Scheme of Evaluation
Internal Assessment Test 1 – March 2019

Sub:	Microprocessor & Microcontroller							**Code:**	17CS44
Date**:**	06/03/2019	Duration:	90mins	Max Marks:	50	**Sem:**	IV	**Branch:**	CSE

Note: Answer Any Five Questions

Question #		**Description**	**Marks Distribution**		**Max Marks**
1	a)	**Draw and explain the internal architecture of the 8086 along with functions of each blocks and registers.**		10 M	10 M
		• Functional Block diagram	3 M		
		• Explanation on BIU,EU	3 M		
		• General purpose and Special Purpose registers	2 M		
		• Flag register	2 M		
2	a)	**Differentiate between physical address and logical address**		4 M	10 M
		• Explanation	2M		
		• Examples	2M		
	b)	**If DS=E210H, CS=A920H, SS=0B01H, BP=1234H, SP=0130H, SI=4321H, DI=1200H, BX=1000Hthen determine the physical address accessed in the following instructions.** **MOV [BP+DI+0A], AH** **MOV AL, [5036H]** **MOV AX, [BX+SI]** **PUSH AX**	4*1.5M	6 M	

3		**Explain addressing modes of 8086**		10 M	10 M
		• Register and Immediate addressing modes with examples	3M		
		• Memory addressing modes with examples	7M		
4		**Develop an assembly language program that Copies 10 word information stored in consecutive memory locations to another reserved block of consecutive word memory location. Use Indexed addressing mode. Write appropriate comments.**		10M	10 M
		• Program • Usage of indexed addressing mode	5M 2M		
		• Comments	3M		
5	a)	**With proper syntax and examples explain ADD,ADC, SUB and SBB instructions** • Syntax plus examples each instruction	1.25*4	5 M	10 M
	b)	**Explain the flag register with necessary diagrams. Choose an appropriate example to show how flag bits are updated.** • Flag register details • Example		2 M 3 M	
6	a)	**What are control transfer instructions** • Jump and Call instruction execution explanation		03 M	10 M
	b)	**Demonstrate all conditional and unconditional jump Instructions available in 8086 assembly level programming with syntax and application.** • Conditional jump instruction with syntax (any 4) • Unconditional jump with proper syntax		5 M 2 M	

7	a)	**Write an assembly language program using 8086 instructions to add 8 byte information stored in consecutive memory locations. Consider the data in such a way that the resulting sum is a word data. Find the average value. Use full segment Definition.** • Program with one sample output data • Full segment definition	 7M 3M	10M	10 M

2 (a) ***Differentiate between physical address and logical address***

Physical Address: Physical address is a 20bit address that is actually put on the address lines of 8086 and decoded by the memory interfacing circuit. This has arrange of 00000H-FFFFFH (1MB). It refers to the actual physical location in the memory.

Logical address: To access a location in 1MB memory, the entire memory is viewed as a group of segments each ranges up to 64KB size. Thus each location within a segment is specified as a combination of two 16bit values called, segment value and offset address. It is represented as
Seg reg: Offset reg
Offset address is a location in 64KB space.
The default segments and offset registers available in 8086

Segment register:	CS	DS	ES	SS
Offset register(s):	IP	SI, DI, BX	SI, DI, BX	SP, BP

(b) ***If DS=E210H, CS=A920H, SS=0B01H, BP=1234H, SP=0130H, SI=4321H, DI=1200H, BX=1000Hthen determine the physical address accessed in the following instructions.***

```
MOV [BP+DI+0A], AH
 MOV AL, [5036H]
MOV  AX, [BX+SI]
PUSH AX :
```

Given DS=E210H, CS=A920H, SS=0B01H, BP=1234H, SP=0130H, SI=4321H, DI=1200H, BX=1000H

1) MOV [BP+DI+0A], AH

As BP is indicated the referred segment is STACK
Logical address - SS: BP+DI+0A
Physical address – (SS*10H) + BP+DI+0A

Physical address – (0B01H*10H) + 1234H+1200H+0Ah = 0D44E H

2) MOV AL, [5036H]
The referred segment is DATA
Logical address - DS: 5036H
Physical address – (DS*10H) + 5036H

Physical address – (E210H*10H) + 5036H = E7136 H

3) MOV AX, [BX+SI]

The referred segment is DATA
Logical address - DS: BX+SI , DS:BX+SI+1
Physical address – (DS*10H) + BX+SI and (DS*10H) + BX+SI+1

Physical address – (E210H*10H) + 1000H+ 4321 = E3230H and E3231 H

4) PUSH AX
The referred segment is stack

Top of the stack: SS: SP
The available location: SS: SP-2
Physical address: – (SS*10H) + SP-2 and (SS*10H) + SP-1
B13H and B13DH

4) Develop an assembly language program that Copies 10 word information stored in consecutive memory locations to another reserved block of consecutive word memory location. Use Indexed addressing mode. Write appropriate comments.

```
.MODEL SMALL
.DATA
SRC DW 1234H,0ABCH,1200H,3456H,1000H,300H,100H,200H,5400H,3200H
DEST DW 10 DUP(0)

.CODE
; INITIALISTION OF DATA SEGMENT REGISTER
START:MOV AX,@DATA
MOV DS,AX

MOV CX,10 ; INITIALISE COUNTER
LEA SI, SRC ; LOAD EFFECTIVE ADDRESS OF SRC TO SI, SI A POINTER TO SRC
LEA DI,DEST; DI POINTER TO DEST
; COPY WORD DATA FROM SOURCE TO DESTINATION
BACK: MOV AX,[SI]
MOV [DI], AX
ADD SI,2 ; MANIPULATE SRC POINTER
ADD DI,2; MANIPULATE SRC POINTER
LOOP BACK ; REPEAT TILL CX BECOMES ZERO
; TERMINATIO CODE
MOV AH,4CH
INT 21H
END START
```

5 (a) With proper syntax and examples explain ADD,ADC, SUB and SBB instructions

ADD/ADC Instruction:
Syntax: ***ADD destination, source ; destination= destination +source***
ADC destination, source; destination= destination +source+Carry

These instructions add a number from source to a number from destination and put the result in the destination. The ADC, instruction also adds the status of carry flag into the result.

- The source may be an immediate number, a register, or a memory location.
- The source and the destination in an instruction cannot both be memory locations.
- The source and destination both must be of same size. If you want to add a byte to a word, you must copy the byte to a word location and fill the upper byte of the word with zeroes before adding.
- Flags affected : AF, CF, OF, PF, SF, ZF.

SUB/SBB Instruction:
Syntax:

SUB destination, source ; destination= destination - source
SBB destination, source; destination= destination – source- Carry

- The source may be an immediate number, a register, or a memory location.
- The source and the destination in an instruction cannot both be memory locations.
- The source and destination both must be of same size.
- Flags affected : AF, CF, OF, PF, SF, ZF.

(b) Explain the flag register with necessary diagrams. Choose an appropriate example to show how flag bits are updated.

- The flag register is a 16-bit register sometimes referred as the ***status*** register. Although the register is 16-bit. Not all the bits are used.
- ***Conditional flags***: 6 of the flags are called the conditional flags, meaning that they indicate some condition that resulted after an instruction was executed. These 6 are: CF, PF, AF, ZF, SF, and OF.
- The 16 bits of the flag registers:

R	R	R	R	OF	DF	IF	TF	SF	ZF	U	AF	U	PF	U	CF

R=	reserved	SF=	sign flag
U=	undefined	ZF=	zero flag
OF=	overflow flag	AF=	auxiliary carry flag
DF=	direction flag	PF=	parity flag
IF=	interrupt flag	CF=	carry flag
TF=	trap flag		

CF, the Carry Flag: This flag is set whenever there is a carry out, either from d7 after an 8-bit operation, or from d15 after a 16-bit data operation.

PF, the Parity Flag: After certain operations, the parity of the result's low-order byte is checked. If the byte has an even number of 1s, the parity flag is set to 1; otherwise, it is cleared.

AF, the Auxiliary Carry Flag: If there is a carry from d3 to d4 of an operation this bit is set to 1, otherwise cleared (set to 0).

ZF, the Zero Flag: The ZF is set to 1 if the result of the arithmetic or logical operation is zero, otherwise, it is cleared (set to 0).

SF, the Sign Flag: MSB is used as the sign bit of the binary representation of the signed numbers. After arithmetic or logical operations the MSB is copied into SF to indicate the sign of the result.

TF, the Trap Flag: When this flag is set it allows the program to single step, meaning to execute one instruction at a time. Used for debugging purposes.

IF, Interrupt Enable Flag: This bit is set or cleared to enable or disable only the external interrupt requests.

DF, the Direction Flag: This bit is used to control the direction of the string operations.

OF, the Overflow Flag: This flag is set whenever the result of a signed number operation is too large, causing the high-order bit to overflow into the sign bit.

The followig example shows, the effect of ADD instruction on Flag bits

```
MOV   AL,9CH      ;AL=9CH
MOV   DH,64H      ;DH=64H
ADD   AL,DH       ;now AL=0
```

Solution:

```
      9C          1001  1100
+     64          0110  0100
      00          0000  0000
```

CF=1 since there is a carry beyond d7
PF=1 since there is an even number of 1s in the result
AF=1 since there is a carry from d3 to d4
ZF=1 since the result is zero
SF=0 since d7 of the result is zero

6 (a) What are control transfer instructions

In the sequence of program execution it may be necessary to transfer the control to another instruction than the next instruction in line. This is achieved through the use of control transfer instruction. The various type of control transfer instruction available with 8086 are

- Conditional jump
- Unconditional jump
- Procedure Call

CALL:

The CALL instruction is used to 8086 Program Execution Transfer Instructions to a subprogram or procedure. There are two basic types of CALLs, near and far.

- A near CALL is a call to a procedure which is in the same code segment as the CALL instruction. When the 8086 executes a near CALL instruction it decrements the stack pointer by two and copies the offset of the next instruction after the CALL on the stack. It loads IP with the offset of the first instruction of the procedure in same segment. The near CALL is also known as intra-segment CALL
- A far CALL is a call to a procedure which is in a different segment from that which contains the CALL instruction. When the 8086 executes a far CALL it decrements the stack pointer by two and copies the contents of the CS register to the stack. It then decrements the stack pointer by two again and copies the offset of the instruction, after the CALL to the stack. Finally, it loads CS with the segment base of the segment which contains the procedure and IP with the offset of the first instruction of the procedure in that segment. The far CALL is also known as inter segment CALL.

RET:

The RET instruction will return 8086 Program Execution Transfer Instructions from a procedure to the next instruction after the CALL instruction in the calling program.

- If the procedure is a far procedure (in a different code segment from the CALL instruction which calls it), then the instruction pointer will be replaced by the word at the top of the stack. The stack pointer will then be incremented by two. The code segment register is then replaced with a word from the new top of the stack. After the code segment word is popped off the stack, the stack pointer is again incremented by two. So 8086 will fetch the next instruction after the CALL.
- If the procedure is a near procedure ,then the instruction pointer will be replaced by the word at the top of the stack. The stack pointer will then be incremented by two. So 8086 will fetch the next instruction after the CALL.

6 (b)Demonstrate all conditional and unconditional jump Instructions available in 8086 assembly level programming with syntax and application.

Unconditional Jump:

JMP label

This instruction will always cause the 8086 Program Execution Transfer Instructions to fetch its next instruction from the location specified in the instruction rather than from the next location after the JMP instruction.

- There are two basic types of JMPs, near and far. A near JMP is a jump where destination location is in the same code segment. In this case only IP is changed. A near JMP is known as intra-segment JMP.
- A far JMP is a jump where destination location is from a different segment. In this case both IP and CS are changed as specified in the destination. A far IMP is known as Inter segment JMP Near and far jumps are further described as either direct or indirect.

- If the destination address for the jump is specified directly within the instruction, then the jump is described as direct.

- If the destination address for the jump is contained in a register or memory location, the jump is referred as indirect, because the 8086 has to access the specified register or memory location to get the required destination address.

Conditional Jump:

Syntax:

Jcond label

Conditional Transfer Instructions: These instructions will cause a jump to a label given in the instruction if the desired condition(s) occurs in the program before the execution of the instruction. The destination must be in the range of – 128 bytes to + 127 bytes from the address of the instruction after the conditional transfer instruction. If the jump is not taken, 8086 Program Execution Transfer Instructions simply goes on to the next instruction.

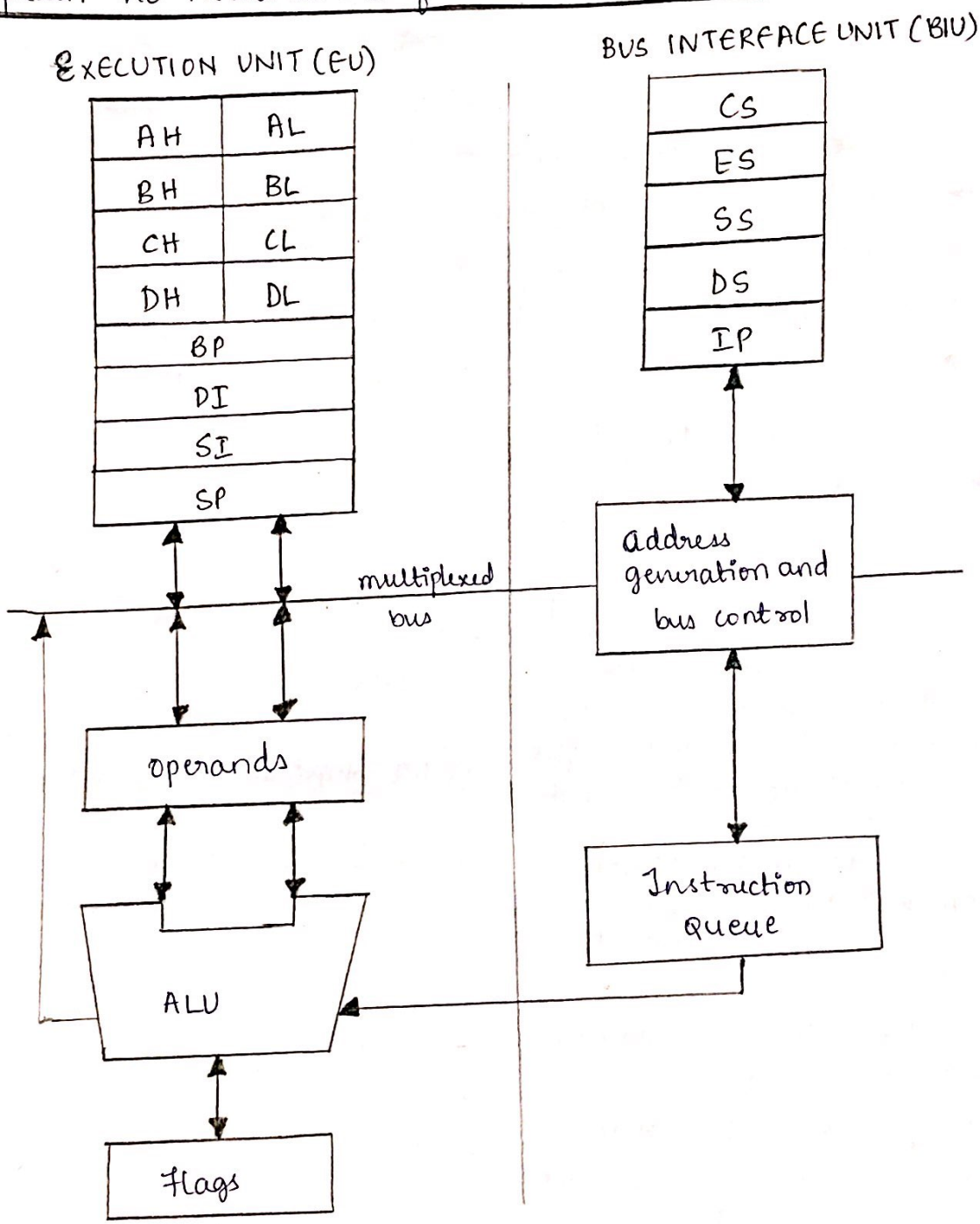
Instruction Code	Description	Condition for jump
JA/JNBE	Jump if above/Jump if not below or equal.	CF = 0 and ZF = 0
JAE/JNB	Jump if above or equal/Jump if not below.	CF = 0 and ZF = 1
JB/JNAE/JC	Jump if below/Jump if not above or equal.	CF = 1 and ZF = 0
JBE/JNA	Jump if below or equal/Jump if not above.	CF = 1 and ZF = 1
JE/JZ	Jump if equal/Jump if zero flag.	ZF = 1
JG/JNLE	Jump if greater/Jump if not less than nor equal.	ZF = 0 and CF = 0
JGE/JNL	Jump if greater than or equal/Jump if not less than.	SF = 0
JL/JNGE	Jump if less than/Jump if not greater than or equal.	SF ≠ 0
JLE/JNG	Jump if less than or equal/Jump if not greater	ZF = 1 or SF ≠ 0
JNC	Jump if no carry	CF = 0
JNE/JNZ	Jump if not equal/Jump if not zero	ZF = 0
JNO	Jump if no overflow	OF = 0
JNP/JPO	Jump if not parity/Jump if parity odd	PF = 0
JNS	Jump if not sign or jump if positive	SF = 0
JO	Jump if overflow flag = 1.	OF = 1
JP/JPE	Jump if parity/Jump if parity even	PF = 1
JS	Jump if sign flag = 1 or jump if negative	SF = 1

7 Write an assembly language program using 8086 instructions to add 8 byte information stored in consecutive memory locations. Consider the data in such a way that the resulting sum is a word data. Find the average value. Use full segment Definition.

```
; DATA SEGMENT DEFINITION
DATA1 SEGMENT     ; DATA1 SEGMENT STARTS
ARR  DB 23H, 45H,12H, 34H,23H, 47H,33H, 56H
LEN DW 8
SUM DB 00,00
```

```
DATA1 ENDS          ; DATA1 SEGMENT ENDS

; CODE SEGMENT DEFINITION
CODE1 SEGMENT       ; CODE1 SEGMENT STARTS

ASSUME CS:CODE1, DS:DATA1  ; The ASSUME directive is used to tell the assembler
that the name of the logical                ;segment should be used for a specified
segment
MOV AX,DATA1
MOV DS,AX

LEA SI,ARR          ; GET THE OFFSET ADDRESS OF DATA IN TO SI REG
MOV CX,LEN          ; LOAD THE COUNT
MOV AX,00           ; INITIALISE TARGET REGISTER FOR SUM

COMPUTE: ADD AL,[SI]
ADC AH,0
INC SI
LOOP COMPUTE


MOV SUM,AL          ; STORE THE SUM IN MEMORY LOCATION
MOV SUM+1,AH        ; STORE THE HIGHER WORD OFSUM IN MEMORY


MOV AH,4CH
INT 21H

CODE1 ENDS          ; CODE1 SEGMENT END
END
```

1. Explain the Architecture of 8086 microprocessor



Internal Block Diagram of the 8086 microprocessor

- As 8086 does 2-stage pipelining (overlapping fetching & execution), its architecture is divided into 2 units:
  1. Bus Interfacing Unit (BIU)
  2. Execution Unit (EU)

1. Bus Interfacing Unit (BIU)

- It provides the interface of 8086 to external memory and I/O devices.
- It operates with respect to bus cycles.
- BIU performs following functions –
  - → It generates 20 bit physical address for memory access.
  - → It fetches instruction from memory.
  - → It transfers data to and from the memory and I/O.
  - → It supports pipelining using 6 byte instruction queue.
- The main components of BIU are as follows :

→ Segment Registers

CS register

- CS holds the base address for the code segment. All programs are stored in the code segment.
- CS is multiplied by 10H to give the 20 bit physical addess of code segment.

Eg: If CS = 4321H then CSX10H = 43210H → Starting address of code segment.

DS register

- DS hols the base address for the data segment.
- It is multiplied by 10H to give the 20 bit physical address of the data segment.

Eg: If DS = 4321H then DSX10H = 43210H → starting address of Data segment.

SS register

- SS holds the base address for the stack segment
- It is multiplied by 10H to give 20 bit physical address of stack segment.

Eg: If SS = 4321H then SSX10H = 43210H → starting address of stack segment

ES register

- ES holds the base address for the extra segment
- It is multiplied by 10H to give the 20 bit physical address of Extra segment

Eg: If ES = 4321H then ESX10H = 43210H → starting address of Code segment.

→ Instruction Pointer (IP) :

- It is a 16 bit register.
- Address of next instruction is calculated as CSX10H + IP.
- IP is incremented after every instruction byte is fetched.
- IP gets a new value whenever a branch occurs.

2. Execution Unit. (EU)

→ It fetches instructions from the queue in BIU, decodes and executes them.

→ It forms arithmetic, logical and internal data transfers operations within the microprocessor.

→ It sends request signals to the BIU to excess the external module.

The main Components of the EU are as follows:

- General Purpose registers:

. 8086 microprocessor has 4 16-bit general purpose registers. AX, BX, CX & DX.

. Each of these can be divided into 2 8-bit registers such as AH, AL; BH, BL; CH, CL; & DH, DL.

★ Special Operations of general purpose register:

(I) AX Registers:

→ holds operands & result during multiplication & division operation.

→ All I/O data transfers using IN & OUT instructions use A register (AL/AH or AX).

→ functions as accumulator during string operations.

(II) BX Registers:

→ holds the memory address (offset address) in indirect addressing modes.

(III) CX Registers:

→ holds count for instructions like loop, rotate, shift & string operations.

(IV) DX Registers:

→ used with AX to hold 32-bit value during multiplication & division.

→ used to hold the address of the I/O port in indirect I/O addressing mode.

Special purpose Registers:

(I) Stack pointer:

→ 16-bit

→ holds the address of top of the stack.

→ used during push, POP, CAL, RET ... etc. types of instruction.

(II) Base Pointer:

→ 16 bits.

→ holds address (offset) of any location in the stack segment.

→ used to access random location of the stack.

(III) Source Index:

→ 16-bit.

→ hold the offset address of data segment.

→ can be also used for other segment using segment overriding.

→ hold the offset address of source data in Data segment during string operation.

(IV) Destination Index:

→ 16 bits

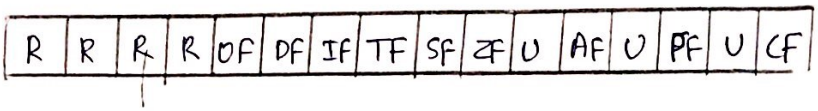
→ holds the offset address for extra segment.

→ holds offset address of destination in Extra segment during string operation.

ALU (Arithmetic Logical Unit).

→ 16 bit.
→ performs 8 & 16-bit arithmetic & logical operations.

Flag Register



→ It has 9 flags.
→ 6 status & 3 control flag.

(I) Carry flag.
(II) Parity flag.
(III) Auxiliary flag
(IV) Zero flag
(V) Sign flag.
(VI) Overflow flag.

(I) trap flag
(II) Interrupt flag
(III) Direction flag.

→ Status flags are affected by the ALU after every Arithmetic & logical operation.
→ Control flags are used to control certain operations.

Operand Register:

→ 16-bit
→ used by the control register to hold the operands temporarily.
→ not available to the Programmer.

• Address Generation Circuit.

- The BIU has a Physical address generation circuit.
- It generates the 20 bit physical address using segment & offset addresses using the formula:

Physical address = Segment Address x10H + Offset address

Features of 8086

★ It is a 16-bit

★ 8086 has a 20 bit address bus can access upto 2^{20} memory locations (1MB)

★ It can support up to 64K I/O ports.

★ It provides 14, 16-bit registers

★ Word size is 16 bits

★ It has multiplexed address and data bus AD0-AD15 and A16 - A19.

★ It requires single phase clock.

Explain the different addressing modes used in 8086 µp with suitable example.

The CPU can access operands in various ways called Addressing modes.
The 8086 µp provides a total of seven distinct addressing modes.

1. register
2. Immediate
3. Direct
4. Register Indirect
5. Based relative
6. Indexed relative
7. Based Indexed relative

1) Register AM:
The register AM involves the use of registers to hold the data to be manipulated. It is relatively fast because memory is not accessed when AM is executed.

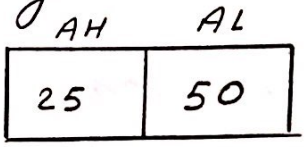
Eg: MOV BX, DX ; copy the contents of

; DX into BX
The src and dest registers must match in size.

2) Immediate AM:
Immediate AM can be used to load information into any of the registers except the segment registers and flag registers. Here the source operand is constant. The destination operand can be register, memory (Indirect)

Eg: MOV AX, 2550H ; move 2550H into AX

The data cannot be directly moved into segment registers (DS).

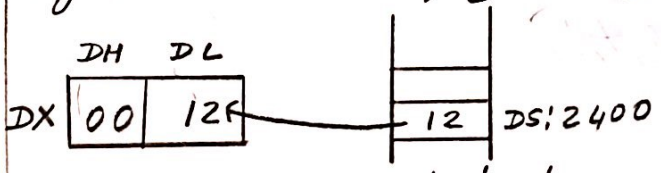


3) Direct Addressing Mode

In Direct AM, the data is in some memory location and the offset address of the data in memory comes immediately after the instruction. Both src and dest cannot be memory

Here the address of the operand is provided with the instruction instead of actual operand. Src can't be immediate.

Eg: MOV DL, [2400] ; move contents of
; DS:2400H into DL

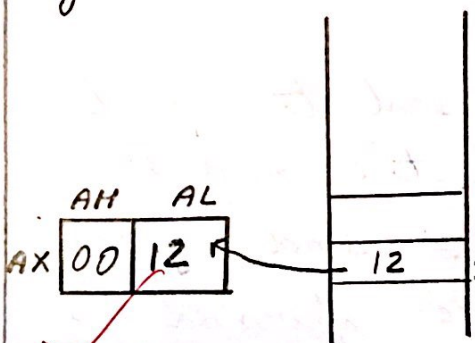


4) Register Indirect AM

In the Register Indirect AM, the offset address of the memory location where the operand resides is held by a register.

The registers used for this purpose are:- SI, DI and BX. Both Src and dest cannot be memory

Eg: MOV BX, 7H
MOV AL, [BX] ; move into AL the
; contents of the memory
; location pointed to by
; DS:BX



Src can be register, memory and immediate. The dest can be register or memory.

5) Based Relative AM

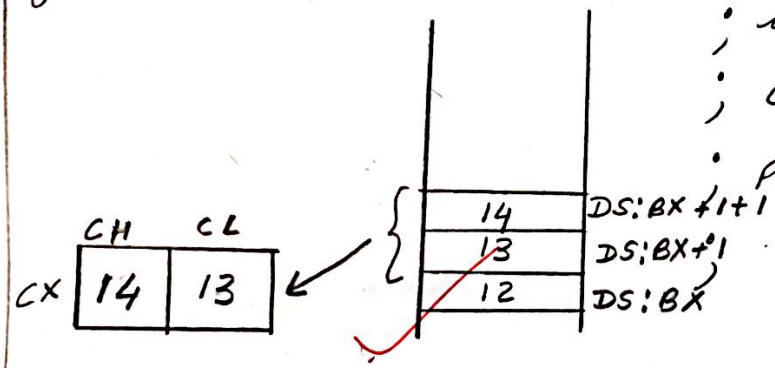
In the based relative AM, base registers BX and BP, as well as a displacement value are used to calculate what is called the Effective Address.

The default segments used for the calculation of the physical address (PA) are:-

DS for BX and SS for BP

Eg: MOV CX, [BX]+1 ; move DS:BX+1 ; and DS:BX+1 +1 into ; CX.

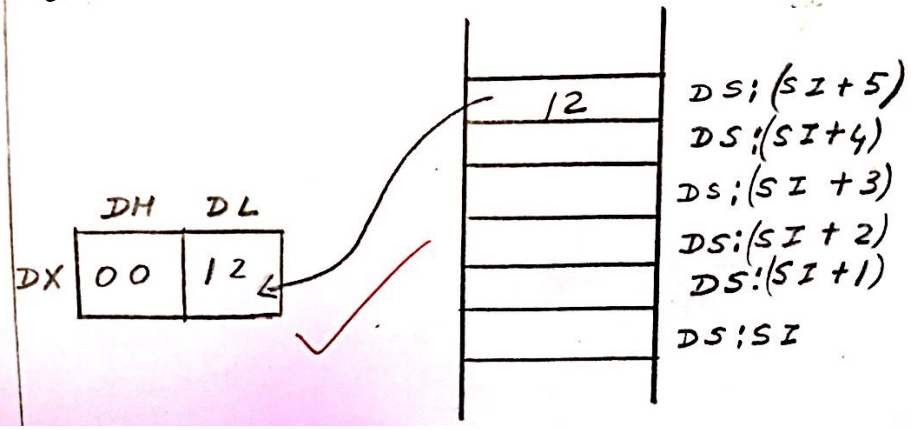
PA = DS(shifted left)+ BX +1



6) Indexed Relative AM

The Indexed Relative AM works the same as the Based Relative AM, except that the registers DI and SI hold the offset address.

Eg: MOV DX, [SI]+5 ; PA = DS(shifted left): SI +5

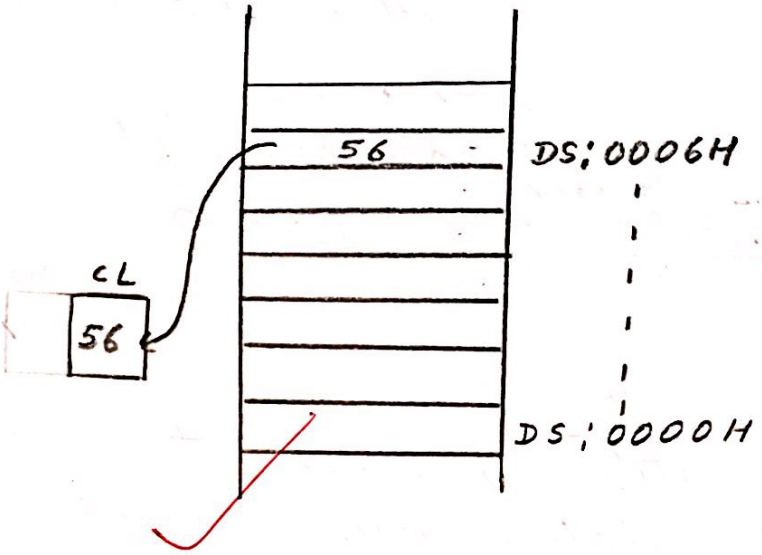


7) Based Indexed AM

By combining based and Indexed AMs, a new AM is derived called the Based Indexed AM.

In this mode, one base register and one index register are used.

Eg: MOV CL, [BX][DI]+2; PA = DS (shifted left) + BX + DI + 2



; BX = 0001H

; DI = 0003H