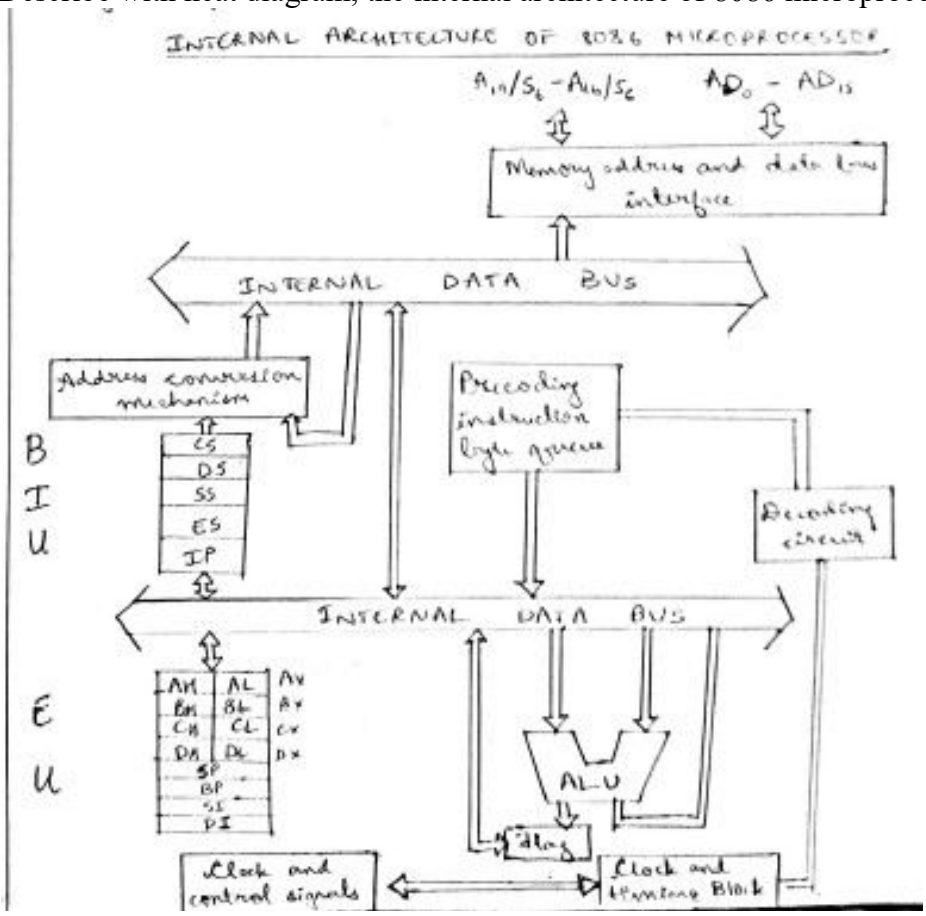
CMR INSTITUTE OF TECHNOLOGY

USN | | | | | | | | | | |

Internal Assesment Test-I									
Sub:	MICROPROCESSORS							Code:	15EC42
Date:	12/03 / 2018	Duration:	90 mins	Max Marks:	50	Sem:	4th	Branch:	ECE(A,B)
Answer FIVE FULL Questions									

		Marks	OBE CO	OBE RBT
1.	Describe with neat diagram, the internal architecture of 8086 microprocessor.	[4+3+3]	CO1	L1



(6)

Architecture of 8086 — (a) Bus Interface Unit (BIU)
(b) Execution unit (EU)

BIU: It contains circuit for physical address calculations
and predecoding instruction byte queue (6 byte long)

→ This unit is responsible for establishing communications with external devices and peripherals including memory via the bus.

– When the opcode is fetched and decoded, the external bus remains free for some time. This time slot is utilised in 8086 to achieve the overlapped fetch and execution cycles.

– While the fetched instruction is executed internally, the external bus is used to fetch the machine code of the next instruction and arrange it in a queue known as predecoded instruction byte queue.

– It is a 6 bytes long FIFO structure.

– Once a byte is decoded, the queue is rearranged by pushing it out and the queue status is checked for the possibility of the next opcode fetch cycle.

→ While the opcode is fetched by BIU, the EU executes the previously decoded instructions concurrently.

- The BIU along with the EU forms a pipelin
- BIU works under the control of timing and control unit.

EU: It contains the register set of 8086 except segment registers and IP.
- It has 16-bit ALU, able to perform arithme and logic operations.
- 16-bit flag register reflects the results of execution by the ALU.

Decoding unit: It decodes the opcode bytes issued from the instruction byte queue.
- The timing and control unit derives the necessary control signals to execute the instructi opcode received from the queue.
- The EU may pass the results to the BIU for storing them in memory.

2.(a) Briefly describe the memory segmentation of 8086. [02+04] CO1 L1

Memory segmentation ④

The 8086 is able to access 1 MB of memory. The memory in an 8086 based system is organised as segmented memory.

→ The complete available memory is divided into a number of logical segments.

→ Each segment is 64k bytes in size and addressed by one of the segment registers.

→ The 16 bit contents of the segment register point to the starting location of a particular segment.

→ We need an offset address to address a specific memory location.

→ offset address is 16 bit long.

∴ maximum 2^{16} = 64k memory locations can be accessed.

Maximum offset value can be FFFFH

→ CPU 8086 able to address 1 MBytes of physical memory.

→ It is divided into 16 segments, each of 64 kBytes size.

1 MBytes = $2^{10} \times 2^{10}$

$= (2^{10} \times 2^{6}) \times 2^{4}$

= 64kBytes x 16

∴ 16 segments.

Address of the segments: 0000H
1000H
:
F000H

Offset address values are from 0000H to FFFFH.

Here the segments are non-overlapping segments.

Advantages

i) Allows the memory capacity to be 1MB, although the actual addresses to be handled are of 16 bit size

ii) Allows the placing of code, data and stack portions of the same program in different parts of memory, for data and code protection

iii) Permits a program and/or its data to be put into different areas of memory each time the program is executed, i.e. provision for relocation is done

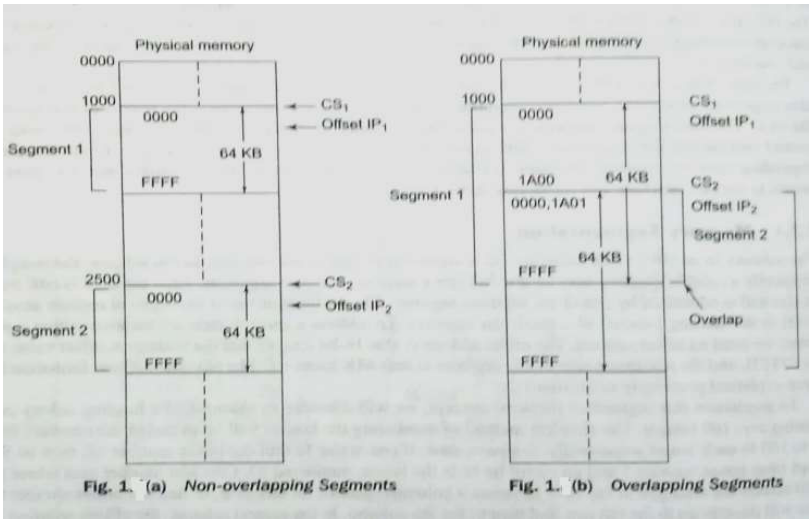


Fig. 1. (a) *Non-overlapping Segments* Fig. 1. (b) *Overlapping Segments*

(b) If CS = 1000 H, DS = 25A0H, SS = 3210H, ES = 5890H, BX = 43A9H, BP = 3400H, find the physical address of the source data for the following instructions: [02+02] CO1 L3
(i) MOV AL, [BX+ 1200H]
(ii) ADD BL, [BP+05]

i) Physical address = 10H * DS + BX + 1200H
= 10H * 25A0H + 43A9H + 1200H
= 2AFA9H

ii) Physical address = 10H * SS + 3400H + 05H
= 10H * 3210H + 3400H + 05H
= 32100H + 3405H
= 35505H

3. Describe briefly any 5 data addressing modes of 8086 with an example for each. [02*05] CO1 L1

Addressing modes of 8086

1. Immediate: Data is a part of the instruction and appears as successive byte or bytes.
MOV AX, 0005H → 16 bit immediate
MOV BL, 06H → 8bit immediate

2. Direct: A 16-bit memory address (offset) or an I/O address is directly specified in the instruction as a part of it.
MOV AX, [5000H] ; 10H * DS + 5000H → effective address
[5000H] → offset address
IN 80H ; ; 80H is IO address

3. Register

MOV BX, AX

ADC AL, BL

→ The operands in these instructions are provided in the registers BX, AX, AL, BL respectively.

4. Register Indirect

In this addressing mode, the offset address of data is in BX, SI, DI register.

The address of the memory location which contains the data or operand is determined using the offset registers.

MOV AX, [BX]

Here data is present in a memory location in DS whose offset address is in BX.

effective address 10H * DS + [BX]

5. Indexed:

offset of the operand in one of the index registers.

DS is the default segment for index registers SI and DI.

In case of string operations DS and ES are the default segments for SI and DI respectively.

MOV AX, [SI]

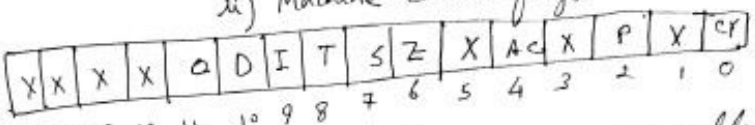
MOV CX, [DI]

Here, data is available at an offset address stored in SI in DS.

The effective address, is, 10H * DS + [SI]

4. Describe the flag register structure of 8086 with a neat diagram. [2+8]

Flag Register — i) condition codes or status flags
ii) Machine control flags



O – overflow flag
D → Direction flag
I → Interrupt flag
T → Trap flag
S - Sign flag
Z → zero flag
Ac → Auxiliary carry flag
P → Parity flag
CY → carry flag
X → not used.

S - Sign flag :- If result of any computation is -ve. For signed computations sign flag is MS. of the result.

Z - zero flag :- If the result of computation or comparison performed by the previous instructions is zero.

P - parity flag: flag is set if lower byte of the result contains even number of 1s.

C – carry flag :- set if carry out of MSB for addition or a borrow for subtraction.

T - Trap flag : If it is set processor enters single step execution mode. Trap interrupt is generated after execution of each instruction. The processor executes the current instruction and control is transferred to Trap interrupt service routine.

I - Interrupt flag : - The maskable interrupts are recognised by CPU if it is set.

D – Direction flag : If this bit is '0' the string is processed beginning from the lowest address, to the highest address, i.e. auto incrementing mode. - otherwise string is processed from the highest address towards the lowest address, autodecrementing mode.

AC – Auxiliary Carry Flag : It is set if there's a carry from the lowest nibble, during addition, or borrow from the lowest nibble, i.e. bit 3, during subtraction.

O – Overflow flag : Set if overflow occurs, i.e. the result of a signed operation is large enough to be accomodated in a destination register. eg. addition of two signed numbers, if the result overflows into the sign bit, i.e. the result is of more than 7-bits in size in case of 8 bit signed operations and more than 15-bits in size in case of 16-bit signed operations, then the overflow flag will be set.

5. Describe the following instructions with example. i) AAD ii) LEA iii) IDIV iv) SAR. [3+2 +3+2] CO2 L1

i)AAD **- ASCII adjust before Division**

The AAD instruction converts two unpacked BCD digits in AH and AL to the equivalent binary number in AL. This adjustment must be made before division.

Example:

```
MOV AH, '6'          ; AH=36
MOV AL, '2'          ; AL=32
MOV BL,'8'           ; BL=38
SUB AX,3030H         ;AX=0602
SUB BL,30H           ; BL=8
AAD                  ; AX=003EH
DIV BL               ;AX=0607
```

ii) LEA

Syntax: LEA Destination, Source

The LEA instruction Loads the offset (effective) address of source operand into specified destination register.

Ex: LEA BX,ARRAY ; BX←Offset address of ARRAY is loaded into
; destination register BX

iii) IDIV

Syntax: IDIV SOURCE

- It divides a signed word or double word by a signed byte or word operand respectively.
- The **source Operand** can be **a general purpose register** or **memory**. Cannot be a constant or immediate data.
- **CF,OF,SF, ZF, AF, PF** are **unpredictable**

Example: 16-bit division:

Let **DX=0000h, AX=0005h**, and **BX=FFFEh**
IDIV BX; AX=FFFE DX=0001

iv) SAR.

SAR Dest, Count

- SAR instruction shifts the contents of destination operand (register/memory location) to the right either one or count specified in CL register.
- As a bit is shifted out of the MSB position, a copy of the old MSB is put in the MSB position.
- The LSB bit is shifted to the carry flag.
- This instruction preserves the sign of the number.

Example:

```
MOV DL,80H     ; DL=80H
MOV CL, 04H    ;CL=04h
SAR DL, CL     ;DL=F8H
```

7. (a) Write an ALP to copy a block of 10 data bytes from location SRC to location DST in memory. [06] CO2 L3

```
        .model small
        .data
src db 20h,21h,22h,23h,24h,25h,26h,27h,28h,29h
count equ $-src
dst db count dup (00)
        .code
        mov ax,@data
        mov ds,ax
        mov cx,count
        lea si,src
        lea di,dst
back:mov al, [si]
        mov [di], al
        inc si
        inc di
        loop back
        mov ah,4ch
        int 21h
        end
```

(b) Write an ALP to multiply two signed bytes data and store the result in memory. [04] CO2 L3

```
         .model small
        .data
num1 db 0FEH
num2 db 0FEH
res     dw 00h

        .code
         mov ax,@data
         mov ds,ax
         mov al, num1
         imul num2
         mov res,ax

         mov ah,4ch
         int 21h
        end
```

RESULT: 0004H

[10] CO2 L3

8.(a) Correct the following instructions if any mistake, and explain the operations performed by each of them:

i. ADD [5000H], 0100H
Instruction is correct.

Here an immediate data 0100H is copied to the memory location in data segment whose Offset address is 5000H.
DS:[5000H]←0100H

ii. INC [BX]
Instruction is correct.
This instruction increases the contents of the memory location pointed by BX by 1.

iii. NOT 34H
Instruction is wrong.
NOT instruction cannot work on immediate data.
This instruction complements (inverts) the contents of register or memory location.
SO take 34H into 8-bit general purpose register or into memory location and the use NOT.
MOV AL,34H
NOT AL

iv. DAA
Instruction is correct.
Working of DAA Instruction:

- If after an ADD or ADC instruction the lower nibble of AL is greater than **9**, or if **AF = 1**, DAA instruction **adds 06 to the lower nibble of AL**.
- After adding 06, If the upper nibble of AL is greater than **9**, or if **CF = 1**, DAA instruction **adds 6 to the upper nibble of AL.**

For example, adding **29H** and **18H** will result in **41 H**, which is incorrect as far as BCD is concerned.

	Hex	BCD	
	29	0010 1001	
+	18	+0001 1000	
	41	0100 0001	AF = 1
+	6	+ 0110	because AF =1 DAA will add 6 to lower nibble
	47	0100 0111	The final result is BCD

v. JNC Label
Instruction is correct.
It is a conditional branch instruction.
This instruction examines the Carry Flag (CF), if it is set as a result of previous operation then execution control is transferred to the address specified by the 'Label' in the instruction.

6.(a) Differentiate between the following instructions: [2*3]

i. SUB & CMP

SUB(Subtract)	CMP(Compare)
SUB instruction subtracts the source operand from the destination operand and the **result is stored** in destination operand.	It performs comparison by subtracting the source operand from destination operand but **does not store the result** anywhere.
Source operand may be a register, memory location or immediate data. **Destination operand** may be a register or memory location.	**Source operand** may be a register, memory location or immediate data. **Destination operand** may be a register or memory location.
All the status flags will be affected by this instruction	The flags are affected depending upon the result of subtraction.

ii. SHIFT & ROTATE

SHIFT	ROTATE
Shift instruction bit-wise shifts the contents of a destination operand which could be in register or memory location to right or left.	Rotation instructions bit-wise rotates the contents of destination operand (register/memory location) to the right or left either one or count specified in CL register .
There are two kinds of shifts: **Logical** and **Arithmetic** The **logical shift** is for **unsigned operands** (SHL, SHR), Here the shifted position is inserted with logical '0'. and The **arithmetic shift (SAR)** is for **signed operands.** This instruction preserves the sign of the number.	The rotation instructions are of two types one is simple rotation of the bits of the operand. ROR, ROL. And other is a rotation through carry. RCR, RCL.

iii. CALL & JMP

Unconditional Branch instructions

Call: Call a subroutine from a main program.
↓
procedure.

Near call i.e (±32K displacement): procedure available in the same segment.

procedure in another segment (far CALL)

- When executed it stores the incremented IP and CS onto the stack and loads the CS and IP registers with the segment and offset address of the procedure to be called.
- Pushes only IP in case of NEAR CALL and both IP and CS in case of FAR CALL

JMP (unconditional Jump):

Unconditionally transfers the control of execution to the specified address using 8-bit or 16-bit displacement (intrasegment relative, short or long) or CS:IP (intersegment direct far).

- No flags are affected.

6.(b) [1*4]

(i)

ALE (Address latch enable): When it is high it indicates availability of valid address on the address/data lines.

(ii)

RESET(I): This signal causes the processor to terminate the current activity and start execution from FFF0h.

- must be high for at least four clock cycles

(iii)

$\overline{TEST}$(I): This input is examined by 'WAIT' instruct
If $\overline{TEST}$ goes low, execution will continue,
else processor remains in idle state.

(iv)

M/$\overline{IO}$ - Memory / IO !

Equivalent to $\overline{S_2}$ in the maximum mode.
When it is low, CPU is having I/O operation.
When it is high, it indicates CPU is having a memory operation.