2/8/17

MODULE 1

1st generation (1971-1973)

* low cost, speed, power.
* PMOS
* Not compatible with high speed (TTL)

2nd generation

* NMOS
* Compatible with TTL
* Subroutine nesting.

3rd generation

* HMOS
* High speed.
* Program relocation
* Virtual memory.
* Segmented memory.

8086.

Bit - A unit of information.

Using 2 bit positions, one can generate 4 addresses.

$2 \Rightarrow 4$

$3 \Rightarrow 8$

$\vdots$

$n \Rightarrow 2^n$

4 bits $\Rightarrow$ nibble

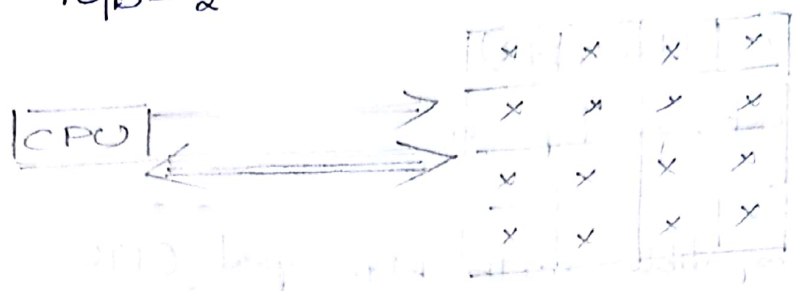
8bits $\Rightarrow$ byte

16 bits $\Rightarrow$ word.

1024 bits = 1KB = 2^{10}.

1024×1024 = 1MB = 2^{20}

1GB = 2^{30}.



8086 is called a 16 bit processor

It means we can read or write 16 bits at at a time.

Total → 40 pins.

1/8/17 Mon

8086 Architecture.

* 16 bit data bus, ALU, registers.
* 20 bit address bus = 2^{20} (1MB) memory
* 40 pins in 8086 chip.

Architecture is divided into two parts

1) Bus Interface Unit (BIU)

* Interface with peripherals - enable 8086 processor to communicate with external peripherals
* Memory address calculation

Entire 1MB memory is divided (logically) into equal parts. Each part is called segment.

Each segment contain 64kB.

Number of segments = $\frac{2^{20}}{2^6 \times 2^{10}} = 2^4 = 16$ segments.

For a data there are two addresses:

Segment address - base address of segment. (16 bit).

offset (16 bit) - distance from the segment address.

eg of segment is 64kB 2^{16}

If there are no segments 00001H.

If there are segments 00001H.

20 bit physical address. =

1> Shift segment address 4 bit, left.

2> Add offset.

Eg: segment address = 2001H

Offset = A320H

Step1>

0010 0000 0000 0001

0010 0000 0000 0010

0010 0000 0000 0010 0

0010 0000 0000 0010 00

0010 0000 0000 0010 000

Step2>

0010 0000 0000 0010 0000 +

1010 0011 0010 0000

0010 1010 0011 0011 0000

2 A 3 3 0

20010
A320
2A330

Physical address = segment address * 10 + offset

2001 * 10 + A320 = 2A330.

* Pipelining

2> Execution Unit (EU)

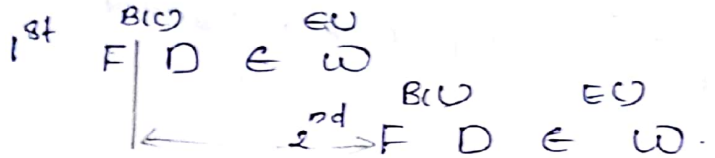
* Execute instruction.

> Pipelining

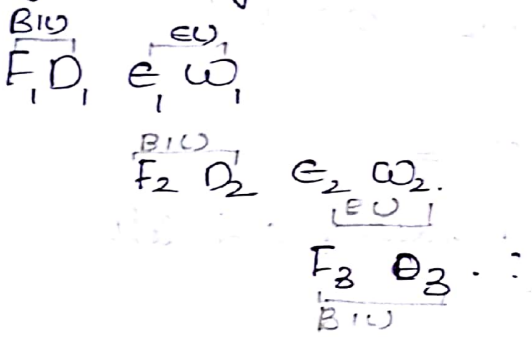
BIU performs — EU performs.

Fetch Decode Execute Write

F D E W.



During pipelining



BIU contains predecoded queue. (6 byte). to hold. the see instruction ↓ while 1st instr is handled by BIU. (temp storage for fetch decoded instruction).

18/12 Tue

8086 registers

1> General Data registers (16) ~~AX~~ EU

holds data used in computations.

There are 4 general data registers AX, BX, CX, DX

	AH (8)	AL (8)	
AX			Accumulator.
BX	BH	BL	
CX	CH	CL	Counter
DX	DH	DL	

2> Segment registers (16) BIU

Used to hold segment address.

CS : Code segment. seg Actual program code is stored
DS : Data segment base add of seg where data is stored
SS : Stack segment addr of stack
ES : Extra segment seg add of another segments

3> Index & Pointer register (16) EU

SP : Stack pointer points to top of stack
BP : Base pointer used for diff. addr modes.
SI : Source Index stores offset of source data
DI : Destination Index stores offset of destination data
IP : Instruction pointer (addr. of next instruction to be executed)

4> flags (16) EU

Machine control

X	X	X	X	O	D	I	T	S	Z	X	Ac	X	P	X	C
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Two groups:

1> Status flag | conditional code (Rest flags)
2> Machine control (D, I, T)

1> O : Overflow flag

If the result of arithmetic operation create more than 16 bit overflag is set to 1.

2> D : Direction flag

Strings can be processed in forward / backward mode (0 -> Start to end Auto increment 1 -> backward Auto decrement)

3> I : Interrupt

0 -> Disable interrupt
1 -> Interrupt enabled.

T: Trap

1 → Single step execution
There is an interrupt after each execution.

S: Sign

0 → Postive number

1 → Negative numbers.

Z: Zero

1: result is zero.

0: non-zero.

Ac: Auxiliary carry.

1: A carry from LSB to MSB

1 → Ac → 1
0000 1000
0000 1000
0000

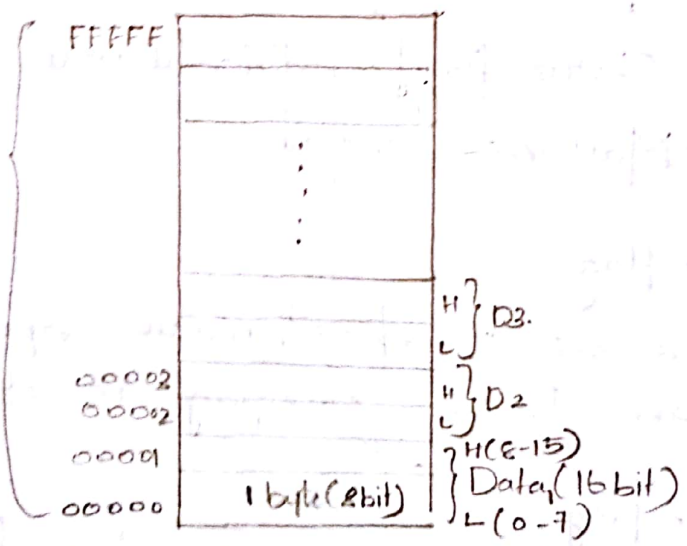
P: Parity

C: Carry

1: Carry is present.

9/8/22 Wed

Physical memory organization:



Data Bus = $D_0 - D_{15}$

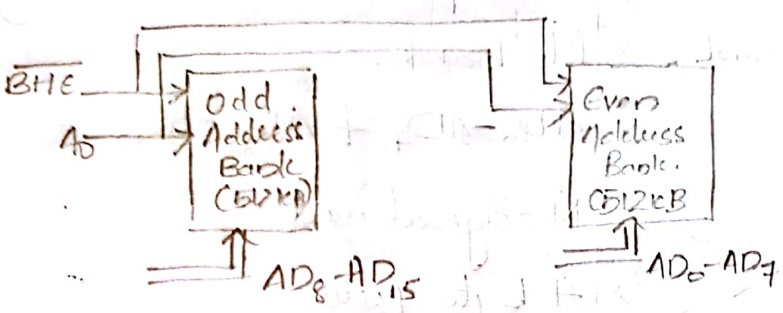
$D_0 - D_7$ lower byte

$D_8 - D_{15}$ higher byte

for a data, the lower bytes bits of data are stored in even bytes. and higher bits are stored in odd m/m locations.

Lower byte (0-7) in even add = Even address bank.
Higher byte (8-15) in odd add = Odd address bank.

Total size of even add = 512kB.
" odd add = 512kB.



BHE, A0 → Two signals to select the banks.

> Modes of Operation of 8086.

1> Minimum mode	2> Maximum mode
→ Only one processor (8086)	More than one processor
→ All signals (RD, WR, HALT..) generated by processor.	8086 generates only status signal S0, S1, S2. Then bus controller generate all three signals.
→ MN/$\overline{MX}$ = 1 MN=1 MX=0. (minimum mode.	MN/$\overline{MX}$ = 0 MN=0 MX=1
→ Latches, transceivers, memory.	

> 8088 | 8086

8086	V/S	8088
* 20 bit address bus 1MB memory.	*	Same.
* 16 bit data bus	*	8 bit data bus.
* 16 bit processor	*	8 bit processor
* Even & odd address bank	*	No bank.
* $AD_0 - AD_{15}$	*	$AD_0 - AD_7$ & $AD_8 - AD_{15}$
* $\overline{BHE}$ signal is used	*	No signal used.
* 6 byte queue	*	4 byte queue.
* BIU fetch next instruction when 2 byte is free in Q.	*	1 byte
* $\overline{IO}/M$	*	IO/$\overline{M}$ (to make 8088 compatible with 8085).
* Maximum current 360mA.	*	340mA.

19/8/17 Mon

PIN DIAGRAM OF 8086. (Figure)

1> Common pins.
2> Maximum mode pins.
3> Minimum mode pins. } 24-31.

→ $AD_0 - AD_{15}$. (Pin 16-2, 39)

These are multiplexed bidirectional address/data bus.

Active low signal (signal) 1 0

Active high signal.

$AD_0 - AD_{15}$ → Active high signal.

→ A_{19}/S_6, A_{18}/S_5, A_{17}/S_4, A_{16}/S_8.

These lines are multiplexed unidirectional address & status bus.

· S_3 S_4. (denotes the segment currently used).

S_3	S_4	
0	0	
0	1	
1	0	
1	1	

S_5 → status of interrupt flag.

→ BHE/S_7.

Pin. 34 Output.

BHE → Bus High Enable.

BHE used to denote transfer of higher order data bus.

→ $\overline{RD}$ (Read)

Output.

It is used for read operation.

→ READY (Pin 22 Input)

This is an acknowledgement signal from slower I/O devices or memory.

→ RESET (Pin 21 Input)

It is a s/m reset

The s/m is reset to FFFF0 address. (OS is stored in this location)

When high, microprocessor enters into reset state

→ INTR.

To generate interrupt signal. For low priority (we can refuse this)

→ NMI Pin 17 Input.

It is a non maskable interrupt signal. (for high priority) we cannot ignored

→ TEST Pin 23 Input.

If low, execution continues else microprocessor is in wait state.

The status of maths co-processor.

→ MN/$\overline{MX}$

MN – 1 min mode.
$\overline{MX}$ – 0.

Minimum mode

→ $\overline{INTA}$ (Pin 24 Output)

Interrupt Acknowledge Signal

→ ALE (Pin 25 output)

Address Latch Enable signal.

It indicates that valid address is available on bus.

$AD_0 - AD_{15}$.

→ $\overline{DEN}$ (Pin 26 output)

This is Data Enable Signal.

It is an active low signal.

Demultiplexing

→ DT/$\overline{R}$ (Pin 27 Output)

This is a data transmit / receive signal.

It decides the direction of flow of data.

→ M/$\overline{IO}$

To check m/m op^n or I/O.

→ $\overline{WR}$ To enable write op^n

→ HOLD

for DMA processor.

→ HLDA.

Max mode pins

→ QS_1 and QS_0 - Status of instruction queue.

→ $\overline{S_0}$ $\overline{S_1}$ $\overline{S_2}$

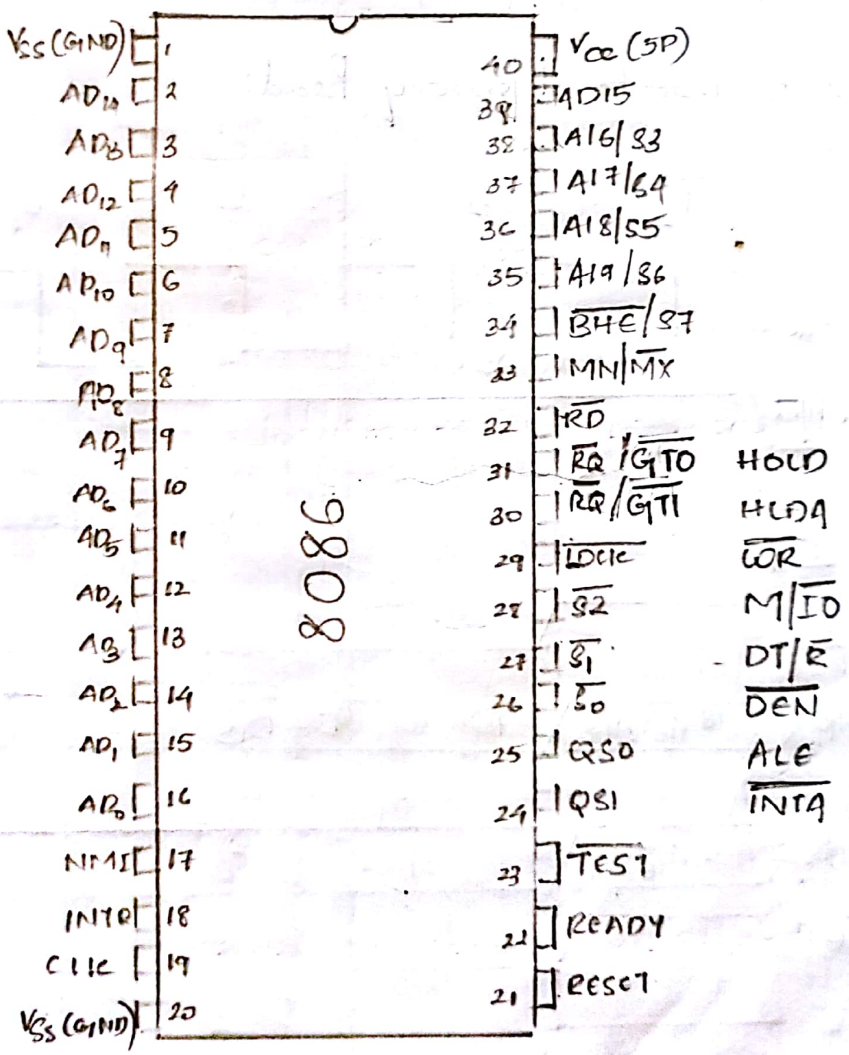
Status signals given to bus controller.

→ $\overline{LOCK}$

This signal indicates that other processor should not ask the CPU. (to lock the bus.

→ $\overline{RQ}/\overline{GT_1}$ and $\overline{RQ}/\overline{GT_0}$

Bus request and grant signal.



16/8/17 Wed

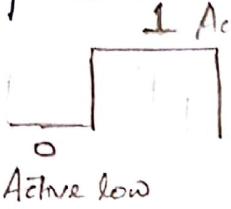
Timing Diagram

It is a diagram representing enabling & disabling of various signals.

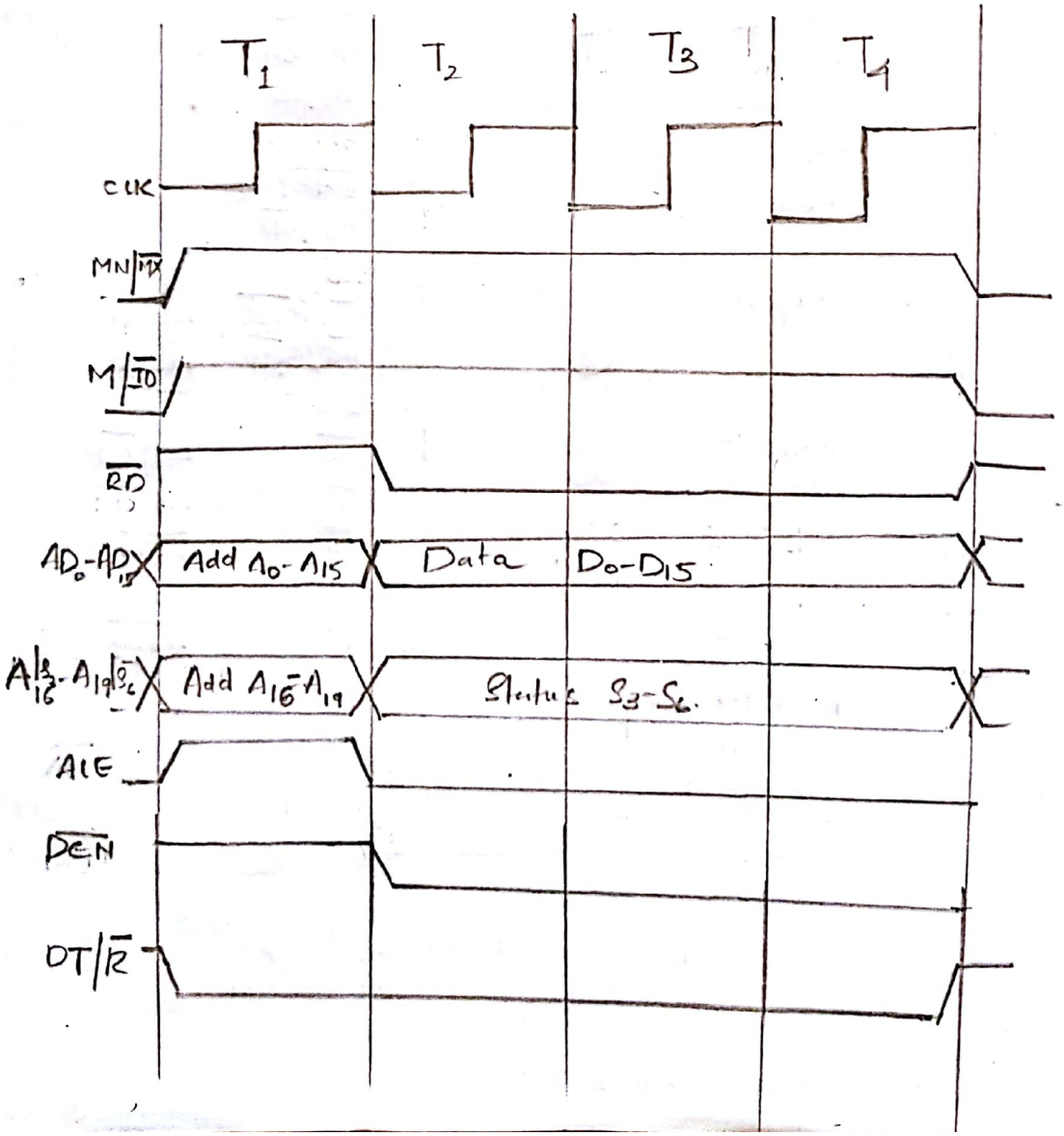
- Minimum Mode / Max.
- Memory / IO.
- Write / Read.

} 1 clock cycle required

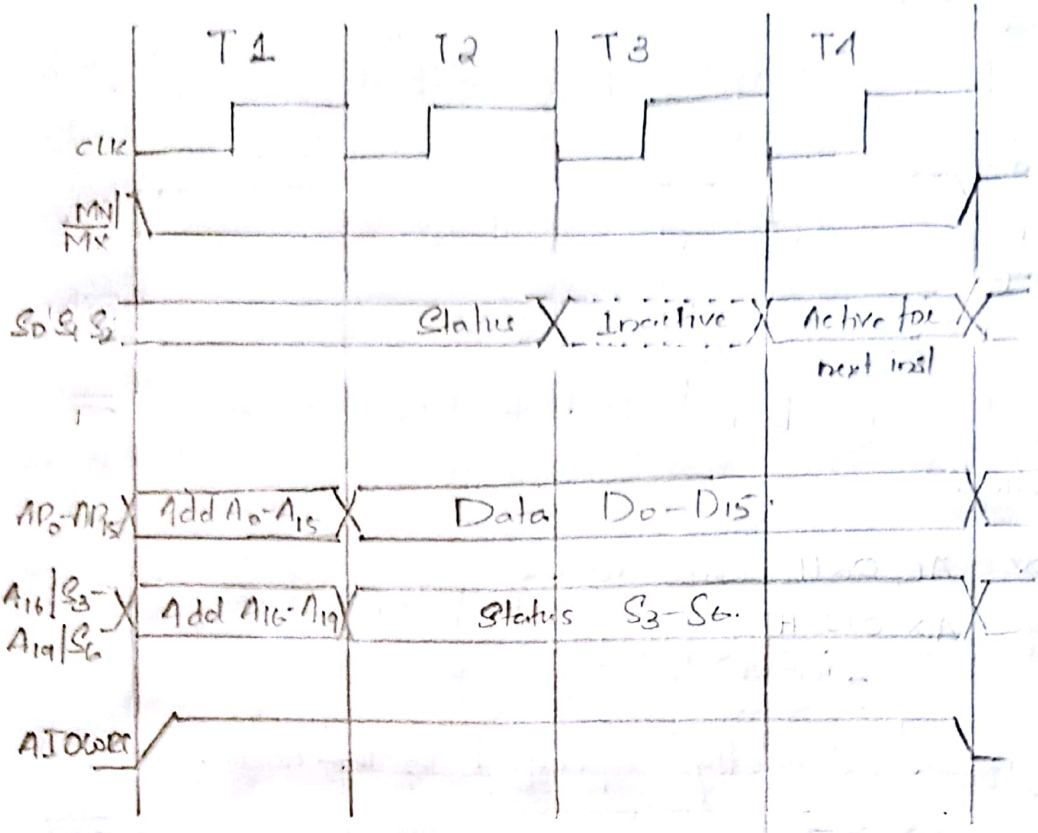
4 clock cycles (4 T States) required.



→ Minimum mode Memory Read:



→ Minimum mode IO write:

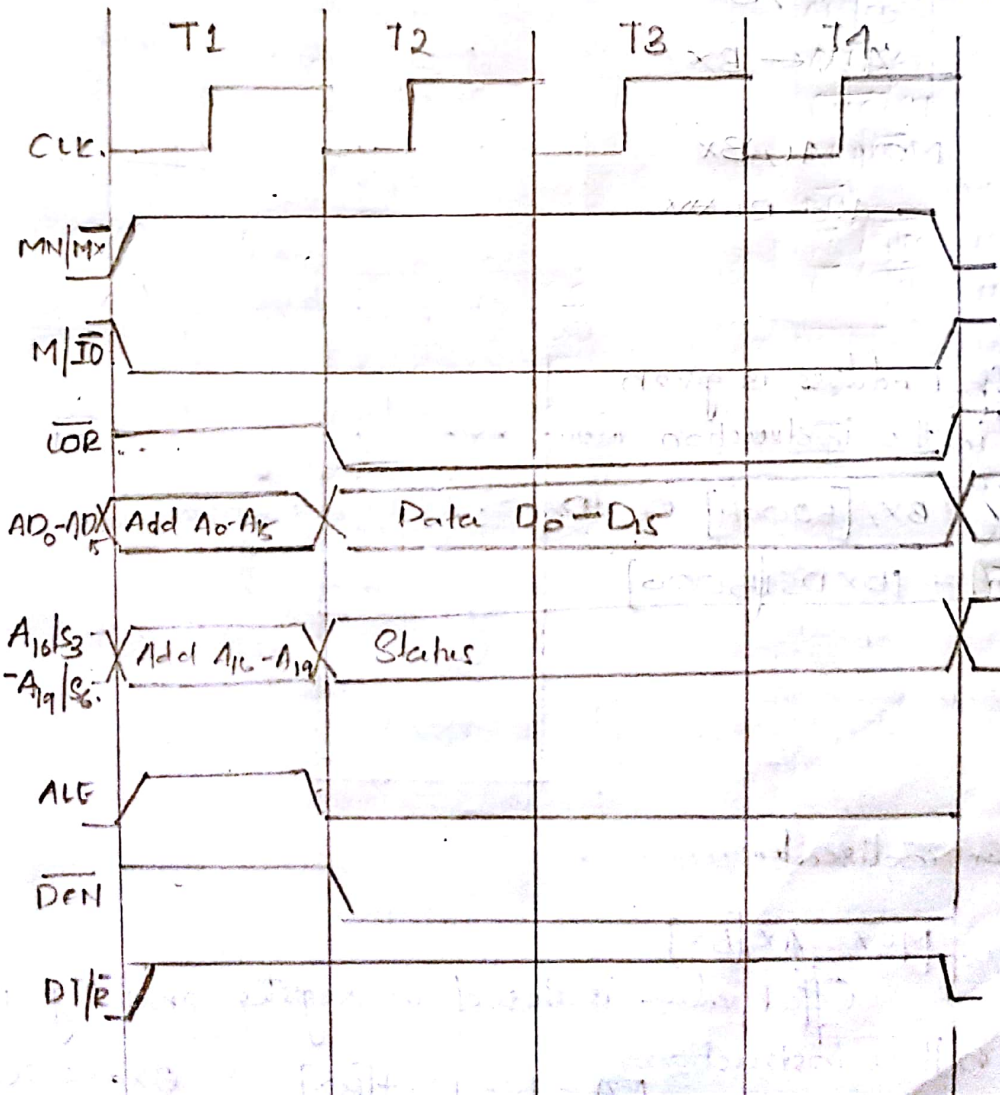


In max mode 8086 generate only status signals

Status signals are activated during T4 of previous inst and active till T1 of this inst is active

To distinguish m/m read write I/o read write in max mode bus controller generates 4 signals:
- AMRC
- AMWRC
- AIORC
- AIOWRC (Advanced-A)

Minimum mode IO Write.



Write - transmit DT
Read - receive R

21/8/17 ADDRESSING MODES: (T

→ Sequential.

1. Implied:

HALT.
Reset
CLR.

Opecand is not directly specified in the instruction

2. Immediate.

MOV AL, 02H → 8 bit AL←02. (02)

MOV AX, 0102H → 16 bit

AH ← 01
AL ← 02.

Opecand is directly specified in the inste.

3. Registee.

MOV AX, BX

AX ← BX.

MOV AL, BX

AL ← BL of BX

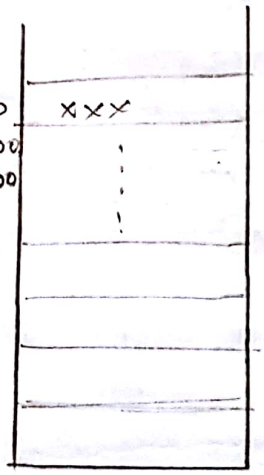
5/17 4. Direct

Offset address is given directly in the instruction.

MOV BX, [5000H]

EA = 10× DS + [5000]

600.
CA



5. Register indirect.

MOV AX [BX]

Offset value is stored in registee and seg is specified in thes instruction.

EA = 10* DS + [BX] BX=5000H

For storing offset address we can use 3 registers in indirect addressing.

Three reg: BX | SI | DI

6. Indexed

MOV AX, [SI]

SI | DI can be used.

Mainly used for string operation.

If SI is used DS is used to get data segment.

If DI used, ES (Extra segment) is used).

EA = 10*DS + SI

10*ES+DI

7. Relative Based.

MOV AX, 50H [BX]
BP

EA = 10*DS + [BX] | [BP] + 50.

8. Based Indexed

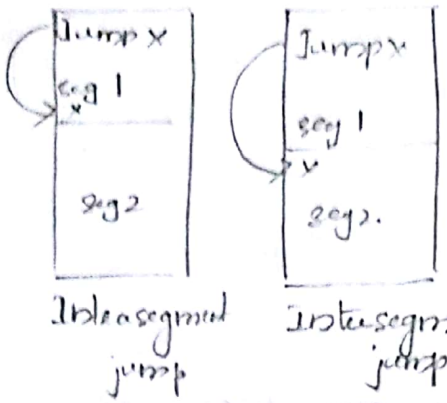
MOV AX, [BX] [SI]
BP DI

EA = 10* DS + [BX | BP] + [SI | DI]

9. Relative Based Indexed

EA = 10*DS + 20H + [BX | BP] + [SI | DI].

→ Non sequential



Intra segment jump — Inter segment jump

10. Relative.

→ Jump 500H. IP-Instruction pointer

EA = 10 × CS + [IP] + 500.

CS = Code segment. — Jumping to another instruction

Problem

1)

CS: 7000
DS: 8000
SI: 0050
DI: 0040
BX: 0052
IP: 0000

12	70050
11	
10	
01	70000
09	80060
:	
02	80052
03	80051
04	80050
:	
05	80001
06	80000

1) MOV AX, BX

Register mode.

AX ← 0052

2) MOV AX, [0001]

= 10 × DS + 0001

= 10 × 8000 + 0001

= [80001]

AX ← 05.

3> MOV AX, [BX] Register indirect

EA = 10X DS + [BX]

= 10X8000 + [0052]

= [80052]

AX ← 02.

4> MOV AX, 08[BX] Rel

EA = 10X DS + [BX] + 08

= 80000 + ~~02+08~~ 0052+08

= 80000 + 0060

= [80060]

AX ← 09.

5> CALL 50H

Relative mode

EA = 10X CS + [IP] + 50

= 10X7000 + [0000] + 50

= 70050

6.

70020: Jump 05H.

70021:

70022:

;

;

CS: 7000H

[IP] 0021

EA = 10X CS + IP + 05

= 70000 + 0021 + 05

= 70026

23/8/17
Wed

Instruction Set of 8086.

opcode	operand	operand

8bit

RESET (8,1) - 1 byte instruction
MOV (8) AX,BX - "
MOV (8) AL,02H (8) - 2 byte
MOV (8) AX,0001H (16) - 3 byte

Classification of Instruction Set

1) Data Transfer Inst.
2) Arithmetic Inst
3) Logical Inst
4) Branch Inst
5) String Inst
6) Flag Manipulation
7) Processor control Inst.

1 Data transfer Instruction.

Transfers data from source to destination.

eg: MOV Des, Src:

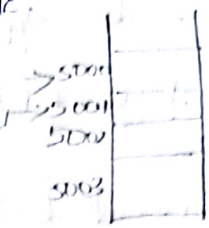
Src operand can be reg, mem location or immediate operand.

* MOV CX, 037AH CX ← 037A.
* MOV AL,BL AL ← [BL]
  MOV DS,5000H (wrong).

* PUSH operand:

It pushes the word into top of stack.

Eg: PUSH BX (First BH then BL).



* POP Des:

It pops operand from the top of the stack to Des.

Eg: POP AX

* XCHG Des, Sec.

→ The instruction exchanges Sec with Des.

→ It cannot exchange two m/m locations directly.

Eg: XCHG DX, AX

[AX] ⟺ [BX]

* XLAT

→ Equals MOV AL, [AL][BX]

AL ← 10* DS + [AL] + [BX]

* IN Accumulator, Port Address:

It read a data from port addres

Eg IN AX, 0028H : AL ← [0028], AH ← [0029]

IN AL, 0028H : AL ← [0028]

↓ Content of 002 is 8bit sin AX is 16bi To fill the rest 8 bit we take next port add

~~LEA~~

* LEA Register, Src: (Load Effective Address).

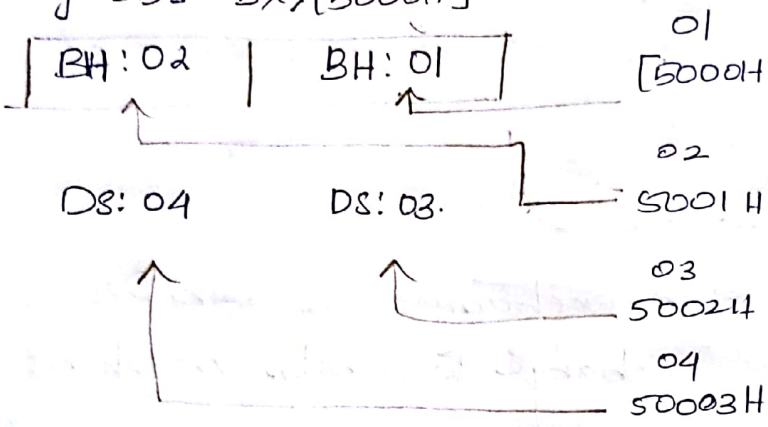
→ Moves the offset address specified by source to given reg.

Eg: LEA [BX], D1.

* LDS Load Data segment

→ Load DX and reg BX

Eg: LDS BX, [5000H].



* LES Des, Src.

→ DS replaced with extra segment

Eg. LES BX, [0301H]

* LAHF: (Load)

It copies the lower byte of flag reg to AH.

* SAHF: (Store)

It copies the contents of AH to lower byte of flag reg.

* PUSHF:

Pushes flag reg to top of stack.

* POPF:

Pops the stack top to flag reg.

2) Bit Manipulation / Logical Instructions:

* NOT Src

Complements each bit of Src to produce 1's complement.

NOT BL.

* AND Des, SRC. Logical AND b/w content of src & des.
* OR Des, SRC. " OR "
* XOR Des, Src " XOR. "
* TEST Des, Src

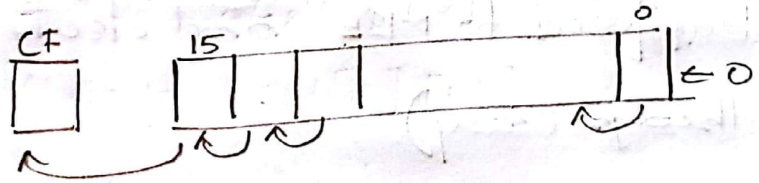
Perform bit by bit AND of operands, result is not stored anywhere.

Used to check flag status.

* SHL (Shift logical left) / SAL (Shift Arithmetic left).

SHL / SAL Des, Count.

SHL/SAL as specified in count.



SHL AL, 02 (Shift left two times) SHL, SAL (Equal)

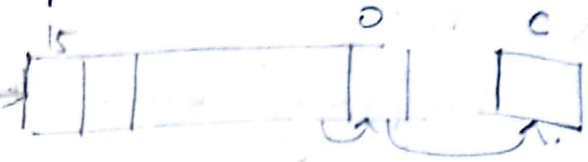
AL = CF
1 0 0 0 1
0 0 0 1 0 ←
0 0 0 0 0 ←

* SHR Des, Count

SHR, SAR (Not equal)

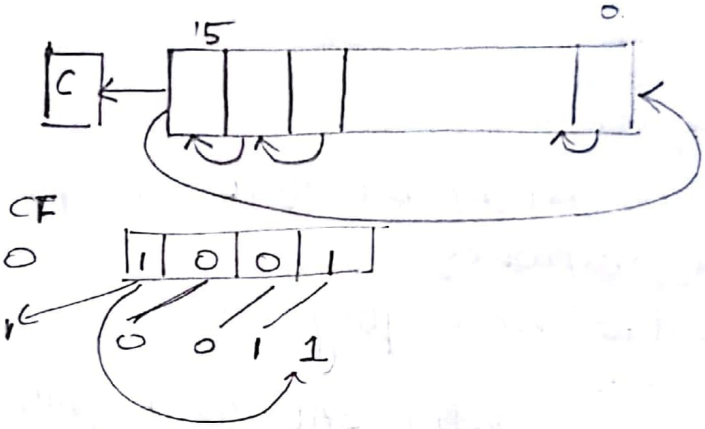
* SAR Des, Count

In SAR the LSB ~~MSB~~ is copied to the vacant space.

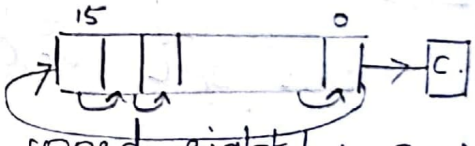


* ROL Des, Count (Without carry).

It rotates bits of byte or word left, by count MSB transferred to LSB and also to CF

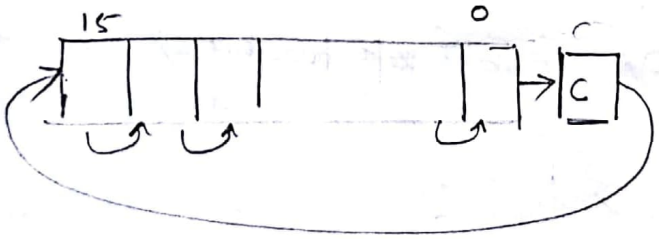


* ROR Des, Count (Without Carry).



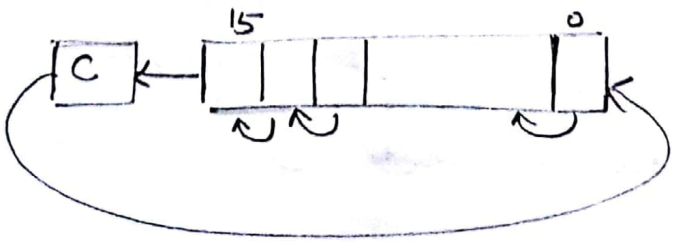
It rotates bits of byte or word right by count LSB transferred to MSB and also to CF

* RCR Des, Cnt (Through Carry).



~~Bears~~

* RCL Des, Cnt (through carry



3. Branch Instructions.

* CALL Des (call a function, subroutine)
* RET (return to.
* JMP Des:
* Jxx Des
* STC
  sets carry flag to 1
* CLC
  clear carry flag to 0.
* CMC
* STD
  Sets directional flag to 1.
* CLD
* WAIT
* HALT : Halt the processor
* NOP
* LOCK : lock the BUS.
* ESC

29/8/17 Tue

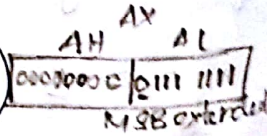
4. String Instructions.

* LODSB / LODSW
  Load AL/AX by the content of string pointed by DS:SI
* STOSB / STOSW
  Load string pointed by ES:DI with content of AL/AX
* CMPS Des, Src:
  It compares the string bytes or words.
* SCAS

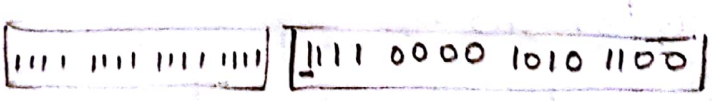
- MOVS/MOVSB/MOVSW

A

- ADD Des, Src
  Add two data
- ADC Des, Src
  Add two data and carry flag.
- SUB Des, Src
- SBB Des, Src
- INC Src
  Increment
- DEC Src
- CMP Des, Src
  Compares two values.
- MUL Src
  unsigned multi^n
- IMUL Src
  Signed mult^n.
- DIV Src
- IDIV Src
- CBW (Convert byte to word) (8bit → 16bit)



- CWD (Convert word to Double word.)
  (16 bit → 32 bit)



- AAA (ASCII Adjust after Addition):
  Data entered from terminal is in ASCII format.

In ASCII 0-9 are represented as 30H - 39H

1> ASCII unpacked. (15).

31	35

2> Unpacked BCD.

01	05

3> Packed BCD - 15

4> Hex - 0F.

Rules for ASCII Adjust after Addition: (use for: After arithmetic op^n the result is converted to same format as operands)

1> Clear AH

2> Move the value to be converted to AL register.

Case 1°.
If the lower nibble of AL (first digit of the number) less than 9 and Auxiliary carry flag ie (AF=0) then set the upper nibble of AL to 0.

Case 2°:
If the lower nibble of AL > 9 then add 06 to AL, set upper nibble of AL to 0. Increment AH.

Case 3°:
Lower nibble of AL < 9 & AF=1 then add 06 to AL, clear upper nibble of AL, increment AH.

Eg:
09+05 = (14

```
      (1)
  0000 1001
  0000 0101  +
  ---------
  0000 1110   → 0E → convert to 14.
```

0000 1110. 6>9.
0000 0110

0001 0100

[0000 0001] [0000 0100]

Eg2:
05+05 10. Case 2.

0000 0101
0000 0101 +

0000 1010 10>9
0000 0110 +

0001 0000

[0000 0001] [0000 0000]

Eg3:
08+08' Case 3

0000 1000
0000 1000

0001 0000 0<9.
0000 0110 +

0001 0110

[0000 0001] [0000 0110]

* DAS (Decimal Adjust After subtraction).

* DAA (Decimal Adjust after Addition).

It only works on AL reg. It is used to make sure that result of adding two BCD numbers is adjusted to be a correct BCD number.

23+45=68

0010 0011
0100 0101 +

0110 1000

0010 0101 + 25+
0010 0101 25
0100 1010 → 4A → convert to dec. 50.

* The value to be converted must be in AL
* If the lower nibble of AL > 9. add 06, then check the upper nibble of AL. If it is > 9 add 60 otherwise do nothing.

4 A
0100 1010 A > 9.
0000 0110 Add 06
0101 0000
5 0

Eg: 49+23.

0100 1001
0010 0011 +
0110 1100 (6C).

0110 1100
0000 0110 +
0111 0010
(72)

1. Assume that the capacity of AL register is infinite and the value in AL is 9C4A. Then what is the result after DAA instruction

9 C 4 A

```
            1      1 1
1001 1100   0100   1010
   0    0      0   0110  +
------------------------
111  1
1001 1100   0101   0000
   0 0110      0      0
------------------------
1  1
1010 0010   0101   0000
0110    0      0      0
------------------------
1 0000 0010 0101   0000
========================
```

2. For addition opeeation the usee entered two values 09 and 05 through keyboard then what could be the result in hex format unpacked BCD and packed decimal.

```
0011 1001
0011 0101
---------
```

39+
35

DA

20/8/17 Wed

8086 Programs — Tutorial

```
ASSUME CS:CODE DS:DATA SS:name
DATA SEGMENT.
    Data DB 01H.
    Msg DB "progee-mmsg"
    Xxx MACRO paramelee
    ....
    ENDM
DATA ENDS.
CODE SEGMENT.
Start: move AX, data
    Mov ds, ax
    Yyy PROC paeam
    ....
    ENDP
CODE ENDS
END
```

data, variables, strings, macros.

cs - code segment
DS: Data segment
SS: Stack "

In Assembly level pgrm
* Assembler directives
* Instructions are present.

```
Data DB 01H.
    Data=1.
Data, DB Dup (?).
msg DB "Hello$"
```

1. Write an assembly level progeem to print string hello world

prog

```
ASSUME CS:CODE DS:DATA.
DATA SEGMENT.
    MESSAGE DB "HELLO WORLD!!!$"
ENDS
CODE SEGMENT.
START: MOV AX,DATA.
    MOV DS,AX
    LEA DX,MESSAGE
    MOV AH,09H
    INT 21H
    MOV AH, 4CH
    INT 21H
CODE ENDS
END START.
```

To print something the value must be in DX.
DX - o/p reg.
AL → 09h - print string.
→ 01h - Read
→ 02h - ~~Write~~ print data.

2> COAP to print a string using MACRO.

```
ASSUME CS:CODE DS:DATA
DATA SEGMENT
   MESSAGE DB "HELLO WORLD!!!$"
   display MACRO xxx
   LEA DX, xxx
   MOV AH, 09H
   INT 21H
   ENDM.
ENDS
CODE SEGMENT.
START: MOV AX, DATA.
       MOV DS, AX.
       display MESSAGE.
       MOV AH, 4CH
       INT 21H.
   CODE ENDS.
  END& START.
```

MACRO -> DS
PROC -> CS.

3> COAP to print a display string with subroutine

```
ASSUME CS:CODE DS:DATA.
DATA SEGMENT.
    MSG DB "HELLO WORLD$"
ENDS
CODE SEGMENT
START: MOV AX, DATA
       MOV DS, AX.
       DISPLAY PROC YYY
       (Push all reg)
       LEA DX, YYY
       MOV AH, 09H
       INT 21H.
RET
ENDP.
ENDS
    CALL display MSG
```

```
    MOV AH,4CH
    INT 21H
CODE ENDS
END START
```

4) WAP to add two immediate values (pgm to add 02, 03)

no data seg

```
ASSUME CS:CODE
CODE SEGMENT
START: MOV BL,02
       MOV CL,03
       ADD BL,CL
CODE ENDS
END START.
```

mov bl, 02
add bl, 03.

5) WAP to add two numbers, located at offset address 5000 and 6000, and store the result to offset address 7000.

no ds

```
ASSUME CS:CODE
CODE SEGMENT.
START: MOV BX,[5000]
       ADD BX,[6000]
       MOV [7000],BX.
CODENDS
END START.
```

6) WAP to move 10 numbers located at offset address 5000 to 5009 to another offset 6000 to 6009 respectively.

```
ASSUME CS:CODE
CODE SEGMENT.
START: MOV CL,0Ah
       mov si,5000
       mov di,6000
XX:    mov BL,[SI]
```

```
mov[di],bl

INC SI
INC DI
DEC CL
JNZ XX

CODE ENDS
END START.
```

For immediate use use SI & DI

7) Write a program to find 1+2+3+....+10.

```
ASSUME CS:CODE
START: MOV CL,0AH
       mov bl,00h
XX:    ADD BL,CL
       DEC CL
       JNZ XX.
CODE ENDS
END START
```

```
mov bl,0ah
mov cl,01h
xx: add bl,cl.
    inc cl
    cmp cl,0ah
    JNZ xx.
```

8) WAP to check whether initialized variable number is even or odd.

even - LSB-0
odd - LSB-1

Use shift right.

```
ASSUME CS:CODE DS:DATA.
DATA SEGMENT.
     odd DB "Number is odd$"
     NUM DB 05H.
     even DB "Number is even$"
ENDS.
CODE SEGMENT.
START MOV AX,DATA.
      MOV DS,AX
      MOV AL,NUM.
      SHR
      JNC XX
      display odd
      JMP Y
XX: display even
Y: End.
```

9. ALP to count the number of 1s in an 8 bit number.

```
ASSUME  CS:CODE  DS:DATA
DATA SEGMENT
        NUM  DB  02H
        COUNT DB  00H
ENDS
CODE SEGMENT.
START: MOV  AX, DATA
       MOV  DS, AX
       MOV  CL, 08
       MOV  BL, COUNT.
       MOV  AL, NUM.
   YY: SHR.
       DEC  CL.
       JNC  XX
       ADD  BL, 01.
   XX: JNZ  YY.
CODE ENDS.
END START.
```

S5 CS B (2nd hour).

Meera (30/8/17) : Meera. M

30/8/17 : Dhanya Pm.

11/9/17 Mon

10) 3 processes connected with 8086 multiprocessor (max mode) in a master-slave configuration. In this configuration 8086 processor transmitting a data to 2050H (m/m location of another processor).

(i) Identify the mode of operation of 8086.

(ii) Identify the type of operation.

(iii) Draw the timing diagram.

Ans.

(i) Maximum mode m/m write.

(ii) m/m write

11. (iii) Write an assembly program to find factorial of a number

Prog
```
ASSUME CS:CODE
CODE SEGMENT.
Start: mov al,04h
       mov cl,01h
       mov bl,01h
LOOP: MUL bl,cl
      INC cl
      CMP cl,al          ~~to find factorial of a number~~
      JLE LOOP
      mov dl,bl  } //print bl
      mov al,02H }
      INT 21H
CODE ENDS
END
```

12. Write a program to print fibonacci series (first 6 fibonacci numbers)

0 1 1 2 3 5 8

bl cl

1 = 0+1

2 = 1+1

3 = 2+1

5 = 3+2

8 = 5+3

Prog
```
ASSUME CS: CODE.
CODE SEGMENT
START: mov al, 04H
       mov bl, 00H.
       mov cl, 01H.
       print bl
       print cl
LOOP: ADD bl,cl
      MOV dl, bl
      MOV bl,cl
      print bl
      DEC al.
      JNZ LOOP.
CODE ENDS
END START.
```