eight service subroutines, one for each peripheral device. To service an interrupt, the 8259 sends an CALL address instruction to 8085. This calls the service subroutine for the interrupting peripheral device.

SENDING THE CALL



When a peripheral needs service, 8259 sends a high bit to the INTR input (Refer Figure 4.2). As soon as the 8085 recognises a high INTR, it returns a low INTRA* (interrupt Acknowledge) to the 8259. The 8259 responds by sending back the data CD, which is the opcode of CALL.

Next, the 8085 sends another low INTRA*. This INTRA* fetches the low address byte from the 8259. Finally, the 8085 sends another low INTRA*. This third INTRA* fetches the high address byte.

8085 INTERRUPTS IN MICROPOWER-i

It is not enough if you know just about the 8085 interrupts. You should actually understand the way in which they are connected in the kit, so that they are used in an efficient way. The following is a table of the vector locations of the interrupts that are used in Micropower-i with 8085 CPU.

Having known the way the interrupts are used in the trainer, you can write the program of your own for the Interrupt service subroutine.

The following points are worth nothing when concerned with interrupts and service subroutine.

i. As the vector table given above shows, the whole interrupt service routine cannot be written in the vectored location itself.

ii. So a jump to the service routine should be done at the vector location.

iii. If the interrupt was done during a main procedure, it is better to save all registers in the interrupt service subroutine and restore them back before going to the main procedure.

Interrupt	Used for	Jumps to RAM addr.
RST 0	-	
RST 1	Software interupt. Can be used by user at end of program to save reg.	-
RST 2	-	-
RST 3	-	-
RST 4	-	-
RST 5	Used in a similar way to RST 1 but will be used in Serial Monitor	-
RST 6	-	-
RST 7	-	-
TRAP	-	40FD
RST 7.5	Keyboard Interrupt.	-
RST 6.5	Single stepping Purpose.	-
RST 5.5	-	4070

Now, let us write a small program to familiarise with the concept of interrupts. Let us assume that RST 5.5 is connected to a peripheral and that is requesting service to store the data. at memory location 4200.

i. RST 5.5 jumps to RAM address 4070 upon it being active.

ii. So, at 4070 jump to your Interrupt Service Routine (ISR).

iii. Save Registers first that will be affected in the ISR.

iv. Restore registers and return to main program.
Let the ISR start at 4150. So at 4070 is

Memory Addr.	Object Codes	Mnemonics
4070	C3	JMP 4150
4071	50	
4072	41	

The ISR is as follows.

Memory Addr.	Object Codes	Mnemonics
4100	3E	MVI A,18H
4101	18	
4102	30	SIM
4103	76	HLT

INTERRUPT SERVICE ROUTINE

Memory Addr.	Object Codes	Mnemonics
4150	3E	MVI A,05H
4151	05	
4152	06	MVI B,05H
4153	05	

Memory Addr.	Object Codes	Mnemonics
4154	80	ADD B
4155	32	STA 4500
4156	00	
4157	45	
4158	CF	RST 1

Enter that op-codes to the micro processor board and execute the location 4100H. Now, upon an active RST 5.5, the ISR will execute the task desired by the processor and check the data '0A' is displayed in the location 4500H.

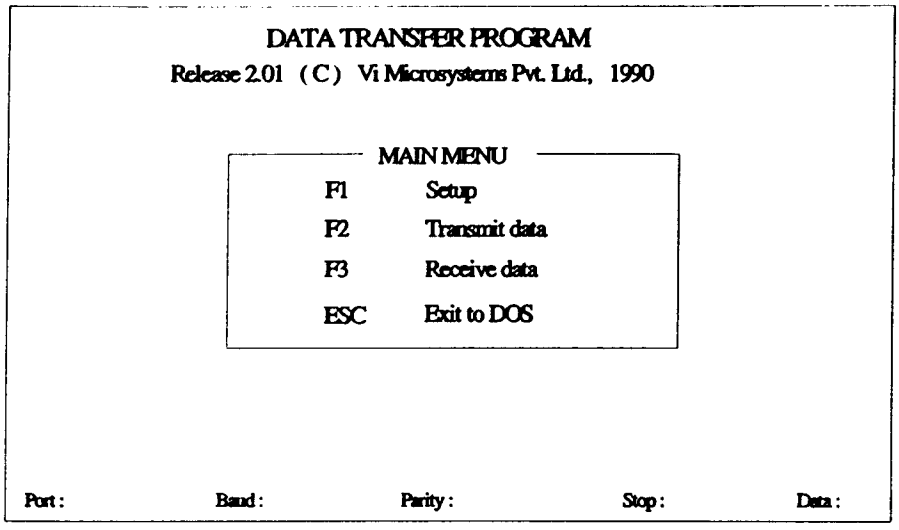
This is just an example to show about the response of interrupts. Follow similar procedures while using other interrupts.

DATACOM PACKAGE

DATACOM Release 2.01 is menu oriented user friendly communication package developed to communicate with any type of data between Vi range of Microprocessor trainer Kits with IBM PC/XT/AT. DATACOM helps the user to download any data from a Computer to a Trainer Kit and vice Versa. Using DATACOM Release 2.01 you can communicate either through COM 1 or COM 2 serial port by configuring the communication protocols such as baud rate, parity etc.

The 'MAIN MENU' is as given below. The function keys F1, F2, F3 and ESC are used as follows.

When you execute the DATACOM Release 2.01 by typing 'DATACOM' and <CR> in the DOS prompt the 'MAIN MENU' will appear on the screen as shown below and ask for four different options and display the current status of the host serial port.



Using the above function keys, you can establish a perfect link between IBM PC/XT/AT and Microprocessor Kit and transfer data between them.

Also, you can select your option by placing the cursor using UP (↑) and DOWN (↓) arrow keys in the number pad and Carriage Return. One more, by pressing the capital letter of the option you can select the option.

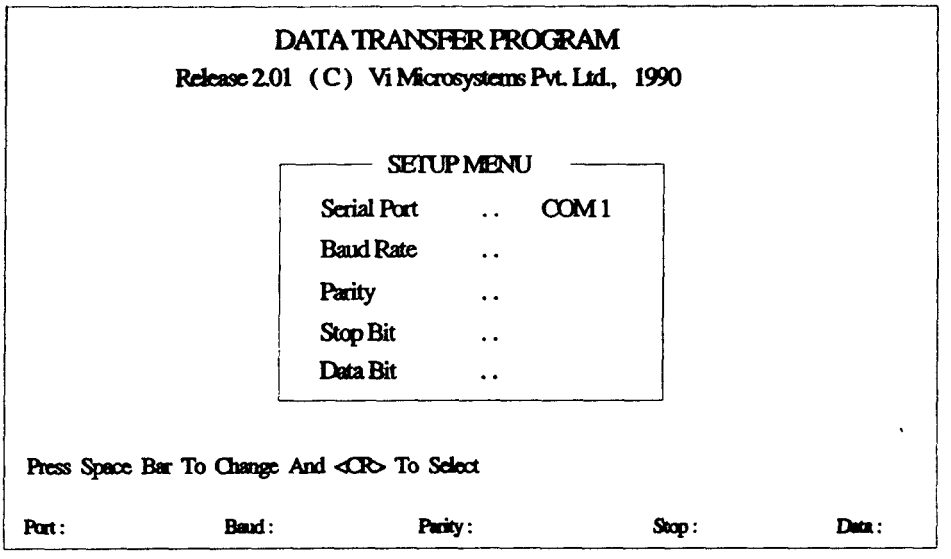
To establish a communication link between any kit with DATACOM package, the kit should have a serial port with software to send data and receive data in accordance with DATACOM format.

The DATACOM package expects '?????' marks to indicate the end of reception. Similarly it sends '??????' marks at the end of transmission to the Microprocessor Kit. Hence when the Kit is sending data to the PC, it should finish with '?????' marks. Similarly the kit expects '?????' marks for the end of reception.

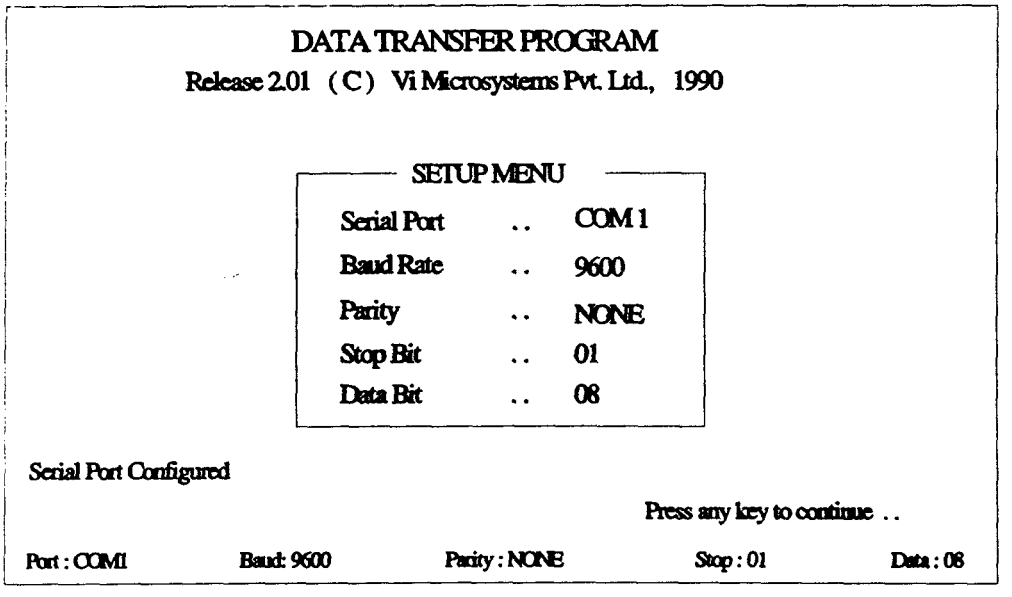
SETUP HOST SERIAL PORT

Communication through DATACOM release 2.01 is possible at any range of Baud rate starting from 110 to 9600 with facilities to control number of data bits, number of stop bits and parity on both COM1 and COM2 serial port.

The moment 'SETUP' option is selected 'SETUPMENU' will appear like this.



As in the 'SETUP MENU' by pressing the <Space Bar> you can change the protocols and confirmations done by Carriage Return. For Example after all parameters are selected, the menu will be



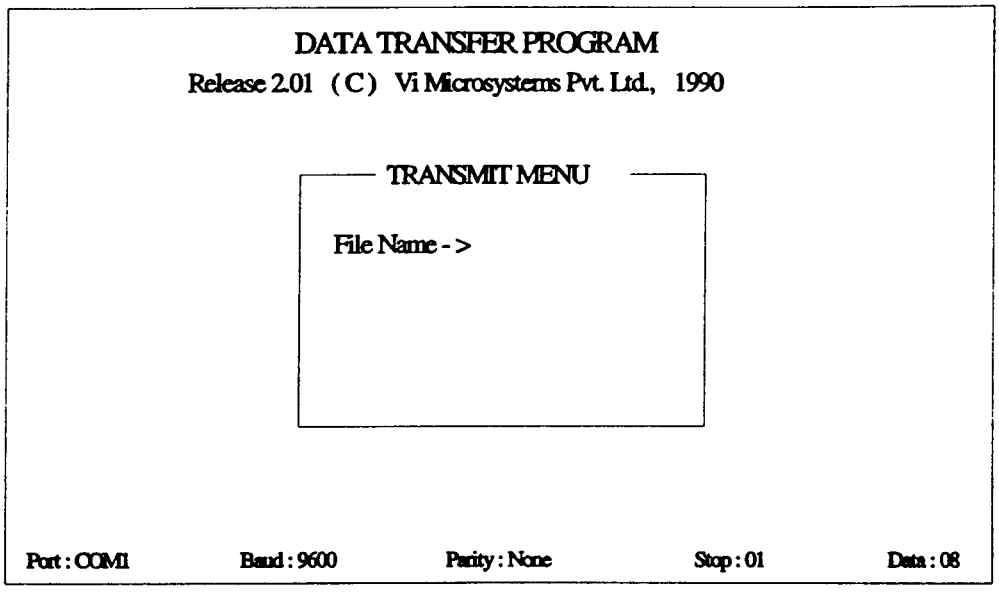
This means your host serial port is configured according to your protocols selected in this menu and this port is ready for communication.

CAUTION: When you invoke the DATACOM Release 2.01 from the DOS prompt, you must 'SETUP' the host serial port. Otherwise it will not respond properly.

TRANSMIT DATA TO KIT

To transmit data form IBM PC/XT/AT to the kit, first setup the kit as serial input mode and it should be ready to receive data form the host serial port. For further details about the kit serial port setup please refer the Kit's manual supplied by Vi Microsystems Pvt. Ltd., Chennai -96.

In DATACOM Release 2.01, by selecting 'Transmit data' option the control will go to the 'TRANSMIT MENU' as shown,



Let us file be, 'XXX' to be transmitted to the kit. Now you enter that file and press <CR>. Then DATACOM Release 2.01 will ask on question to the user to display the count while transmitting like this,

Display the count (Y/N)?

If the user response is 'N' or 'n' then the byte count will not display on the screen, otherwise the byte count will display on the screen.

Then if the file is present, it will wait for green signal from the user to start the data transmission. by displaying like this.

Press any key continue .....

Now make the trainer ready in the Serial receive mode by using SER. IN followed by the address. Once the trainer is ready to receive data, press any at the host to start transmission.

Else if the file is not present, it will give error message and wait for the user's response for futher action to be taken by displaying like this

Cannot open <file name> Wish to try again? (Y/N)

If the user's response in 'N' or 'n' the control goes to 'MAIN MENU', otherwise it will go to the 'File Name ->' prompt to transmit another file.

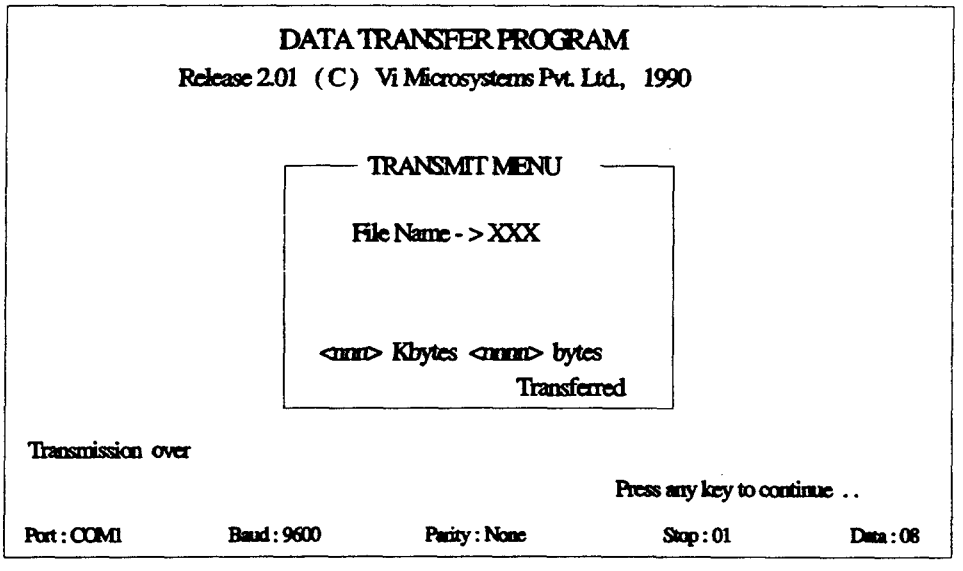
While transmission in progress the message and the count will display like this

<nnn> kbytes <nnnn> Bytes

Press ESC to abort

Transmission in progress....

By pressing 'ESC' key you can abort the current process. After data transmission is over the screen display will be,



This means the transmission is over without any problem. You can see that in the trainer also, command prompt will be displayed, telling that it has received the file in case of successful reception.

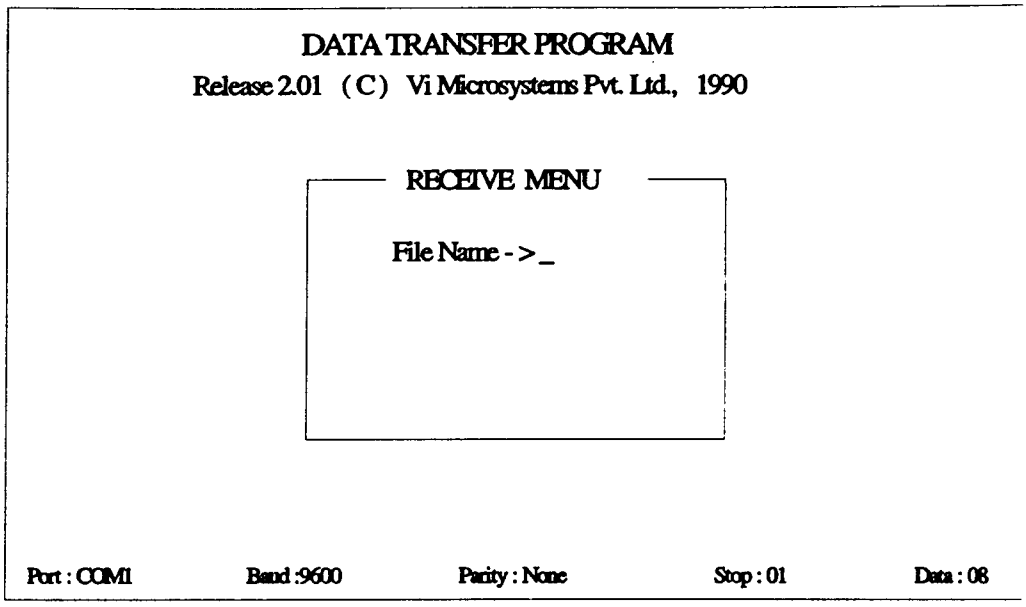
In case if the communication is not proper then, the message.

Time out Error Press any key to continue...

will appear and wait for a key, then the control goest to 'MAIN MENU'

RECEIVE DATA FROM KIT

To receive data from trainer to IBM PC/XT/AT. select 'Receive data' from the 'MAIN MENU'. Then another 'RECEIVE MENU' will appear like this



Enter the file name into which the data that are coimng from the trainer are to be stored and strike <CR>. If the file already exists then it wil ask whether it can be overwritten or not. After this task is over, it will wait for green signal from the user by displaying like this,

Press any key to continue ...

Now set the trainer to send the data to the host serial port by SER, OUT and the start address and end address of the block to be transmitted.

Now, visualize one practical problem, while uploading.

In case of down loading, the kit is first made to be in a constant receive mode and then the file is transmitted from PC.

In case of up loading, the PC cannot be made to be in a constant receive mode if there is any problem in the communication it has to be booted again, which is a long process.

Hence, when the computer is made to be in the mode receive, it checks the serial port for 50 times whether and put it in the file name mentioned. If it is not ready till that 50 times, then

Time out Error, Task Aborted

Press any key to continue ...

While checking for the data in the serial port, DATACOM will display,

Receiving data to file: <file name>

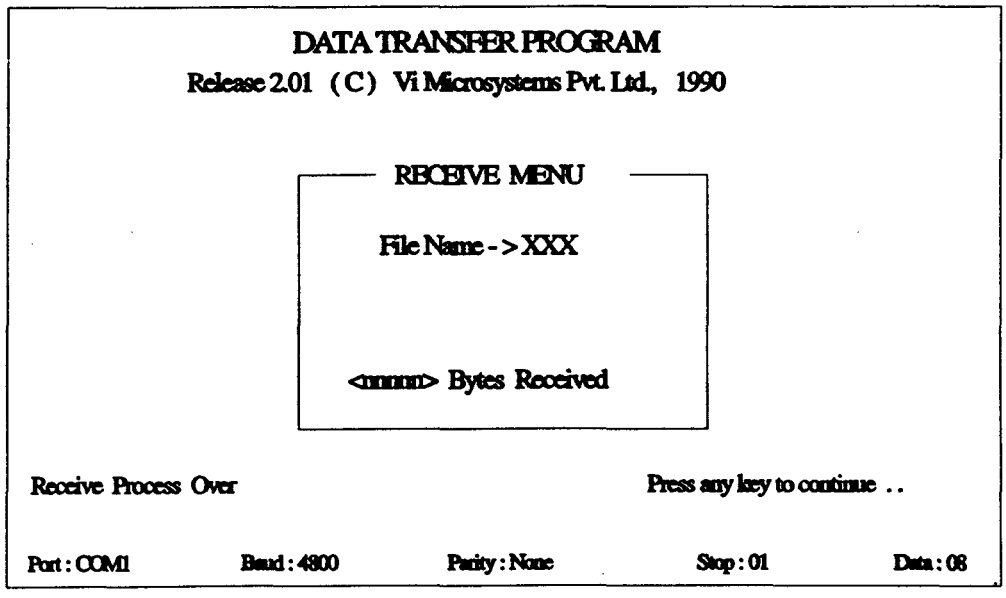
Hence additional care must be taken while transmitting to PC.

Make 'DATACOM' ready to receive. When it displays,

Receiving the file: <file name>
Press the 'NEXT' key in the Trainer.

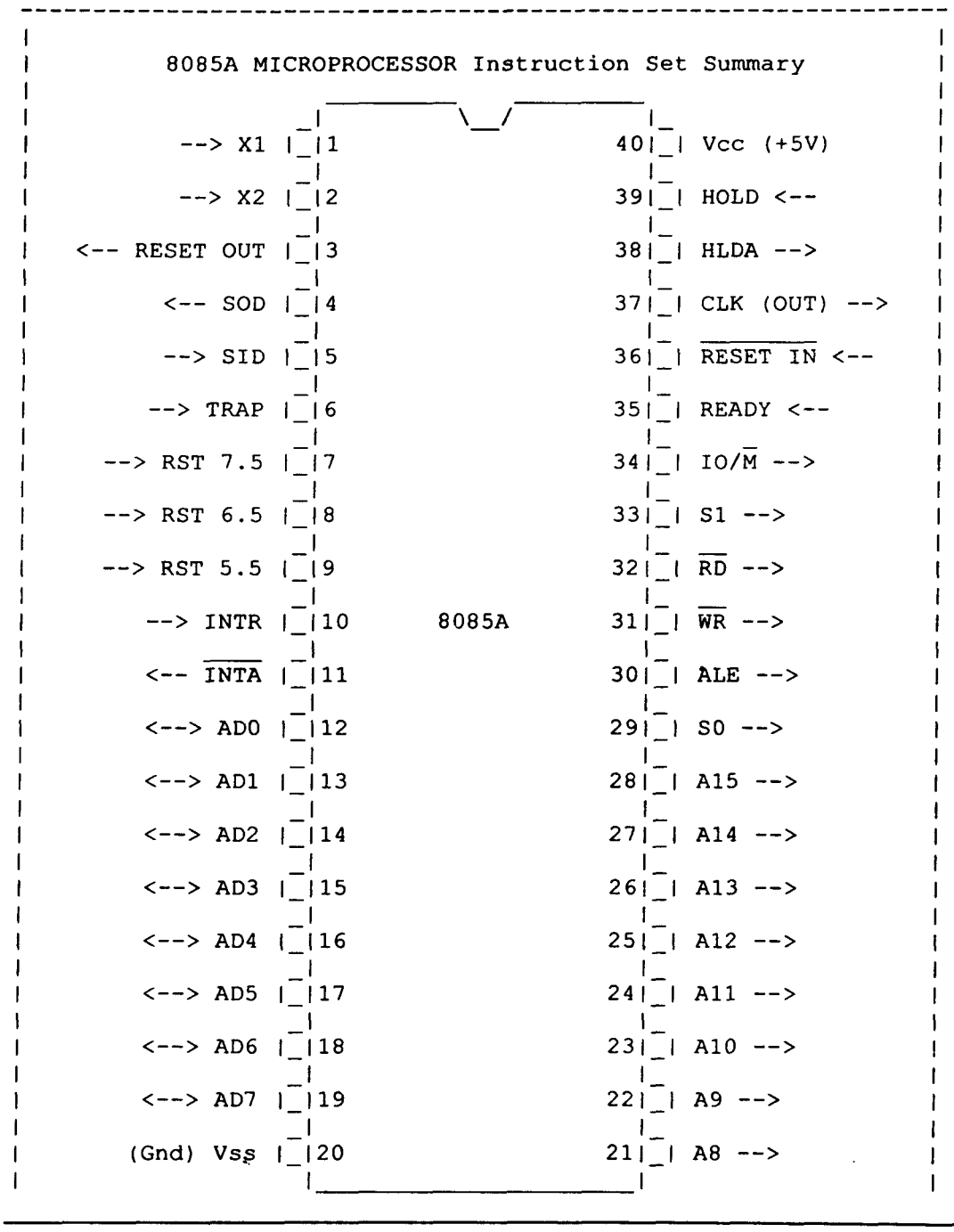
Also be sure that the final 5 values of your data is filled with '?' marks, which is the EOF mark for 'DATACOM'.

After successful reception, 'DATACOM' displays the number of bytes received and wait for a key form the user by displaying like below and then comes to the 'MAIN MENU'.



Quit the 'DATACOM' by selecting 'Exit to Dos' and verify the contents of 'XXX'.

The 8085 Instruction Set



Instructions can be categorized according to their method of addressing the hardware registers and/or memory.

Implied Addressing:

The addressing mode of certain instructions is implied by the instruction's function. For example, the STC (set carry flag) instruction deals only with the carry flag, the DAA (decimal adjust accumulator) instruction deals with the accumulator.

Register Addressing:

Quite a large set of instructions call for register addressing. With these instructions, you must specify one of the registers A through E, H or L as well as the operation code. With these instructions, the accumulator is implied as a second operand. For example, the instruction CMP E may be interpreted as 'compare the contents of the E register with the contents of the accumulator.

Most of the instructions that use register addressing deal with 8-bit values. However, a few of these instructions deal with 16-bit register pairs. For example, the PCHL instruction exchanges the contents of the program counter with the contents of the H and L registers.

Immediate Addressing:

Instructions that use immediate addressing have data assembled as a part of the instruction itself. For example, the instruction CPI 'C' may be interpreted as 'compare the contents of the accumulator with the letter C. When assembled, this instruction has the hexadecimal value FE43. Hexadecimal 43 is the internal representation for the letter C. When this instruction is executed, the processor fetches the first instruction byte and determines that it must fetch one more byte. The processor fetches the next byte into one of its internal registers and then performs the compare operation.

Notice that the names of the immediate instructions indicate that they use immediate data. Thus, the name of an add instruction is ADD; the name of an add immediate instruction is ADI.

All but two of the immediate instructions uses the accumulator as an implied operand, as in the CPI instruction shown previously. The MVI (move immediate) instruction can move its immediate data to any of the working registers including the accumulator or to memory. Thus, the instruction MVI D, OFFH moves the hexadecimal value FF to the D register.

The LXI instruction (load register pair immediate) is even more unusual in that its immediate data is a 16-bit value. This instruction is commonly used to load addresses into a register pair. As mentioned previously, your program must initialize the stack pointer; LXI is the instruction most commonly used for this purpose. For example, the instruction LXI SP,3OFFH loads the stack pointer with the hexadecimal value 30FF.

Direct Addressing:

Jump instructions include a 16-bit address as part of the instruction. For example, the instruction JMP 1000H causes a jump to the hexadecimal address 1000 by replacing the current contents of the program counter with the new value 1000H.

Instructions that include a direct address require three bytes of storage: one for the instruction code, and two for the 16-bit address

Register Indirect Addressing:

Register indirect instructions reference memory via a register pair. Thus, the instruction MOV M,C moves the contents of the C register into the memory address stored in the H and L register pair. The instruction LDAX B loads the accumulator with the byte of data specified by the address in the B and C register pair.

Combined Addressing Modes:

Some instructions use a combination of addressing modes. A CALL instruction, for example, combines direct addressing and register indirect addressing. The direct address in a CALL instruction specifies the address of the desired subroutine; the register indirect address is the stack

pointer. The CALL instruction pushes the current contents of the program counter into the memory location specified by the stack pointer.

Timing Effects of Addressing Modes:

Addressing modes affect both the amount of time required for executing an instruction and the amount of memory required for its storage. For example, instructions that use implied or register addressing, execute very quickly since they deal directly with the processor's hardware or with data already present in hardware registers. Most important, however is that the entire instruction can be fetched with a single memory access. The number of memory accesses required is the single greatest factor in determining execution timing. More memory accesses therefore require more execution time. A CALL instruction for example, requires five memory accesses: three to access the entire instruction and two more to push the contents of the program counter onto the stack.

The processor can access memory once during each processor cycle. Each cycle comprises a variable number of states. (See below and the appendix of "USING THE SDK-85 MICROPROCESSOR TRAINER"). The length of a state depends on the clock frequency specified for your system, and may range from 480 nanoseconds to 2 microseconds. Thus, the timing for a four state instruction may range from 1.920 microseconds through 8 microseconds. (The 8085 have a maximum clock frequency of 5 MHz and therefore a minimum state length of 200 nanoseconds.)

Instruction Naming Conventions:

The mnemonics assigned to the instructions are designed to indicate the function of the instruction. The instructions fall into the following functional categories:

Data Transfer Croup:

The data transfer instructions move data between registers or between memory and registers.

MOV	Move
MVI	Move Immediate
LDA	Load Accumulator Directly from Memory
STA	Store Accumulator Directly in Memory
LHLD	Load H & L Registers Directly from Memory
SHLD	Store H & L Registers Directly in Memory

An 'X' in the name of a data transfer instruction implies that it deals with a register pair (16-bits);

LXI	Load Register Pair with Immediate data
LDAX	Load Accumulator from Address in Register Pair
STAX	Store Accumulator in Address in Register Pair
XCHG	Exchange H & L with D & E
XTHL	Exchange Top of Stack with H & L

Arithmetic Group:

The arithmetic instructions add, subtract, increment, or decrement data in registers or memory.

ADD	Add to Accumulator
ADI	Add Immediate Data to Accumulator
ADC	Add to Accumulator Using Carry Flag
ACI	Add Immediate data to Accumulator Using Carry
SUB	Subtract from Accumulator
SUI	Subtract Immediate Data from Accumulator
SBB	Subtract rom Accumulator Using Borrow(Carry) Flag
SBI	Subtract Immediate from Accumulator Using Borrow Carry) Flag
INR	Increment Specified Byte by One
DCR	Decrement Specified Byte by One
INX	Increment Register Pair by One
DCX	Decrement Register Pair by One
DAD	Double Register Add; Add Content of Register air to H & L Register Pair

Logical Group:

This group performs logical (Boolean) operations on data in registers and memory and on condition flags.

The logical AND, OR, and Exclusive OR instructions enable you to set specific bits in the accumulator ON or OFF.

ANA	Logical AND with Accumulator
ANI	Logical AND with Accumulator Using Immediate Data
ORA	Logical OR with Accumulator
OR	Logical OR with Accumulator Using Immediate Data
XRA	Exclusive Logical OR with Accumulator
XRI	Exclusive OR Using Immediate Data

The Compare instructions compare the content of an 8-bit value with the contents of the accumulator;

CMP	Compare
CPI	Compare Using Immediate Data

The rotate instructions shift the contents of the accumulator one bit position to the left or right:

RLC	Rotate Accumulator Left
RRC	Rotate Accumulator Right
RAL	Rotate Left Through Carry
RAR	Rotate Right Through Carry

Complement and carry flag instructions:

CMA	Complement Accumulator
CMC	Complement Carry Flag
STC	Set Carry Flag

Branch Group:

The branching instructions alter normal sequential program flow, either unconditionally or conditionally. The unconditional branching instructions are as follows:

JMP	Jump
CALL	Call
RET	Return

Conditional branching instructions examine the status of one of four condition flags to determine whether the specified branch is to be executed. The conditions that may be specified are as follows:

NZ	Not Zero (Z = 0)
Z	Zero (Z = 1)
NC	No Carry (C = 0)
C	Carry (C = 1)
PO	Parity Odd (P = 0)
PE	Parity Even (P= 1)
P	Plus (S = 0)
M	Minus (S = 1)

Thus, the conditional branching instructions are specified as follows:

Jumps	Calls	Returns	
C	CC	RC	(Carry)
INC	CNC	RNC	(No Carry)
JZ	CZ	RZ	(Zero)
JNZ	CNZ	RNZ	(Not Zero)
JP	CP	RP	(Plus)
JM	CM	RM	(Minus)
JPE	CPE	RPE	(Parity Even)
JP0	CPO	RPO	(Parity Odd)

Two other instructions can affect a branch by replacing the contents or the program counter:

PCHL	Move H & L to Program Counter
RST	Special Restart Instruction Used with Interrupts

Stack I/O, and Machine Control Instructions:

The following instructions affect the Stack and/or Stack Pointer:

PUSH	Push Two bytes of Data onto the Stack
POP	Pop Two Bytes of Data off the Stack
XTHL	Exchange Top of Stack with H & L
SPHL	Move content of H & L to Stack Pointer

The I/O instructions are as follows:

IN	Initiate Input Operation
OUT	Initiate Output Operation

The Machine Control instructions are as follows:

EI	Enable Interrupt System
DI	Disable Interrupt System
HLT	Halt
NOP	No Operation

Mnemonic	Op	S%APC	~s	Description	Notes
ACI n	CE	*****	7	Add with Carry Immediate	A=A+n+CY
ADC r	8F	*****	4	Add with Carry	A=A+r+CY(21X)
ADC M	8E	*****	7	Add with Carry to Memory	A=A+[HL]+CY
ADD r	87	*****	4	Add	A=A+r (20X)
ADD M	86	*****	7	Add to Memory	A=A+[HL]
ADI n	C6	*****	7	Add Immediate	A=A+n
ANA r	A7	****0	4	AND Accumulator	A=A&r (24X)
ANA M	A6	****0	7	AND Accumulator and Memory	A=A&[HL]
ANI n	E6	**0*0	7	AND Immediate	A=A&n
CALL a	CD	-----	18	Call unconditional	-[SP]=PC,PC=a
CC a	DC	-----	9	Call on Carry	If CY=1(18~s)
CM a	FC	-----	9	Call on Minus	If S=1 (18~s)
CMA	2F	-----	4	Complement Accumulator	A=~A
CMC	3F	----*	4	Complement Carry	CY=~CY
CMP r	BF	*****	4	Compare	A-r (27X)
CMP M	BF	*****	7	Compare with Memory	A-[HL]
CNC a	D4	-----	9	Call on No Carry	If CY=0(18~s)
CNZ a	C4	-----	9	Call on No Zero	If Z=0 (18~s)
CP a	F4	-----	9	Call on Plus	If S=0 (18~s)
CPE a	EC	-----	9	Call on Parity Even	If P=1 (18~s)
CPI n	FE	*****	7	Compare Immediate	A-n
CPO a	E4	-----	9	Call on Parity Odd	If P=0 (18~s)
CZ a	CC	-----	9	Call on Zero	If Z=1 (18~s)
DAA	27	*****	4	Decimal Adjust Accumulator	A=BCD format
DAD B	09	----*	10	Double Add BC to HL	HL=HL+BC
DAD D	19	----*	10	Double Add DE to HL	HL=HL+DE
DAD H	29	----*	10	Double Add HL to HL	HL=HL+HL
DAD SP	39	----*	10	Double Add SP to HL	HL=HL+SP
DCR r	3D	****-	4	Decrement	r=r-1 (0X5)
DCR M	35	****-	10	Decrement Memory	[HL]=[HL]-1
DCX B	0B	-----	6	Decrement BC	BC=BC-1
DCX D	1B	-----	6	Decrement DE	DE=DE-1
DCX H	2B	-----	6	Decrement HL	HL=HL-1
DCX SP	3B	-----	6	Decrement Stack Pointer	SP=SP-1
DI	F3	-----	4	Disable Interrupts	
EI	FB	-----	4	Enable Interrupts	
HLT	76	-----	5	Halt	
IN p	DB	-----	10	Input	A=[p]
INR r	3C	****-	4	Increment	r=r+1 (0X4)
INR M	3C	****-	10	Increment Memory	[HL]=[HL]+1
INX B	03	-----	6	Increment BC	BC=BC+1
INX D	13	-----	6	Increment DE	DE=DE+1
INX H	23	-----	6	Increment HL	HL=HL+1
INX SP	33	-----	6	Increment Stack Pointer	SP=SP+1
JMP a	C3	-----	7	Jump unconditional	PC=a
JC a	DA	-----	7	Jump on Carry	If CY=1(10~s)
JM a	FA	-----	7	Jump on Minus	If S=1 (10~s)
JNC a	D2	-----	7	Jump on No Carry	If CY=0(10~s)
JNZ a	C2	-----	7	Jump on No Zero	If Z=0 (10~s)
JP a	F2	-----	7	Jump on Plus	If S=0 (10~s)
JPE a	EA	-----	7	Jump on Parity Even	If P=1 (10~s)
JPO a	E2	-----	7	Jump on Parity Odd	If P=0 (10~s)
JZ a	CA	-----	7	Jump on Zero	If Z=1 (10~s)

LDA a	3A	-----	13	Load Accumulator direct	A=[a]
LDAX B	0A	-----	7	Load Accumulator indirect	A=[BC]
LDAX D	1A	-----	7	Load Accumulator indirect	A=[DE]
LHLD a	2A	-----	16	Load HL Direct	HL=[a]
LXI B,nn	01	-----	10	Load Immediate BC	BC=nn
LXI D,nn	11	-----	10	Load Immediate DE	DE=nn
LXI H,nn	21	-----	10	Load Immediate HL	HL=nn
LXI SP,nn	31	-----	10	Load Immediate Stack Ptr	SP=nn
MOV r1,r2	7F	-----	4	Move register to register	r1=r2 (1XX)
MOV M,r	77	-----	7	Move register to Memory	[HL]=r (16X)
MOV r,M	7E	-----	7	Move Memory to register	r=[HL] (1X6)
MVI r,n	3E	-----	7	Move Immediate	r=n (0X6)
MVI M,n	36	-----	10	Move Immediate to Memory	[HL]=n
NOP	00	-----	4	No Operation	
ORA r	B7	**0*0	4	Inclusive OR Accumulator	A=Avr (26X)
ORA M	B6	**0*0	7	Inclusive OR Accumulator	A=Av[HL]
ORI n	F6	**0*0	7	Inclusive OR Immediate	A=Avn
OUT p	D3	-----	10	Output	[p]=A
PCHL	E9	-----	6	Jump HL indirect	PC=[HL]
POP B	C1	-----	10	Pop BC	BC=[SP]+
POP D	D1	-----	10	Pop DE	DE=[SP]+
POP H	E1	-----	10	Pop HL	HL=[SP]+
POP PSW	F1	-----	10	Pop Processor Status Word	{PSW,A}=[SP]+

Mnemonic	Op	SZAPC	~s	Description	Notes
PUSH B	C5	-----	12	Push BC	-[SP]=BC
PUSH D	D5	-----	12	Push DE	-[SP]=DE
PUSH H	E5	-----	12	Push HL	-[SP]=HL
PUSH PSW	F5	-----	12	Push Processor Status Word	-[SP]={PSW,A}
RAL	17	----*	4	Rotate Accumulator Left	A={CY,A}<-
RAR	1F	----*	4	Rotate Accumulator Righ	A=->{CY,A}
RET	C9	-----	10	Return	PC=[SP]+
RC	D8	-----	6	Return on Carry	If CY=1(12~s)
RIM	20	-----	4	Read Interrupt Mask	A=mask
RM	F8	-----	6	Return on Minus	If S=1 (12~s)
RNC	D0	-----	6	Return on No Carry	If CY=0(12~s)
RNZ	C0	-----	6	Return on No Zero	If Z=0 (12~s)
RP	F0	-----	6	Return on Plus	If S=0 (12~s)
RPE	E8	-----	6	Return on Parity Even	If P=1 (12~s)
RPO	E0	-----	6	Return on Parity Odd	If P=0 (12~s)
RZ	C8	-----	6	Return on Zero	If Z=1 (12~s)
RLC	07	----*	4	Rotate Left Circular	A=A<-
RRC	0F	----*	4	Rotate Right Circular	A=->A
RST z	C7	-----	12	Restart (3X7)	-[SP]=PC,PC=z
SBB r	9F	*****	4	Subtract with Borrow	A=A-r-CY
SBB M	9E	*****	7	Subtract with Borrow	A=A-[HL]-CY
SBI n	DE	*****	7	Subtract with Borrow Immed	A=A-n-CY
SHLD a	22	-----	16	Store HL Direct	[a]=HL
SIM	30	-----	4	Set Interrupt Mask	mask=A
SPHL	F9	-----	6	Move HL to SP	SP=HL
STA a	32	-----	13	Store Accumulator	[a]=A
STAX B	02	-----	7	Store Accumulator indirect	[BC]=A
STAX D	12	-----	7	Store Accumulator indirect	[DE]=A
STC	37	----1	4	Set Carry	CY=1
SUB r	97	*****	4	Subtract	A=A-r (22X)

SUB M	96	*****	7	Subtract Memory	A=A-[HL]
SUI n	D6	*****	7	Subtract Immediate	A=A-n
XCHG	EB	-----	4	Exchange HL with DE	HL<->DE
XRA r	AF	**0*0	4	Exclusive OR Accumulator	A=Axr (25X)
XRA M	AE	**0*0	7	Exclusive OR Accumulator	A=Ax[HL]
XRI n	EE	**0*0	7	Exclusive OR Immediate	A=Axn
XTHL	E3	-----	16	Exchange stack Top with HL	[SP]<->HL

PSW	-*01	Flag unaffected/affected/reset/set
S	S	Sign (Bit 7)
Z	Z	Zero (Bit 6)
AC	A	Auxilary Carry (Bit 4)
P	P	Parity (Bit 2)
CY	C	Carry (Bit 0)

a p	Direct addressing
M z	Register indirect addressing
n nn	Immediate addressing
r	Register addressing

DB n(,n)	Define Byte(s)
DB 'string'	Define Byte ASCII character string
DS nn	Define Storage Block
DW nn(,nn)	Define Word(s)

A B C D E H L	Registers (8-bit)
BC DE HL	Register pairs (16-bit)
PC	Program Counter register (16-bit)
PSW	Processor Status Word (8-bit)
SP	Stack Pointer register (16-bit)

a nn	16-bit address/data (0 to 65535)
n p	8-bit data/port (0 to 255)
r	Register (X=B,C,D,E,H,L,M,A)
z	Vector (X=0H,8H,10H,18H,20H,28H,30H,38H)

+ -	Arithmetic addition/subtraction
& ~	Logical AND/NOT
v x	Logical inclusive/exclusive OR
<- ->	Rotate left/right
<->	Exchange
[ ]	Indirect addressing
[ ]+ -[ ]	Indirect address auto-inc/decrement
{ }	Combination operands
(X)	Octal op code where X is a 3-bit code
If (~s)	Number of cycles if condition true