MICROPROCESSORS AND MICROCONTROLLER

ASSIGNEMENT-2 Solution

LONG QUESTIONS

1. Explain about the following instructions of 8086 microprocessor. (Syntax, Function and Example)
Move, Compare, Addition, Negate, Move string, Multiplication, Division, Jump if No carry, Jump If Zero.

ANS:

Move: Data transfer group Instruction

Syntax: MOV Dst, Src
Functions: Moves (Copy) data from Destination (Dst) to Source (Src)
Example: MOV SI, DI
MOV CX, [1200H]
MOV CL, 02H
MOV AL, [SI]

Compare: Arithmetic group Instruction because compare using Subtraction

Syntax: CMP Dst, Src
Functions: Compare the content of Source with destination using subtraction, but not stored the subtraction result only modified the status flags to reflect the comparision result.

compare	ZF	CF
Dst > Src	0	0
Dst < Src	0	1
Dst = Src	1	0

Example: CMP SI, DI
CMP CX, 005FH
CMP AL, [SI]

Addition: Arithmetic group Instruction

Syntax: ADD Dst, Src
Functions: Add the content of Source with destination and store the result in destination.
Example: ADD CL, BL
ADD [3000H], BX

Negate: Logical group Instruction

Syntax: NEG dst
Functions: Find the 2's compliment of destination and store the result in destination.
Example: NEG SI

Move string: String Instruction

Syntax: MOVSB/MOVSW
Functions: Copy data byte or word from source string to destination string and then updated the index registers. Source string is in data segment pointed by offset SI and Destination string is in Extra segment pointed by DI.
Example: MOVSB
MOVSW

Multiplication: Arithmetic group Instruction

Syntax: MUL Src/ IMUL Src

Functions: Perform Unsigned or Signed multiplication of source data with AL (Byte multiplication) or AX (Word multiplication) and store the result in AX (Byte multiplication) or DXAX (Word Multiplication).
Example: MUL BH (AL X BH = AX)
MUL SI (AX X SI = DXAX)

Division: Arithmetic group Instruction

Syntax: DIV Src/ IDIV Src
Functions: Perform Unsigned or Signed division of AX (Byte division) or DXAX (Word division) by source and store Quotient in AL and Remainder in AH (Byte division) or Quotient in AX and Remainder in DXAX(Word division).
Example: DIV SI

Jump if No carry: Branch Instruction

Syntax: JNC *label*
Functions: Jump to label, if CF=0 else execute the next instruction.
Example: JNC 3400H

Jump If Zero. Branch Instruction

Syntax: JZ *label*
Functions: Jump to label, if ZF=1 else execute the next instruction.
Example: JZ 1000H

2. What is minimum and maximum mode of 8086 processor?

ANS:

Difference between MAX and MIN mode

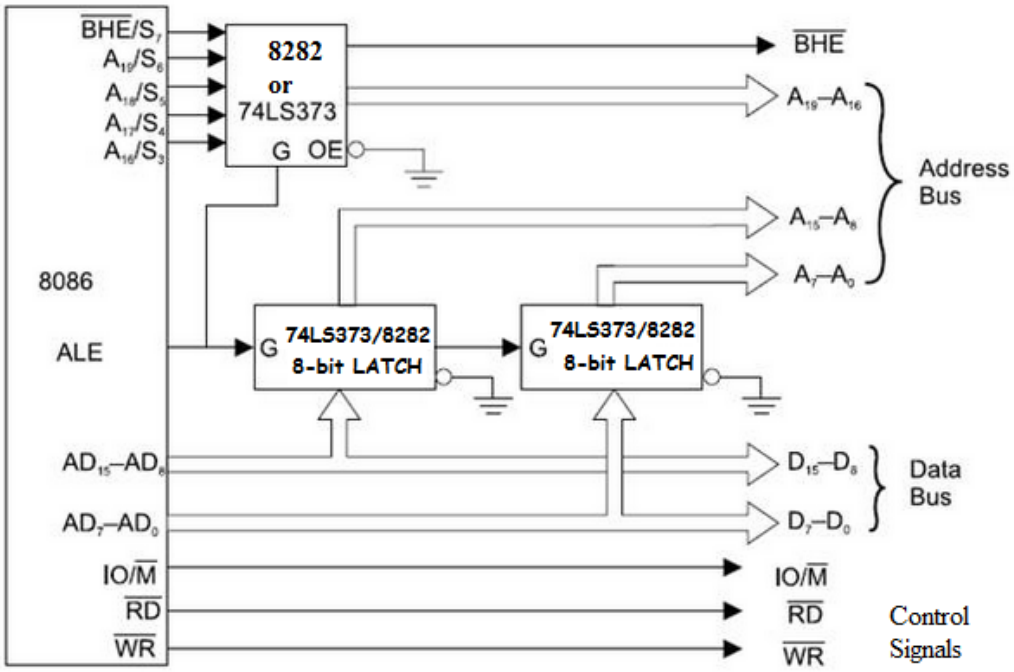
Maximum mode	Minimum Mode
When MN/MX(bar) low 8086 is in maximum mode.	When MN/MX(bar) high 8086 is in minimum mode.
In maximum mode 8086 generates QS1,QS0,S0 (bar),S1(bar),S2(bar), LOCK(bar),RQ(bar)/GT1,RQ (bar)/GT0 control signals.	In minimum mode 8086 generates INTA(bar), ALE, DEN(bar), DT/R(bar), M/IO(bar), HLDA,HOLD and WR (bar) control signals.
So clearly there are multiple processors in the system.	There is only one processor in the system minimum mode.
Whereas in maximum mode interfacing, master/slave and multiplexing and several such control signals are required	In minimum mode no interfacing or master/slave signals is required.
In maximum mode a bus controller is required to produce control signals. This bus controller produces $\overline{\text{MEMRDC}}$, $\overline{\text{MEMWRC}}$, $\overline{\text{IORDC}}$, $\overline{\text{IOWRC}}$, ALE, DEN, DT/$\overline{\text{R}}$ control signals.	In minimum mode direct $\overline{\text{RD}}$ $\overline{\text{WR}}$ signals can be used. No bus controller required. A simple demultiplexer would do the job. of producing the control signals. This demultiplexer produces $\overline{\text{MEMRD}}$, $\overline{\text{MEMWR}}$, $\overline{\text{IORD}}$, $\overline{\text{IOWR}}$ control signals.

3. What are the different multiplex signals of 8086 processor and how to de-multiplex it?

ANS:

Multiplexing is the process of combining 2 or more signals together into one pin. De-multiplexing is the process of separating the multiplexed signal into its individual signals.
In case of 8086 Multiples signals are:
AD0 to AD15, A16/S3 to A19/S6 and BHE'/S7

Demultiplexing of system bus in 8086

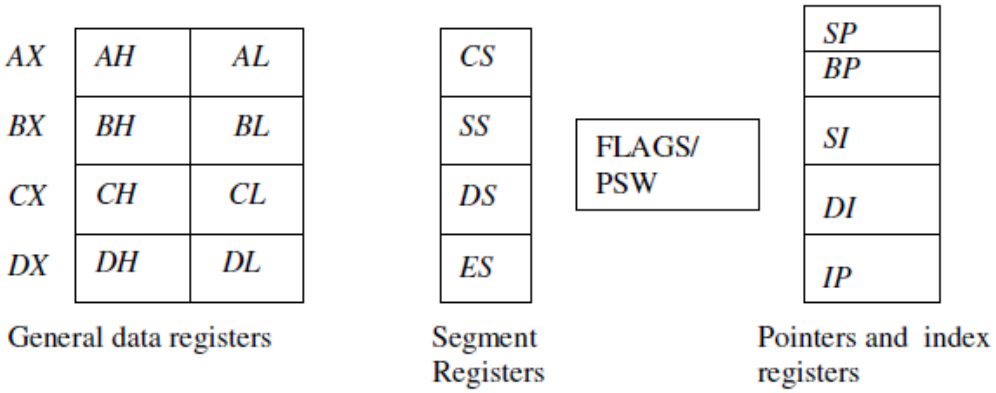


Demultiplexing of 20-bit address bus in 8086 processor

ALE=1, during T1 state of the every bus cycle to enable the Latches and external hardware (three 74LS373 or 8282) latch the 20-bit address generated on multiples bus.
ALE=0, during the next clock pulses and hence multiplexed signals are either float or change to Data and Status signals.

4. Draw the register organization of the 8086 microprocessor and explain typical functions of each register.

ANS:



General Data Registers:

The registers *AX*, *BX*,*CX* and *DX* are the general purpose 16-bit registers.

AX is used as 16-bit accumulator. The lower 8-bit is designated as *AL* and higher 8-bit is designated as *AH*. *AL* can be used as an 8-bit accumulator for 8-bit operation. All data register can be used as either 16 bit or 8 bit.

BX is a 16 bit register, but *BL* indicates the lower 8-bit of *BX* and *BH* indicates the higher 8-bit of *BX*. The register *BX* is used as offset storage for forming physical address in case of certain addressing modes.

CX is used default counter in case of Rotate, Shift, String and loop instructions.

DX register is a general purpose register which may be used as an implicit operand or destination in case of a few instructions.

Segment Registers:

The 8086 architecture uses the concept of segmented memory. 8086 able to address to address a memory capacity of 1 megabyte and it is byte organized. This 1 megabyte memory is divided into 16 logical segments. Each segment contains 64 k bytes of memory.

There are four segment register in 8086

• Code segment register (CS)

• Data segment register (DS)

• Extra segment register (ES)

• Stack segment register (SS)

Code segment register (CS): is used to create the code segment memory, where the executable program is stored.

Data segment register (DS): Used to create the data segment memory, where the executable data is stored.

Extra Segment Register (ES): Used to create Extra segment memory, where destination string data is stored.

Stack Segment Register (SS): Used to create Stack Segment memory, which is used to store stack data. While addressing any location in the memory bank, the physical address is calculated from two parts:

Pointers and Index Registers

The pointers contain offset within the particular segments.

The **pointer register *IP (Instruction Pointer)*** contains offset within the code segment. It always points the address of the next instruction to be executed.

The **pointer register *BP*** contains offset within the stack segment. It is used as an Offset to access data from the stack memory.

The **pointer register *SP*** contains offset within the stack segment. It always point the top of the stack memory.

Index registers

The register *SI (Source Index)* is used to store the offset of source data in data segment.

The register *DI* is used to store the offset of destination in data or extra segment. The index registers are particularly useful for string manipulation.

Flag Register

The 8086 flag register contents indicate the results of computation in the *ALU*. It also contains some flag bits to control the *CPU* operations.

5. Write an ALP for 8086 for arranging a set of data in ascending order. Which instruction/ instructions of your program will you change if you wish to arrange the data in descending order?

ANS:

LABEL	MNEMONICS	COMMENTS
	MOV DI, 1000H	LOAD THE OFFSET ADDRESS OF THE DESTINATION IN DI REGISTER
	MOV CH, 05H	LOAD THE COUNT VALUE (destination count) IN CH REGISTER
L3	MOV SI,DI	COPY THE DESTINATION OFFSET ADDRESS IN SI REGISTER (ADDRESS OF SAMPLE INPUT)
	MOV CL,CH	LOAD THE COUNT VALUE (source count) IN CL REGISTER
L2	MOV AL, [SI]	LOAD THE INPUT DATA BYTE IN AL REGISTER FROM INPUT MEMORY
	CMP [DI], AL	COMPARE THE DATA BYTES
	JC **L1**	IF CF=1, JUMP TO L1
	XCHG [DI], AL	EXCHANGE THE DATA BYTES
	MOV [SI], AL	COPY THE CONTENT OF AL IN SOURCE MEMORY
L1	INC SI	INCREMENT THE SI BY ONE
	DEC CL	DECREMENT THE CONTENT OF CL BY ONE

	JNZ **L2**	IF ZF=0, JUMP TO L2
	INC DI	INCREMENT THE CONTENT OF DI BY ONE
	DEC CH	DECREMENT THE CONTENT OF CH BY ONE
	JNZ **L3**	IF ZF=0, JUMP TO L3
	HLT	STOP THE PROGRAM

To arrange the data in Descending order we use **JNC L1** instead of **JC L1**.

6. **Write a program to subtract two 16 bit numbers BB10 H and 800C H and store the result in the offset address 3400h and 3401h using 8086 processor.**

ANS:

MNEMONICS	COMMENTS
MOV DI, BB10H	LOAD BB10H (MINUED) IN DI REGISTER
MOV AX, 800CH	LOAD BB10H (SUBTRAHEND) IN AX REGISTER
SUB DI, AX	SUBTRACT AX FROM DI
MOV [3400H], DI	STORE THE RESULT IN THE OFFSET 3400H AND 3401H
HLT	STOP THE PROGRAM

7. **Multiply 12H and (-13H). Store the result in 30000H [3000:0000] and 30001H locations.**

ANS:

MNEMONICS	COMMENTS
MOV AX, 3000H	LOAD SEGMENT BASE FOR DATA SEGMENT
MOV DS, AX	
MOV AL, 12H	LOAD MULTIPLICAND IN AL
MOV BL, 13H	LOAD MULTIPLIER IN BL
NEG BL	CHANGE MULTIPLIER TO -13H
IMUL BL	MULTIPLY BL WITH AL
MOV [0000H], AX	**STORE 16-BIT RESULT IN THE PHYSICAL ADDRESS 30000H (DS:0000H) AND 30001(DS:0001H)H**
HLT	STOP THE PROGRAM

8. **Write an Assembly language program in 8086 to search the largest number from a data array of Five bytes. Data array is available from the physical address 25000H.**

ANS:

Physical address of Data Array is = 25000H = 2000H: 5000H
Hence, offset address of data array is = 5000H

LABEL	MNEMONICS	COMMENTS

	MOV AX, 2000H	LOAD SEGMENT BASE FOR DATA SEGMENT
	MOV DS, AX	
	MOV DI, 5000H	LOAD THE OFFSET ADDRESS OF THE INPUT DATA IN DI REGISTER
	MOV CH, 05H	LOAD THE COUNT VALUE (NO. OF BYTES TO BE SORTED) IN CH REGISTER
	MOV AL, [DI]	LOAD THE INPUT DATA BYTE IN AL REGISTER FROM INPUT MEMORY [DS:DI]
L2	CMP AL,[DI]	COMPARE THE DATA BYTES
	JNC **L1**	IF CF=0, JUMP TO L1
	MOV AL, [DI]	COPY THE LARGEST NUMBER IN AL
L1	INC DI	INCREMENT THE DI BY ONE
	DEC CH	DECREMENT THE CONTENT OF CH BY ONE
	JNZ **L2**	IF ZF=0, JUMP TO L2
	MOV [2000H], AL	STORE THE LARGEST NUMBER OF ARRAY IN 2000H OFFSET ADDRESS
	HLT	STOP THE PROGRAM