


CMR INSTITUTE OF TECHNOLOGY		USN <input type="text"/>							
Internal Assessment Test-II									
Sub:	Microprocessor and Microcontroller						Code:	15CS44	
Date:	17 / 04/ 2018	Duration:	90 mins	Max Marks:	50	Sem:	IV	Branch:	CSE (A,B,C)
Answer Any FIVE FULL Questions									

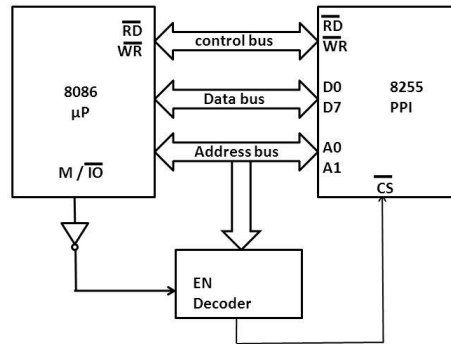
		OBE	
		Marks	
			CO RBT
1	Write note on programmable peripheral interface, 8255. Explain the block diagram, Control word register and IN & OUT instructions. [10]		CO3 L1
2	Interface a general purpose I/O device with 8086 using 8255 PPI. Write an assembly language program to read data from port A and send it to port B, read data from port C lower and send it to port C upper. (Explain how to find out the required control word). Port address is from 0E880H to 0E883H [10]		CO3 L3
3	(a) Explain the prefixes : REP, REPNE and REPE [04]		CO2 L2
	(A3b) Write a program to store byte AAH in 200 memory locations and also test the contents of each location to see if AAH is present. If it fails, the system should display the message "Bad Memory". [06]		CO2 L3
	(b) Explain the syntax and usage of XLAT instruction in 8086 assembly language programming with the help of a suitable example. [06]		CO2 L2
4	Scan the string 'MICROPROCESSOR' for the character 'R'. If the character is found replace it with 'r'. [10]		CO2 L3
5	(a) Explain in detail about 16 Bit memory Interfacing. [06]		CO3 L1
	(b) Design a NAND gate based address decoder to select a 32K*8 memory unit with address range 08000H to 0FFFFH. [04]		CO3 L3
6	(a) Explain the use of CBW and CWD instructions with appropriate examples. State the significance of overflow flag in signed data manipulations. [06]		CO2 L1
	(b) Write an assembly language program to find the largest number among a given ten signed byte information. [04]		CO2 L3
7	1. (A5a) Write note on ISR and IVT. [05]		CO2 L1
	2. (A5b) Explain how 8086 processes an interrupt request. [05]		CO2 L1
	(A7 a) Write an ALP using XLAT to retrieve the value of y in equation $y = x^2 + 2x + 3$. Let x takes values from 0 to 9. Store the result in memory. [6]		CO2 L3
	(A7 b) Explain any two function codes associated with INT 10H. [4]		CO2 L1

1. Write note on programmable peripheral interface: 8255. Explain the block diagram, Control word register and IN & OUT instructions.

To connect any general purpose I/O device to 8086 a programmable interface is needed. One kind of such an interface is 8255 PPI.

Figure 1 depicts the interfacing of 8086 with 8255.

8255 – 8086 Interfacing - 8 Bit Input - Output



Interfacing the 8255 PPI to the 8086 microprocessor

FIGURE 1

Block Diagram 8255 PPI

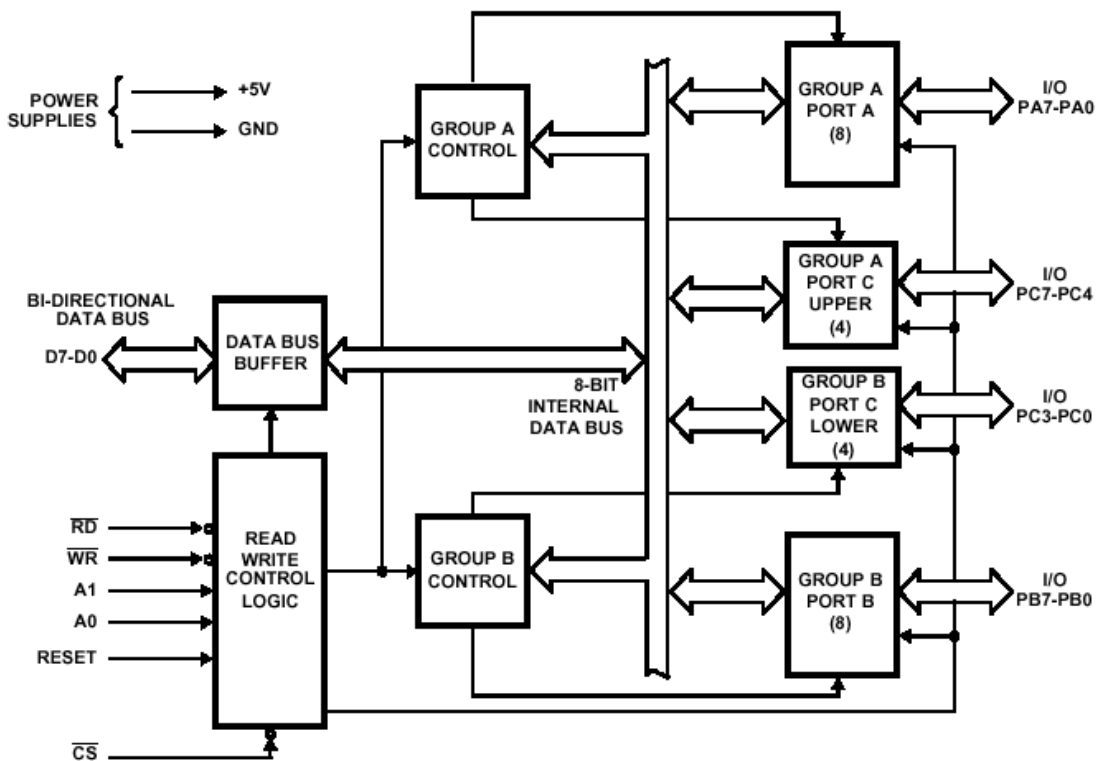


FIG 1.2 BLOCK DIAGRAM

8255 features:

The 8255 is a 40-pin DIP chip. It has three separately accessible ports. The ports are each 8-bit, and are named A, B, and C. The individual ports of the 8255 can be programmed to be input or output, and can be changed dynamically. In addition, 8255 ports, have handshaking capability, thereby allowing interface with devices needs handshaking signals, such as printers.

PA0 – PA7

The 8-bit port A can be programmed as all input, or as all output, or all bits as bidirectional input/output.

PB0 – PB7

The 8-bit port B can be programmed as all input or as all output. Port B cannot be used as a bidirectional port.

PC0-PC7

This 8-bit port C can be all input or all output. It can also be split into two parts, CU (upper bits PC4 – PC7) and CL (lower bits PC0 – PC3). Each can be used for input or output. Each pin can be accessed bitwise too.

RD and WR

These two active-low control signals are inputs to the 8255. The IORD and WR signals from the 8086 are connected to these inputs.

DO – D7 data pin

The data pins of the 8255 are connected to the data pins of the microprocessor allowing it to send data back and forth between the processor and the 8255 chip.

RESET

This is an active-high signal input into the 8255 used to clear the control register. When RESET is activated, all ports are initialized as input ports. In many designs this pin is connected to the RESET output of the system bus or grounded to make it inactive. Like all 1C input pins, it should not be left unconnected.

A0, A1, and \overline{CS}

While CS (chip select) selects the entire chip, it is A0 and A1 that select specific ports. These three pins are used to access ports A, B, C, or the control register as shown in Table. Note that CS is active-low.

\overline{CS}	A1	A0	Selection
0	0	0	Port A
0	0	1	Port B
0	1	0	Port C
0	1	1	Control register
1	x	x	8255 is not selected

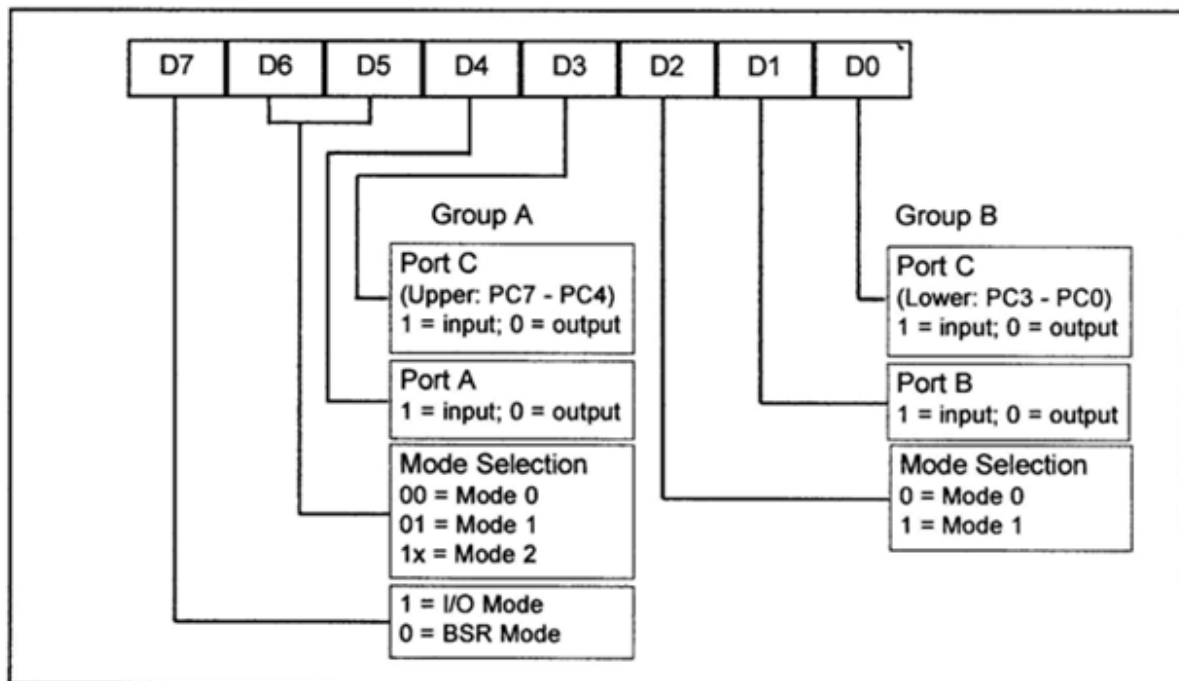
Mode selection of the 8255

While ports A, B, and C are used to input or output data, it is the control register that must be programmed to select the operation mode of the three ports. The ports of the 8255 can be programmed in any of the following modes.

1. Mode 0, simple I/O mode. In this mode, any of the ports A, B, CL, and CU can be programmed as input or output. In this mode, all bits are out or all are in. In other words, there is no such thing as single-bit control as we have seen in PO – P3 of the 8051. Since the vast majority of applications involving the 8255 use this simple I/O mode, we will concentrate on this mode in this chapter.
2. Mode 1. In this mode, ports A and B can be used as input or output ports with handshaking capabilities. Handshaking signals are provided by the bits of port C.
3. Mode 2. In this mode, port A can be used as a bidirectional I/O port with handshaking capabilities whose signals are provided by port C. Port B can be used either in simple I/O mode or handshaking mode 1.
4. BSR (bit set/reset) mode. In this mode, only the individual bits of port C can be programmed.

Control word Register Format :

I/O mode



Control Word Format 8255A

The 8255 chip is programmed in any of the 4 modes mentioned earlier by sending a byte to the control register of the 8255. We must first find the port addresses assigned to each of ports A, B, C, and the control register. This is called *mapping* the I/O port.

Instructions for input and output port transfer

- **IN** – Used to read a byte from the provided port to the accumulator.
- **OUT** – Used to send out a byte from the accumulator to the provided port.

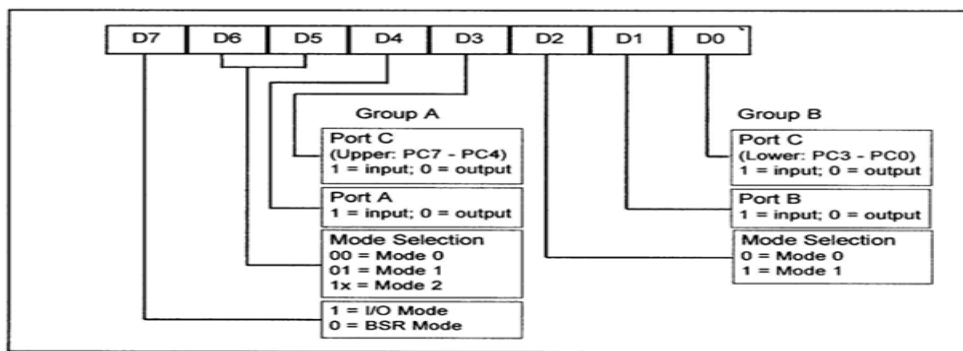
Format:

```
A) MOV DX, PORTADDRESS1
   IN AL, DX ;READ DATA FROM THE PORTADDRESS1
```

```
B) MOV DX, PORTADDRESS2
   MOV AL,10H
   OUT DX,AL ; WRITE DATA FROM AL ONTO PORTADDRESS2
```

2. *Interface a general purpose I/O device with 8086 using 8255 PPI. Write an assembly language program to read data from port A and send it to port B, read data from port C lower and send it to port C upper. (Explain how to find out the required control word). Port address is from 0E880H to 0E883H*

Control word:



Control Word Format 8255A

1	0	0	1	0	0	0	1
---	---	---	---	---	---	---	---

CONTROL WORD: 10010001 = 91H

.MODEL SMALL

.STACK 100

.DATA

PA EQU 0E880H

PB EQU 0E881H

PC EQU 0E882H

CT EQU 0E883H

.CODE

```
MOV AX, @DATA
MOV DS, AX
MOV DX, CT
MOV AL, 91H
OUT DX, AL
MOV DX, PA
IN AL, DX
MOV DX, PB
OUT DX, AL
MOV DX, PC
IN AL, DX
AND AL, 0FH
MOV CL, 04
ROL AL, CL
OUT DX, AL
MOV AH, 4CH
INT 21H
END
```

3. (a) Explain the prefixes : **REP, REPNE and REPE**

The REP prefix, when set before a string instruction, for example - REP MOVSB, causes repetition of the instruction based on a counter placed at the CX register. REP executes the instruction, decreases CX by 1, and checks whether CX is zero. It repeats the instruction processing until CX is zero.

The Direction Flag (DF) determines the direction of the operation.

- Use CLD (Clear Direction Flag, DF = 0) to make the operation left to right.
- Use STD (Set Direction Flag, DF = 1) to make the operation right to left.

The REP prefix also has the following variations:

- REP: It is the unconditional repeat. It repeats the operation until CX is zero.
- REPE or REPZ: It is conditional repeat. It repeats the operation while the zero flag indicates equal/zero. It stops when the ZF indicates not equal/zero or when CX is zero.

For example if the programmer wants to check whether 2 strings are equal:

REPE CMPSB ; This will continue checking both strings byte by byte till Zero flag is not set or CX is zero whichever happens first.

- REPNE or REPNZ: It is also conditional repeat. It repeats the operation while the zero flag indicates not equal/zero. It stops when the ZF indicates equal/zero or when CX is decremented to zero.

REPNE SCASB ; This will continue searching the string, for the character present in AL, byte by byte till Zero flag is set or CX is zero whichever happens first.

(b) Explain the syntax and usage of XLAT instruction in 8086 assembly language programming with the help of a suitable example.

XLAT Instruction:

The 80x86 instruction set includes an instruction that allows us to perform table lookup. The instruction is XLAT, and it is a no operand instruction.

The XLAT instruction to work the entries of the lookup table must be in sequential order and have one to one relation with the index. The table is specified as DS: BX (BX being the memory offset to the beginning of the table). AL is the index into the table. XLAT will simply return the value at index to AL register. A lookup table fetching will save a lot of machine cycle time if it is used for calculating frequently required arithmetic data.

The programmer must first load the offset address of the table into BX register and then load AL with the position of the byte invoke the XLAT instruction. The byte stored at the specified position in the table will be in AL register after the instruction executes. In other words, the instruction does the equivalent of the following:

$$AL \leftarrow [AL+BX]$$

EXAMPLE:

A TABLE OF ASCII CODES FOR HEXADECIMAL DIGITS :

```
.MODEL SMALL
```

```
.DATA
```

```
TABLE DB '0123456789ABCDEF' ;This generates the table
```

```
DIGIT DB 7 ; Assume that we need to fetch ASCII code for 7
```

```
ASC_DIGIT DB ?
```

```
.CODE
```

```
MOV DX,@DATA
```

```
MOV DS,DX
```

```
LEA BX, TABLE ; Offset of table
```

```
MOV AL, DIGIT ; Position of table entry
```

```

XLAT          ; Now AL contains ASCII code
MOV  ASC_DIGIT, AL  ;Store it

```

```

MOV AH,4CH
INT 21H

```

```

END

```

(A3b)Write a program to store byte AAH in 200 memory locations and also test the contents of each location to see if AAH is present. If it fails, the system should display the message “Bad Memory”.

```

.MODEL SMALL

```

```

.STACK 64H

```

```

.DATA

```

```

    M1 DB 'BAD MEMORY$'

```

```

    DEST DB 200 DUP(00)

```

```

.CODE

```

```

    MOV AX,@DATA

```

```

; INITIALISE DATA SEGMENT AND EXTRA SEGMENT REG

```

```

    MOV DS, AX

```

```

    MOV ES, AX

```

```

    LEA DI, DEST  ; DI POINTING TO THE STRING

```

```

    MOV AL, 0AAH  ; DATA TO BE STORED FOR IN AL REG

```

```

    CLD          ; CLEAR DIRECTION FLAG, DI INCREMENTS AFTER
                ; EXECUTION OF EACH STRING INSTRUCTION

```

```

    MOV CX, 200  ; INITAILISE COUNTER

```

```

    REP STOSB   ;STORE THE DATA TO 200 LOCATIONS

```

```

    MOV CX,200

```



```

        LEA SI, DEST
CHECK:  LODSB
        CMP AL,0AAH
        JNZ FAIL
        LOOP CHECK
        JMP EXIT
FAIL:   LEA DX, M1
        MOV AH, 09H
        INT 21H

EXIT:   MOV AH, 4CH
        IN 21H
        END

```

4. *Write a program that scan the string “MICROPROCESSOR” and replaces all the uppercase R with lowercase r.*

```
EXTRA1 SEGMENT
```

```

        M1 DB 'MICROPROCESSOR$'
        LEN EQU 14

```

```
EXTRA1 ENDS
```

```
CODE1 SEGMENT
```

```
ASSUME ES: EXTRA1, CS: CODE1
```

```
; INITIALISE EXTRA SEGMENT REG
```

```

        MOV AX,EXTRA1
        MOV ES, AX

```

```
        LEA DI, M1      ; DI POINTING TO THE STRING
```

```
        MOV AL, 'R'     ; CHARACTER TO BE SEARCHED FOR IN AL REG
```

```
        CLD             ; CLEAR DIRECTION FLAG, DI INCREMENTS
```

```
AFTER                                     ; EXECUTION OF EACH STRING INSTRUCTION
```

```

MOV CX, LEN ; INITIALISE COUNTER
MOV DL, 'r'
; SCAN THE STRING FOR R, IF FOUND REPLACE WITH r
CHECK: REPNE SCASB
      JE CHANGE
      JMP DIS
CHANGE: MOV [DI-1], DL
      CMP CX, 0
      JNE CHECK

MOV AH, 4CH
IN 21H
CODE1 ENDS
END

```

5. (a) Explain in detail about 16 Bit memory Interfacing.

The microprocessor must be able to read and write data to any 16-bit location in addition to any 8-bit location. To facilitate the same the data bus and memory are divided into banks. The processor uses two control signals to access even and odd memory banks: BHE (Bus high enable) and the lower significant address bit A_0

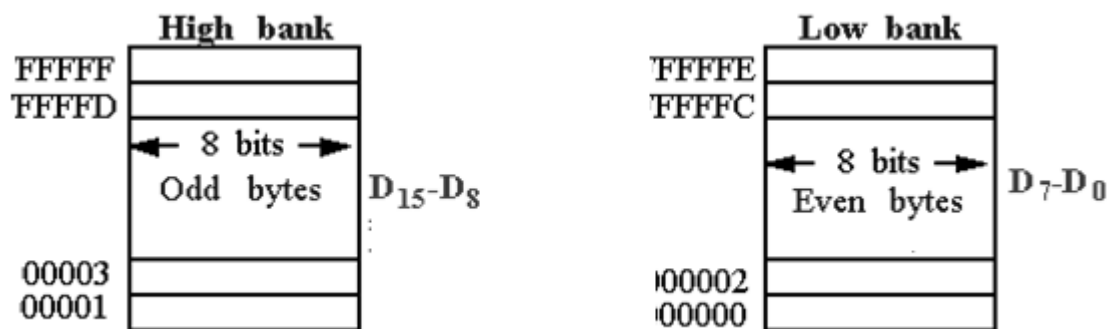


Fig 1

BHE and A₀ are used to select one or both:

BHE	A₀	Function
0	0	Both banks enabled for 16-bit transfer
0	1	High bank enabled for an 8-bit transfer
1	0	Low bank enabled for an 8-bit transfer
1	1	No banks selected

Note: A₀ does not connect to memory and bus wire A₁ connects to memory pin A₀, A₂ to A₁, etc.

Fig 7 depicts the 16 bit memory interfacing block diagram. The address decoding circuitry is not shown in the diagram. It can be a 3:8 line decoder, CPLD or FPGA based.

The system makes use of 2 bidirectional data buffers, one for even and other for odd banks.

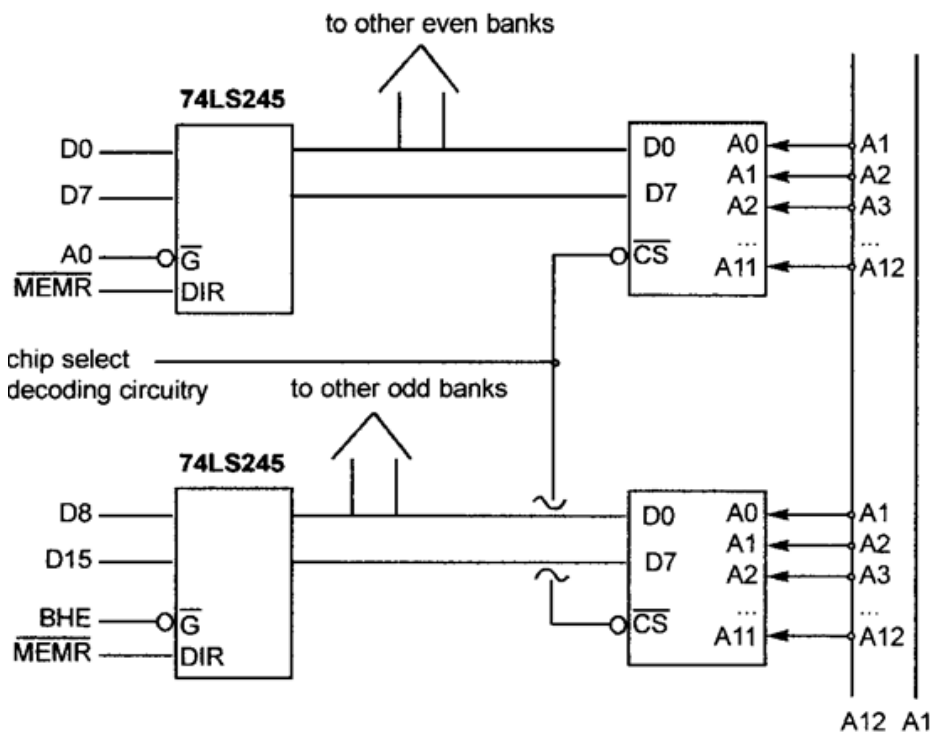


Fig 7

Bidirectional data buffer (74LS245):

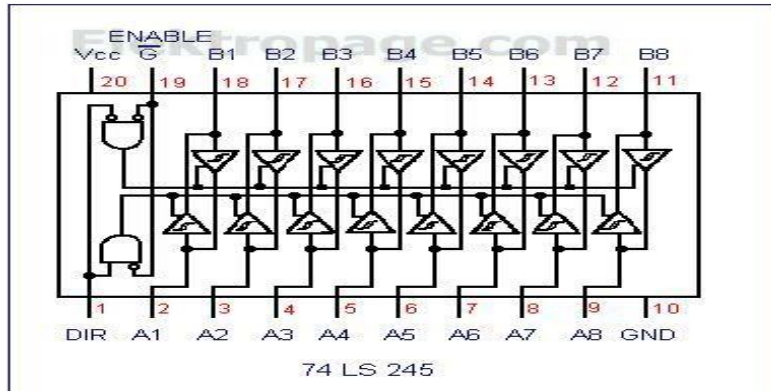


Fig 3

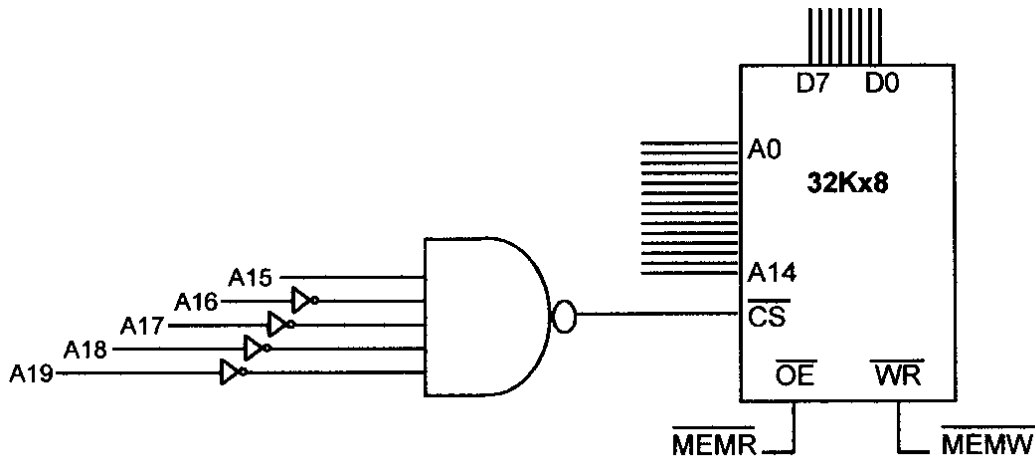
It is an 8bit data buffer. The direction of data transfer is controlled by the memory read signal from the processor. If the request is for memory read, then the processor places low on this line thus enabling data movement from memory to processor. If the request is for memory write, then the processor places high on this line thus enabling data movement from processor to memory.

(b) Design a NAND gate based address decoder to select a 32K*8 memory unit with address range 08000H to 0FFFFH.

Address	A ₁₉ - A ₁₆				A ₁₅ - A ₁₂				A ₁₁ - A ₈				A ₇ - A ₄				A ₃ - A ₀			
Start:08000H	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0AFFFH	0	0	0	0	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1
END:0FFFFH	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

The address line A19-A15 can be used to generate the chip select signal. As NAND gate and NOT gates are employed the implementation diagram will be :

A ₁₉ - A ₁₆	A ₁₅	Out put	Function
0	0	0	$f = \overline{A_{19} A_{18} A_{17} A_{16} A_{15}}$



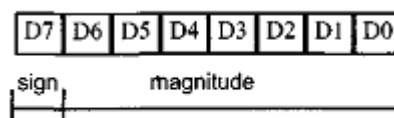
6. (a) Explain the use of CBW and CWD instructions with appropriate examples. State the significance of overflow flag in signed data manipulations.

Signed Number Representation:

The most significant bit (MSB) is set aside for the sign (+ or -), while the rest of the bits are used for the magnitude. The sign is represented by 0 for positive (+) numbers and 1 for negative (-) numbers. Signed byte representation is discussed below.

Signed 8-bit operands:

In signed byte operands, D7 (MSB) is the sign and D0 to D6 are set aside for the magnitude of the number. If D7 = 0, the operand is positive and if D7 = 1, it is negative. The format is depicted



Overflow problem in signed number operations:

When using signed numbers, a serious problem arises that must be dealt with. If the result of an operation on signed numbers is too large for the register, an overflow has occurred and the programmer must be notified. This is the overflow problem. The 8086 indicates the existence of an error by raising the overflow flag, but it is up to the programmer to take care of the erroneous result.

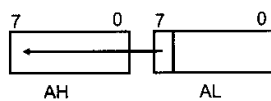
In 8-bit signed number operations, OF is set to 1 if either of the following two conditions occurs:

1. There is a carry from D6 to D7 but no carry out of D7 (CF = 0).
2. There is a carry from D7 out (CF = 1) but no carry from D6 to D7.

In other words, the overflow flag is set to 1 if there is a carry from D6 to D7 or from D7 out, but not both. This means that if there is a carry both from D6 to D7 and from D7 out, OF = 0.

To overcome the overflow related issues the programmer can opt for sign extension of operands.

- The CBW (convert byte to word) instruction extends the sign bit of AL into the AH register. This preserves the number's sign:



.DATA

byte_val db -101

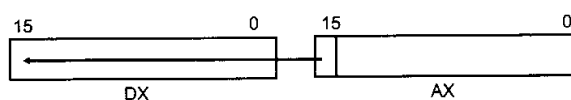
.CODE

mov al, byte_val ; AX = 9Bh

cbw ; AX = FF9Bh

Both **9Bh** and **FF9Bh** both equal decimal **-101**, the only difference is the storage size.

- The CWD (convert word to double-word) instruction extends the sign bit of AX into the DX register:



.DATA

word_val dw -101 ; FF9Bh

.CODE

mov ax, word_val ; AX = FF9Bh

cwd ; DX:AX = FFFFh:FF9Bh

Example: Signed overflow condition.

```

DATA1    DB +96
DATA2    DB +70
          ....
          MOV    AL,DATA1    ;AL=0110 0000 (AL=60H)
          MOV    BL,DATA2    ;BL=0100 0110 (BL=46H)
          ADD    AL,BL        ;AL=1010 0110 (AL=A6H= -90 invalid!)

+ 96      0110 0000
+ 70      0100 0110
+166      1010 0110 According to the CPU, this is -90, which is wrong. (OF = 1, SF = 1, CF = 0)

```

If in the previous example CBW instruction was used to sign extend both operands,

S	AH	AL	
0	000 0000	0110 0000	+96 after sign extension
0	000 0000	0100 0110	+70 after sign extension
0	000 0000	1010 0110	+166

The OF=0 and the answer is correct.

(b) Write an assembly language program to find the largest number among a given ten signed byte information.

```

.MODEL SMALL
.STACK 64H
.DATA
    TEMP DB -12, 13, 4, 5, 6, -18, 19, 12, -5, -2
    LEN DB LEN-TEMP
    RES DB 00

.CODE

MOV AX, @DATA
MOV DS, AX
LEA SI, TEMP

MOV BL, LEN
MOV BH, 00

DEC BX; N-1 COMPARISONS

MOV AL, [SI]; TAKE THE FIRST DATA, PRESENT LARGEST
L2: INC SI

```

```
CMP AL, [SI]; COMPARE THE PRESENT LARGEST WITH NEXT DATA
JGE L1
MOV AL, [SI] ; UPDATE LARGEST
L1: DEC BX
JNZ L2
MOV RES, AL
MOV AH, 4CH
INT 21H
END
```

7. (a) (A5a) Write note on ISR and IVT.

Interrupt is the method of creating a temporary halt during program execution and allows peripheral devices to access the microprocessor. The microprocessor responds to that interrupt with an **ISR** (Interrupt Service Routine), which is a short program to instruct the microprocessor on how to handle the interrupt.

Interrupt Service Routine (ISR):

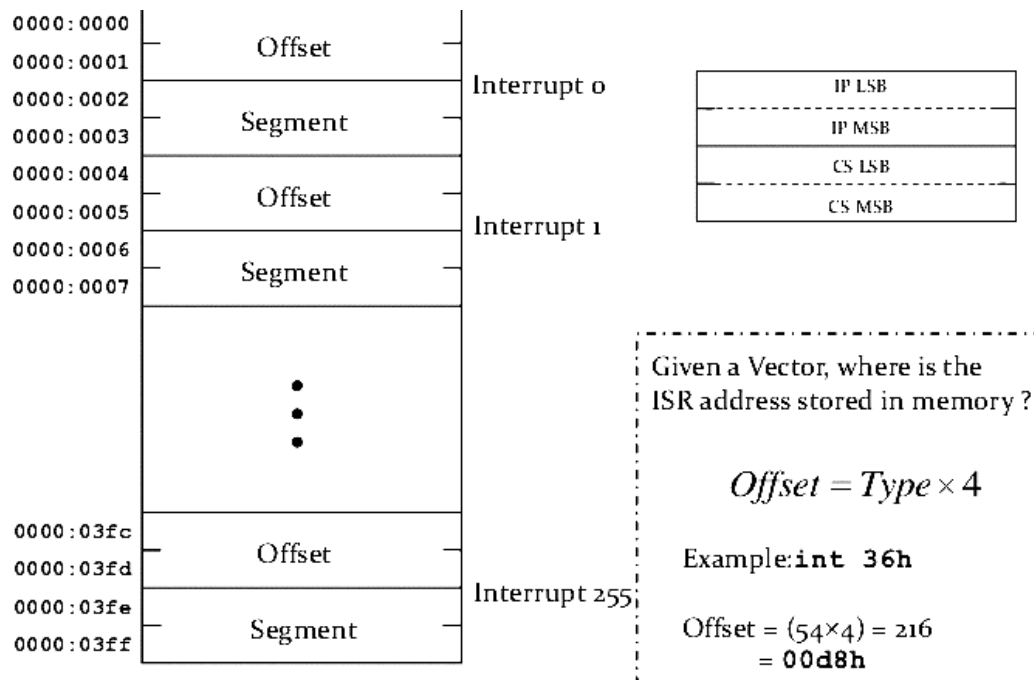
For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler. When an interrupt is invoked, the microprocessor runs the interrupt service routine. For every interrupt, there is a fixed location in memory that holds the address of its ISR. The group of memory locations set aside to hold the addresses of ISRs is called the interrupt vector table.

When an interrupt is occurred, the microprocessor stops execution of current instruction. It transfers the content of flag register, CS and IP onto stack. After this, it jumps to the memory location specified by Interrupt Vector Table (IVT). Then ISR will be executed. The last instruction on ISR is IRET. When IRET is executed the IP, CS and FR are replenished from stack. The execution control is given back to the program.

Interrupt Vector Table(IVT):

The first 1Kbyte of memory of 8086 (00000 to 003FF) is set aside as a table for storing the starting addresses of Interrupt Service Routines(ISR). Since 4-bytes are required for storing starting addresses of ISRs (CS and IP), the table can hold 256 Interrupt procedures.

The starting address of an ISR is often called the Interrupt Vector or Interrupt Pointer. Therefore the table is referred as Interrupt Vector Table. In this table, IP value is put at lower word of the vector & CS is put at higher vector.



(b) (A5b) Explain how 8086 processes an interrupt request.

Processing Interrupts in 8086:

If an interrupt has been requested, the 8086 responds to the interrupt by stepping through the following series of major actions:

- Decrements the stack pointer by 2 and pushes the flag register on the stack.
- Disables the 8086 INTR interrupt input by clearing the interrupt flag in the flag register.
- Resets the trap flag in the flag register.
- Decrements the stack pointer by 2 and pushes the current code segment register contents on the stack.
- Decrements the stack pointer again by 2 and pushes the current instruction pointer contents on the stack.
- The interrupt type number is multiplied by 4 to get the physical location in IVT to fetch the CS and IP of corresponding ISR
- Processor executes ISR
- The last instruction is IRET, on execution of this the processor gets back IP, CS and FR from stack and continues with execution of the program.

(A7 a) Write an ALP using XLAT to retrieve the value of y in equation $y = x^2 + 2x + 3$. Let x takes values from 0 to 9. Store the result in memory.

```

.MODEL SMALL
.DATA
TABLE DB 3,6,11,18,27,38,51,66,83,102 ;This generates the table
X DB 7 ; Assume that we need to fetch function value for 7
VALUE DB ?

.CODE
MOV DX,@DATA
MOV DS, DX

LEA BX, TABLE ; Offset of table
MOV AL, X ; Position of table entry
XLAT ; Now AL contains VALUE
MOV VALUE, AL ;Store it

MOV AH,4CH
INT 21H

END

```

(A7 b) Explain any two function codes associated with INT 10H.

BIOS INTERRUPT (INT 10H):

One way to display text on the screen quickly is to use the BIOS interrupt 10h functions. Four function codes associated with INT10h is given below.

INT 10h / AH = 0 - set video mode.

input:

AL = desired video mode.

these video modes are supported:

00h - text mode. 40x25. 16 colors. 8 pages.

03h - text mode. 80x25. 16 colors. 8 pages.

13h - graphical mode. 40x25. 256 colors. 320x200 pixels. 1 page.

INT 10h / AH = 2 - set cursor position.

input:

DH = row.

DL = column.

BH = page number (0..7).

INT 10h / AH = 06h - scroll up window.

INT 10h / AH = 07h - scroll down window.

input:

AL = number of lines by which to scroll (00h = clear entire window).

BH = attribute used to write blank lines at bottom of window.

CH, CL = row, column of window's upper left corner.

DH, DL = row, column of window's lower right corner.