MICRO 86/88 LCD

User Manual

Version 2.0

Technical Clarification / Suggestion :

✍ / ☎

Technical Support Division,
Vi Microsystems Pvt. Ltd.,
Chennai - 600 096.
Ph: 044-2496 3142, 044-5201 5899
E-Mail :service@vimicrosystems.com
Website: vimicrosystems.com

Vi Microsystems Pvt. Ltd.,
Chennai - 96

MICRO 86/88 LCD

User Manual

Version 2.0

Technical Clarification / Suggestion :

✍ / ☎

Technical Support Division,
Vi Microsystems Pvt. Ltd.,
Chennai - 600 096.
Ph: 044-2496 3142, 044-5201 5899
E-Mail :service@vimicrosystems.com
Website: vimicrosystems.com

Vi Microsystems Pvt. Ltd.,
Chennai - 96

N

PREFACE

The ***"Micro-86/88 LCD User Manual/Student Workbook"*** has been devised for the Micro-86/88 LCD trainer which can work with 8086 CPU. It shall serve as a valuable aid to a learner, new to the software, hardware and interface programming aspects of the iAPX-86 family of processors.

The Manual has been formulated with consideration to the capabilities of a beginner and has a systematic organisation of all the chapters. Commencing with an introduction, each of the chapters has a whole complement of examples and exercises.

Chapter 1 serves as an introduction to the 8086 CPU.

Chapter 2 which has been further segmented into various sections deals with the software features of the CPUs under arithmetic, data movement, stack operations, flags and such other heads.

To further enhance your capability, ***Chapter 3*** gives you typical examples on hardware programming with the peripherals and the numeric coprocessor available on the trainer.

Chapter 4 Deals communication of kit and computer using our data communication package.

While working through the exercises, please refer to the Appendices for information on Instruction Set. To promote your skills to higher levels, you can consult the titles quoted in the Bibliography of References.

You shall no doubt emerge as a number one programmer, after working with the trainer along with this manual in all sincerity. Suggestions to enhance the substance provided in this manual are welcome.

Write to:
The Customer-Support Division,
Vi Microsystems Pvt. Ltd.,
Plot No.75, Electronics Estate,
Perungudi, Chennai - 600 096.
Phone # : (044) 24961842, 2496 1852.
Fax # : (044) 24961536.
E-mail : service@vimicrosystems.com

TABLE OF CONTENTS

LIST OF APPENDICES

CHAPTER-1

AN INTRODUCTION TO THE 8086/8088

This chapter is intended to serve as an introduction to the enhanced software capabilities of the 8086/8088 family of microprocessors and shall not deal with the hardware organization of the chip. User's are requested to read 8086/8088 throughout this manual, as this trainer support 8086 the CPU and software features of both the CPU's are similar.

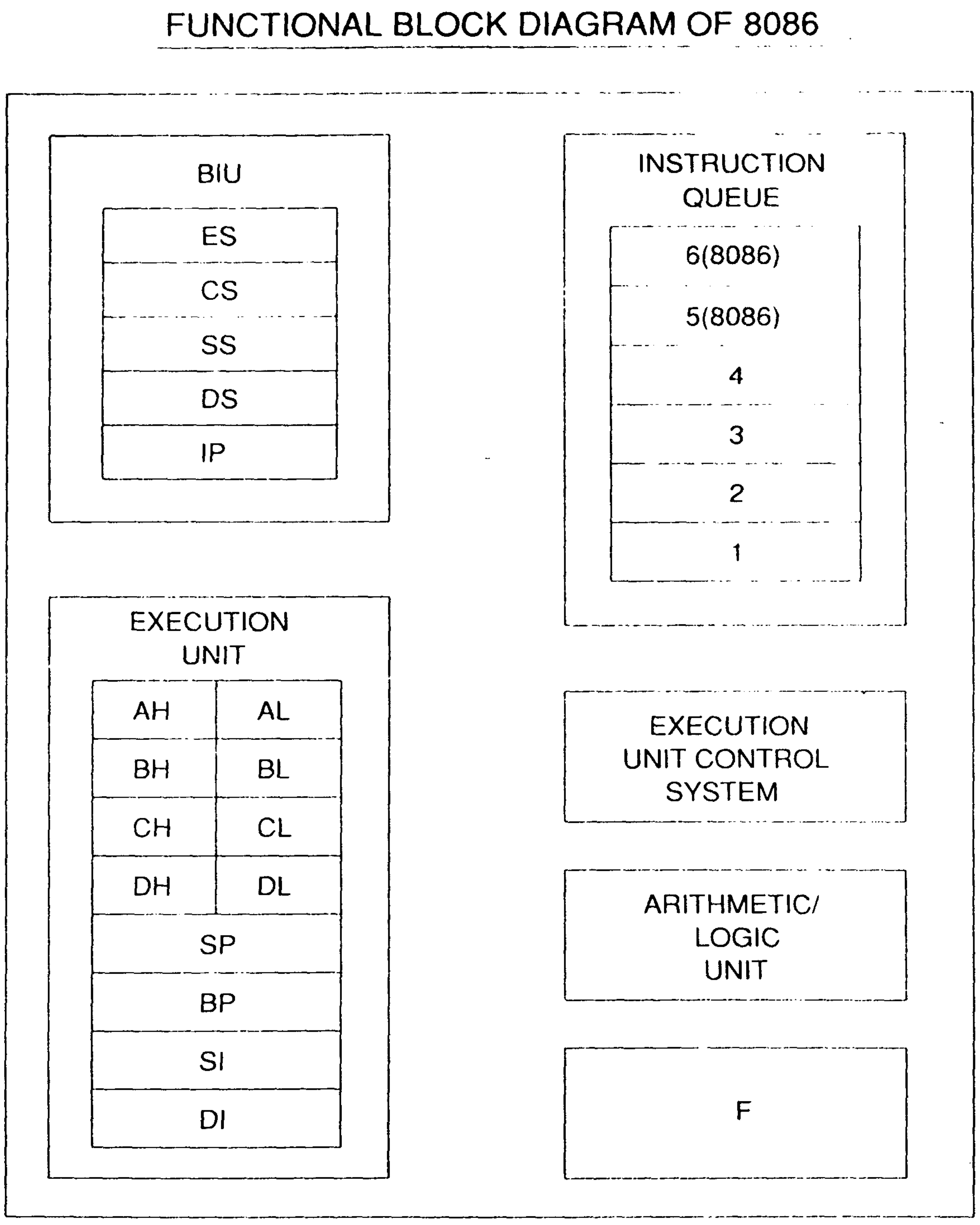
1.1 Registers in the 8086:

To get an idea of what either CPU has, go through the Figure F1.1. All the blocks are similar for both the CPUs 8086 and 8088, except the Instruction Queue, which is 6 bytes for the 8086.

The list of registers present in either CPU is given below. The segment, pointer and index registers may seem novel concepts to the learner. But it is the power that is provided by the segment registers that makes the 8086/8088 family of CPUs very powerful.

CS	-	CODE SEGMENT REGISTER
SS	-	STACK SEGMENT REGISTER
DS	-	DATA SEGMENT REGISTER
ES	-	EXTRA DATA SEGMENT REGISTER
IP	-	INSTRUCTION POINTER
DI	-	DESTINATION INDEX REGISTER
SI	-	SOURCE INDEX REGISTER
AX	-	ACCUMULATOR PAIR [AH, AL]
BX	-	BX PAIR [BH, BL]
CX	-	CX PAIR [CH, CL]
DX	-	DX PAIR [DH, DL]
BP	-	BASE POINTER
SP	-	STACK POINTER
F	-	FLAGS REGISTER

Functional Block Diagram of 8086



BIU —— Bus Interface Unit

Figure F1.1

1.2 Software Features in 8086:

The forthcoming sections highlight the following with reference to 8086.

1. Segmented Addressing.
2. Arithmetic Instructions.
3. String Operations and Indexed Addressing.
4. Procedure Calls.

These features are improvements over the usual software capabilities of 8 bit microprocessor.

1.2.1 Segmented Addressing:

The physical address of any fetch or write operation with the memory is 20-bits, while any instruction is associated only with a 16-bit address. The additional 4bits are obtained from the contents of the corresponding segment register, say code segment in the case of an instruction fetch and so on. Hence in effect a 1 megabyte memory of 64KB overlapping segments is addressable with the 8086.

This also facilitates easy relocation of code at the required segment address.

With the CS = 0200, the IP = 2BC0 and the displacement (Disp) = 5119 a branch instruction would cause execution at the physical address (PA) 09DC9 as shown below.

```
IP    =     2BC0 +
Disp =      5119
           ________
            7CD9 +
 CS   =     02000
           __________
 PA   =     09CD9
           _________
```

1.2.2 Arithmetic Instructions:

Byte, word and double word operations:

By virtue of the fact that the register organisation of the CPU is 16-bit, the instruction set supports byte, word as well as longword operations. Hence a 16-bit addition is a simple add of two register pairs and a 32-bit addition is two 16-bit additions. This means that both

```
          ADD   AH,AL  ; (8-bit addition)
  and  ADD   AX,BX  ; (16-bit addition) are permitted with the 8086.
```

Arithmetic ADJUST Instructions:

A host of ADJUST instructions for manipulation of **decimal operations** is an added feature of the 8086.

The DAA (Decimal Adjust for packed BCD add),
DAS (Decimal Adjust for packed BCD subtraction),
AAA (Adjust for unpacked BCD addition)
AAS (Adjust for unpacked BCD subtraction) and
AAM (Adjust for unpacked BCD multiplication) are remarkable examples.

When subtraction of negative numbers is involved then the negative number has to be sign extended when the operands are of different sizes. This is facilitated by the

CBW (Convert Byte to Word) and the
CWD (Convert Word to Double Word) instructions.

MUL and DIV:

The 8086 has provision for both 8 and 16-bit multiplication and division of a long word by a word and a word by a byte in

MUL (8 and 16-bit)
DIV (32 and 16-bit)

In addition the IMUL and IDIV instructions perform signed multiply and divide.

1.2.3 String Operations and Indexed Addressing:

The 8086 instruction set has a set of instructions which it calls string primitives and an array can be operated upon as a whole entity by employing these primitives. They can be listed as,

MOVS : Move String byte or word
CMPS : Compare two Strings
LODS : Load AL or AX with a String content
STOS : Store AL or AX in a String
SCAS : Scan a String for a byte or word.

By default all string operations use

SI + DS * 16_{10} as the source address and
DI + ES * 16_{10} as the destination address unless otherwise specified.

To move a block of memory of FF bytes to another location,the 8085 would require initialising an index register say the HL pair for the source address and move the content to the destination address after moving the content of the source to the accumulator. then increment both the addresses and perform the same operation till end ofarray which again requires a counter.

All this tedium is avoided in the 8086. The following set of instructions conveniently performsthe above (STRING1 and STRING2 being the source and destination).

```
MOV        SI,OFFSET STRING1
MOV        DI,OFFSET STRING2
REP   MOVSB       STRING1,STRING2
```

Another advantage is that the CX register can be effectively used as a counter with the help of the LOOP, JCXZ, LOOPZ, LOOPNZ instructions. Hence the CX register can be conveniently used to execute a SOFTWARE DELAY.

1.2.4 Procedure Calls:

In addition to all the above mentioned,the 8086 provides subroutine CALL by means of PROCEDURES. A procedure may be in

i) The same code segment as the calling statement.

ii) A different code segment but in the same source module.

iii) A different source module and segment and accordingly declared as NEAR, FAR or PUBLIC and EXTRN.

To supplement the above there is a host of CONDITIONAL AND UNCONDITIONAL JUMPS and SHIFT AND ROTATE instructions.

1.3 Conclusion:

As a concluding word we can say this:

This workbook is intended to serve as a guide to the vast area of applications involving the 8086 and hence has the chapters and examples arranged in an increasing order of complexity and utility.

The instructions that have not been utilised herein may be considered by the learner as a new terrain to explore.

The exercises that follow each example shall certainly challenge the intellect and transform the learner into a proficient programmer. You can always contact us in case of difficulty.

List of Symbols and Notations Used:

S.No.	ABBREVIATION	DESCRIPTION
1.	MSB	Most Significant Bit
2.	LSB	Least Significant Bit
3.	HN	Higher Nibble
4.	2nd LN	Bits 8 to 11 of a Word
5.	LN	Lower Nibble
6.	1st LN	Bits 4 to 7 of a word
7.	MSD	Most Significant Digit (Byte)
8.	LSD	Least Significant Digit
9.	[XXXX] = YY	Contents of the memory location XXXX is the byte YY
10.	[XXXX] = YYZZ	Contents of memory location XXXX is ZZ Contents of memory location XXXX+1 i s YY
11.	XXXX XXXXB	Binary
12.	RAM	Random Access Memory
13.	*	Active Low Signal

Note: All the user programs start at origin 1000 and the default code segment is 0000. The addresses are all in hex form.

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 8086 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 Experimental sections, each in turn 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 REQUIREDresults.

iii) Proceed to perform the EXERCISES totest 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 M - 86/88 LCD.

This Chapter has been organised as 6 sections, namely:

1. Data Transfer Operations.
2. Flag Operations.
3. Arithmetic, Logical, Shift and Rotate Operations.
4. Loop and Branch Operations.
5. Array and String Operations.
6. Stack and Procedures.

2.2 Data Transfer Operations:

This section is intended to introduce the programmer to the various addressing modes of the 8086, which are classified into FIVE groups:

i) **Register and Immediate Addressing** for accessing immediate data and data in registers.
ii) **Memory Addressing** for accessing data in memory.
iii) **I/O Addressing** for accessing I/O Ports.
iv) **Relative Addressing** modes.
v) **Implied Addressing** modes.

The following programs shall illustrate register, immediate, memory and relative addressing modes each with an example.

2.2.1 Example 1 - Register and Immediate Addressing Mode:

Objective:

To initialise the accumulator, a data register, and a segment register to a particular value.

Theory:

While using immediate addressing mode, the data is specified within the instruction. These data are located in the memory addressed by the CS and IP registers,because the data is considered part of the instruction fetched from memory. The source operand can only be immediate.

Example:

In this example the registers are initialised as below.

(AH) = FF
(CS) = AX
(BX) = 2000

Program:

```
MOV  AH,AF        ;Initialise AH to FF
MOV  AH,AL        ;move the data in AL to AH
MOV  CS,AX
MOV  BX,2000      ;BX contains 2000
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C6	MOV AH,FF
1001	C4	
1002	FF	
1003	88	MOV AH,AL
1004	C4	
1005	8E	MOV CS,AX
1006	C8	
1007	C7	MOV BX,2000
1008	C3	
1009	00	
100A	20	
100B	F4	HLT

Procedure:

i) Enter the above opcodes in RAM memory from location 1000, using SUB command.

ii) Using STEP command, as illustrated in the manual M - 86/88 LCD Technical Reference, execute the program instruction by instruction.

iii) After each instruction verify register contents and see that they are initialised to the determined values.

Discussion:

From the above program, it should be clear that both 16 and 8 bit data transfers are possible with the 8086 instruction set. The segment registers can be initialised only using register addressing or memory addressing as shall be seen from the following example.

Exercises:

i) Initialise register CX to value FFFF and register AX to value 0000. Write a program to exchange the contents of both these registers.

Data : (AX) = 0000
(CX) = FFFF

Result:(AX) = FFFF
(CX) = 0000

2.2.2 Memory Addressing:

When memory addressing is involved, the operands are to be obtained from a location in memory. Memory addressing includes six different modes of addressing. In each of these modes the effective address is obtained as below.

Direct Addressing:

The effective address is obtained directly from the displacement field of the instruction.

Register Indirect Addressing:

The effective address is obtained from either a pointer register (BX or BP) or an index register (SI or DI).

Based Addressing Mode:

Effective address is obtained by adding a displacement to the contents of BX or BP. For memory access BX and for stack addressing BP are used along with DS and SS respectively.

Indexed Addressing Mode:

With a displacement, the contents of SI or DI are added to obtain the effective address.

Based Indexed Addressing Mode:

Combining the above two, the effective address is obtained as the sum of a displacement, the contents of an index register and a base register.

Relative Addressing:

The operand address is an 8-bit displacement relative to the current contents of the IP.

2.2.3 Example 2 - Direct and Indirect Addressing:

Objective:

To illustrate the use of LEA, LDS, LES instructions and to initialise registers using these instructions.

Theory:

In the LEA instruction the offset address is loaded into the destination register. In the LDS and LES instructions DS and ES registers are loaded with an effective address. In the MOV [BX], DX instruction the memory location pointed to by BX is loaded with the contents of DX register.

Example: The registers are initialised as below in this example.

(AX)	=	array	:	1200 if array starts at 1200
(CX)	=	[1300]		
(ES)	-	[1302]		
(DX)	=	[1400]		

Using the final MOV instruction, the content of memory location 1400 is moved to the memory location pointed to by AX.

Program:

```
LEA   AX,1200      ; AX gets 1200
LES   CX,[1300]    ; CX gets contents of 1300.
                   ; ES gets contents of 1302
MOV   DX,[1400]    ; DX gets contents of 1400.

MOV   AX,DX        ; Contents of 1400 moved to
                   ; location pointed by AX
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8D	LEA AX, [1200]
1001	06	
1002	00	
1003	12	
1004	C4	LES CX, [1300]
1005	0F	
1006	00	
1007	13	
1008	C7	MOV DX, 1400
1009	C2	
100A	00	
100B	14	
100C	89	MOV AX, DX
100D	D0	
100E	F4	HLT

Procedure:

i) Enter the above opcodes in RAM memory from location 1000, using SUB command.

ii) Using STEP command, as illustrated in the manual M-86/88 LCD Technical Reference, execute the program instruction by instruction.

iii) After each instruction verify register contents and see that they are initialised to the required values.

Discussion:

Two new instructions have been illustrated in this example. These examples are only to understand the use of various instructions and the program as such is a do-nothing program.

In the next example we shall see the more complex methods of addressing.

2.2.4 Example 3 - Based & Based Indexed Addressing:

Objective:

To illustrate the more advanced modes of addressing namely based indexed and based addressing.

Theory:

BX and BP are the base registers and SI and DI are the index registers. The effective address is obtained using these registers. Since only data in memory are addressed the physical 20 bit address is obtained using the DS register.

Example:

In the example below AD is the location 1050 and RES is the location 1052. The SI index register is loaded using the LEA instruction and the AX register is loaded with an effective address using the based indexed addressing mode.
[1050] = FF

Program:

```
MOV  BX,1200
LEA  SI,[BX+AD]
MOV  AX,1000
MOV  BP,AX
MOV  AX,RES
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C7	MOV BX, 1200
1001	C3	
1002	00	
1003	12	
1004	8D	LEA SI, [BX+1050]
1005	B7	
1006	50	
1007	10	
1008	C7	MOV AX,1000
1009	C0	
100A	00	
100B	10	
100C	89	MOV BP, AX
100D	C5	
100E	C7	MOV AX,1052
100F	C0	
1010	52	
1011	10	
1012	F4	HLT

Procedure:

i) Enter the above opcodes in RAM memory from location 1000, using SUB command.

ii) Using STEP command, as illustrated in the manual M - 86/88 LCD Technical Reference, execute the program instruction by instruction.

iii) After each instruction verify register contents and see that they are initialised to the required values.

Discussion:

With this example we complete the preliminaries involved in initialising the registers of the 8086 using various addressing modes. The implied addressing mode is left as an exercise to the learner. In the next section we shall learn about the various flags of the 8086.

2.3 Flag Operations:

The 8086 has one 16-bit Flags Register, also referred to as a Status Register or Program Status Word. This register may be illustrated as follows:

-	-	-	-	O	D	I	T	S	Z	-	A	-	P	-	C
15							8	7							

C - Carry Flag :

Set on high-order bit carry or borrow; cleared otherwise.

P - Parity Flag :

Set if low-order 8-bits of result contain an even number of 1 bits; cleared otherwise.

A - Auxiliary Carry Flag :

Set on carry or borrow to the low-orderCarry Flag4-bits of AL; cleared otherwise.

Z - Zero Flag :

Set if result of preceding operation is zero; cleared otherwise.

S - Sign Flag :

Set equal to high-order bit of result (0 if positive, 1 if negative).

T - Single Step Flag :

Once set, a single-step interrupt occursStep Flag after the next instruction executes; T flag is cleared by the Single-Step interrupt.

I - Interrupt enable flag :

When set, maskable interrupts will causeEnable Flag the CPU to transfer control to an interrupt vector specified location.

D - Direction flag :

Causes string instructions to auto-Flagdecrement the appropriate index register when set; clearing it causes auto-increment.

O - Overflow flag :

Set if the signed result cannot be Flagexpressed within the number of bits in the destination operand; cleared otherwise.

The above mentioned flags always monitor the accumulator and the registers and signal the presence of a specific condition. It is only based upon the status of these flags that the conditional branches occur. So, a profound knowledge about the FLAGS REGISTER is very essential to enhance your ability as a programmer.

2.3.1 Example 4 - Flag Manipulation:

Objective:

To manipulate the Carry (C), Parity (P), Auxiliary Carry (A), Zero (Z), Sign (S), Direction (D), Interrupt enable/disable (I) and the Overflow (O) flags and to determine the conditions that set and reset these flags.

Example:

The example given below is a "DO NOTHING" program. It just manipulates the flags and enables you to visualise conditions that set or reset a particular flag. Since you have to view register contents, you should use the TRACE COMMAND for this purpose.

Program:

```
MOV  AL,F5
ADD  AL,0C
XOR  AL,AL
SUB  AL,04
DAA
STI
CLI
STD
CLD
AND  AL,00
INC  AL
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C6	MOV AL, F5
1001	C0	
1002	F5	
1003	80	ADD AL, 0C
1004	C0	
1005	0C	
1006	30	XOR AL, AL
1007	C0	
1008	80	SUB AL, 04
1009	E8	
100A	04	
100B	27	DAA
100C	FB	STI
100D	FA	CLI
100E	FD	STD
100F	FC	CLD
1010	80	AND AL, 00
1011	E0	
1012	00	
1013	FE	INC AL
1014	C0	
1015	F4	HLT

Procedure:

i) Enter the program given above from 1000.

ii) Enter the TRACE command with address 1000 from where single step is to be done and press enter key.

iii) Press INT key and using the reg. command view the register contents.

iv) Press INT key to terminate the register command.

v) Once again enter TR and press enter key now the next instruction is executed and repeat the step 3 and 4 upto HLT instruction

Step No.	Mnemonics	Flags Register									Accumulator	Program Counter
		O	D	I	T	S	Z	A	P	C		
1	MOV AL,F5											
2	ADD AL, 0C											
3	XOR AL,AL											
4	SUB AL,04											
5	DAA											
6	STI											
7	CLI											
8	STD											
9	CLD											
10	AND AL,00											
11	INC AL											
12	HLT											

Discussion:

In the above example, you stepped through a simple "DO-NOTHING" program that manipulated the 8086 flags. You can verify for yourself the reason for the setting and clearing of flags from the table, after single stepping through the program.

These flag bits are used to automatically detect specificconditions within the ALU of the microprocessor. They can be conveniently tested by specialised instructions, so that specific action can be taken in response to the condition tested. Do not feel intimidated by the apparent complexity of the flags. Their use will become clearer as we examine programs.

Exercises:

i) State the flags that are to be tested for the following branch instructions.

JNGE	JPE
JG	JNS
JA	JNB
JNB	JAE
JO	JS

ii) With the use of the Direction Flag, verify the change in SI and DI registers depending on the status of that flag, when used in string operations.

iii) Examine the following instructions. Store the status of the flag register after the execution of each instruction from the memory location starting at 1200.

```
MOV  AX,9A
SUB  AL,B0
AND  AL,93
DAA
SHR  AL,1
AAA
```

2.4 Arithmetic, Logical, Shift & Rotate Instructions:

In this section which is the most exhaustive you will learn about the four basic arithmetic operations ADD, SUB, MUL and DIV, a few of the logical instructions, shift and rotate instructions.

For these instructions, unless the source operand is immediate, one of the operands must be in a register. The other may have any addressing mode as discussed in the first section.

The exercises in this section shall help you to learn most of the instructions of the 8086.

2.4.1 Example 5 - One's Complement of a 16-Bit Number:

Objective:

To find the one's complement of the data in register pair AX and store the result at 1400.

Theory:

In the one's complement of a binary number the ones are changed to zeros and vice versa. It is one way of representing negative numbers. All negative numbers start with a 1 at the MSBit.

For instance considering the hex number 5600

5600	=	0101 0110 0000 0000
One's complement	=	1010 1001 1111 1111
	=	A9FF

Example:

The example given is to find the one's complement of 1234 and store it in memory location 1400.

Input :			
Data:	(AX)	=	0001 0010 0011 0100 = 1234
Result:	[1400]	=	1110 1101 1100 1011 = EDCB

Program:

```
MOV  AX,1234            ;word in AX register
NOT  AX
MOV  [1400],AX          ;One's complement in
                        ;AX and memory
HLT
```

Object Codes:

Memory Address	Object Codes	Mnemonics
1000	C7	MOV AX, 1234
1001	C0	
1002	34	
1003	12	
1004	F7	NOT AX
1005	D0	
1006	89	MOV [1400], AX
1007	06	
1008	00	
1009	14	
100A	F4	HLT

Procedure:

i) Enter the above mnemonics into RAM memory from 1000 using the assembler command.

ii) Using GO command execute the program and enter 1000. This is the address from where execution of your program starts.

iii) Press ENTER key to start execution.

iv) Reset the kit using RESET key.

v) Check up for the result at 1400.

Discussion:

In binary arithmetic, the two's complement of a number is the negative of that number. It is obtained by adding 1 to the one's complement or by inverting all bits after the first '1' starting from the LS Bit. For data transfer with peripherals using negative assertion logic, the above is utilised.

Exercises:

i) Obtain the one's complement of a 16-bit number stored at 1100 and store the result at 1200.

Sample Data:	[1100]	7654
Result:	[1200] =	89AB

ii) Obtain the two's complement of a number at memory location 1100 and store the result at 1200.

Sample Data:	[1100] =	7654
Result :	[1200] =	89AC

2.4.2 Example 6 - Masking Off Bits Selectively:

Objective:

To clear 8 selected bits, the 2nd HN and the HN in a 16 bit number.

Theory:

The logical AND instruction is used for masking off bits. The bits which have to be cleared are to be ANDed with a logical zero and the other bits are to be high. Hence to achieve the above objective, AND with 0F0F.

Example: The 16 bit number is at location 1200 and the result is at location 1400.

Input:	[1200] =	FF
	[1201] =	FF
Result:	[1400] =	0F
	[1401] =	0F

Program:

```
MOV  AX,[1200]     ;data in AX
AND  AX,0F0F
MOV  [1400],AX     ;result in 1400
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV AX,[1200]
1001	06	
1002	00	
1003	12	
1004	81	AND AX, 0F0F
1005	E0	
1006	0F	
1007	0F	
1008	89	MOV [1400], AX
1009	06	
100A	00	
100B	14	
100C	F4	HLT

Procedure:

The procedure outlined for previous exercises is to be followed for this program also.

Discussion:

The AND instruction is generally used to mask off bits and to check for the status of certain devices by checking the bits in the status word output by them.

Exercises:

i) Write a program to mask off the MSBit and LSBit of a 16-bit word at 1100 and store result at 1200.

```
Sample:    [1100] = 35
           [1101] = F6

Result:    [1200] = 34
           [1201] = 76
```

ii) Write a program to store successive powers of 2 starting from memory location 1100, given data input 0000.

```
Result:   [1100] =    8000
          [1102] =    4000
          [1104] =    2000
```

[1106] =	1000
[1108] =	0800
[110A] =	0400
[110C] =	0200
[110E] =	0100
[1110] =	0080
[1112] =	0040
[1114] =	0020
[1116] =	0010
[1118] =	0008
[111A] =	0004
[111C] =	0002
[111E] =	0001

[Hint: Mask off successive bits in FFFF]

2.4.3 Example 7 - Setting Bits Selectively in a 16-Bit Number:

Objective:

To set the LN and the 2nd LN of a 16-bit number in AX and store the result at memory location 1100.

Theory:

The logical OR of a bit with 1 produces a result of 1. Hence bits can be set selectively by ORing the particular bit with a 1.

The result required is 0000 1111 0000 1111B.

Example:

The accumulator AX is initially cleared.

Input : (AX) = 0000
Result: [1100] = 0F0F

Program:

```
MOV  AX,0         ;clear accumulator
OR   AX,0F0F      ;OR with 0F0F
MOV  [1100],AX    ;Store result in memory
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C7	MOV AX, 0000
1001	C0	
1002	00	
1003	00	
1004	81	OR AX, 0F0F
1005	C8	
1006	0F	
1007	0F	
1008	89	MOV [1100],AX
1009	06	
100A	00	
100B	11	
100C	F4	HLT

Procedure:

The same procedure as illustrated in the previous examples. Try setting various bits.

Discussion:

The logical OR instruction can likewise be used to check whether a particular bit is high or notin any word or byte input from a peripheral.

Exercises:

The number at location 1100 is FFFF.

i) Logical AND this with 0F0F and store the result at 1200.

Sample: [1100] = FFFF
Result: [1202] = 0F0F

ii) Logical OR this with 0F0F and store the result at 1202.

Sample: [1100] = FFFF
Result: [1200] = FFFF

iii) What is the difference between logical ORing with zero and logical ANDing with zero?

2.4.4 Example 8 - Computing a Boolean Expression:

Objective:

To obtain a Boolean expression F which has 4 terms and 8 variables A,B,C,D,E,F,G,H.

$$F = \{(AB'CDE' + A'BCD(BCD+EFGH)\}$$

Theory:

Evaluation of Boolean expressions through minimisation procedures is customary. But this example seeks to do the same using the 8086 registers. The 4 minterms are in FOUR 8 bit registers. Use of logical instructions to perform this is consequential. Don't care variables are represented by set bits.

The correspondence is,

ABCDEFGH —— D7 D6 D5 D4 D3 D2 D1 D0

Example:	Input:	AL	=	10110111B -------	B7
		AH	=	01111111B ------	7F
		BL	=	11111111B ------	FF
		BH	=	11111111B ------	FF
	Result: [1100]		=	11111111B ------	FF

Program:

```
MOV  AL,10110111B             ;operands formed in
                          ;al,ah,bl,bh
MOV  AH,01111111B             ;f = al+ah(bl+bh)
MOV  BL,11111111B             ;value of f in al
MOV  BH,11111111B
OR   BL,BH
AND  AH,BL
OR   AL,AH
MOV  [1100],AL                ;value of f in 1100
HLT
```

OBJECT CODES:

Memory Address	Object Codes	Mnenonics
1000	C6	MOV AL, B7
1001	C0	
1002	B7	
1003	C6	MOV AH, 7F
1004	C4	
1005	7F	
1006	C6	MOV BL, FF
1007	C3	
1008	FF	
1009	C6	MOV BH, FF
100A	C7	
100B	FF	
100C	08	OR BL, BH
100D	FB	
100E	20	AND AH, BL
100F	DC	
1010	08	OR AL, AH
1011	E0	
1012	88	MOV [1100], AL
1013	06	
1014	00	
1015	11	
1016	F4	HLT

Procedure:

A similar procedure of entering the opcodes and executing the program and checking the result is to be followed.

Discussion:

When the number of Boolean variables used in a function is very large and conventional Karnaugh maps become very complicated, this idea can be followed.

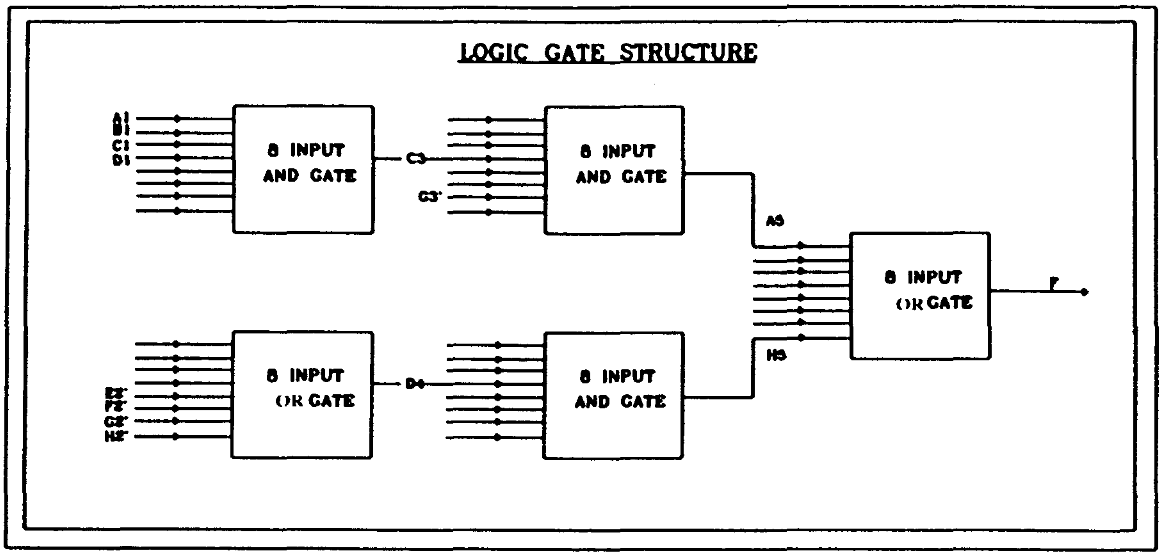
In essence, the use of the logical instructions to implement gate functions is outlined in this example.

Exercises:

i) Obtain F = A'B'C'D' + [E'F'AB.A'BCD]. Store the result F in memory location 1100.

[1100] 7F

ii)



1 to 8 are inputs and
9 is the output of the gate in the above figure.
Obtain F in the above figure and store at 1200.
Result : [1200] = FF

2.4.5 Example 9 - Addition of Two 16-Bit Numbers:

Objective:

To add two 16 bit hex numbers residing in memory and store the result in memory.

Theory:

The "add" instruction requires either the addend or the augend to be in a register, unless the source operand is immediate since the addressing modes permitted for the source and destination are register-register, memory-register, register -memory, register-immediate, and finally memory-immediate. Our objective is to obtain the sumof two numbers in memory.

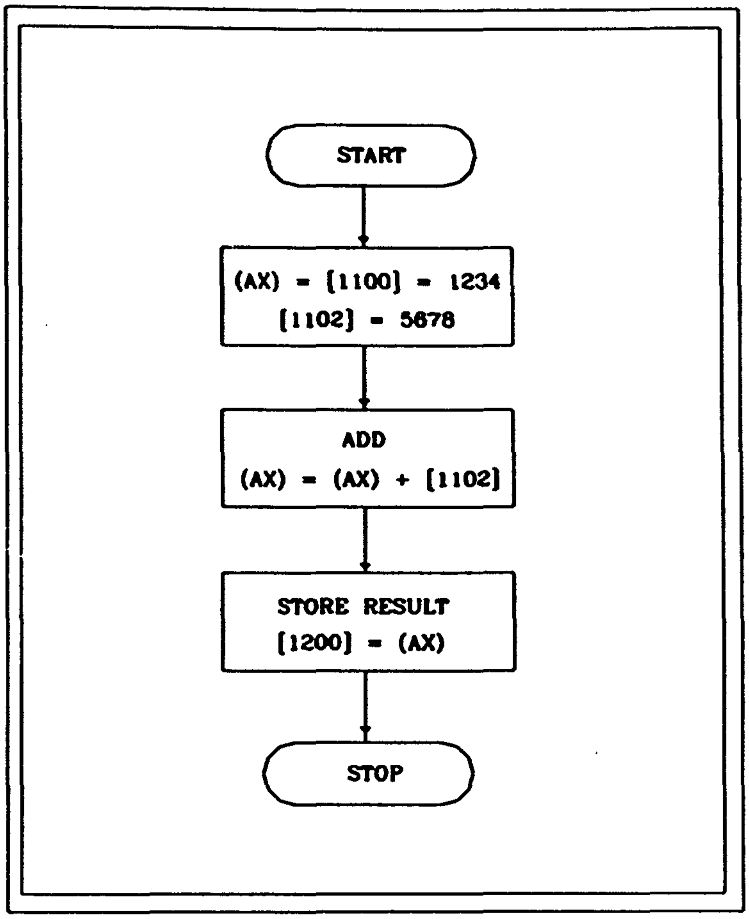
Hence one of the operands is initially moved to AX. Then using the add instruction, 16 bit addition is performed.

Example:

Let us add two numbers in locations 1100 and 1102 and store the result at 1200.

Input :	[1100] =	1234
	[1102]	5678
Result :	[1200] =	68AC

Flow Chart:



Program:

```
MOV  AX,[1100]    ;addend in AX
ADD  AX,[1102]    ;add
MOV  [1200],AX    ;result
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV AX, [1100]
1001	06	
1002	00	
1003	11	
1004	03	ADD AX,[1102]
1005	06	
1006	02	
1007	11	
1008	89	MOV [1200], AX
1009	06	
100A	00	
100B	12	
100C	F4	HLT

Procedure:

Proceed in a similar manner as for the previous examples.

Discussion:

Equipped with the knowledge of ADD and MOV use them and perform addition with the source operand being immediate and the destination operand in register and memory.

The 8086 provides packed and unpacked BCD adjust instructions namely DAA, AAA. In both cases the sum in AL is adjusted by adding 6's. In AAA, AH gets the value AH + carry from AL adjustments.

Exercises:

i) Add FFFF and 0001 and store the result starting from 1100.

Sample:	[1200] = FFFF	
	[1202] = 0001	
Result:	[1100] = 0000	
	[1102] = 1	[Use LAHF]

ii) What is the status of the flags in exercise (i)?

iii) Perform decimal addition of contents of location 1100 and 1102 and store the result in RES at 1200.

[1100]	8889
[1102]	0001
[1200] =	8890

[Hint: Use DAA]

iv) Check flag status before and after using DAA in Exercise 3.

2.4.6 Example 10 - Double Precision Addition:

Objective:

Two 32 bit numbers stored as double words in 8 consecutive memory locations are to be added and the result stored as a double word in memory.

Theory:

The ADD instruction is only one of the two arithmetic ADD instructions. The other instruction ADC performs the following.

(Dest) <——— (Dest)+ (Source) + CF

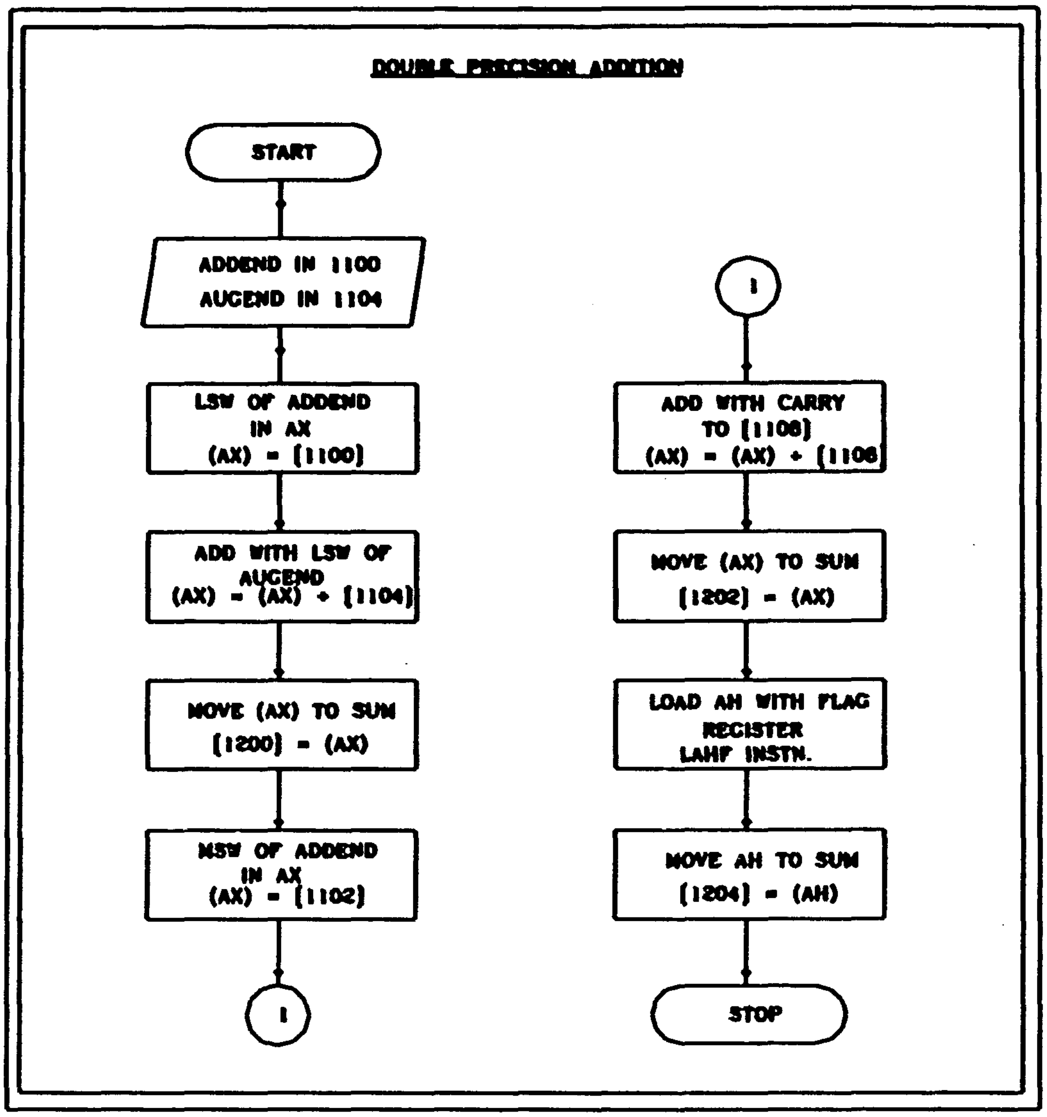
Hence for a double word addition the status of the carry flag must be taken into account using the ADC instruction while adding the 2nd word.

The operands must be in either of the prescribed addressing modes, as detailed in 16 bit addition.

Example:

Input : [1100] = 12345678
[1104] = 12345678
Result:[1200] = 2468ACF0

Flow Chart:


DOUBLE PRECISION ADDITION
START
ADDEND IN 1100
AUGEND IN 1104
LSW OF ADDEND IN AX (AX) = [1100]
ADD WITH LSW OF AUGEND (AX) = (AX) + [1104]
MOVE (AX) TO SUM [1200] = (AX)
MSW OF ADDEND IN AX (AX) = [1102]
1
1
ADD WITH CARRY TO [1106] (AX) = (AX) + [1106
MOVE (AX) TO SUM [1202] = (AX)
LOAD AH WITH FLAG REGISTER LAHF INSTN.
MOVE AH TO SUM [1204] = (AH)
STOP

Program:

```
MOV  AX,[1100]
ADD  AX,[1104]
MOV  [1200],AX
MOV  AX,[1102]
ADC  AX,[1106]
MOV  [1202],AX
LAHF
MOV  [1204],AH
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV AX,[1100]
1001	06	
1002	00	
1003	11	
1004	03	ADD AX, [1104]
1005	06	
1006	04	
1007	11	
1008	89	MOV [1200], AX
1009	06	
100A	00	
100B	12	
100C	8B	MOV AX,[1102]
100D	06	
100E	02	
100F	11	
1010	13	ADC AX,[1106]
1011	06	
1012	06	
1013	11	
1014	89	MOV [1202], AX
1015	06	
1016	02	
1017	12	
1018	9F	LAHF
1019	89	MOV [1204], AX
101A	06	
101B	04	
101C	12	
101D	F4	HLT

Procedure

Proceed in lines similar to the first example and check up the result.

Discussion:

Now that use of a new instruction ADC has been learnt, the user can understand how multiple precision addition is performed by computers. If we were dealing with the one's complement representation of numbers then AAAA is a negative word. Therefore before adding, a sign extension is to be performed. Sign extension for a byte is CBW and for a word is CWD.

Exercises:

i) Add the double precision number AAAABBBB and the single precision number AAAA. Store the result in memory. Consider AAAA as an unsigned word.

Sample:	[1100] BBBB
	[1102] AAAA
	[1104] = AAAA
Result:	[1200] = 6665
	[1202] = AAAB

ii) Do the same addition considering AAAA as a signed word.

Result:	[1200] = 6665
	[1202] = AAAA
	[1203] = 1

[Hint: Use the sign extension instruction CWD]

2.4.7 Example 11 - 16-Bit Subtraction:

Objective:

To subtract two words in memory and place the difference in a memory location.

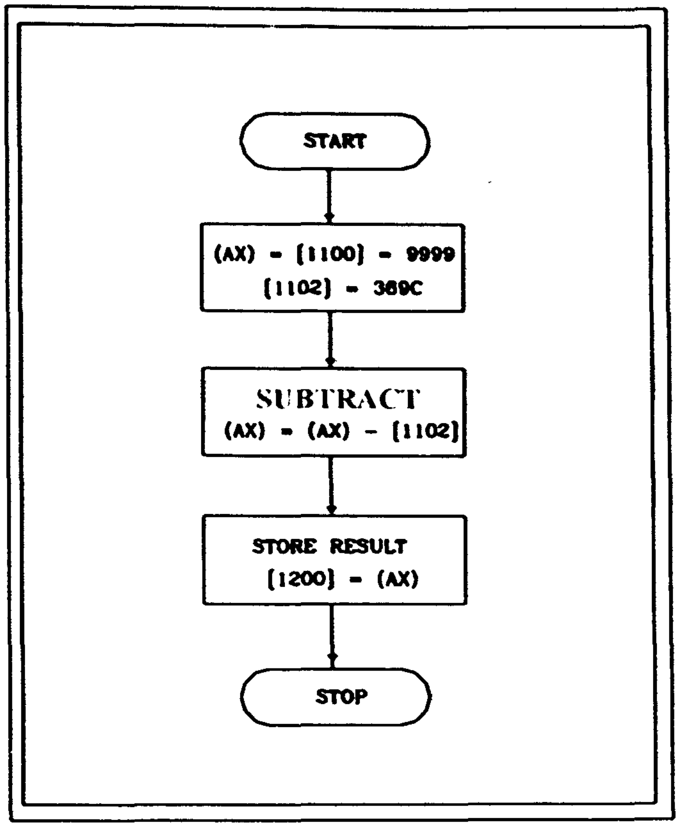
Theory:

The next arithmetic primitive is SUB. As discussed in ADD it permits the same modes of addressing. Hence moving the minuend to a register pair is necessary. Then the result is moved to a location in memory.

Example:

Input: Minuend	:	[1100] = 9999
Subtrahend	:	[1102] = 369C
Result:		[1200] = 62FD

Flow Chart:



Program:

```
MOV  AX,[1100]
SUB  AX,[1102]
MOV  [1200],AX            ;[diff] = [oper]-[oper+2]
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV AX, [1100]
1001	06	
1002	00	
1003	11	
1004	2B	SUB AX,[1102]
1005	06	
1006	02	
1007	11	
1008	89	MOV [1200], AX
1009	06	
100A	00	
100B	12	
100C	F4	HLT

Procedure:

Proceed to enter and execute the program in the same manner as detailed in Program 1 and check up result for various data.

Discussion:

A new instruction SUB has been introduced in this example. The use of DAS instruction and AAS instruction in performing packed and unpacked BCD subtraction shall become obvious when you work out the exercises below.

Exercises:

i)The content of location 1100 is 3456. Subtract 1234 from this and store result at 1200.

Sample:	[1100] = 3456
	[1102] = 1234
Result:	[1200] = 2222

ii) Perform decimal subtraction of content of memory location 1100 and the immediate data $(7865)_{10}$ and store result in 1200.

Sample:	[1100] = $(8756)_{10}$
Result:	[1200] = $(0891)_{10}$
Key :	Use DAS.

iii) Investigate the use of AAS, in subtraction of two unpacked decimal operands.

2.4.8 Example 12 - Double Precision Subtraction: ✓

Objective:

To subtract two 32-bit numbers and store the result in memory. The input data is also to be fetched from memory.

Theory:

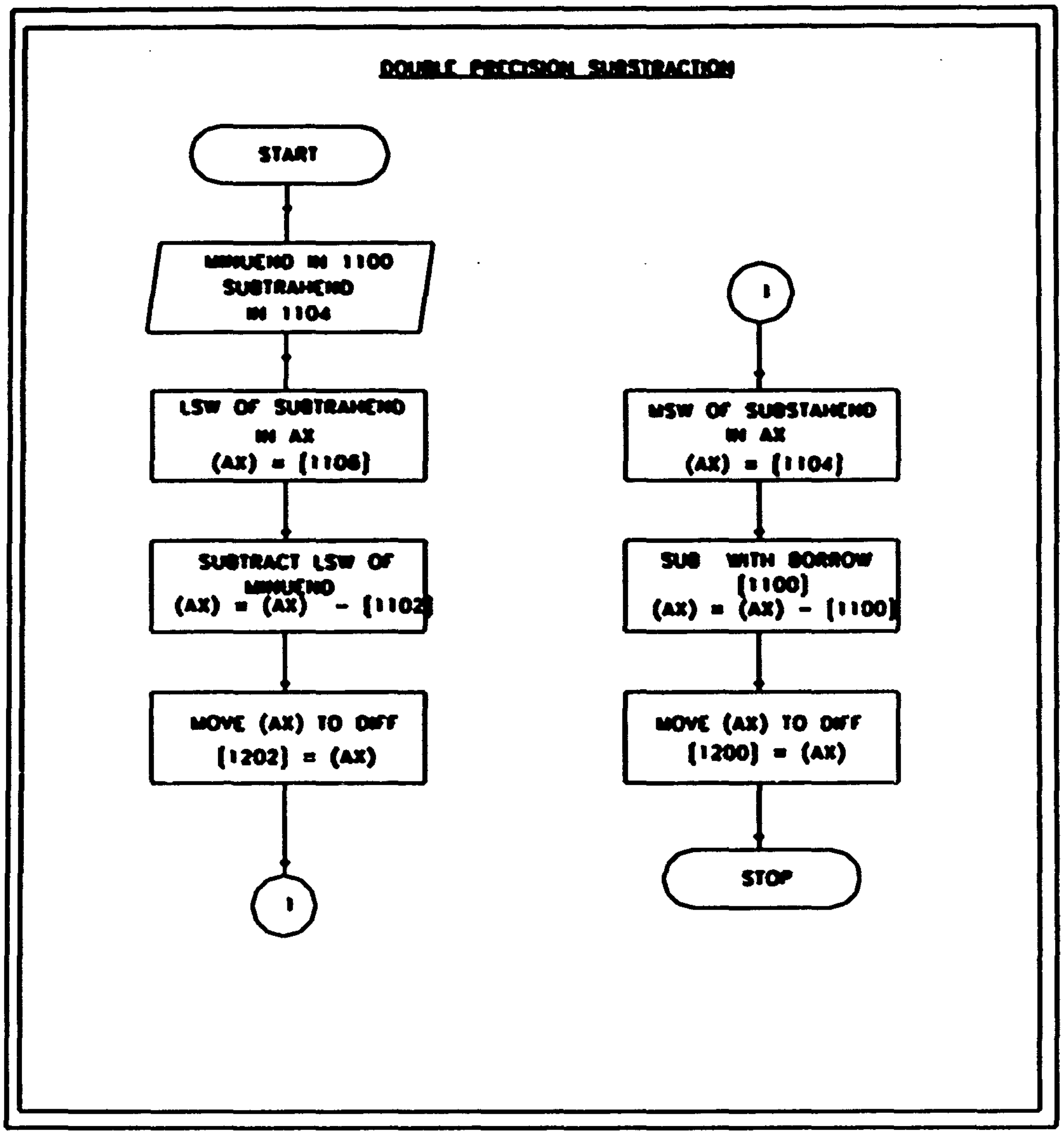
Two long words are stored in memory starting from 1100. It is required to obtain the double word difference of these 32-bit numbers. The SBB instruction is used here. The two least significant words are first subtracted using SUB and the result is stored. Then the higher order words are subtracted using SBB to account for the borrow (if any) produced during the first subtraction. The operation of the SBB (Subtract With Borrow) instruction which is effective if the Carry Flag is set is as depicted below.

(Dest) <—— (Dest) - (Source) - CF.

Example:

Input:	Minuend:	[1100] = 5678
		[1102] = 123A
	Subtrahend:	[1104] = 123A
		[1106] = 5678
	Result:	[1200] = BBC2
		[1202] = 443E

Flow Chart:


DOUBLE PRECISION SUBSTRACTION
START
MINUEND IN 1100 SUBTRAHEND IN 1104
LSW OF SUBTRAHEND IN AX (AX) = [1106]
SUBTRACT LSW OF MINUEND (AX) = (AX) - [1102]
MOVE (AX) TO DIFF [1202] = (AX)
1
1
MSW OF SUBSTAHEND IN AX (AX) = [1104]
SUB WITH BORROW [1100] (AX) = (AX) - [1100]
MOVE (AX) TO DIFF [1200] = (AX)
STOP

Program:

```
MOV  AX,[1106]     ;fetch lower order word
                         ;of subtrahend
SUB  AX,[1102]
MOV  [1202],AX
MOV  AX,[1104]     ;fetch higher order word
                   ;of subtrahend
SBB  AX,[1100]
MOV  [1200],AX
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV AX, [1106]
1001	06	
1002	06	
1003	11	
1004	2B	SUB AX, [1102]
1005	06	
1006	02	
1007	11	
1008	89	MOV [1202], AX
1009	06	
100A	02	
100B	12	
100C	8B	MOV AX, [1104]
100D	06	
100E	04	
100F	11	
1010	1B	SBB AX, [1100]
1011	06	
1012	00	
1013	11	
1014	89	MOV [1200], AX
1015	06	
1016	00	
1017	12	
1018	F4	HLT

Procedure:

Perform entering mnemonics, execution and checking the result as outlined before.

Discussion:

The application of SBB instruction is detailed in the above program. The learners can perform multiple precision arithmetic using the instructions SBB and ADC.

Exercises:

i) Subtract the two numbers AAAAAAAA and BBBB using two's complement addition.

Sample:	[1100] = AAAA	[1104] = BBBB
	[1102] = AAAA	
Result:	[1200] = 6665	
	[1202] AAAB	

ii) Subtract the two numbers :

2^{30} and [$(2^{28} + 2^7) - 1$]
[Hint: Use OR to find 2^{30}, 2^{28} and 2^7.

Use double precision addition to obtain $2^{28} + 2^7$.

Use double precision subtraction to obtain the result.]

Result: EFFFFF01.
[1200] = FF01
[1202] = EFFF

2.4.9 Example 13 - 16-Bit Multiplication:

Objective:

To multiply two 16 bit numbers in memory and store the result in memory.

Theory:

Unlike most of the 8 bit processors which do not have an arithmetic multiply instruction, 16 bit processors from 8086 upward provide both signed and unsigned multiply in their instruction sets to overcome the loss of efficiency in performing repeated addition.

The MUL instruction can have both 16 and 8 bit operands and the multiplicand is AX or AL, accordingly the result for a byte multiply is a 16 bit number in AX while that for a word multiply is a 32 bit number, the lower word of which is in AX and the higher word in DX.

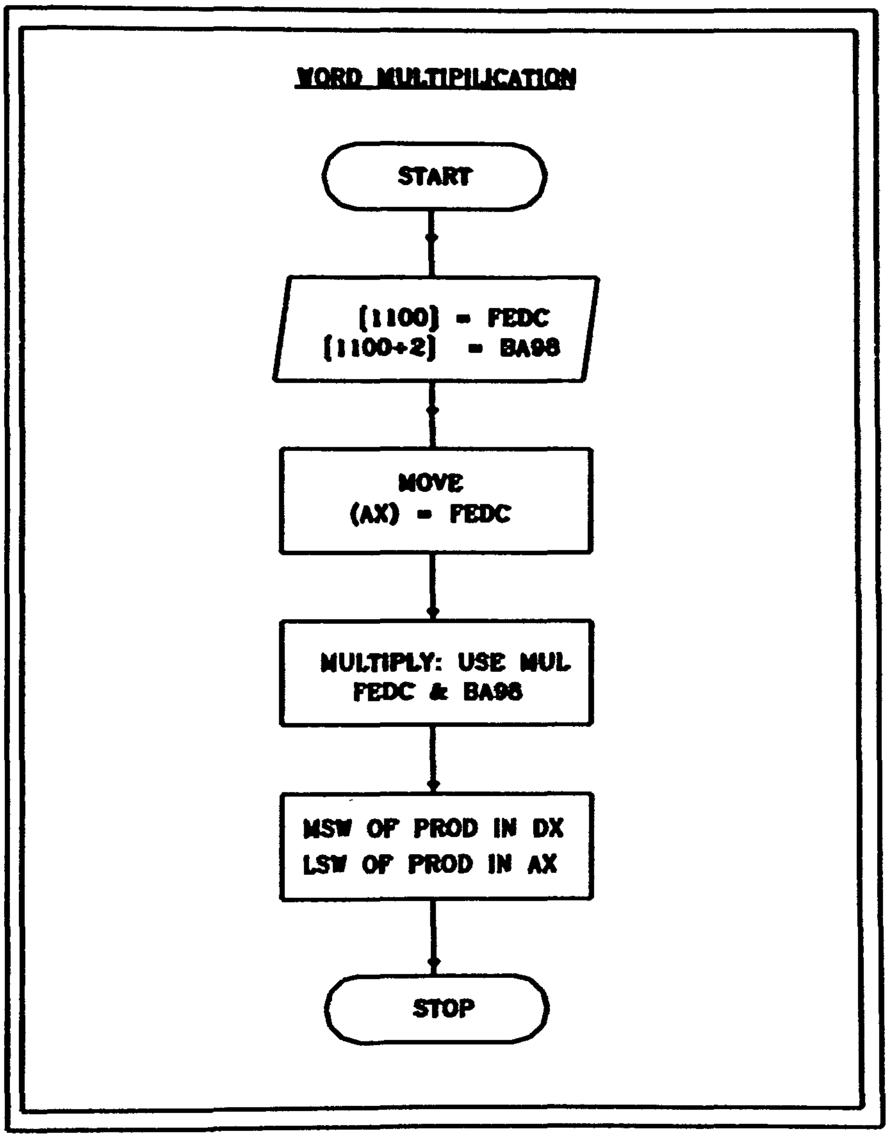
Example:

Multiply the 16 bit numbers:

FEDC and BA98 at 1100 and 1102

Input:	[1100]	FEDC
	[1102]	BA98
Result:	[1200]	B9C3
	[1202]	2AA0

Flow Chart:



Program:

```
MOV   AX,[1100]
MOV   BX,[1102]    : Moving BX since it has the higher order
MUL   BX
MOV   [1200],DX    : Word of the resultant product
MOV   [1202],AX
HLT
```

Object Codes:

Memory Address	Object Codes	Mnemonics
1000	8B	MOV AX, [1100]
1001	06	
1002	00	
1003	11	MOV BX,[1102]
1004	8B	
1005	1E	
1006	02	
1007	11	
1008	F7	MUL BX
1009	E3	
100A	89	MOV [1200],DX
100B	16	
100C	00	
100D	12	
100E	89	MOV [1202], AX
100F	06	
1010	02	
1011	12	
1012	F4	HLT

Procedure:

Enter data and object code in the appropriate RAM locations and execute the program and check if result is stored properly.

Discussion:

Use of the unsigned multiply instruction has been pictured above. The signed multiply instruction IMUL can be used to perform multiplication of a positive number by a negative number and vice versa. In the result, the sign is extended to the higher order registers AH or DX. In this program the results have been stored MSW first and LSW next.

Exercise:

i) Obtain the product $(-47_{10}) * 47_{10}$. Store the resultant word in memory.

Result: [1200] F75F

Key: 1. Obtain hex equivalent of 47 and -47 as an 8-bit number.
2. Use IMUL.

ii) Obtain the product of two 24-bit numbers: 887FFF and 443666.

Key: 887FFF = (88 * FFFF) + 8087.
443666 = (44 * FFFF) + 36AA.

Sample : [1100] = 7FFF [1104] = 3666
[1102] = 0088 [1106] = 0044

Algorithm:

$$\begin{array}{l} a2^{16} + b \\ c2^{16} + d \\ \hline ad2^{16} + bd \\ ac2^{32} + bc2^{16} \\ \hline ac2^{32} + (ad+bc)2^{16} + bd \quad (2^{16} = 10000) \\ \hline \end{array}$$

Result = 245ED1599A.

[1200] = 599A
[1202] = 5ED1
[1204] = 0024

2.4.10 Example 14 - 32-Bit Division:

Objective:

To perform division of a 32 bit number by a 16 bit number and store the quotient and remainder in memory.

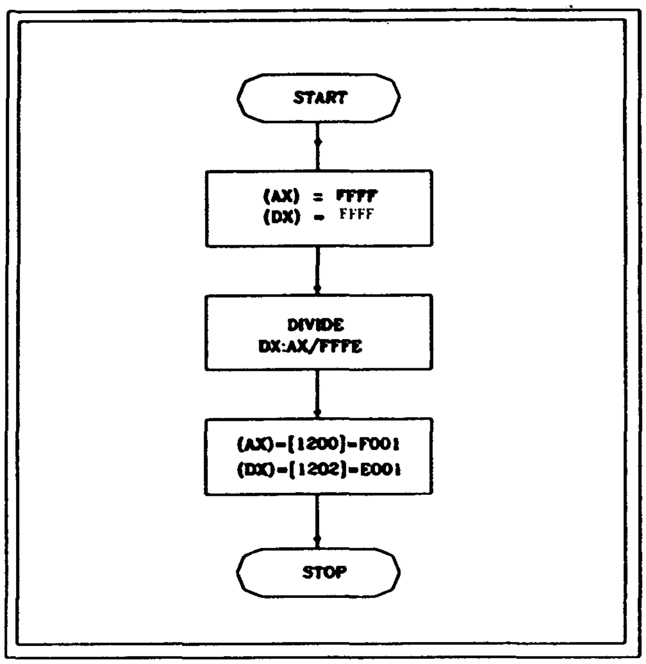
Theory:

Using the DIV instruction of 8086, both division of a double word by a word and a word by a byte can be performed. For the first case, the lower order word of the dividend is in AX and the higher order word is in DX. The quotient is in AX and remainder in DX. For the second case, the dividend is in AL, the quotient is in AL and the remainder in AH.

Example:

Data:	Dividend:	[1100]	0000
		[1102]	FFFF
	Divisor:	[1104] =	FFFF
Result:	Quotient :	[1200] =	0001
	Remainder:	[1202] =	0000

Flow Chart:



Program:

```
MOV   DX,[1100]     ;moving to dx and ax since
                    ;DIV requires
MOV   AX,[1102]     ;higher and lower order words
                    ;to be in these registers
DIV   [1104]
MOV   [1200],AX     ;since DIV gives quotient in
                    ;ax and remainder in dx
MOV   [1202],DX
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	8B	MOV DX, [1100]
1001	16	
1002	00	
1003	11	
1004	8B	MOV AX, [1102]
1005	06	
1006	02	
1007	11	
1008	8B	MOV CX,[1104]
1009	0E	
100A	04	
100B	11	
100C	F7	DIV CX
100D	F1	
100E	89	MOV [1200], AX
100F	06	
1010	00	
1011	12	
1012	89	MOV [1202], DX
1013	16	
1014	02	
1015	12	
1016	F4	HLT

Procedure:

Enter mnemonics codes into the trainer RAM and execute to check result as done before.

Discussion:

The user might be wondering about the consequences if the divisor is zero. The 8086 performs a type 0 interrupt: starts execution at 0000 if this is attempted.

For division of an 8 bit number by an 8 bit divisor, the AH register is to be cleared using the SUB AH, AH instruction.

For division of a word by a word, the DX register is likewise to be cleared.

Exercises:

i) [1100] = FF
Divide FFFF by [1100].
Store quotient at 1200.
Store remainder at 1202.
Result: [1200] = 0101
[1202] = 0000

ii) Investigate the results.
a) IDIV on FF/0A
b) DIV on FF/0A
For case (a) should AH = 0 or FF ?

Substantiate your answer with proper explanations.

2.5. Loop and Branch Operations

A special feature of 8086 lines in the set of LOOP instructions using CX register as the counter. A count is initially loaded in CX and the LOOP, LOOPZ or LOOPE, LOOPNZ or LOOPNE instructions decrement CX register and jump to the label location depending on the condition. That is CX is an implied counter for these instructions. The use of LOOP instructions shall become obvious when performing string operations, array operations and delays.

Branch operations are based on the flags of 8086. The contents of IP are changed depending upon the condition and the displacement of the label location. They use the relative addressing mode and the displacement is only an 8-bit number in the case of conditional branch instructions.

Conditional branch instructions include.

JZ	-	Branch on zero
JNZ	-	Branch on non-zero
JS	-	Branch on sign set
JNS	-	Branch on sign clear
JO	-	Branch on overflow
JNO	-	Branch on no overflow
JB/JC	-	Branch on carry
JNB/JNC	-	Branch on no carry
JNA	-	Branch on not above
JA	-	Branch on not below

The same set for signed operation is:

JNGE	-	Branch on less
JGE	-	Branch on not less
JNG	-	Branch on not greater
JG	-	Branch on greater

In the signed case, the sign flag, overflow and zero flags are examined before branching, while in the unsigned case carry and zero flags are used.

In unconditional branching the displacement is 16-bits.

To illustrate branching, a program to find length of a block is used.

2.5.1 Example 15 - Calculating the Length of aString

Objective:

To find the number of characters in a string

Theory:

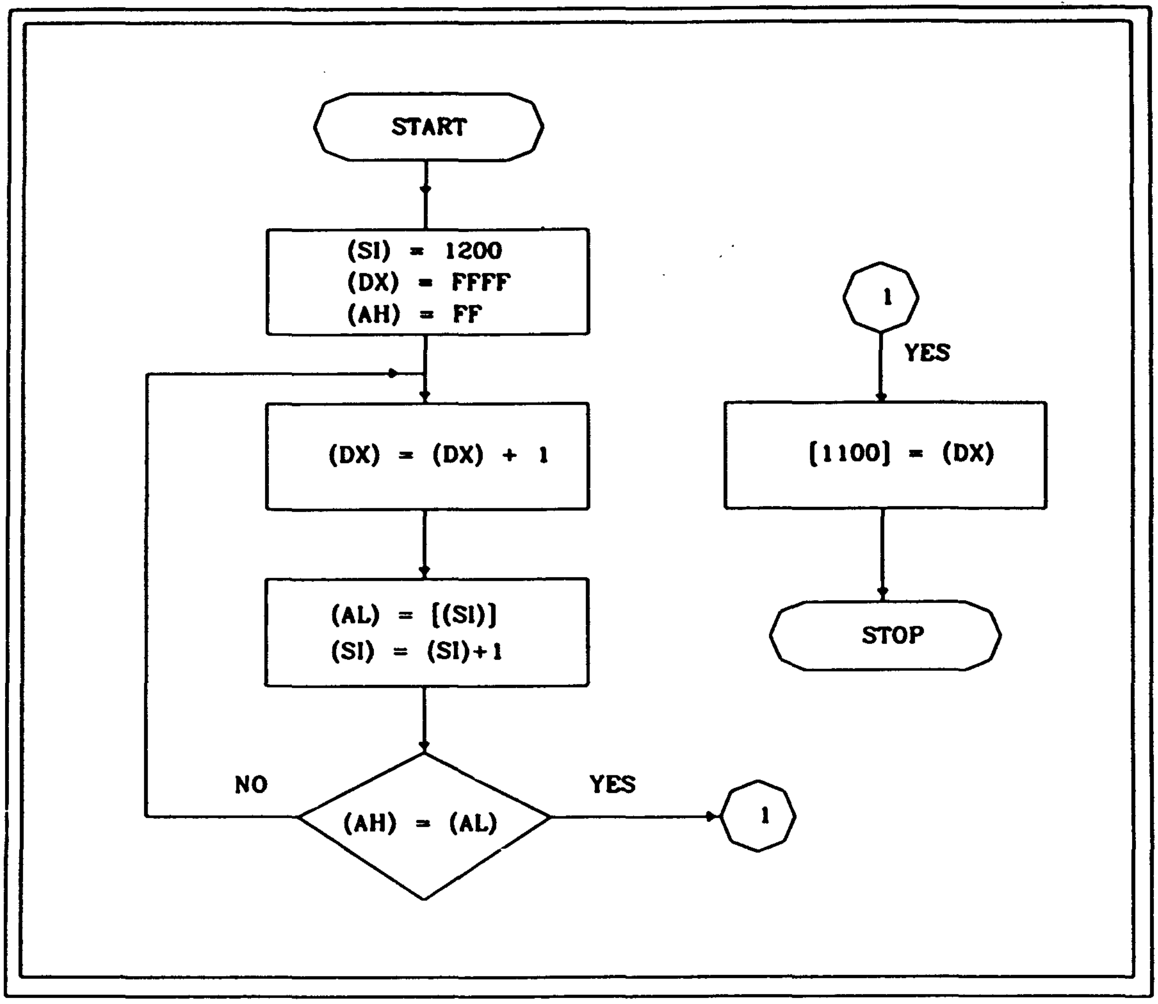
Addressing the string is done using SI register, and the DX register is used to store the number of characters. End of string is detected using FF. Hence each character is detected using FF. Hence each character is fetched from memory and is compared with FF. If the zero flag is set, then it denotes end of string, the count having been stored in DX, by incrementing it after each comparison.

Example:

In this example the string is assumed to start at 1200 and end of string is the value FF. Let the content of location 1300 be FF. Then the result, which is the length of the string is checked in 1100 to be 0100. Please note that FF is not part of the string.

RESULT: [1100] = 0100

Flow Chart:



Program:

```
            MOV  SI, 1200       ;String start in SI
            MOV  DX, FFFF       ;
            MOV  AH, FF         ;End of string = FF
NOTEND:INC  DX                  ;String length in DX
            MOV  AL, [SI]       ;Get string character to AL
            INC  SI             ;Increment string index
            CMP  AH, AL         ;
            JNZ  NOTEND         ;If comparison not
            MOV  [1100],DX      ;successful increment count
                                ;Else store string length
            HLT
```

Object Codes:

Memory Address	Object Codes	Mnemonics
1000	C7	MOV SI, 1200
1001	C6	
1002	00	
1003	12	
1004	C7	MOV DX, FFFF
1005	C2	
1006	FF	
1007	FF	
1008	C6	MOV AH, FF
1009	C4	
100A	FF	
100B	42	NOTEND: INC DX
100C	8A	MOV AL,[SI]
100D	04	
100E	46	INC SI
100F	38	CMP AH, AL
1010	C4	
1011	75	JNZ 100B
1012	F8	
1013	89	MOV [1100],DX
1014	16	
1015	00	
1016	11	
1017	F4	HLT

Procedure:

Enter the mnemonics from 1000. Enter the string starting from 1200. After last character in the string enter FF. Execute the program and check for result in 1100.

Discussion:

In this program use of the instruction JNZ has been used for conditional branching. Use of other instructions for conditional branching shall become evident in later sections.

Exercises:

i) Simulate the rotate instructions with count #1 using the conditional branch.

ii) Simulate the instructions CWD and CBW using conditional transfer depending upon sign flag status.

iii) Using the code developed in the previous exercise convert a byte in memory to a word and store result in memory.

Sample:[1100] = EF
Result: [1200] = FFEF

2.5.2 Example 16 - Illustration of Loop Instruction:

Objective:

To store numbers from 00 to FF in consecutive memory locations.

Theory:

We have to store 256 numbers from 00 to FF in successive memory locations. Hence the process has to be done 256 times. The CX register is initialised with 0100 and used for the looping or continuation of the process.

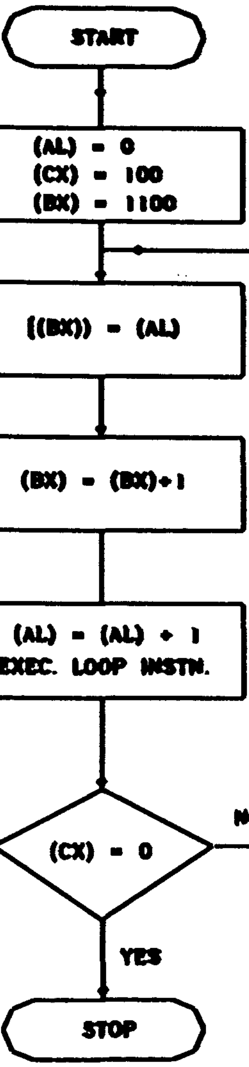
Example:

In this example, the number array is assumed to start at location 1100 as indicated by BX. After execution [1100] = 00; [1101] = 01 and so on upto [11FF] = FF.

Program:

```
            XOR     AL, AL          ;Clear AL to get 0
            MOV     CX, 0100        ;Count in CX
            MOV     BX, 1100
CONT:       MOV     [BX],AL         ;Store successive numbers
                                    ;from 1100
            INC     BX
            INC     AL
            LOOP    CONT            ;Stop storing if CX=0
            HLT
```

Flow Chart:


START
(AL) = 0
(CX) = 100
(BX) = 1100
[(BX)] = (AL)
(BX) = (BX)+1
(AL) = (AL) + 1
EXEC. LOOP INSTN.
(CX) = 0
NO
YES
STOP

Object Codes:

Memory Address	Object Codes	Mnemonics
1000	30	XOR AL, AL
1001	C0	
1002	C7	MOV CX, 0100
1003	C1	
1004	00	
1005	01	
1006	C7	MOV BX, 1100
1007	C3	
1008	00	
1009	11	
100A	88	CONT: MOV [BX], AL
100B	07	
100C	43	INC BX
100D	FE	INC AL
100E	C0	
100F	E2	LOOP CONT
1010	F9	
1011	F4	HLT

Procedure:

Enter the mnemonics from 1000. Execute the program and check if the contents of 1100 is 00 and so on upto 11FF, which contains FF.

Discussion:

In this example we have used the LOOP instruction. Similar instructions are LOOPZ and LOOPNZ which shall be clear after working through the exercises. The same objective can be realised using string primitives as shall become evident in the next section. The loop instruction is equivalent to a branch if CX is not zero (CX #0).

Exercises:

i) A byte array consists of FF elements. Search for the byte 80 in the array. If match is found store address of match in a location; else store FFFF in the same location. [Use LOOPNE]

a)	Sample:	[1200] to [12FF] 41
	Result :	[1100] - FFFF
b)	Sample:	[1200] to [1210] 41 [1211] 80
	Result:	[1100] = 1211

2.6 Array and String Operations:

The 8086 instruction set includes a set of instructions called the string primitives. Each string primitive instruction performs a sequence of operations normally handled by an instruction loop. The string primitive instruction performs an operation specified by the primitive, then increments or decrements the pointer registers involved in the operation. On each iteration the affected pointer registers can be either incremented or decremented, by 1 or 2.

Pointer registers will be incremented if the value of the Direction Flag in the Flags Register is 0; affected pointer will be decremented if the value of the Direction Flag is 1. The affected pointer registers will be incremented or decremented by 1 if the low-order bit of the string primitive operation code is 0. If the low-order bit of the string primitive operation code is 1, the affected pointer registers will be incremented or decremented by 2.

There are five string primitives;

MOV -	Move 8 or 16 bit data in memory
LODS -	Load 8 or 16 bits of data from memory into the AL or AX register
STOS -	Store the AL (8-bit operation) or AX (16-bit operation) register into memory
SCAS -	Compare the AL (8-bit operation) or AX (16-bit operation) register with memory
CMPS -	Compare two strings of memory locations.

Use of index registers and string primitives along with direction flag status enables efficient array and string manipulation as shall be evident from the following examples.

2.6.1 Example 17 - Sum of the Numbers in a Word Array:

Objective:

To obtain the sum of a 16-bit array in memory, using index register SI and store the result in memory.

Theory:

Initially the index register SI is made to point to the location where the array starts. The length of the array is loaded in the CX register. Then the accumulator is cleared and by adding the contents of the location pointed to by the index register with the accumulator, incrementing the index register to point to the next word and simultaneously decrementing the CX register till CX is 0, the sum of the elements in the array can be obtained.

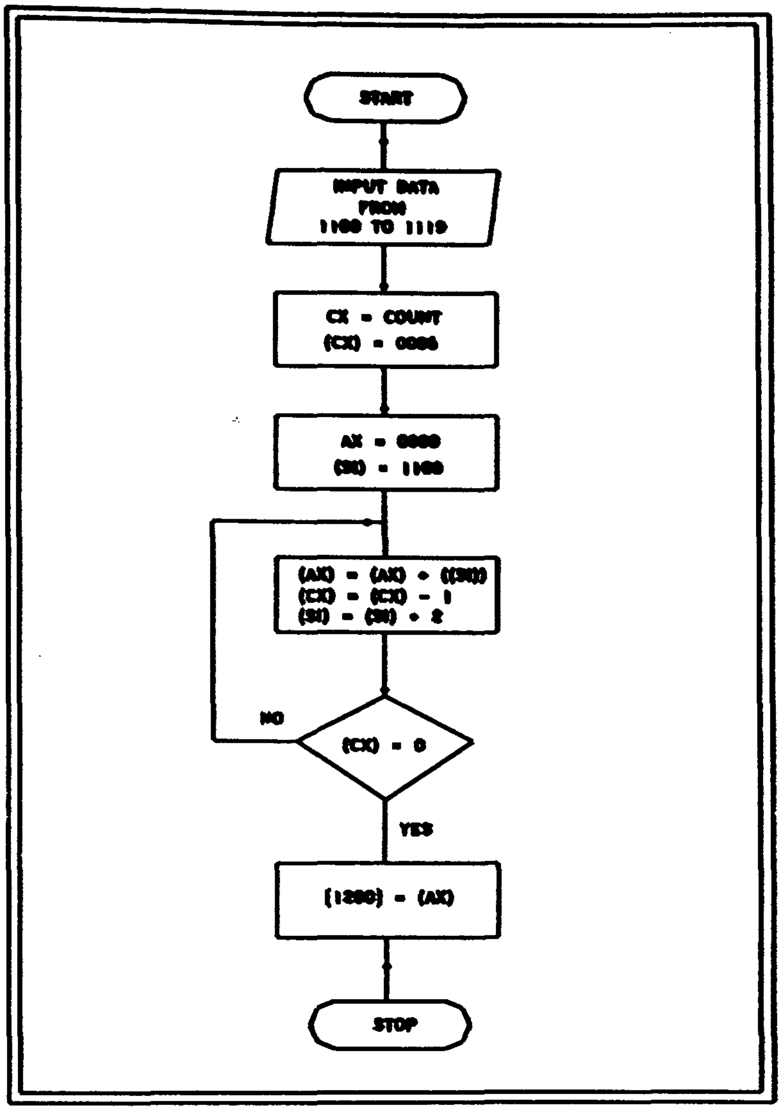
Example:

Data in array from START = 1110

```
[1110] = 0FFF
[1112] = 0FFF
[1114] = 0FFF
[1116] = 0FFF
[1118] = 0FFF
```

Result: [1200] = 4FFB

Flow Chart:



Program:

```
                MOV  CX, 5                 ;number of elements in CX
                MOV  AX, 0
                MOV  SI, AX                ;initialise SI to start of array
 LOOP1:    ADD  AX, START[SI]
                ADD  SI,2                  ;decrement CX and check
                                           ;if zero
                LOOP LOOP1
                MOV  [SUM],AX
                HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C7	MOV CX, 5
1001	C1	
1002	05	
1003	00	
1004	C7	MOV AX, 00
1005	C0	
1006	00	
1007	00	
1008	89	MOV SI, AX
1009	C6	
100A	03	LOOP1: ADD AX,[1100]
100B	06	
100C	10	
100D	11	
100E	81	ADD SI, 2
100F	C6	
1010	02	
1011	00	
1012	E2	LOOP LOOP1
1013	F6	
1014	89	MOV [1200], AX
1015	06	
1016	00	
1017	12	
1018	F4	HLT

Procedure:

Enter above data and program using Assembler command and execute from 1000. Using GO and EXEC, check for the result in 1200.

Discussion:

In this example two concepts have been employed: namely, use of index registers and use of LOOP instruction which decrements CX and jumps to the label location if CX # 0.

Exercise:

i) Compute the 24 bit sum of 5 elements in an array, each equal to FFFF and store the result as an array of 4 bit numbers.

```
Sample:     [1100] = FFFF
            [1102] = FFFF
            [1104] = FFFF
            [1106] = FFFF
            [1108] = FFFF

Result:     [1200] = 00
            [1201] = 04
            [1202] = 0F
            [1203] = 0F
            [1204] = 0F
            [1205] = 0B
```

ii) Write a program that gives the same result using MUL.

iii) Which of the above is efficient ?

iv) When is array addition useful ?

v) Perform the same array addition using LODS.

2.6.2. Example 18 - String Operations (Move):

Objective:

To move a byte string of length FF from a source to a destination.

Theory:

Since the 8086 includes the string primitives, which require initialisation of the index registers, the SI and DI registers are initialised to start of the source and start of the destination array respectively. The direction flag is cleared to facilitate autoincrementing of the index registers. The CX register is used to perform the operation repeatedly. The string primitive used in MOVS. In the case of MOVE operation, the status of the direction flag is however immaterial.

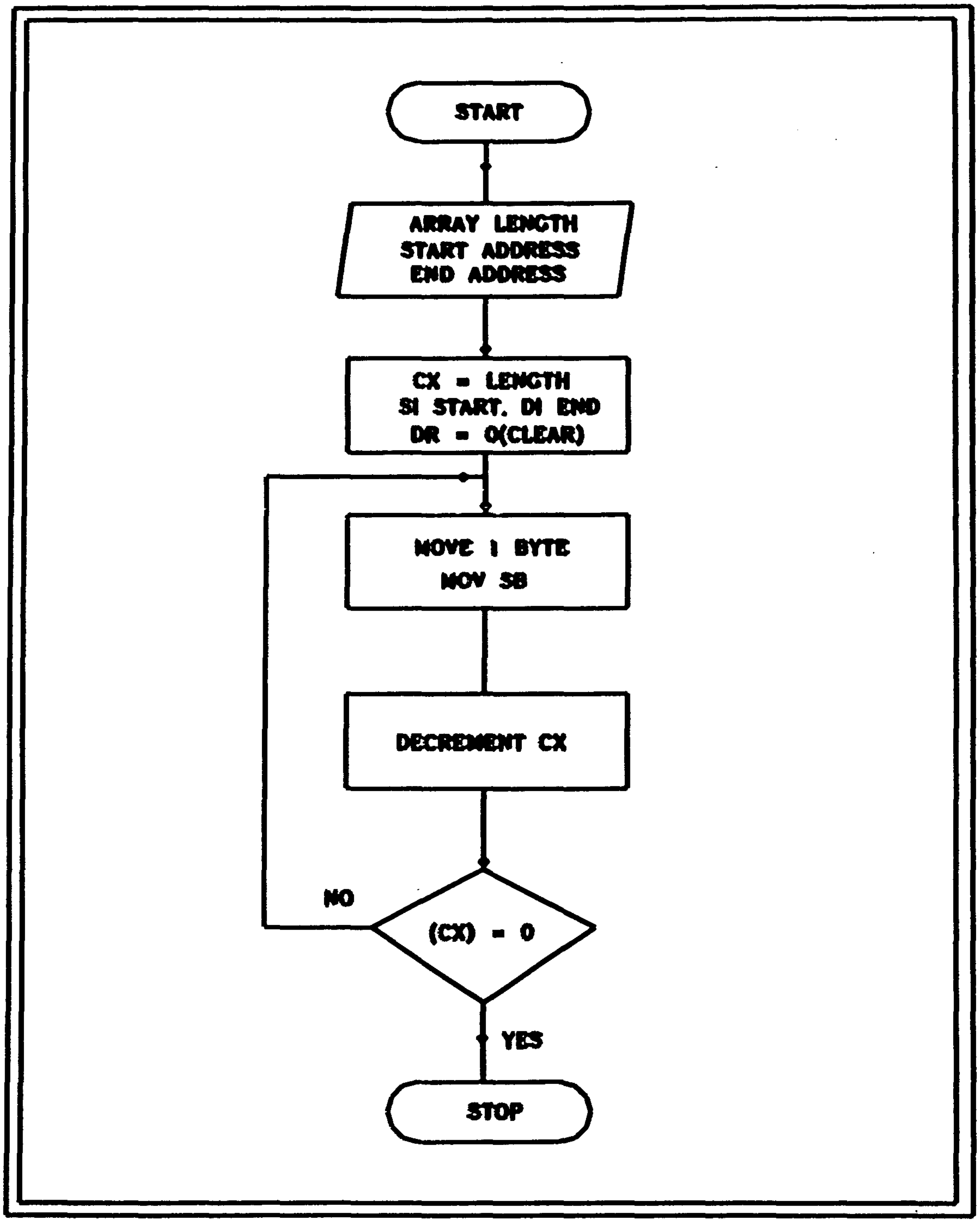
Example:

The data for the source array has to be initially entered. Hence let us fill the source locations starting from S-ARRAY using the FILL command in the kit. Fill locations from 2000 to 20FF with 41.

S-ARRAY 2000
D-ARRAY = 20FF

Result: [2000] TO [20FF]- 41

Flow Chart:



Program:

```
MOV     SI, OFFSET S_ARRAY
MOV     DI, OFFSET D_ARRAY
MOV     CX, OFFH
CLD                                 ;facilitates auto-incrementing
                                    ;of the index registers
MOVE:   MOVSB
LOOP    MOVE
HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C7	MOV SI,2000
1001	C6	
1002	00	
1003	20	
1004	C7	MOV DI,2100
1005	C7	
1006	00	
1007	21	
1008	C7	MOV CX, FF
1009	C1	
100A	FF	
100B	00	
100C	FC	CLD
100D	A4	MOVE: MOVSB
100E	E2	LOOP MOVE
100F	FD	
1010	F4	HLT

Procedure:

Enter the program from location 1000.

Fill FF locations as stated above with a particular data, 41

Execute the program. Check if the contents are duplicated to another 255 locations using the compare command in the kit.

Discussion:

A small program can be written to fill the contents of memory with a particular byte using Stosb or Stosw, to duplicate the contents at another set of memory locations and to compare the contents of the two blocks of memory. The same program can be implemented with 'REP' in a more efficient manner. The same purpose can be achieved using MOVSW but the count should end on word boundaries.

Exercises:

In a symbol table of 100 entries starting at location TABLE, each entry is comprised by 80 bytes. The first 8 bytes represent the LABEL field and the remaining bytes, the info. Search this table for a given label of 8 characters stored in LABEL. If this LABEL is found copy the associated 72 bytes to a location starting at INFO. Else fill info with blank.

Key: TABLE is an array of 80 strings. LOAD AL with first 8 bytes of LABEL successively SEARCH each string of the array for these 8 bytes. If a match is found store the 72 bytes following LABEL by MOVSB in INFO. Else use STOSB to fill INFO with blanks.

2.6.3 Program 19 - Use of Store String Primitive:

Objective:

To fill the locations 1100 to 11FF in memory with the byte 34.

Theory:

This program uses the string primitive STOS. The function of this is that it will store the byte in AL or the word in AX depending upon the operand size, from the location pointed to by the destination index DI. So if we want to fill a block with a particular data then we should set destination index to the beginning of the block and then use the STOSW instruction or the STOSB instruction and use CX to get the required length. S_ARRAY is the location 1100 in this program.

Example:

Input:	[AX] = 0034
Result:	[1100] to [11FF] = 34

Program:

```
        MOV     CX, 0100
        MOV     DI, OFFSET S_ARRAY
        MOV     AX, 0034
        CLD
L:      STOSB
        LOOP    L
        HLT
```

Object Codes:

Memory Address	Object Codes	Mnenonics
1000	C7	MOV CX, 0100
1001	C1	
1002	00	
1003	01	
1004	C7	MOV DI,1100
1005	C7	
1006	00	
1007	11	
1008	C7	MOV AX, 0034
1009	C0	
100A	34	
100B	00	
100C	FC	CLD
100D	AA	L: STOSB
100E	E2	LOOP L
100F	FD	
1010	F4	HLT

Procedure:

Enter the above program and execute it. Check if locations 1100 to 11FF contain 34.

Discussion:

Use STOSW and fill locations similarly. Alternatively employ REPZ STOSW to achieve the same objective.

Exercises:

i) Fill locations 1100 to 11FF with 41 using the above program.
Fill from 1200 to 12FF with 41 using FILL command in the kit.
Compare from 1100 to 11FF and 1200 to 12FF using CMP key in the trainer.

ii) Compare the above two blocks using a routine which uses CMPSB.

CHAPTER-3

PERIPHERAL INTERFACING

3.1 Introduction:

This Chapter emphasizes more on the INTERFACE PROGRAMMING ASPECTS of the various peripherals available on M - 86/88 LCD.

As in the previous Chapter, you will have to try and work with the given EXAMPLES AND EXERCISES to strengthen your technical muscle, in understanding the operational capabilities of the peripheral devices used. Chapters 2 & 3 of this manual should no doubt be of great help in shaping you into a well equipped and 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 Intel, OKI Semiconductor or

Note: The signals specified with an "*" to the right are active low signals.

3.2 Timer Interface: U29

3.2.1 Component:

The timer interface is comprised by the Intel 8253, which is used as a general purpose timer and also to provide baud rates for the USART for serial communication.

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
* Binary or BCD Count

Typical applications of 8253 are,

i) Event Counter
ii) Binary Rate Multiplier
iii) Real Time Clock
iv) Programmable Rate Generator and so on.

3.2.2 Component Description:

The 8253 is organised as **THREE 16-BIT COUNTERS** each with a count rate of upto 2MHz. 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.

The **CONTROL WORD REGISTER** controls the operation mode of each counter, selection of binary or BCD counting and the loading of each count register depending upon the information stored in it.

This **CONTROL REGISTER CAN ONLY BE WRITTEN INTO**; it cannot be read from.

a) Basic System Interface:

The system interface of 8253 uses the RD*, WR*, CS*, A0, A1 and D0-D7 lines, described below:

RD*	-	Active low Read input
WR*	-	Active low Write input
CS*	-	Active low chip select
A0, A1	-	Select Counter / Control Register
D0-D7	-	Bi-directional data bus.

The basic operation of the timer using the above signals is as follows.

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

In the M - 86/88 LCD trainer, the Channel 0 of the timer is used for the baud rate generation for the USART. If the 8251A is not going to be used, the timer can be used for different applications. Channels 1 and 2 are available to users for their own development purpose.

3.2.3 Programming Description:

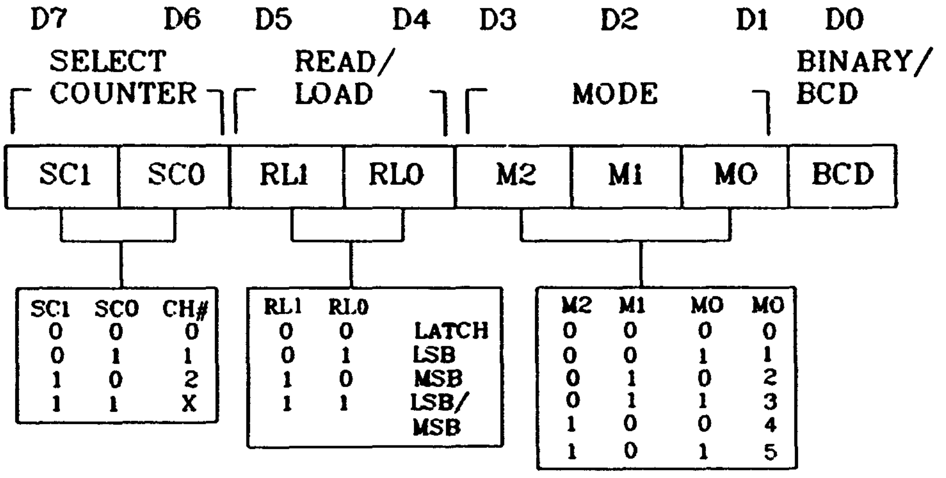
The complete functional definition of the 8253 is programmed by the system software. A set of control words must be sent out by the CPU to initialise each counter of 8253 with the desired mode and quantity information.

gPrior to initialisation, the MODE, the COUNT and OUTPUT of all counters is undefined. The control words program the mode, Loading sequence and selection of binary or BCD counting.

a) Programming the Control Register:

Control Word Format:

The Control Word Format of 8253 is as given in Figure F3.1.



SC1	SC0	CH#
0	0	0
0	1	1
1	0	2
1	1	X

RL1	RL0	
0	0	LATCH
0	1	LSB
1	0	MSB
1	1	LSB/ MSB

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

Figure F3.1

Referring to the Figure F3.1, it is evident that we have got options to Select Counter, Read/Load mode, Six different modes of operations and to select either binary or BCD counting.

The six modes of operation enable the 8253 to be used for various applications.

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 count. It is re-triggerable, hence the output will remain low for the full count after 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 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 8251A 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.

b) Programming Sequence:

The following are the steps to be followed during the programming of the counters.

i) Measure the input clock to the counter.
ii) Decide the mode of operation.
iii) Calculate the COUNT for the desired output.
iv) Write the MODE CONTROL WORD.
v) Write the count according to the RL1, RL0 bit positions in the control register.

For instance, if we want to initialise 8253 for the following parameters, say **"Channel 0 - Square Wave - at 125 KHz"**. Then,

SC1,SC0	=	0,0	—> Select Channel 0
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	=	36 (hex)	

Using the OUT instruction of 8086, output the control word to the control register and the count to Channel 0 so that you get an output frequency of 125 KHz whose input clock is nearly 1.255 MHz.

3.2.4 Circuit Implementation:

The Figure F3.2 gives a clear idea about the interface between the 8253 and the 8086 microprocessor.

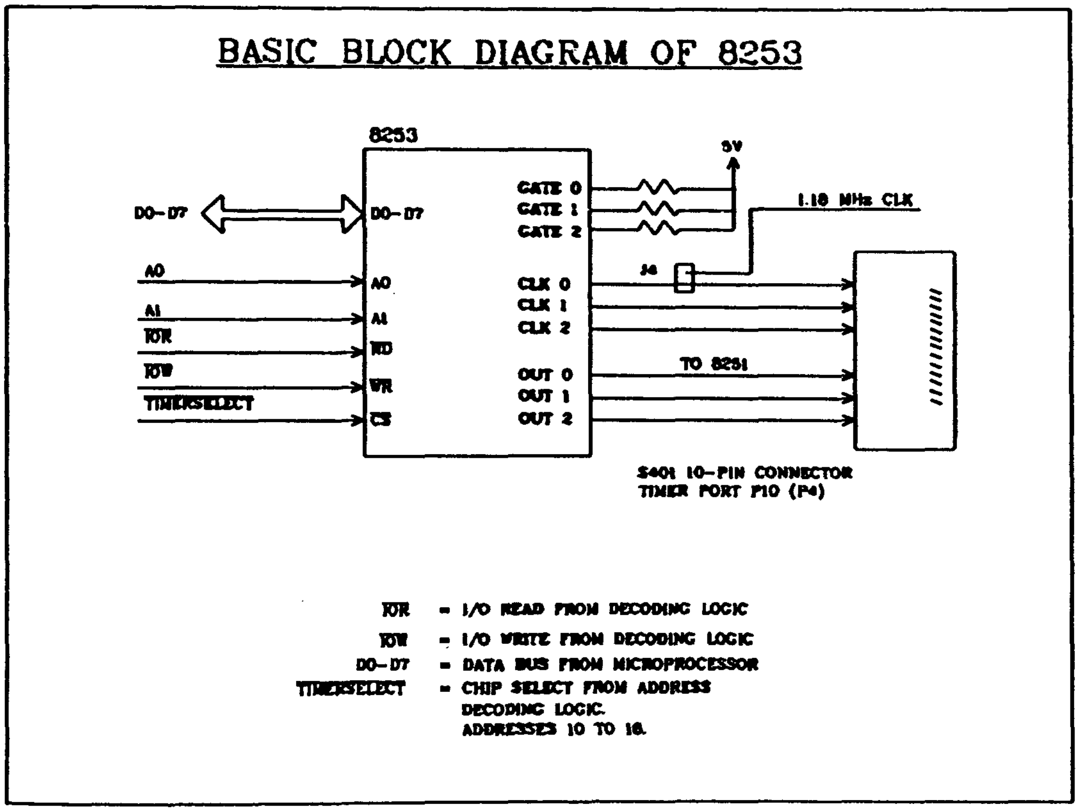

BASIC BLOCK DIAGRAM OF 8253
8253
D0-D7
A0
A1
IOR
IOW
TIMERSELECT
D0-D7
A0
A1
RD
WR
CS
GATE 0
GATE 1
GATE 2
CLK 0
CLK 1
CLK 2
OUT 0
OUT 1
OUT 2
5V
1.18 MHz CLK
J4
TO 8251
8401 10-PIN CONNECTOR
TIMER PORT P10 (P4)
IOR = I/O READ FROM DECODING LOGIC
IOW = I/O WRITE FROM DECODING LOGIC
D0-D7 = DATA BUS FROM MICROPROCESSOR
TIMERSELECT = CHIP SELECT FROM ADDRESS DECODING LOGIC. ADDRESSES 10 TO 16.

Figure F3.2

As is clear from the above block diagram, the Data bus D0-D7 is connected to the microprocessor data bus. The IOR* and IOW* are connected to the RD* and WR* of 8253 respectively. The TIMERSELECT* which is got from the address decoding logic provides the Chip Select to the 8253. This Chip Select is low for I/O addresses from 10 to 16. The A0 and A1 lines are connected to the A1 and A2 address lines of the microprocessor respectively.

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

IC No.	Function	Address
U29	Control Register	0016
U29	Channel 0	0010
U29	Channel 1	0012
U29	Channel 2	0014

Now that you have learnt about the hardware organisation and functional description of 8253, let us write a simple program to generate a square wave of certain frequency from Channel 0 of the timer.

3.2.5 Example 1 - Square Wave Generation Using 8253:

Objective:

To generate a square wave of frequency 125KHz from Channel 0 of 8253.

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 can be got is Mode 3 Square Wave Generator. As explained before, the output frequency of clock from Channel 0 will depend upon the count to which channel 0 is initialised. To get an output clock frequency of 125KHz, where the input clock frequency is 1250KHz, the count to be initialised must be 1250 / 125 = 10 = 0A (hex). So, if Channel 0 is initialised to Mode 3 and count 0A, the output clock frequency will be 125KHz (square wave).

The control word that must first be written into the control register will be 0 0 1 1 0 1 1 0 b. This control word initialises the control register to

Select Channel 0,
Read/Load LSB first, then MSB,
Mode 3,
Binary counter 16-bits.

Program:

```
                    ;This program is to generate a clock
                    ;of frequency 125 KHz at the Channel 0
                    ;8253 whose input clock is 1.25MHz .
                    ;
                       cont:   equ     16
                       ch0:    equ     10
1000  C6 C0 36                         mov    al,36
1003  E6 16                            out    16, al
1005  C6 C0 0A                         mov    al,0a
1008  E6 10                            out    10,a2
100A  C6 C0 00                         mov    al,00
100D  E6 10                            out    10,al
100E  F4                               hlt    f4
```

Verification:

Enter the program given above as shown from the address and then execute the program. Using an oscilloscope, verify at OUT0 of 8253 (Pin no. 10 of 8253) whether you get a square wave of frequency 125 KHz.

3.2.6 Exercises:

i) What is the output of 8253 at Channel 1, if Channel 1 is initialised for Square wave generator mode and the count being 05.
ii) Write a program to give a pulse at Channel 1 of 8253 at a time delay of one second.
iii) In the above program what mode have you initialised the timer in ?
Are you performing a complete read of the count (both bytes) if in mode 0 ? If not, you are wrong.]

3.3 RS232C Serial Interface: U8, U9

3.3.1 Component:

The RS232C interface of M - 86/88 LCD comprises the UNIVERSALSYNCHRONOUS/ ASYNCHRONOUS RECEIVER/TRANSMITTER 8251A (USART). The main features of 8251A 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.

3.3.2 Component Description:

The 8251A is used here as a peripheral device for serial communication and is programmed by the CPU to operate using virtually any serial data transmission technique. The USART accepts data characters from the CPU in parallel format and then converts 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 8251A commands are written into two registers, namely the MODE INSTRUCTION REGISTER and the COMMAND INSTRUCTION REGISTER. These two registers cannot be read from.

The STATUS REGISTER is read for the same address as Mode/Control.

a) Basic System Interface:

The pins with which the 8251A communicates with the CPU are the RESET, CLK, WR*, RD*, CS*, C/D*, DO-D7, A0.

The basic operation of 8251A can be briefed as.

C/D*	RD*	WR*	CS*	FUNCTION
0	0	1	0	8251A DATA => DATA BUS
0	1	0	0	DATA BUS => 8251A 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 8251A are,

TxRDY	—	Indicates to CPU, that 8251A is ready to accept a character for transmitting serially.
TxEMPTY	—	Indicates that the 8251A 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 to the CPU that the 8251A has received a character ready to be inputted to the CPU.
RxC*	—	It is similar to TxC* with 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 sync & 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.

In M - 86/88 LCD to convert TTL signals to RS232 levels and vice versa, MAX232 driver is used. Whereas in the case of M - 86/88 LCD MC1488 (quad TTL-to-RS232C driver) and MC1489 (quad RS232C-to-TTL receivers) are used. 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 for serial communication, these signals will connect with the CTS*, RTS*, TxD and RxD signals of that system respectively. These signals can be explained as:

i) CTS*: Clear to Send	-	Enables 8251A to transmit serial data, if TxEnable bit in the command byte is set to one.
ii) RTS*: Request to Send	-	When low, indicates that 8251A 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 8251A must be loaded with a set of control words generated by the CPU. These control signals define the complete functional definition of the 8251A 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

i) Mode Instruction Definition:

This format defines the general operational characteristics of the 8251A. It must follow a Reset operation immediately. 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 8251A can be made to operate in the Synchronous mode.

Figure F3.3 and F3.4 gives the Mode Instruction Format for the Asynchronous and Synchronous Mode respectively.

ii) Command Instruction Definition:

This format defines a status word that is used to control the actual operation of the 8251A. All control words written into the 8251A after the Mode Instruction will load the Command Instruction. The Command Instructions can be written into the 8251A at any time in the data block during the operation of the 8251A. 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 F3.5 gives the complete information of command instruction format.

The 8251A has facilities that allow the programmer to read the status of the device at any time during the functional operation. Figure F3.6 gives the information regarding the status register format.

MODE INSTRUCTION FORMAT

ASYNCHRONOUS MODE 8251A

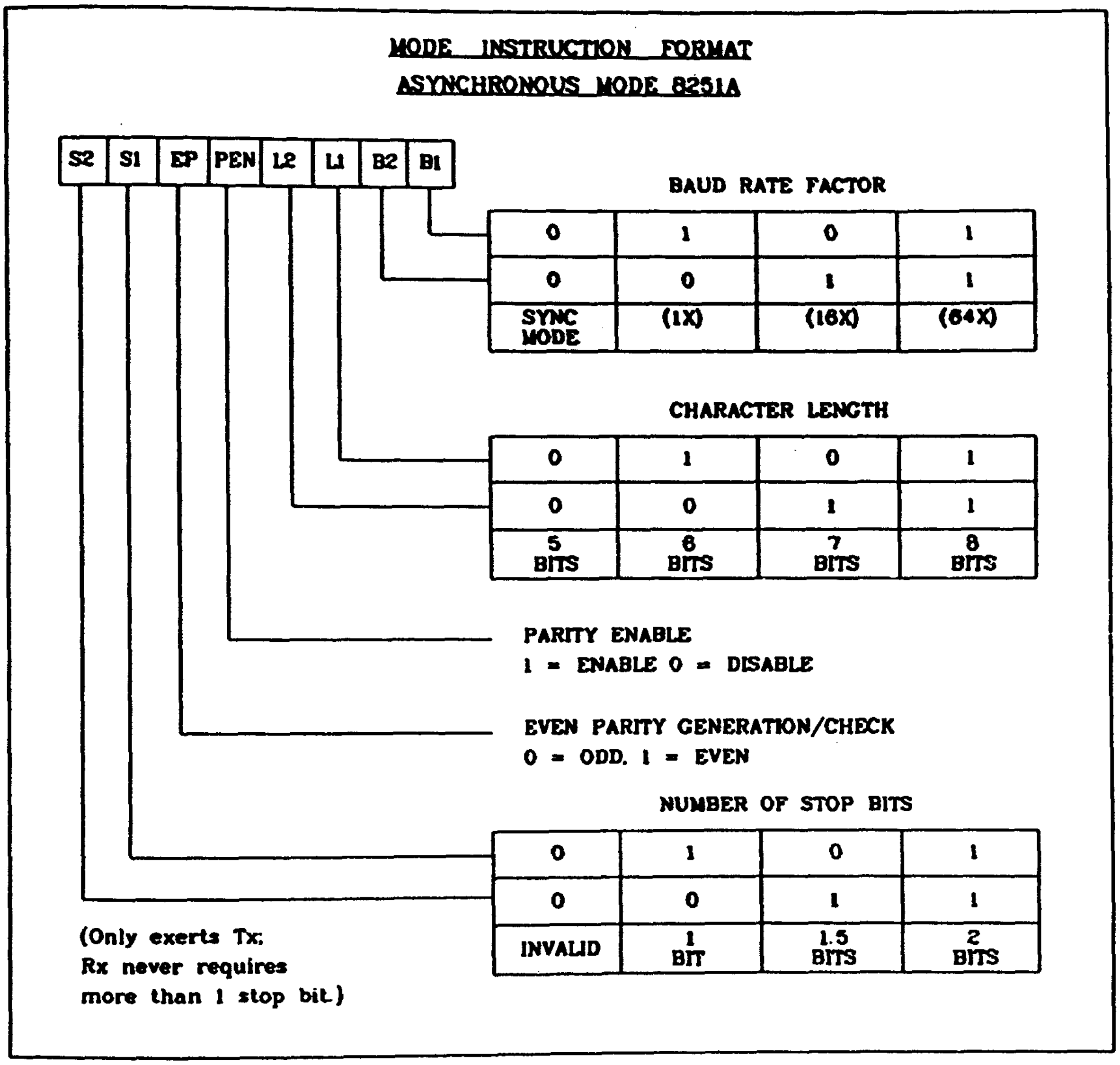


Figure F3.3

SYNCHRONOUS MODE 8251A

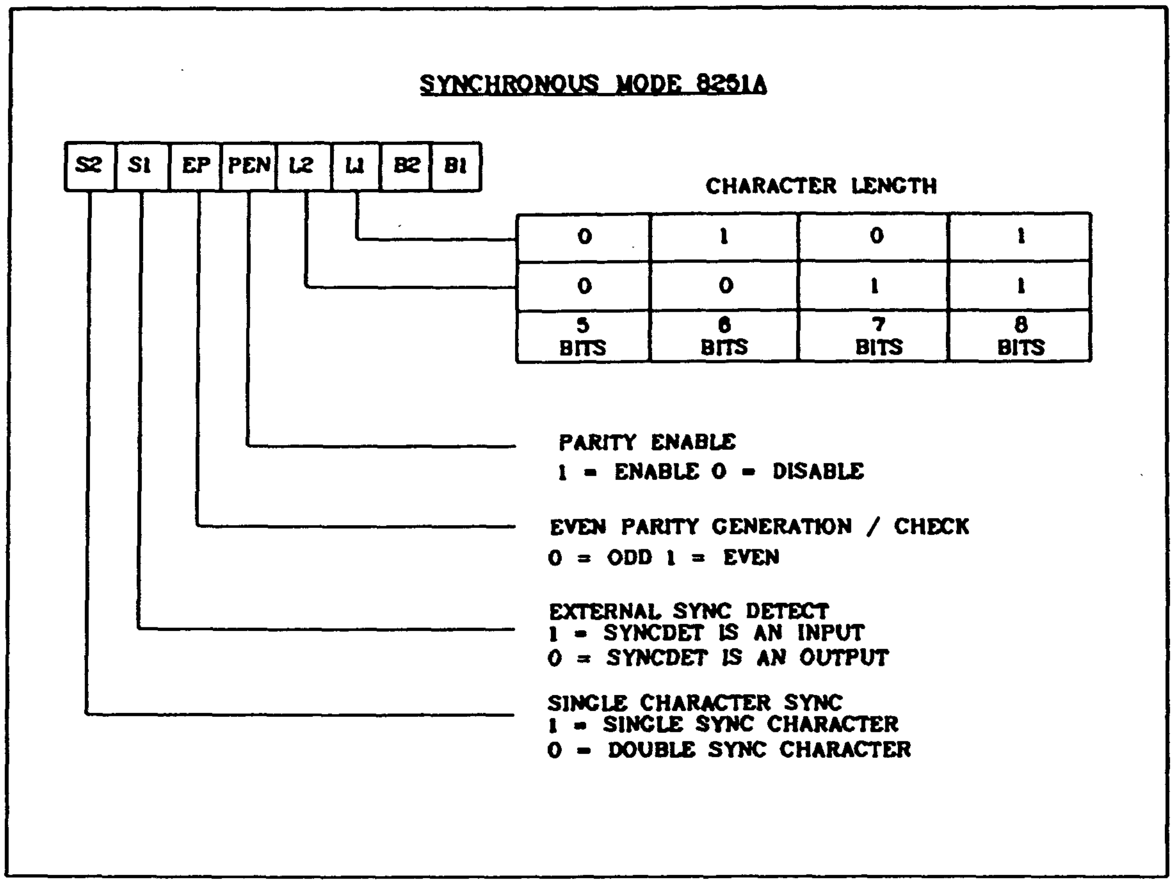

SYNCHRONOUS MODE 8251A
S2
S1
EP
PEN
L2
L1
B2
B1
CHARACTER LENGTH
0 1 0 1
0 0 1 1
5 BITS
6 BITS
7 BITS
8 BITS
PARITY ENABLE
1 = ENABLE 0 = DISABLE
EVEN PARITY GENERATION / CHECK
0 = ODD 1 = EVEN
EXTERNAL SYNC DETECT
1 = SYNCDET IS AN INPUT
0 = SYNCDET IS AN OUTPUT
SINGLE CHARACTER SYNC
1 = SINGLE SYNC CHARACTER
0 = DOUBLE SYNC CHARACTER

Figure F3.4

COMMAND INSTRUCTION FORMAT 8251A

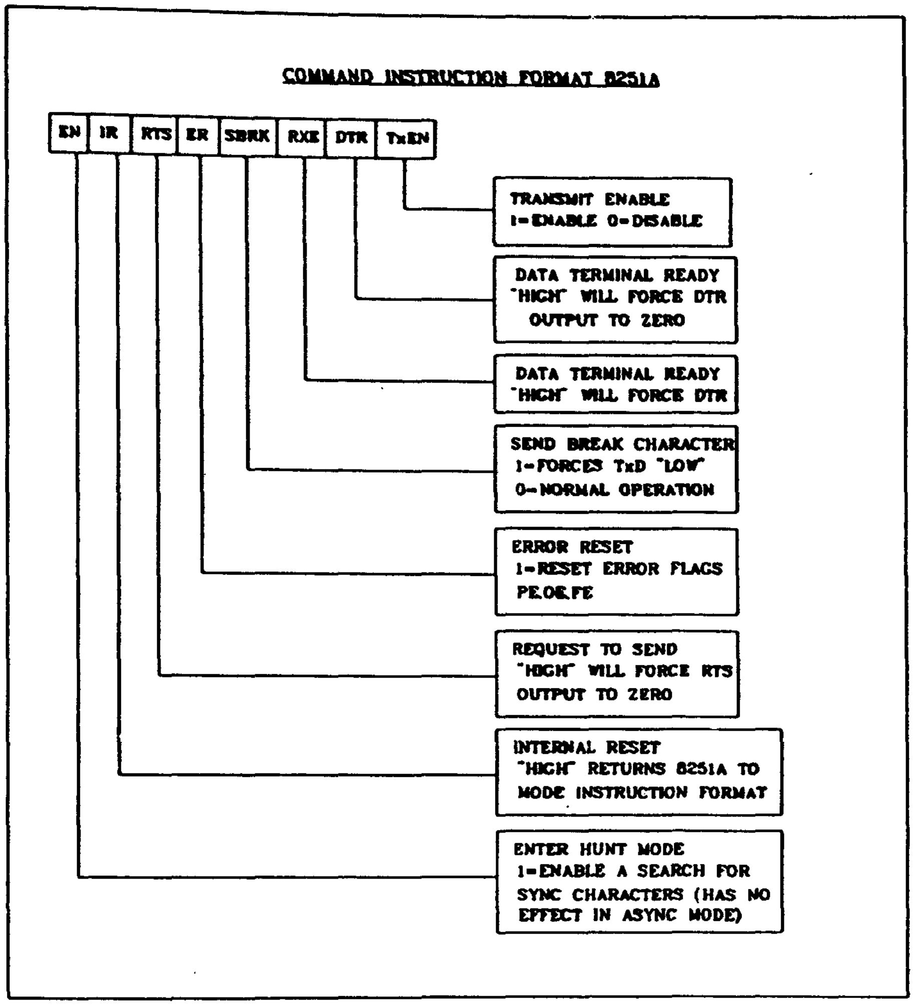

COMMAND INSTRUCTION FORMAT 8251A
EH
IR
RTS
ER
SBRK
RXE
DTR
TxEN
TRANSMIT ENABLE 1=ENABLE 0=DISABLE
DATA TERMINAL READY "HIGH" WILL FORCE DTR OUTPUT TO ZERO
DATA TERMINAL READY "HIGH" WILL FORCE DTR
SEND BREAK CHARACTER 1=FORCES TxD "LOW" 0=NORMAL OPERATION
ERROR RESET 1=RESET ERROR FLAGS PE.OE.FE
REQUEST TO SEND "HIGH" WILL FORCE RTS OUTPUT TO ZERO
INTERNAL RESET "HIGH" RETURNS 8251A TO MODE INSTRUCTION FORMAT
ENTER HUNT MODE 1=ENABLE A SEARCH FOR SYNC CHARACTERS (HAS NO EFFECT IN ASYNC MODE)

Figure F3.5

NOTE: ERROR RESET MUST BE PERFORMED WHENEVER RxENABLE AND ENTER HUNT ARE PROGRAMMED

STATUS READ FORMAT 8251A

D7	D6	D5	D4	D3	D2	D1	D0
DSR	SYDD ET	FE	OE	PE	TxEMPT Y	RxRD Y	TxRD Y

Figure F3.6

TxRDY	— TxRDY 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).
RxRDY	— Receiver Ready. This bit indicates that the 8251A contains a character that is ready to be input to the CPU.
TxEMPTY	— Transmitter Empty. When the 8251A has no character to transmit this bit will go high.
PE	— Parity Error. The PE flag is set when a parity error is detected. It is reset by the ER bit of the command instruction. PE does not inhibit operation of the 8251A.
OE	— Overrun Error. The OE flag is set when the CPU does not read a character before the next one becomes available. OE is reset by the ER bit of the command instruction. OE does not inhibit the operation of 8251A 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 the end of every character. It is reset by the ER BIT of the command instruction. FE does not inhibit the operation of 8251A.
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.

a) Programming Sequence :

i) Decide upon baud rate factor, data bits, parity and stop bits depending upon the system with which the kit is to be connected for serial transmission.
ii) Give an external or an internal RESET to 8251A.
iii) Write the mode word to 8251A.
iv) Find out the Command Word as per your requirement and write it to 8251A.
v) Now all writes to 8251A are command words and all reads are status informations.
vi) To go back to the Mode word a RESET is a must (either external or internal)

Initialising 8251A using the Mode instruction to the following

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

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 8251A can be used for serial transmission or reception. It has an inbuilt buffer for both transmission and reception. The Status Register in the 8251A 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 F3.7.

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 IOW* respectively. A clock is given to the CLK input. The active high RESET is given to the Reset input of the 8251A. Address line A1 is connected to C/D* of 8251A. So, when A1 is high, then it is Control and if it is low, it is Data. The CS* is given the USARTSELECT which goes low for addresses 08 and 0A for the 8251A.

The outputs of 8251A are brought out through the RS232C Driver Chips, ICL232. The outputs of the driver chips are terminated at connector P3.

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

The clock input to 8253 is 1.25 MHz. To get a 9600 Baud rate on 8251A, the clock to 8251A should be 153 KHz, as an internal dividing factor of 16x is selected for 8251A. Hence to initialise 8253 to give a clock frequency of 153 KHz, the counter of 8253 is to be loaded with a count which is derived as,

Clock input to 8253 = 1255 KHz
Output clock required = 153.6 KHz.

Then, Count = 1255/153.6 = 8.1706 = 8

Block diagram

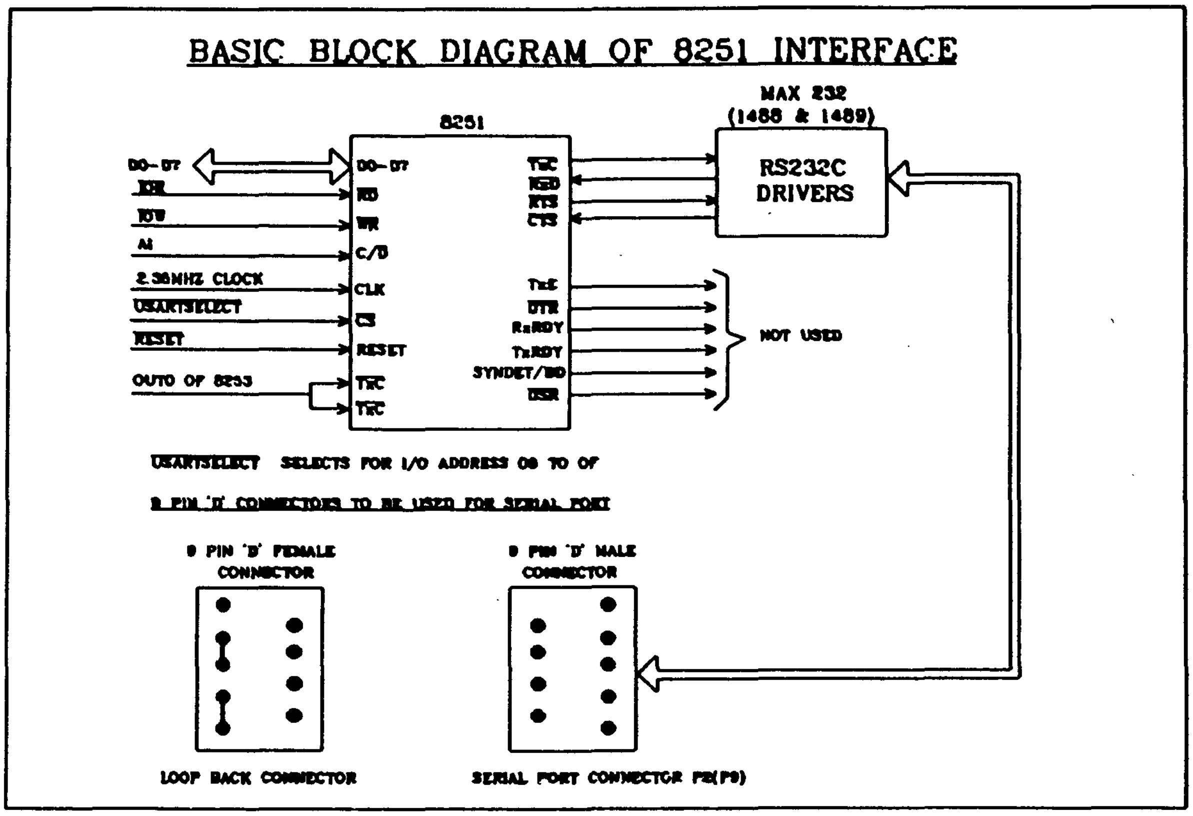


Figure F3.7

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 Micro-86/88 EB 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 8251A, so that the data transmitted can be read back without having another system to check the serial port. The connections for the loop back connector are,

```
1 —> NC      6 —> NC
2 —> ┐       7 —> NC
3 —> ┘       8 —> NC
4 —> ┐       9 —> NC
5 —> ┘
```

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 8251A and status will be set accordingly so that CPU can read the data from the 8251A.

The I/O address for the USART is as follows:

IC No.	Function	Address
U29	Control Status	000A
U29	Data	0008

3.3.5 Example 2 - Transmitting And Receiving a Character Using 8251:

Objective:

To check the transmission and reception of a character.

Theory:

The program reads the status register and checks 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 1100. 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:

```
                        ;Check for Txempty
                        ;then send data 41 to 8251
                        ;
1010  E4 0A             loop: in     al,0A
1012  F6 C0 04                test   al,04
1015  74 F9                   jz     loop
1017  C6 0 41                 mov    al,41
101A  E6 08                   out    08,al
                          ;
                          ;Check for RxRdy and get data
                          ;Store data at 1100
                          ;
101C  E4 0A             loop1:   in     al,0A
101E  F6 C0 02                   test   al,02
1021  74 F9                      jz     loop1
1023  E4 08                      in     al,08
1025  88 06 00 11                mov    [1100],al
1029  F4                         hlt
                                 end
```

Verification:

Check at location 1100 using substitute command. If the contents of this location is 41 then the serial port is OK. Else check the loop back connector and verify RTS*, CTS*, TxD and RxD using an oscilloscope.

3.3.6 Exercises:

i) Write a program to transmit data 00 to FF (stored in memory) to the Serial Port and receive them back and store at address starting from 1500. So, after executing the program, check at locations starting from 1500 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.

iv) Reset the USART. Re-initialize it and transmit and receive data 42.

3.4 Parallel Interface: U13, U23

3.4.1 Component:

The Intel 8255, Programmable Peripheral Interface is the device used to provide the parallel interface for the M - 86/88 LCD. One PPI 8255 IC is available in this trainer providing 24 I/O lines for user interface.

The 8255 is a widely used, programmable, parallel I/O device. It can be programmed to transfer data under various conditions, from simple I/O to interrupt I/O. It is flexible, versatile and economical (when multiple I/O ports are required), but somewhat complex. It is an important general purpose I/O device that can be used with almost any microprocessor.Its unique features are listed below:

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

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. It has a CONTROL REGISTER to which the modes of the three ports are written into. It cannot be read.

a) Basic System Interface:

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

RD*	-	Active Low Read
WR*	-	Active Low Write
CS*	-	Active Low Chip Select
A0,A1	-	Select Control Register ' Port
D0-D7	-	Bi-directional Data Bus

The basic operation of the 8255 using the above signals is as below:

A1	A0	RD*	WR*	CS*	FUNCTION
					INPUT OPERATION (READ)
0	0	0	1	0	PORT A => DATA BUS
0	1	0	1	0	PORT B => DATA BUS
1	0	0	1	0	PORT C => DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS => PORT A
0	1	1	0	0	DATA BUS => PORT B
1	0	1	0	0	DATA BUS => PORT C
1	1	1	0	0	DATA BUS => CONTROL
					DISABLE FUNCTION
X	X	X	X	1	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.

Programming the Control Register:

The Figure F3.8 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.
For Basic mode definition

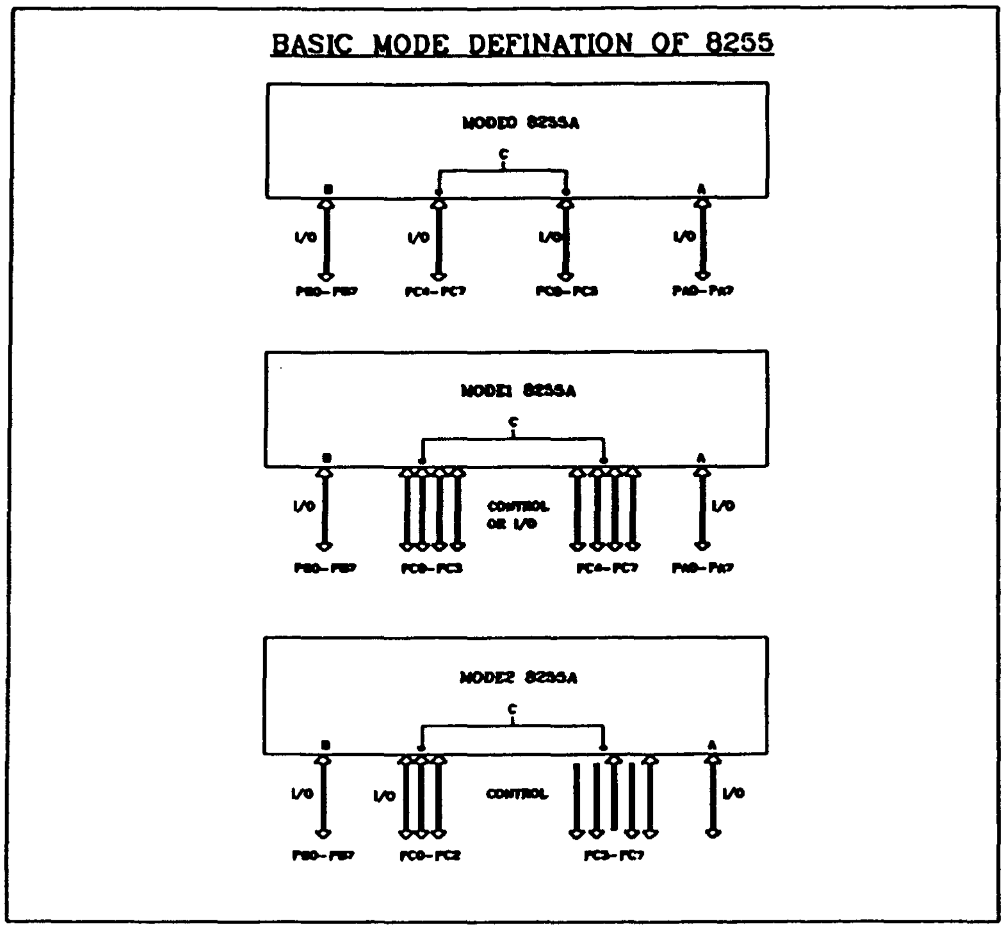


Figure F3.8

The Figure F3.9 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 F3.10. 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.

Programming Sequence:

From the above discussion it should be clear that the sequence involved in configuring the 8255 consists of

i) Writing the mode definition format byte into the control register.

ii) Writing/Reading the data from the ports according to their configuration

3.4.4 Circuit Implementation:

The Figure F3.11 gives a clear idea of the interface between the 8255 and the 8086 microprocessor.

As seen from the block diagram the D0-D7 lines are connected to the processor data lines. The RESET, RD* and WR* are connected to RESET, IOR* and IOW* lines of the circuit respectively. The CS* is got from 8255SELECT* which goes low for I/O addresses 20 to 26 for 8255 (U30), 50 to 56 for 8255 (U31).

MODE DEFINITION FORMAT 8255

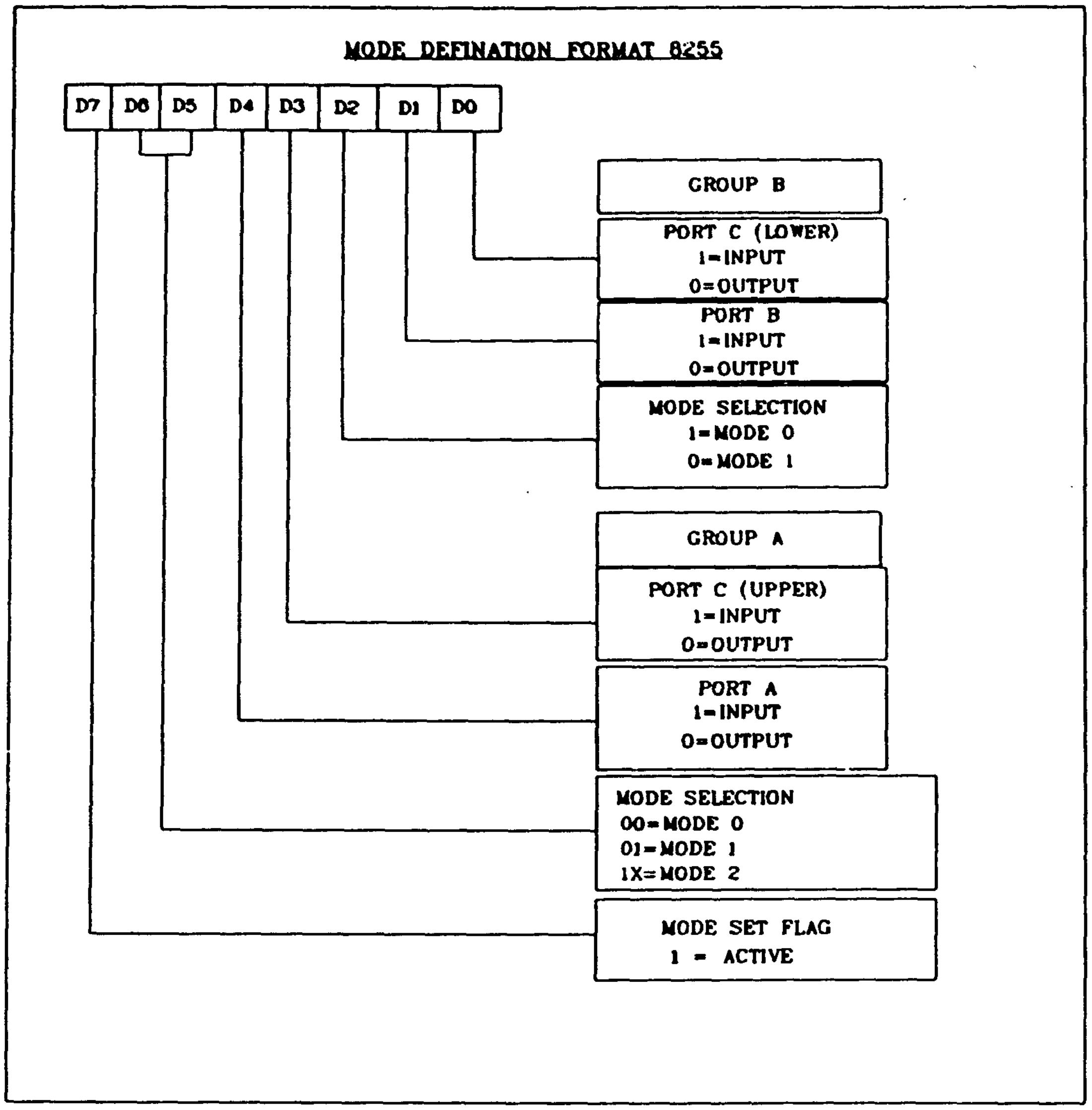

MODE DEFINATION FORMAT 8255
D7
D6
D5
D4
D3
D2
D1
D0
GROUP B
PORT C (LOWER) 1=INPUT 0=OUTPUT
PORT B 1=INPUT 0=OUTPUT
MODE SELECTION 1=MODE 0 0=MODE 1
GROUP A
PORT C (UPPER) 1=INPUT 0=OUTPUT
PORT A 1=INPUT 0=OUTPUT
MODE SELECTION 00=MODE 0 01=MODE 1 1X=MODE 2
MODE SET FLAG 1 = ACTIVE

Figure F3.9

BIT SET / RESET FORMAT 82552

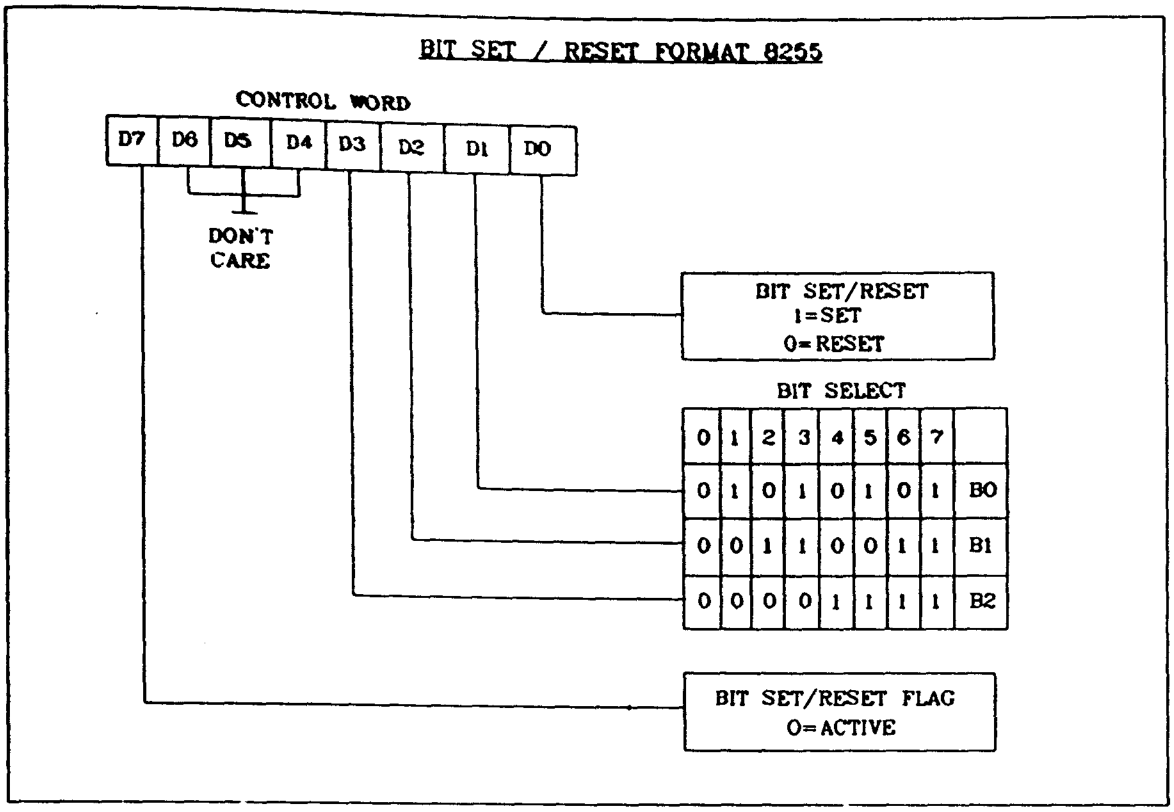


Figure F3.10

Block diagram

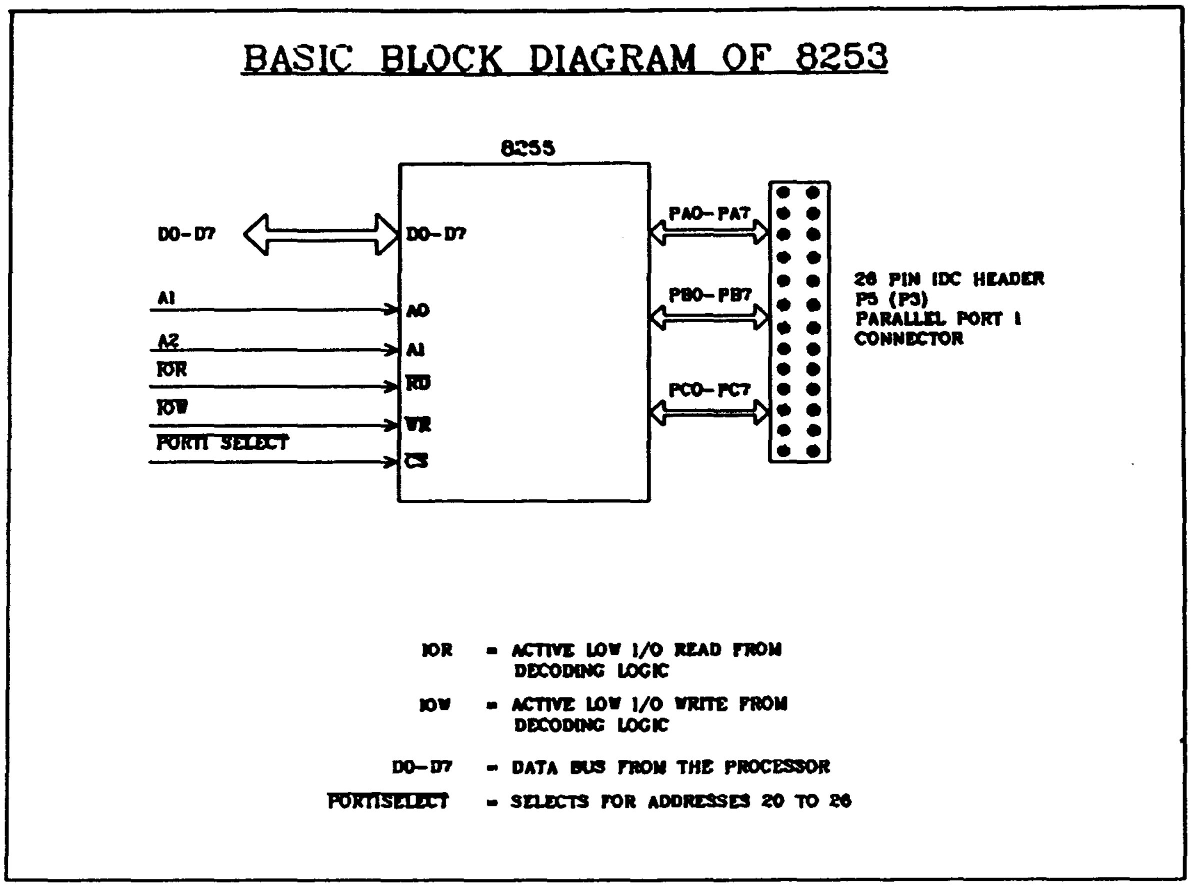

BASIC BLOCK DIAGRAM OF 8253
8255
D0-D7
D0-D7
A1
A0
A2
A1
IOR
RD
IOW
WR
PORT1 SELECT
CS
PA0-PA7
PB0-PB7
PC0-PC7
26 PIN IDC HEADER
P5 (P3)
PARALLEL PORT 1
CONNECTOR
IOR = ACTIVE LOW I/O READ FROM DECODING LOGIC
IOW = ACTIVE LOW I/O WRITE FROM DECODING LOGIC
D0-D7 = DATA BUS FROM THE PROCESSOR
PORT1SELECT = SELECTS FOR ADDRESSES 20 TO 26

Figure F3.11

The following are the I/O addresses for 8255 :

IC No.	Function	Address
U13	Control Register	0026
U13	Port A	0020
U13	Port B	0022
U13	Port C	0024
U23	Control Register	0056
U23	Port A	0050
U23	Port B	0052
U23	Port C	0054

3.4.5 Example 3 - Square Wave Generation:

To generate a square wave at all 24 I/O lines of 8255. The frequency of the square wave depends on the time delay routine used.

Theory:

To generate a square wave using 8255 first initialise all the ports of the 8255 as output ports by writing the corresponding control word.

Program:

```
1000  C6 C0 80                  mov   AL,80
1003  E6 26                     out   26,AL
1005  C6 C0 FF          start:  mov   AL,FF
1008  E6 20                     out   20,AL
100A  E6 22                     out   22,AL
100C  E6 24                     out   24,AL
100E  30 C0                     xor   AL,AL
1010  E6 20                     out   20,AL
1012  E6 22                     out   22,AL
1014  E6 24                     out   24,AL
1016  E9 EC FF                  jmp   START
```

Verification:

After entering the program given above and executing it, verify for the square wave at the output lines of the 8255.

Exercises:

i) Configure 8255-U13 as output and 8255-U23 as input ports. Connect a 26-core flat cable between the two connectors as shown in the block diagram. Output data to 8255-U13 and check them by inputting through 8255-U23.

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

1. Output 01 to Port A. give a delay of 2 secs.
2. Output 02 to Port A. give a delay of 3 secs.
3. Output 04 to Port A. give a delay of 5 secs.
4. Output 08 to port A. give a delay of 4 secs.
5. Repeat again from step 1.

iii) Using the same board, write a program to count from 0 to FF.

Audio cassete interface for M - 86/88 LCD: (U11,U24)

The Audio cassette interface of M - 86/88 LCD allow the user to store their programs onto a audio cassette and retrieve them back also. The data is stored and retrieved 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 M - 86/88 LCD 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 ke y 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 and keep the volume at the maximum level.

When you issue a"TAPE READ" command by pressing "TR" key and enter the file name, the system searches for the file. When a afile is read the file name is displayed. If that is not the required file then it again searches for another one. If that is the required one then data is transferred from the tape into the memory . The starting and ending address of the memory block are read from the tape itself. A achecksum which is also recorded during a atape write operation will be read back and compared with the checksum 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. Inc such cases, acheck forasny loose connections and re-execute the command.

The I/O addresses for the tape read and write are as follows.

I.C. No.	Function	Address
U11	Tape Write	0040B (0026,0024)
U11, U24	Tape Read	0048B (001A)

The audio cassette is a fairly reliable source for a preservation of afiles. However, to improve the reliability of the operation, the following hints may be considered.

* Ensure that the volume, treble andbass controles are at maximum position.

* Use a good quality recorder (preferable AM354 from philips) and a very good quality Audio cassette.

* Avoid storing relatively large files. If the file is really large split into multiple small files. Though this scheme reduces the utilisation of the tape space, it improves the chances of restoring the completer file correctly.

Theory of Operation:

The Program or data to be stored on the tape is sent out in a serial fashion via D0 line.

The file is read back from the EAR outlet of the recorder. The signal is read through the 74LS125 to the D1 line of the data bus.

In the case of M - 86/88 LCD data is sent out in serial form via PC7 line of 8255(U9) and read back using 74LS244.

Data Format:

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

1. Bit Format:

Each data byte consist 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 2 KHZ pulses followed by 2 cycles of 1KHz pulses. Similarly, a "0" is a combination of 4 cycles of 2KHz 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 bits are sent with theLSB taking the leading position. Each and every bit is tested and the corresponding signals are sent out in a serial fashion. After the end of the data bits, 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 signal as shown in the Fif 3.15. The starting of afile is indicated by a lead sync 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 2 seconds is given. This is followed by the data itself. The file ends with a tail sync of 2KHz pulses for 2 seconds.

3.6 8087 Numeric Data Processor: (U2)

3.6.1. Introduction:

The 8086 is a general purpose microprocessor suitable for most data processing tasks. However, many "real world" problems cannot be solved with integer arithmetic and the four basic math functions provided by the 8086. For example, calculating the sine of an angle or the area of a circle requires a fractional number system and transcendental math functions.

Traditionally, software routines have been written to implement these functions. For example, sin x can be approximated as,

As you can imagine, in some applications the time to execute these functions can becomeexcessive. For this reason, math coprocessors have become popular accessories in

$$\sin x = x - \frac{X^3}{3!} + \frac{X^5}{5!} - \frac{X^7}{7!} + \dots$$

microprocessors intended for scientific or other numeric intensive applications.

In fact, Intel offers two additional processors for the 8086 family. These are the 8087 Numeric Data Processor (NDP) and the 8089 I/O Processor (IOP). Each of these chips is quite complex-truly "microprocessors" themselves, capable of executing special coprocessor instructions in parallel with the host CPU. Indeed, Intel would have us consider the coprocessor as an extension of the main CPU.

In this section we consider, the programming model of 8087 which is quite different from that of the 8086. It includes registers capable of processing numbers in eight different data formats. An 8087 procedure written to calculate the area of a circle is also presented in this section.

3.6.2. Host Interface

Figure F3.29 shows how the 8087 is interfaced to the 8086 microprocessor. Basically, the two chips are wired in "parallel" with common connections for the address, data, and most control signals. Note that the host must be operated in the maximum mode. This is to accommodate the RQ/GT DMA protocol of the coprocessor and also allows the coprocessor to monitor the host's status via S0-S2 and QS0-QS1.

3.6.3. Typical Operation:

As the host fetches instructions from memory the 8087 does likewise - "looking" for an instruction with a special ESC prefix. Because of the host's prefetch queue, instructions being fetched are not the same as those being executed. QS0 and QS1 allow the coprocessor to track instructions as they proceed through the queue. S0-S2 provide information on the bus status. The decoding of S0-S2 is given in Chapter 8 of the Manual "Micro-86/88 EB Technical Reference".

When the host encounters an ESC instruction it calculates the effective address (EA) and peroms a "dummy" read cycle from this location. That is, the data read is not stored. The 8087, however, captures this address. Now depending on the coprocessor instruction, a memory read or write cycle will be performed by the NDP from this address. Thus the 8087 works in tandem with the host CPU, using it to generate the EA of coprocessor memory operands. The 8087 itself has no EA generation capability.

As you will see, many of the data formats used by the 8087 require multiple word memory operands. To read or write these numbers, the 8087 requests control of the local bus via RQ/GT. All the required bytes are then read or written in consecutive cycles.

The 8087 BUSY output signal alerts the CPU (via its Test input) that an 8087 calculation is in progress. Only when TEST is high should the CPU attempt to access an 8087 result. The 8086 WAIT instruction is provided for this purpose.

3.6.4 System Configuration :

SYSTEM CONFIGURATION - MAXIMUM MODE

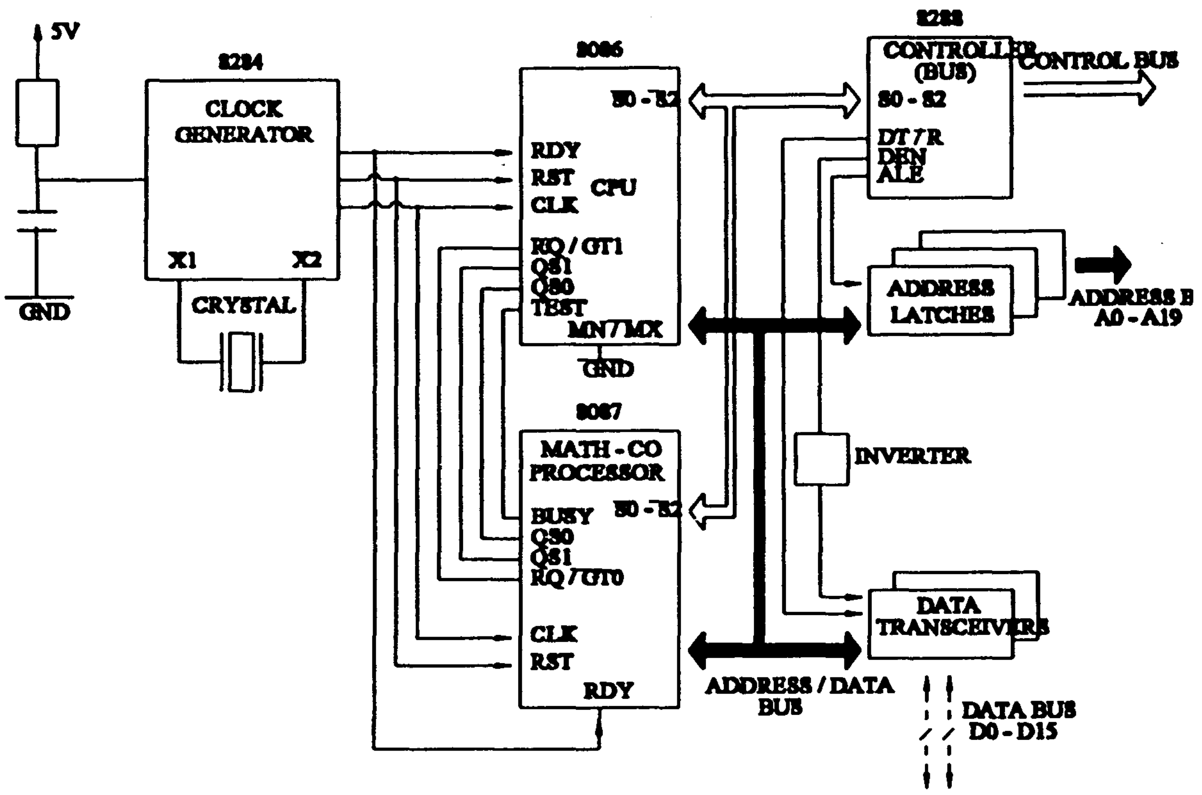


FIGURE F 3.19

3.6.5. Registers in the 8087:

The 8087 has eight 80 bit data registers which can be accessed either as a stack or randomly relative to the top of the stack. An operand may be popped from this stack or pushed onto it. The top of stack element is pointed to by the ST bits, which are bits 13, 12 and 11 of the status register. A push operation first decrements ST by 1 and then loads the operand into the new top of the stack element, and a pop operation retrieves the top of the stack and then increments ST by 1.

As a conventional register file, each register may be referenced by using an index to the stack pointer. This is called relative stack addressing. For relative stack addressing the registers are considered to be in a circle with register 7 being next to register 0. For instance, if ST contains a 3, then ST(2) and ST(6) represent register 5 and register 1, respectively. Because all numbers are internally stored in the temporary real format, each register consists of 80 bits.

The status register is 16 bits wide. It reports various errors, stores the condition code for certain instructions, specifies which register is the top of the stack and indicates the busy status. The bit definitions are given below. A 1 in bit 0 through 5, 7 or 15 indicates that the given condition exists.

Bit o-I, an invalid operation such as stack overflow, stack underflow, invalid operand, square root of a negative number, etc.

Bit 1-D, the operand is not normalized.

Bit 2-Z, a divide by zero error.

Bit 3-0, an exponent overflow error, i.e., the biased exponent is too large.

Bit 4-U, an exponent underflow, i.e. the biased exponent is too small.

Bit 5-P, a precision error, i.e., the result is not exactly representable in the destination format and roundoff has been performed. This indication is normally used only for applications where exact results are required.

Bit 6-Reserved.

Bit7-IR, an interrupt request is being made.

Bits 8, 9, 10 and 14 - C0, C1, C2 and C3 indicate the condition code. The condition code is set by the compare tne examine instructions.

Bits 13, 12 and 11-ST, indicates which element is the top of the stack.

Bit 15-B, the current operation is not complete.

After the 8087 is reset or initialised, all status bits except the condition code are cleared.

Although 8087 recognises six eror types, each error type may be individually masked by causing an interrupt by setting the corresponding mask bits to 1 in the control register. These mask bits are denoted as IM (invalid operation), DM (denormalised operand), ZM (divide by zero), OM (overflow), UM (Under flow), and PM (precision error). If masked the error will not cause an interrupt but, instead, the 8087 will perform a standard response and then proceed with the next instruction in sequence. The precisionn error, for which the standard response is to "return the rounded result", should be masked for floating point arithmetic because for most applications precision errors will occur most of the time. Precision errors are of consequence only in special situations.

An interrupt request, including error-related requests, will be generated also depending upon the interrupt enable mask bit (IEM) in the control register. When this bit is set to 1, all interrupts are disabled except when the CPU is executing a WAIT instruction. If IEM is 0, an unmasked error can cause an interrupt to be sent to the CPU so that the error can be handled by an error-handling-interrupt routine. In the error-handling routine, the current instruction pointer and operand pointer, which are stored in the 8087, can be examined by the 8086 by first putting them into memory. This can be done by using the appropriate 8087 instructions. The contents of these pointers identify the instruction and operand addresses when the error occurred. Note that only the least significant 11 bits of the instruction ocde are kept in the instruction pointer register because the most significant 5 bits are always 11011, the op code of an ESC instruction.

The remaining bits in the control register provide flexibility in controlling precision (PC), rounding (RC), and infinity representations (IC). These bits are defined as follows,

PC bits	-	00 means 24-bit precision
		01 is reserved
		10 means 53-bit precision
		11 means 64-bit precision
RC bits	-	00 means round to nearest
		01 means round toward - x
		10 means round toward + x
		11 means truncation.
IC bit	-	0 indicates that +x and -x are treated as a single unsigned infinitive.
		1 indicates that +x and -x are treated as two signed infinitives.

During a reset or initialisation of the 8087, it sets the PC bits to 11, RC bits to 00, IC bit to 0, IEM bit to 0, and all eror mask bits to 1.

The tag register hold the status of the contents of the data registers. Each data register is associated with two tag bits which indicate whether its contents are valid (00), zero (01), a special value - i.e. NAN, infinity, or denormal - (10), or empty (11).

Note: 'x' indicates infinity.

3.6.6 EXAMPLE 9 - TO DETECT THE PRESENCE OF 8087

OBJECTIVE : To find the co-processor 8087 is present in the trainer or not.

THEORY : Upon initialisation using the finit instruction, the control register of the 8087 gets initialised to a certain value. The precision control bits are set to 11 upon 8087 reset. The program loads these bits and compare with 03. If equal it indicates the presence of 8087 and displays a co-processor is present checks whether the co-processor is precent in the system or not and displays messages accordingly.

PROGRAM:

```
                              ; Program to find the presence
                              ; of Co - Processor in the
                              ; Micro - 86/88 trainer!
                              ;
                              ;
1000  B0 90                          mov   al, 90h
1002  E6 02                          out   (02), al
1004  B9 0800                        mov   cx, 08
                              ;
1007  DB E3                          finit
1009  32 E4                          xor   ah, ah
100B  88 26  3D01                    mov   byte ptr control+1, ah
100F  D9 3E 3C01                     fstcw byte ptr control
1013  8A 26 3D01                     mov   ah,byte ptr control +1
1017  80 FC 03                       cmp   ah,03
101A  75 0B                          jne   no_coproc
101C                  co_proc:
101C  BF 2C10                        mov   di, offset msg1
101F                  Ye_coproc:
101F  8A 05                          mov   al,[di]
1021  E6 00                          Out   (0), al
1023  47                             inc   di
1024  E2 F9                          loop  Ye_coproc
1026  F4                             hlt
                              ;
1027                                 no_coproc:

1027  BF 3410                        mov   di, offset msg2
102A  EB F3                          jmp   ye_coproc
                              ;
102C  91 BF C6               msg1   db    91h, 0bfh, 0c6h
102F  A3 8C AF                      db    0a3, 8ch, oafh
1032  A3 C6                         db    0a3h, 0c6h
1034  C8 BF C6               msg 2  db    0c8h
```

```
1037  A3 8C AF
103A  A3 C6
                                ;
103C  00                        Control db      00
```

3.6.7 EXAMPLE 8 - TO FIND THE AREA OF A CIRCLE

OBJECTIVE : To write a small program to find the area of a circle, fetching the radius from memory and storing the area in memory.

THEORY : Referring to the program below :

The first instruction resets (initializes) the 8087. Then the radius is to be fetched from memory location 1500. Since the radius is assumed to be an integer, we are employing an integer load instruction. Now the stack top contains the radius. Since the 8087 employs relative stack addressing we obtain r^2 by multiplying ST with ST (0). Then π is loaded employing a constant load operation. Now since top of stack element contains by π and the r^2 value is in the next element, we use ST and ST (1) for the next multiplication. Before storing the result, it is covered to packed decimal and the result stored at DS: (1600).

PROGRAM

```
                                ; Program to find the Area
                                ; a Circle whose radius is
                                ; at location 1500h. The
                                ; Area is stored as BCD at
                                ; 1600 ! ST = Top of Stack
                                ;

1000  9B DB E3                  finit                   ; Initialise
                                                        ; 8087
1003  PB DF 06 0015             fild    ds:[1500h]      ; radius
1008  9B D8 C8                  fmul    st, st(0)       ; St = r*r
100B  9B D9 EB                  fldpi                   ; St (0) = pi
                                                        ; St (1) = r*r
100E  9B D8  C9                 fmul    st, st(1)       ; St = pi*r*r
1011  9B DF 36   0016           fbstp   ds: [1600h]     ; Area
1016  F4                        hlt
```

3.7 Programmable Interrupt Controller: (8259A) :U30

3.7.1 Component :

The interrupt interface of Micro-86/88 EB comprises of th Intel programmable interrupt Controller 8259A.

Its distinctive characteristics may be briefed as.

i) Eight levle priority controller.
ii) Expandable to 64 levels.
iii) Programmable interrupt modes.
iv) Individual Request Mask capability.
v) Software programmable capability of all modes of interrupt servicing.

3.7.2 Component Description :

The 8259A can handle upto eight vectored priority interrupts and is also cascadable upto 64 vectored priority interrupts without additional circuitry using 8 numbers of 8259A./

The 8259A is designed to minimzie 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 priotity specification and rotating priotiry.

The two registers that are to be written into before actual Operation Command Word Registers. These may require 2 to 4 initialisation command words and operation command words depending upon the design. these two registers must be initialised for proper operation of the 8259A.

Basic System Interface :

The 8259A interacts with the CPU by D0-D7. CS*, RD*, WR*, A0, INT, INTA*.

RD*	-	Active Low Read
WR*	-	Active Low Write
A0	-	Command select address
CS*	-	Active low chip select
INT	-	Interrupt output
INTA*	-	Interrupt Acknowledge input.

The other pins of the 8259 which control interrupt handing are

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

SP / EN*	-	SLAVE PROGRAM / ENABLE BUFFER :In Buffered Mode, control buffer treanceivers (EN). In other modes, indicates whether a Master (SP=1) or a slave (SP=0) 8259A.
IR0 - IR7	-	INTERRUPT REQUESTS:An Interrupt Request is executed by raising an IR input high. It can be either Edge or Level Trigerred Mode.

3.7.3 Programming Description :

Interrupt Sequence :

The powerful features of 8259A in a micro computer system are its programmibility 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 device. The normal sequence of events during an interrupt depends on the type of CPU being used.

The events occur as follows in an 8086 system :

If one or more of the interrupt request are raised high then the 8259A sends an interrupt to the CPU if the interrupt is appropriate.

The CPU acknowledges the interrupt by an INTA*
Upon receiving an INTA* from the CPU, the highest priority ISR bit is reset.

The 8086 initiates a second INTA* pulse. During this pulse the 8259A releases and 8 - bit pointer onto the data bus which is read by the CPU.

This completes the interrupts cycle. In the automatic end of interrupt mode, the ISR bit is reset at the end of the second INTA* pulse. Otherwise it remains set until an appropriate E01 is issued.

After initialisation, the 8259 A will enter into "FULLY NESTED MODE" unless another mode is programmed. The interrupts requests are ordered in priority from 0 thorugh 7 (0-highest). When an interrupt is acknowledged, the highest priority request is determined and its vector is placed on the bus. Additionally, a bit of the Interrupt Service Register (ISO-7) is set. This bit remains set until the Micro-processor issues an End of 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 (Auromatic 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 an interrupt and receive an acknowledge if the processor's Interrupt Enable Flip Flop is enabled.

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

i) Initialisation Command Word (ICW) :

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

ii) Operation Command Word :

These are command words which decide in what interrupt mode the 8259 is operating in These modes are.

a) Fully - Nested Mode.

b) Rotating - Priority Mode.

c) Special Mask Mode.

d) Polled Mode.

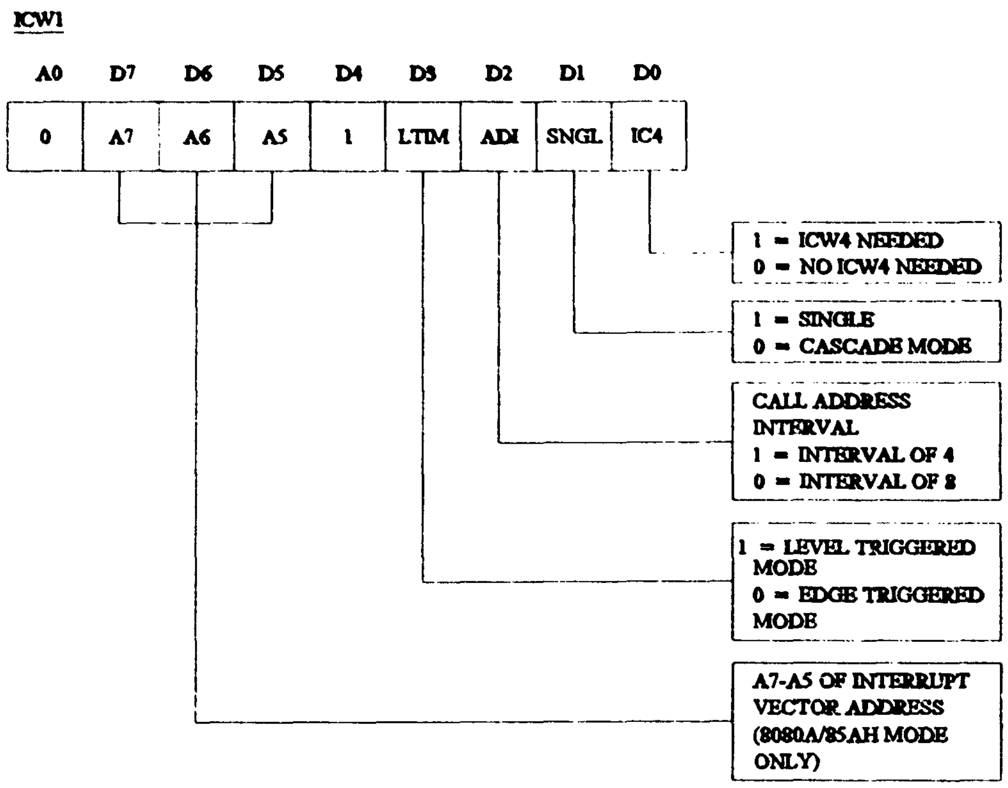
The OCWs can be written into the 8259A at any time after initialisation.

Initialisation :

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

Figure F3.12 defines the initialisation command word format.

\LIZATION COMMAND WORD FORMAT



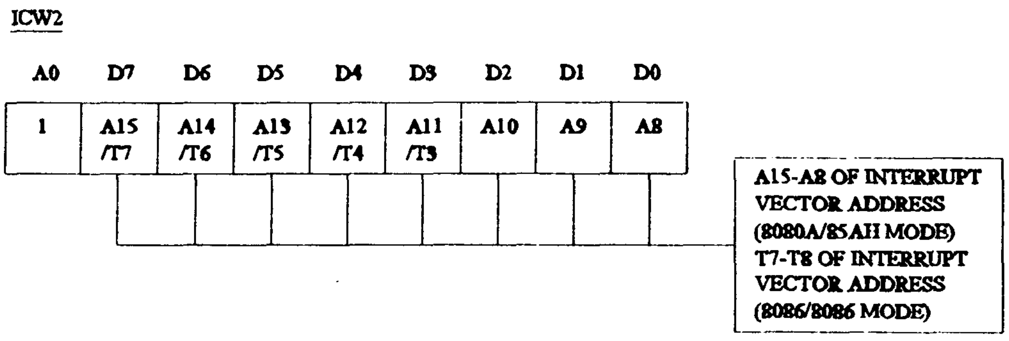
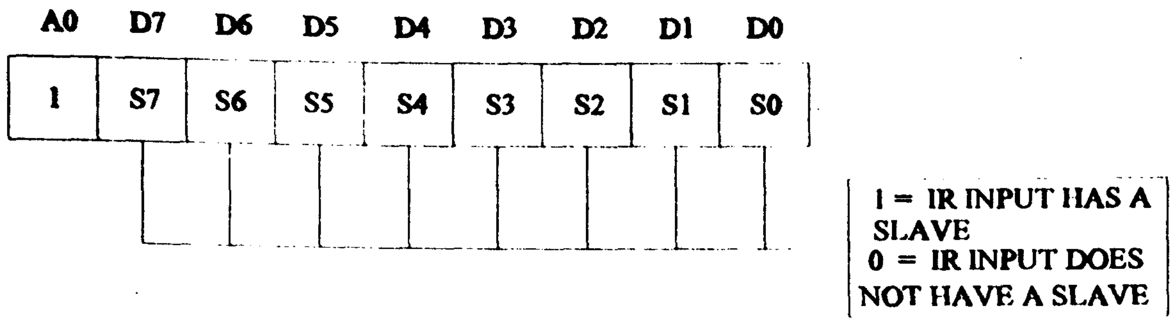

ICW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
1 A15/T7 A14/T6 A13/T5 A12/T4 A11/T3 A10 A9 A8
A15-A8 OF INTERRUPT VECTOR ADDRESS (8080A/85AH MODE)
T7-T8 OF INTERRUPT VECTOR ADDRESS (8086/8086 MODE)

FIGURE F3.12

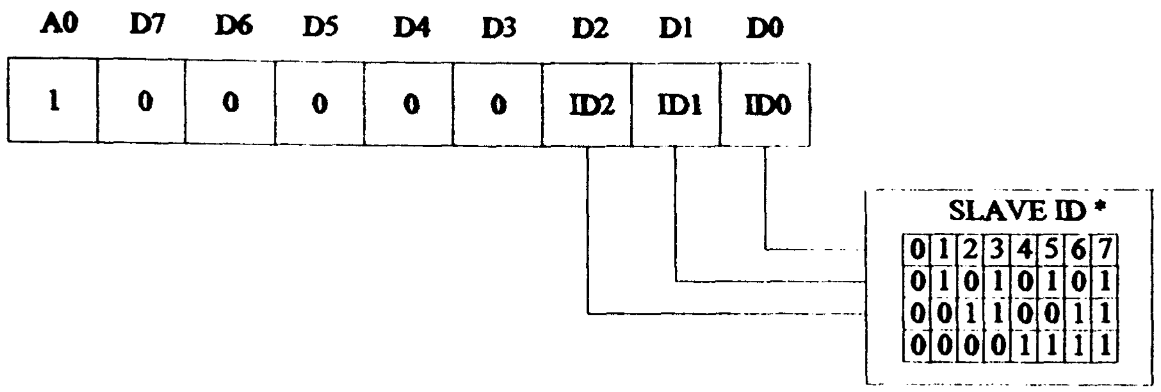
ICW3 (MASTER DEVICE)



A0	D7	D6	D5	D4	D3	D2	D1	D0
1	S7	S6	S5	S4	S3	S2	S1	S0

1 = IR INPUT HAS A SLAVE
0 = IR INPUT DOES NOT HAVE A SLAVE

ICW3 (SLAVE DEVICE)



A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	ID2	ID1	ID0

SLAVE ID *

0	1	2	3	4	5	6	7
0	1	0	1	0	1	0	1
0	0	1	1	0	0	1	1
0	0	0	0	1	1	1	1

SLAVE ID=CORRESPONDING MASTER IR INPUT

FIGURE F3.12

ICW4

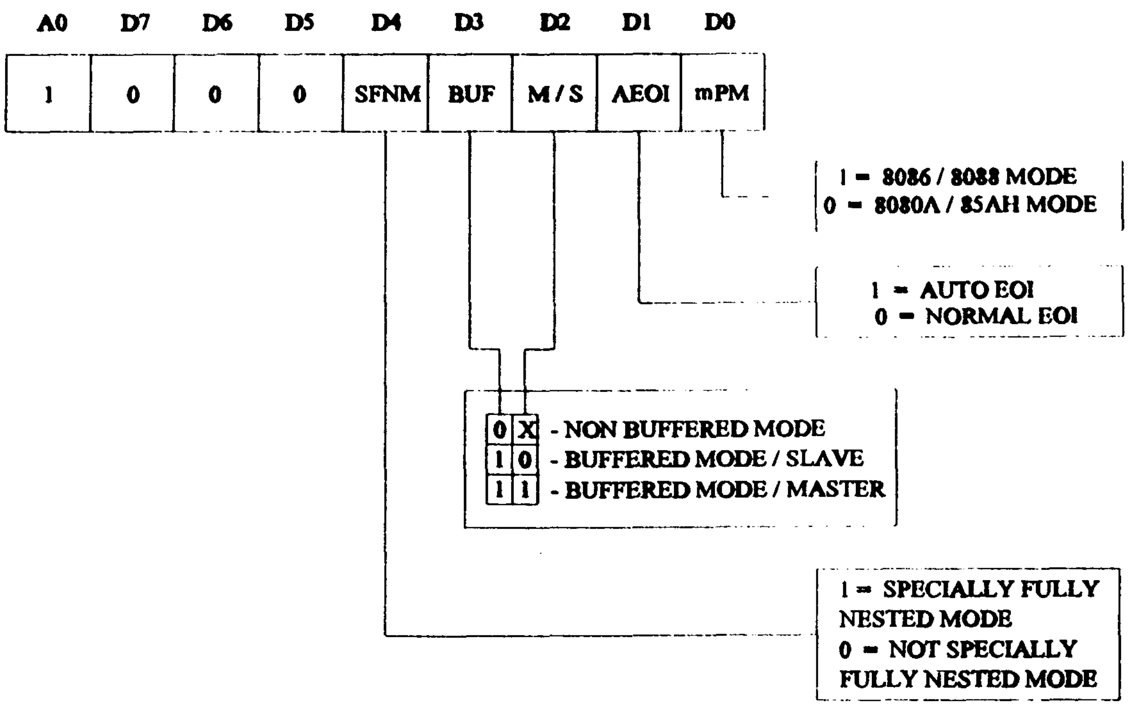
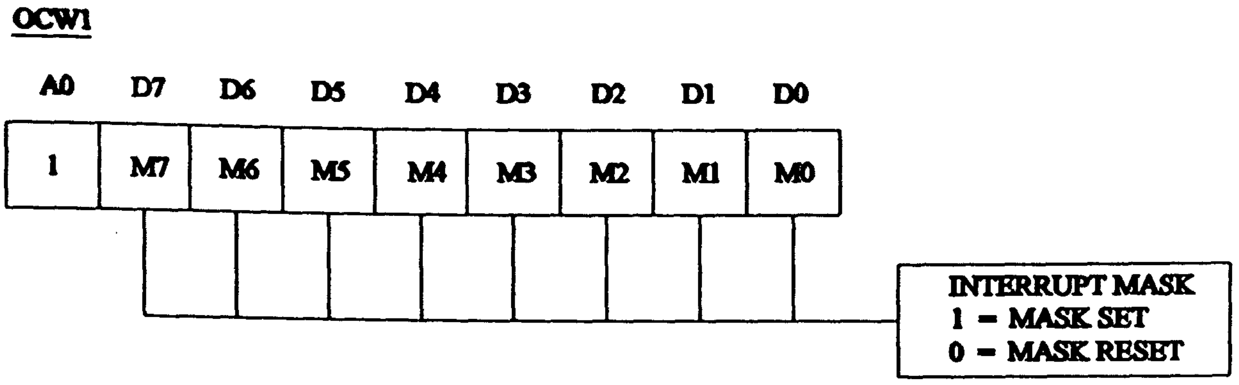


FIGURE F3.12

OPERATION COMMAND WORD FORMAT (8259A)



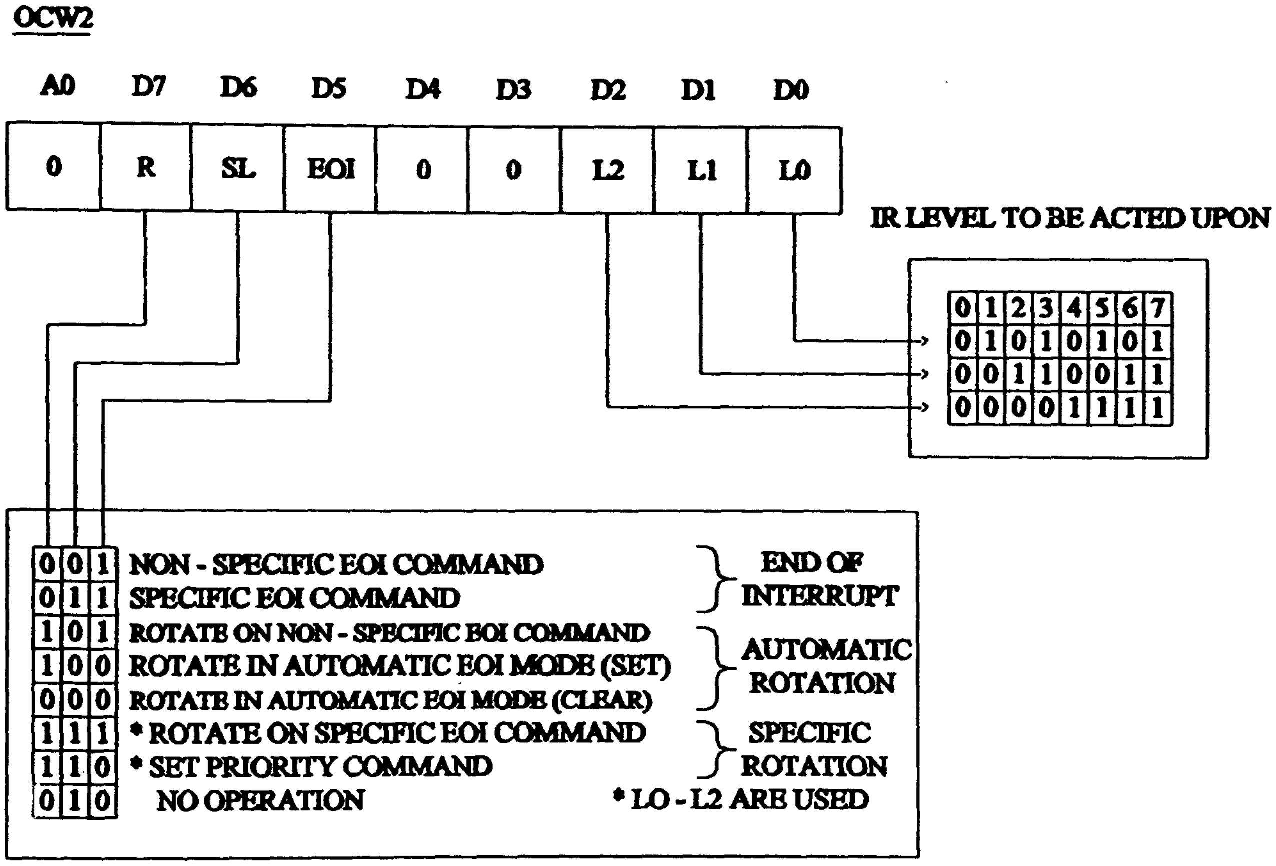

OCW2
A0 D7 D6 D5 D4 D3 D2 D1 D0
0 R SL EOI 0 0 L2 L1 L0
IR LEVEL TO BE ACTED UPON
0 1 2 3 4 5 6 7
0 1 0 1 0 1 0 1
0 0 1 1 0 0 1 1
0 0 0 0 1 1 1 1
0 0 1 NON - SPECIFIC EOI COMMAND
0 1 1 SPECIFIC EOI COMMAND
END OF INTERRUPT
1 0 1 ROTATE ON NON - SPECIFIC EOI COMMAND
1 0 0 ROTATE IN AUTOMATIC EOI MODE (SET)
0 0 0 ROTATE IN AUTOMATIC EOI MODE (CLEAR)
AUTOMATIC ROTATION
1 1 1 * ROTATE ON SPECIFIC EOI COMMAND
1 1 0 * SET PRIORITY COMMAND
SPECIFIC ROTATION
0 1 0 NO OPERATION
* L0 - L2 ARE USED

FIGURE F 3.12

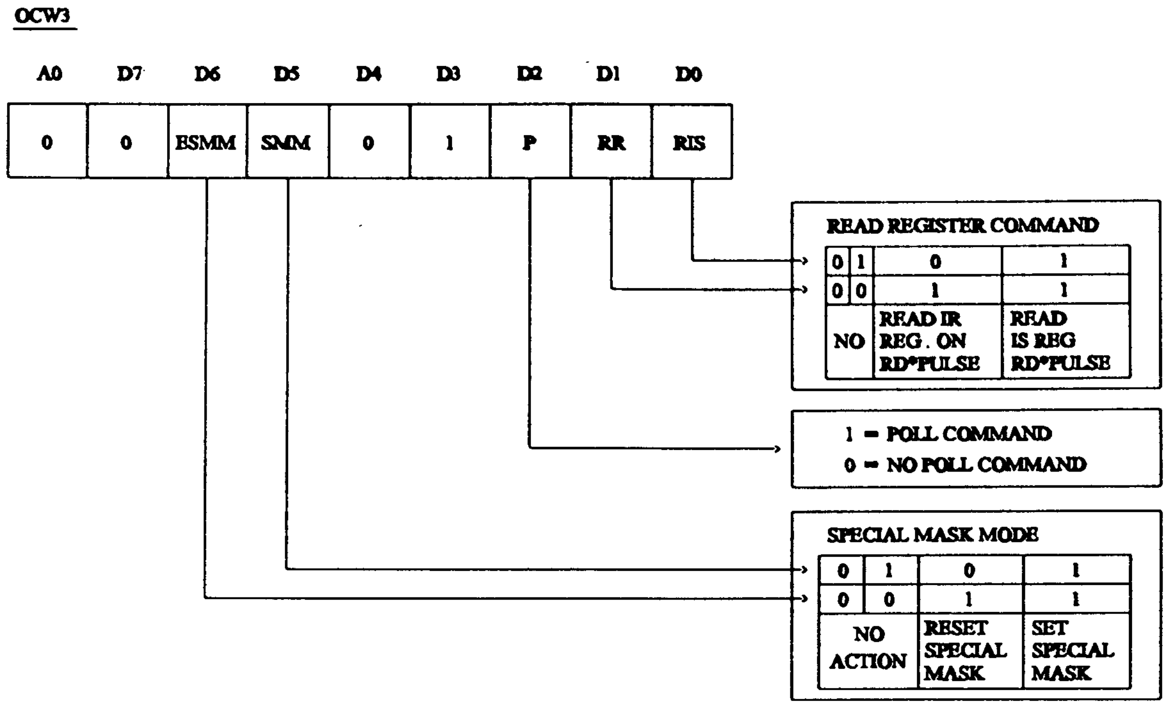


FIGURE F 3.12

Programming the Initialisation Command Registers:

ICW1 :

This word is written to port 0030H. Bits D7 - D5 and D2 are not used in an iAPX 86 system. D4=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 only one 8259A is used ; D0=1 for an 8986 system so that processor can be selected in the ICW4.

Therefore ICW1 = 0001 0011 B (Binary) = 13 (hex)

ICW2 :

This is written to port 0032H. This specfies the five high order bits of the interrupt type number which is to be output by the PIC during INTA bus cycle. Since it is type 8 in your case, ICW2 = 0000 1000 B (Binary) - 08 (hex)

ICW3 :

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

ICW4 :

This word is also output to port 0032H. D0 = 1 for 8086 system, D1 = 1 for Automatic End of Interrupt (AEOI). D2 and D3 should be 1 to specify buffered mode. D4 0 for fully Nested Mode.

Therefore, ICW4 = 0000 1111 B (Binary) = 0F (hex)

Programming the Operation Control Registers :

After the three (four) initialisation control words have been written the PIC is ready to receive interrupts on IR0 - 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.

OCW1 :

This word is written to or read form port 0032H. It is used to mask the interrupts. To enable all mask bits should be reset.

Therefore , OCW1 = 0000 0000 H

OCW2 :

This is written to port 0030H and this is used to specify the EOI command to the PIC. This word is not used as "AEOI" is selected.

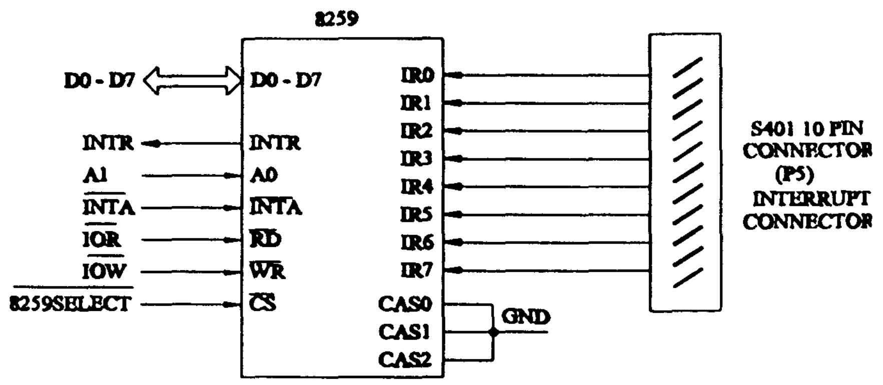
OCW3 :

This is written to port 0030H. Here D3 = 1 and D4 = 0 to distinguish from ICW1 and OCW2. D5 & D6 allow special mask mode to be programmed. This word is not used.

3.5.4 Circuit Implementation :

The Figure F3.13 given below gives a clear idea of the interface between the 8529A and the Microprocessor.

BASIC BLOCK DIAGRAM OF 8259 INTERFACE



IOR = ACTIVE LOW I/O READ FROM DECODING LOGIC
IOW = ACTIVE LOW I/O WRITE FROM DECODING LOGIC
D0 - D7 = DATA BUS FROM THE PROCESSOR
8259 SELECT = CHIP SELECT FROM ADDRESS DECODING LOGIC ADDRESSES 30 TO 32
INTR = INTERRUPT REQUEST TO CPU
INTA = INTERRUPT ACKNOWLEDGE FROM CPU
CAS = CASCADE LINES NOT USED
IR = INTERRUPT REQUEST INPUT TO 8259

FIGURE F 3.13

As shown in the block diagram, the D0 - D7 lines are connected to microprocessor data bus. The RD, WR, A0, INT, INTA lines of 8259A are connected the IOR, IOW, A1, INT, INTA lines of the circuit respectively. The IR0 - IR7 interrupt request level are given to a 10 pin connector for user interface. The CAS0 - CAS2 lines are not used, for this circuit consist of only a single 8259A.

The I / O Addresses for the 8259A are as follows :

IC.NO.	FUNCTION	ADDRESS
U30	ICW1	0030
U30	ICW2	0032
U30	ICW3	0032
U30	ICW4	0032
U30	OCW1	0032
U30	OCW2	030
U30	OCW3	030

As you have learnt the organisation and programming of the programmable 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 intialise the programmable interrupt controller 8259A and write its interrupts service routine and use IRO of the 8259A.

Theory :

Let us initialise the interrupt vector type as 0008. The procedure involves loading the ICW1, ICW2, the ICW4 and the OCW1 and the address of the ISR at the interrupt vector. These should initialise the 8259 to type 8, buffered mode master, auto end of the interrupt, 8086 / 8088 mode in the main routine and the address. The Interrupt vector for the interupt IRO is 0000 : 0020. So at 0000 : 0020, enter the address 0000 :1200H if the ISR is at the address 0 :1200.

```
0000 : 0020 = 00        0000 : 0022 = 00
000C : 0021 = 12        0000 : 0023 = 00
```

The following software will initialise 8259A as said before. The interrupt service Routine at 1200H is also given here.

Program :

```
                    ; Interrupt generation thro'
                    ; 8259A using IRO Interrupt
                    ;
                    ;
1000                  init:
1000  B0 13                  mov   al,13
1002  E6 30                  out   30,al
1004  B0 08                  mov   al,08
1006  E6 32                  out   32,al
1008  B0 0F                  mov   al,0f
100A  E6 32                  out   32,al
100C  B0 00                  mov   al,00
100E  E6 32                  out   32,al
1010  FB                     sti
1011  EB    FE         here :  jmp   here
                    ;
                    ; Interrupt Service Routine
                    ; This Routine will display
                    ; "Int - 00" in the display !
                    ;
```

```
1200  B3 0A         serve :   mov   bl,0ah
1202  BF 13 12              mov   di, offset table
1205  B0 90                 mov   al,90h
1207  E6 02                 out   02,al
                 :
1209                disp :
1209  8A 05                 mov   al, [di]
120B  E6 00                 out   00,al
120D  47                    inc di
120E  FE CB                    dec   bl
1200  75 F7                 jnz disp
1202  F4                    hlt
                 :
1213                table :
1213  F9 AB    87 BF           db  0f9h, 0abh, 087h, 0bfh
1217  C0 C0 FF FF           db  0c0h, 0c0h, 0ffh, 0ffh
```

3.5.6 Exercise :

i) Write a program to set the 8259A 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 ?

CHAPTER-4

SERIAL DATA COMMUNICATION

It becomes a very tedious task to find out opcodes and enter into the trainer in the form of hex codes, when the program size becomes large. It then becomes a necessity to use a PC based assembler for large programs, for effective time utilisation. Now, the opcodes can be transferred from the PC to the trainer. We provide a Data Communication Package for such a transfer from the PC/trainer to the trainer/PC, which transfers data in a serial fashion.

In this chapter, we will be dealing with the usage of this communication package called **DATACOM**, with the M-86/88LCD Trainer.

Note that the messages in **boldface** denote messages that will be displayed on the host terminal while working with DATACOM.

4.1. DATACOM - Basic Features:

DATACOM is a Menu-oriented, User friendly, Serial Communication Package. You can either download to a trainer or upload to a host using Datacom. Selection of Baud rates from 110 to 9600, parity bits and stop bits are determined by the initialisation.

When your host is ready with the DOS prompt, type in

A>DATACOM <CR>

Now the MAIN MENU is displayed as shown below:

```
                    DATA TRANSFER PROGRAM
          Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
               ------------ MAIN MENU ------------
               F1      Setup
               F2      Transmit Data
               F3      Receive Data
               ESC     Exit to DOS

  Port: COM1   Baud: 9600   Parity: NONE   Stop: 01   Data: 08
```

As seen, the Function keys F1, F2 and F3 are used for the three main operations. The three functions can also be selected using either Cursor Keys or pressing the first letter of the function, for example "S" takes you to "Setup", "T" to "Transmit" and so on.

4.1.1. End of File (EOF):

DATACOM can transmit any type of data. To denote the End of File to the receiving system, it sends out Five question marks ("????") as the EOF.

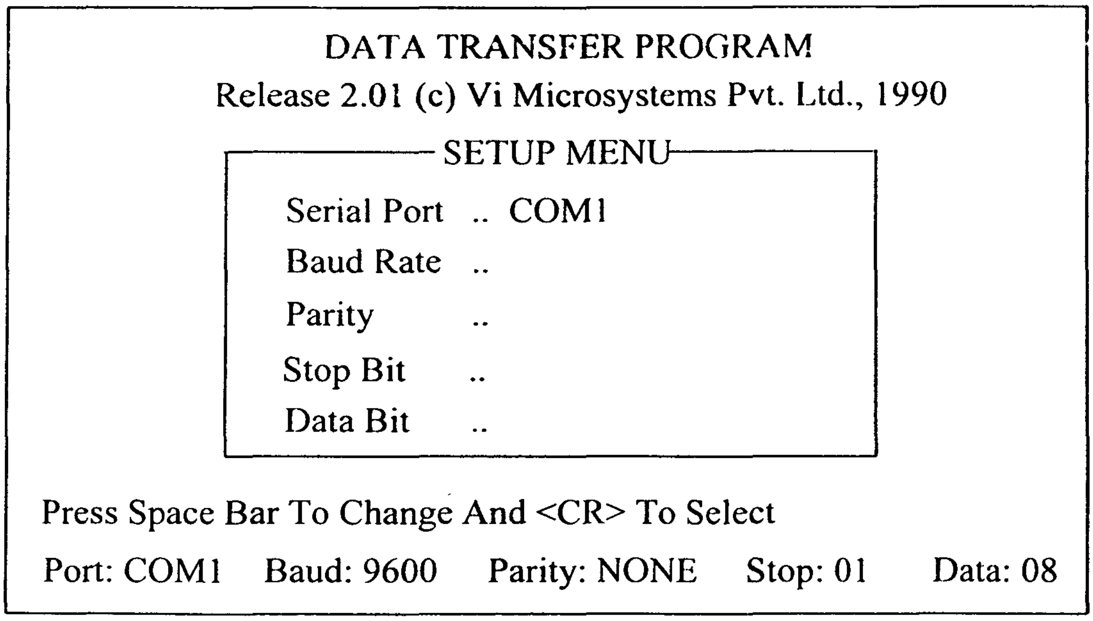
Similarly, DATACOM expects "????" as an EOF when it is receiving data.

The serial port software for the M-86/88LCD Trainer supports this EOF requirement of DATACOM.

The above mentioned are the features available in the DATACOM package. And now to the part which explains communication with the M-86/88 LCD Trainer.

4.2. Setup Host Serial Port:

Select the "Setup" option in the MAIN MENU using either "S" of "F1" or using Cursor keys. The SETUP MENU now appears on the screen as shown below:



```
                 DATA TRANSFER PROGRAM
    Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990

          ----------- SETUP MENU -----------
          |  Serial Port  ..  COM1          |
          |  Baud Rate    ..                |
          |  Parity       ..                |
          |  Stop Bit     ..                |
          |  Data Bit     ..                |
          ----------------------------------

Press Space Bar To Change And <CR> To Select
Port: COM1   Baud: 9600   Parity: NONE   Stop: 01   Data: 08
```

From the MENU, you have got options to change the

Baud rates	---	110, 150, 300, 600, 1200, 2400, 4800, 9600
Parity	---	Even, Odd, None
Stop bits	---	1,2
Data bits	---	7,8

Now use the SPACEBAR to select the protocols and confirm your selection using <CR>. After selection of all the protocols, the current serial port setting is displayed as a Status in the MENU.

```
                        DATA TRANSFER PROGRAM
            Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
               ┌───────────── SETUP MENU ──────────────┐
               │    Serial Port  ..  COM1              │
               │    Baud Rate    ..  9600              │
               │    Parity       ..  NONE              │
               │    Stop Bit     ..  01                │
               │    Data Bit     ..  08                │
               └───────────────────────────────────────┘
  Serial Port Configured                    Press any key to continue ...

  Port: COM1    Baud: 9600    Parity: NONE    Stop: 01    Data: 08
```

Now, the serial port at the host system has been configured as per your selection and is ready for serial transmission.

NOTE: i) Be sure that the other system that is sending/receiving data is also configured with the same protocols. Else proper transmission of data will not occur.

ii) Be sure to SETUP the serial port before transmission and after every time you POWER-UP your system. No default settings are provided during POWER-ON.

4.3. Transmit Data to Trainer:

Ensuring that both the host and trainer are initialised with the same protocols connect the Serial cable from the host to the trainer. For an IBM PC/XT, the cable is a 25-pin D female to a 9-pin D female, the 9-pin D female side going to the trainer. For an IBM PC/AT, it is a 9-pin D female to a 9-pin D female or a 25-pin D female to 9-pin D female depending on the termination. Distinguish between the trainer side female connector and the host side connector.

In the M-86/88 LCD trainer, the default serial port settings are 9600 baud, 8 data bits, no parity and 1 stop bit.

Select the "Transmit data" option in the MAIN MENU of DATACOM. Enter the filename, that is to be transmitted. Assuming a filename of "SA.BIN", the display is as below:

```
                    DATA TRANSFER PROGRAM
         Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
              -----------TRANSMIT MENU-----------
                   File Name ->  SA.BIN <CR>


                   Display the count (Y/N) ?

 Port: COM1   Baud: 9600   Parity: NONE   Stop: 01   Data: 08
```

Only if the file specified is found in the default drive and directory setting, will the option for displaying the count be displayed. Else the user response whether to try again or not is confirmed as,

Cannot open SA.BIN **Wish to try again (Y/N) ?**

If the file is present, the display is as above. The system prompts you for a reply asking you,

Display the count (Y/N) ?

Now, press "Y" or "N" depending upon whether you want the system to display the number of bytes being transferred or not. After this, the system is ready for transmission and says:

Press any key to continue...

Now setup the trainer in the receive mode. A quick glance through the M-86 NT Technical Reference will help you to set up the trainer in the receive mode using the SYNTAX SER, IN, address and NEXT.

Now, if the trainer is ready to receive data, press any key in the host to start transmission. During transmission, the message displayed is

Press ESC to abort

Transmission in progress...

The count is displayed according to user setup to display or not. Without the display of the count, the transmission is fast. You can abort serial transmission halfway by using ESC key. After successful transmission of data, the display is as below:

```
                     DATA TRANSFER PROGRAM
          Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
            ┌─────────────TRANSMIT MENU─────────────┐
            │                                       │
            │         File Name -> SA.BIN           │
            │                                       │
            │                                       │
            │                                       │
            │     <nnn> Kbytes <nnnn> bytes         │
            │                    Transferred        │
            └───────────────────────────────────────┘
  Transmission over
                                  Press any key to continue ......
  Port: COM1   Baud: 9600   Parity: NONE   Stop: 01   Data: 08
```

Now, if the trainer's serial port is proper, then it will display command prompt, indicating successful reception of data.

In case, the communication is not proper, then the display is as

Timer out Error

Press any key to continue...

Try once again now. Check for mismatches in the protocols of the two systems. Even now if the communication is not proper, then contact our Customer Support Division for further clarification.

4.4. Receive Data From Trainer:

Ensure that the protocols of the two systems are identical. If not, initialise properly before transmission of data from trainer.

Selection of "R" or "F3" or the cursor keys takes you to the RECEIVE MENU of DATACOM. Now enter the filename under the incoming serial data from trainer will be stored. Assuming the filename to be "TEMP", the display is as below:

```
                         DATA TRANSFER PROGRAM
           Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
            ┌──────────────── RECEIVE MENU ────────────────┐
            │                                              │
            │             File Name -> TEMP                │
            │                                              │
            │                                              │
            │                                              │
            └──────────────────────────────────────────────┘

                                     Press any key to continue ......
 Port: COM1    Baud: 9600    Parity: NONE    Stop: 01    Data: 08
```

Now the host is ready for reception. Now use the SYNTAX SER, OUT, Start address, NEXT, End address in the trainer to make it ready for serial transmission.

Put the host in receive mode by pressing any key. Instantaneously, press the NEXT key in the trainer.

Notice one practical problem now. The HOST cannot be put in the receive mode for long. It receives data using a hardware based interrupt routine and hence displays 0 bytes received if the required interrupt does not occur.

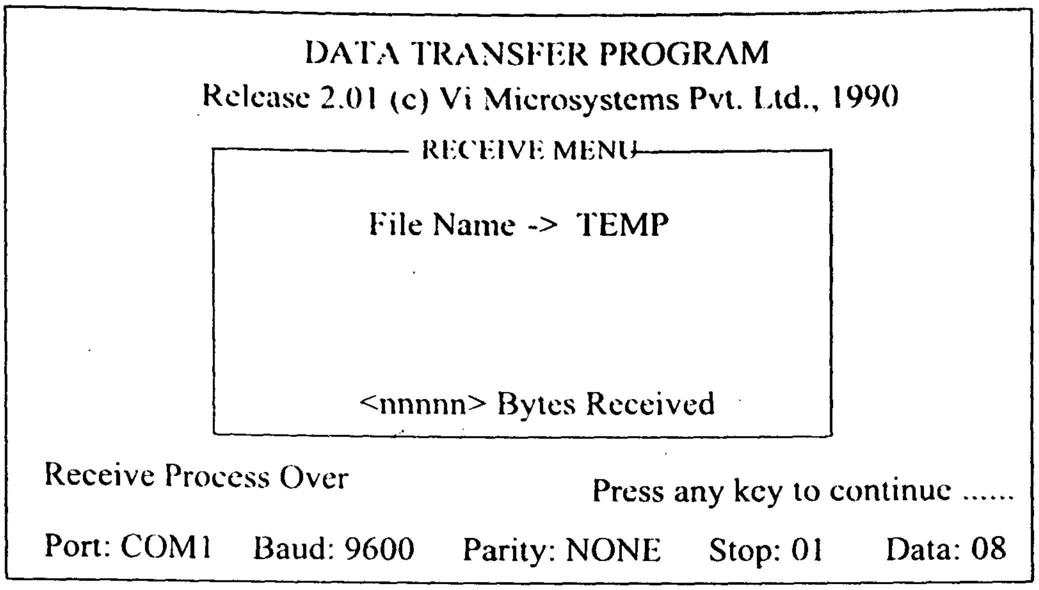
So, do not put the host in the receive mode for long. Send data from the trainer as soon as the Host has come to the receive mode. Hence additional care must be taken during uploading.

When it is receiving data, the display is as,

Receiving to file TEMP

Ensure that the file you are transmitting contains the "?????" EOF mark.

After successful reception, the DATACOM displays the message:



```
                    DATA TRANSFER PROGRAM
        Release 2.01 (c) Vi Microsystems Pvt. Ltd., 1990
          ---------------- RECEIVE MENU ----------------
                     File Name -> TEMP



                   <nnnnn> Bytes Received

Receive Process Over
                                     Press any key to continue ......
Port: COM1   Baud: 9600   Parity: NONE   Stop: 01   Data: 08
```

This ends the DATACOM's Serial Transmission and Reception of data.

Quit DATACOM by using either "ESC" or "E" or using Cursor Keys.

APPENDIX - A

8086 INSTRUCTION SET

This summary is presented only for reference use. For those instructions that can have a variety of operands, the different possibilities are listed. Please refer to outside documentation for more detailed explanations of the exact use of these instructions.

Data Transfer Instructions

A. MOV - Move:

- register or memory location to or from register
- immediate to register or memory location
- immediate to register
- memory location to accumulator
- accumulator to memory location
- register or memory location to segment register
- segment register to register or memory location

B. PUSH - Push:

- register or memory location
- register
- segment register

C. POP - Pop:

- register or memory location
- register
- segment register

D. XCHG - Exchange:

- register or memory location with register
- register with register

E. IN - Input from:

- fixed port
- variable port

F. OUT - Output to:

- fixed port
- variable port

G. XLAT - Translate byte to AL:

H. LEA - Load EA (Effective Address) to register

I. LDS - Load pointer value to DS and register

J. LES - Load pointer value to ES and register

K. LAHF - Load AF with flags

L. SAHF - Store AH into flags

M. PUSHF - Push flags

N. POPF - Pop flags

Arithmetic Instructions

A. ADD - Add:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

B. ADC - Add with carry:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

C. INC - Increment

- register or memory location
- register

D. AAA - ASCII adjust for addition

E. DAA - Decimal adjust for addition

F. SUB - Subtract:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

G. SBB - Subtract with borrow:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

H. DEC - Decrement:

- register or memory location
- register

I. NEG - Negate the contents of a specified register or memory location

J. CMP - Compare:

- register or memory location with register
- immediate with register or memory location
- immediate with accumulator

K. AAS - ASCII adjust for subtract

L. DAS - Decimal adjust for subtract

M. MUL - Multiply (unsigned) accumulator by register or memory location

N. IMUL - Integer multiply (signed) accumulator by register memory location

O. AAM - ASCII adjust for multiplication

P. DIV - Divide (unsigned) accumulator by register or memory location

Q. IDIV - Integer divide (signed) accumulator by register or memory location

R. ADD - ASCII adjust for division

S. CBW - Convert byte to word and perform sign extension from AL to AX

T. CWD - Convert word to doubleword and perform sign extension from AX to DX

Logical Instructions:

A. NOT - Ones complement of register or memory location

B. SHL - Logical left shift of register

C. SAL - Arithmetic left shift of register

D. SHR - Logical right shift of register

E. SAR - Arithmetic right shift of register

F. ROL - Rotate register left

G. ROR - Rotate register right

H. RCL - Rotate register left through carry flag

I. RCR - Rotate register right through carry flag

J. AND - Logical and:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

K. TEST - Logical AND test with result stored in flags

- register or memory location with register
- immediate data and register or memory location
- immediate data and accumulator

L. OR - Logical or:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

M. XOR - Exclusive or:

- register or memory location with register with result stored in either
- immediate to register or memory location
- immediate to accumulator

String Manipulation Instructions

A. REP	-	Repeat
B. MOVS	-	Move byte or word
C. CMPS	-	Compare byte or word
D. SCAS	-	Scan byte or word
E. LODS	-	Load byte or word to AL or AX
F. STDS	-	Store byte or word from AL or AX

Control Transfer Instructions

A. CALL - Call Subroutine:

- direct within segment
- indirect within segment
- direct intersegment
- indirect intersegment

B. JMP - Unconditional jump:

- direct within segment
- direct within segment-short
- indirect within segment
- direct intersegment
- indirect intersegment

C. RET - Return from CALL:

- within segment
- within segment and add immediate to stack pointer
- intersegment
- intersegment and add immediate to stack pointer

D. JE or JZ - Jump on equal to zero **

** The jump instructions that are listed in pairs are exactly identical and can be used inter-changeable.

E. JL or JNGE	- Jump on less or not greater or equal
F. JLE or JNG	- Jump on less or equal or not greater
G. JB or JNAE	- Jump on below or not above or equal
H. JBE or JNA	- Jump on below or equal or not above
I. JP or JPE	- Jump on parity or parity even
J. JO	- Jump on overflow
K. JS	- Jump on sign
L. JNE or JNZ	- Jump on not equal or not zero
M. JNL or JGE	- Jump on not less or greater or equal
N. JNLE or JG	- Jump on not less or equal or greater
O. JNB or JAE	- Jump on not below or above or equal
P. JNBE or JA	- Jump on not below or equal or above
Q. JNP or JPO	- Jump on not parity or parity odd
R. JNO	- Jump on not overflow
S. JNS	- Jump on not sign
T. JCXZ	- Jump on CX zero
U. LOOP	- Subtract one from CX and jump if greater than zero
V. LOOPZ or LOOPE	- Loop while zero or equal
W. LOOPNZ or LOOPNE	- Loop while not zero or equal
X. INT	- Interrupt - Type specified
Y. INTO	- Interrupt on overflow
Z. IRET	- Return from interrupt

Processor Control Instructions

A.	CLC	-	Clear carry
B.	CMC	-	Complement Carry
C.	STC	-	Set Carry
D.	CLD	-	Clear Direction
E.	STD	-	Set Direction
F.	CLI	-	Clear Interrupt
G.	STI	-	Set Interrupt
H.	HLT	-	Halt
I.	WAIT	-	Wait
J.	ESC	-	Escape to external device
K.	LOCK	-	Bus lock prefix

APPENDIX - B

SYSTEM CALLS QUICK REFERENCE CARD

AL	AH	DX	OUTPUT	FUNCTION
X	00	X	X	Kit Reset
X	01	X	AL = Keycode	RDKBD
LN	02	X	Data in DX	GETHEX
0-Addr				
1-Data				
HN				
0-Word				
1-Byte				
LN	03	Data	X	PUTHEX
0-Addr				
1-Data				
HN				
0-Word				
1-Byte				
0-Addr	04	Memory	X	Message
1-Data		Pointer		Display
2-Both				
HN	05	X	X	Addrblk
0-Blank				Datablk
1-Prompt				Addrpmt
LN				Datapmt
0-Addr				
1-Data				
X	06	X	AL = Data from the serial port	SERIN

X	-	Don't Care	Addrblk	-	Address Field Blank
Addr	-	Address	Datablk	-	Data Field Blank
HN	-	Higher Nibble	Addrpmt	-	Address Field Prompt
LN	-	Lower Nibble	Datapmt	-	Data Field Prompt

RDKBD	-	Read a key from the keyboard
GETHEX	-	Get a Hex data from the keyboard
PUTHEX	-	Display a Hex data in the display
SERIN	-	Input a byte from the serial port