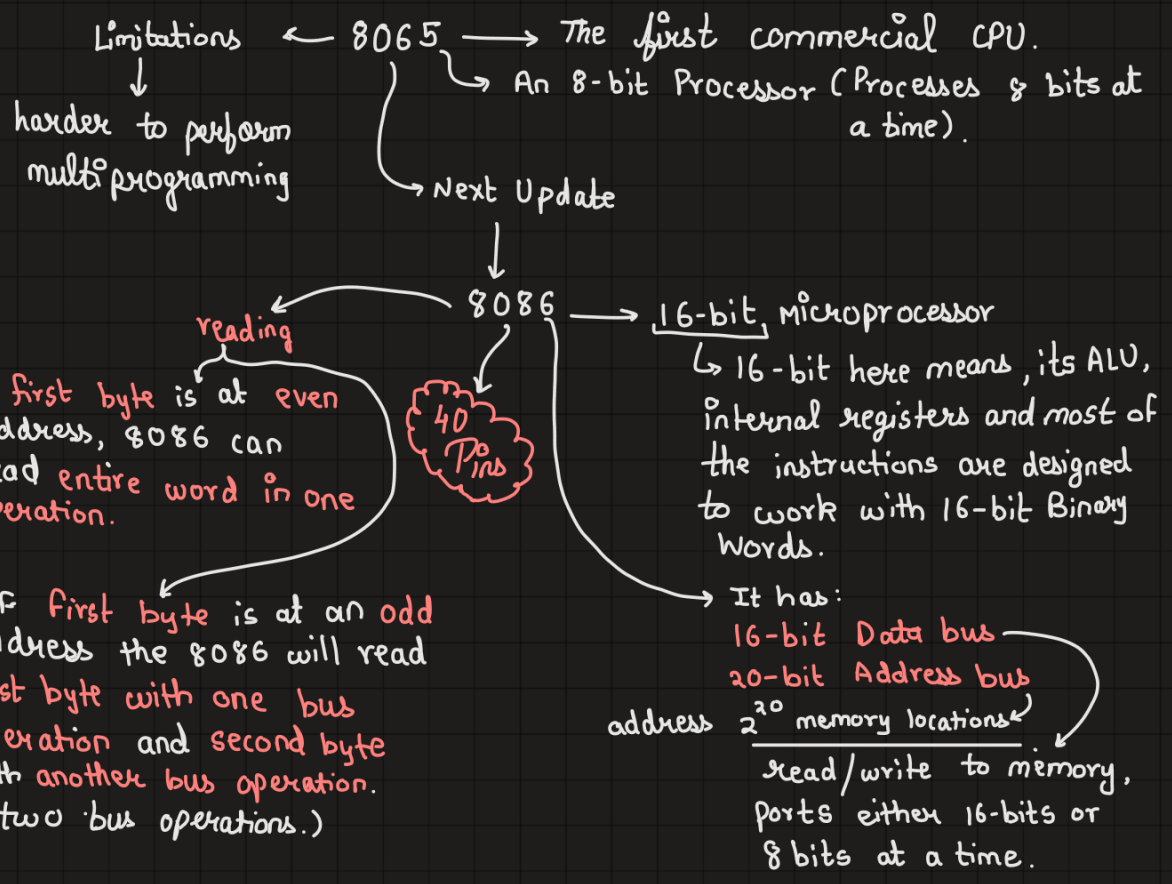
27/02/2023

Microprocessor Techniques

Microprocessor is nothing but a single chip CPU.



Class Notes Comments

In the real address board of 8086 functioning of a processor is
equal to that of a 8-bit processor which assumes only one process executes at a time.
It is the virtual address of 8086 where multiprogramming is done

Data and programs are supplied to CPU from Primary Memory
↓
aka → semi-conductor memory.

Prerequisite for subject

i) Tri-state

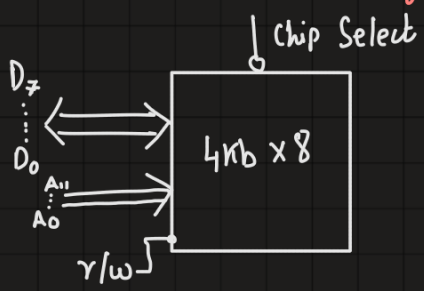
ii) Primary Memory → Address Memory
→ Read & write Cycle
→ Memory access time

Future Problem Question to be Answered

A 16 bit processor being interfaced to an 8-bit memory

16 bit Processor

40-pins
16-bits of Data lines
- These data lines are clubbed together and data bus is created.



"Everything Memory does and should do is for the Processor."

Address line Difference

CPU's address lines are output lines, all others are input lines.

CPU → Synchronous circuit, Combination of Combinational and sequential circuit.

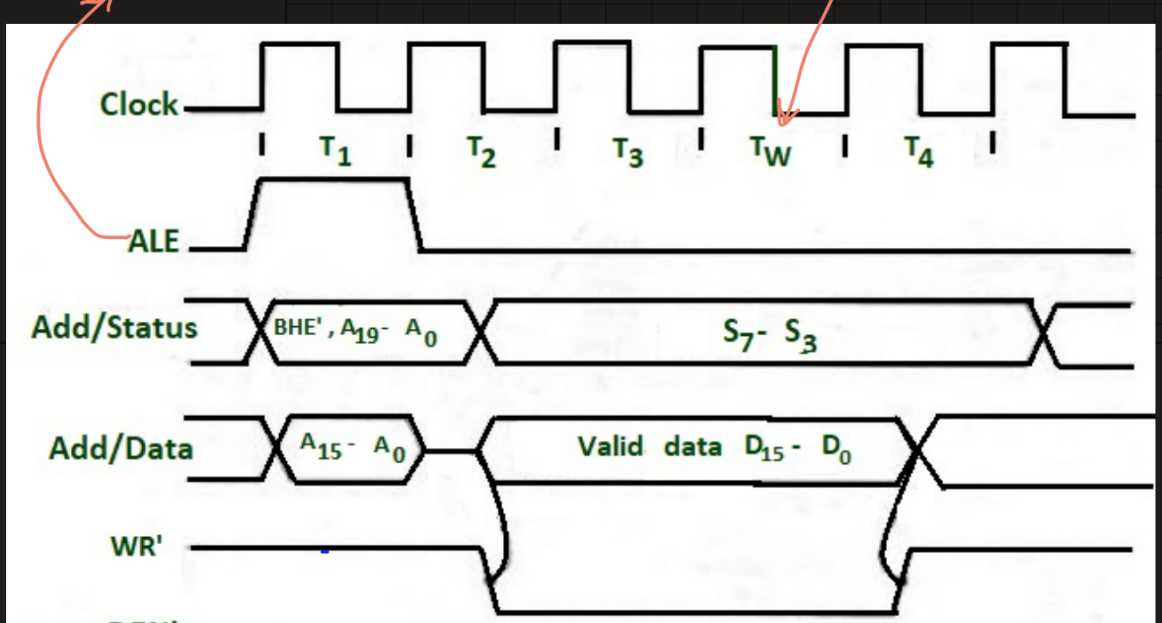
ALU + CU
- ALU → Arithmetic Logical Unit
- CU → Control Unit

µp → Single Chip Processor.
↑
Micro processor

Address Latch Enable.

Wait State

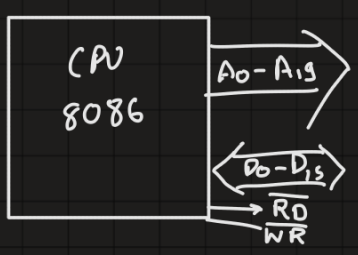
02/03/23



02/03/23

BHE → Bus High Enable
ALE → Address Latch Enable

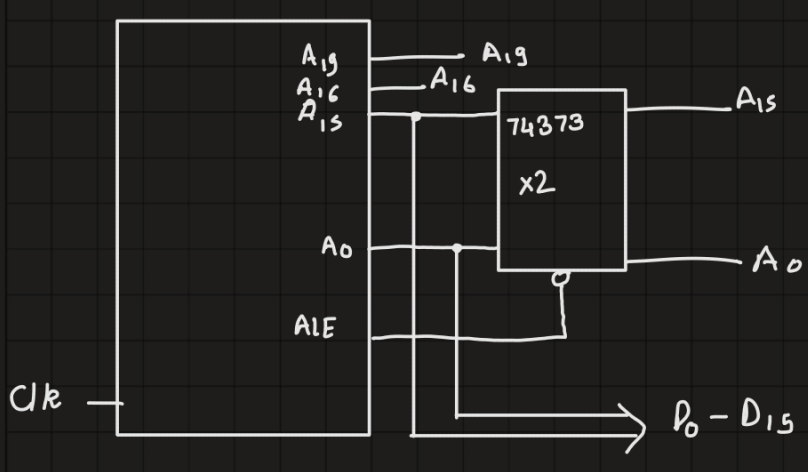
Address lines are clubbed together with data lines. (Multiplexed)
↓
double use.



← Conceptual CPU diagram

CPU's address lines are always o/p lines.
Control lines —11—

74373 is a simple latch.

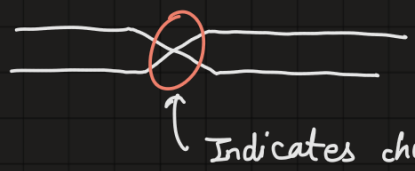


- 5 MHz Max speed.
- The clock pulses are given by external device 8284 Clock generator.

5 MHz
Max SP

Waveform diagram:

clock pulses are → 33% on, 66% off



Indicates change in signal.

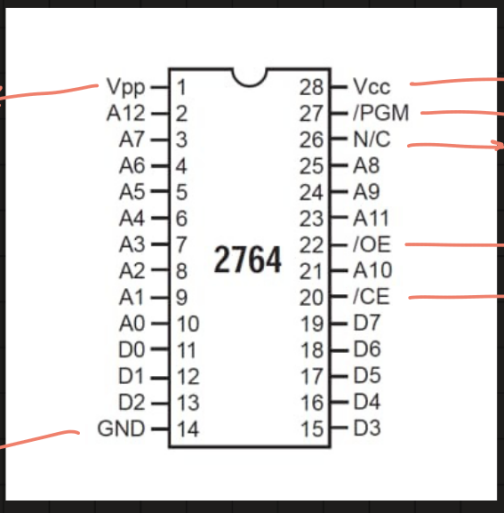
- - - - - - - - - - ← Indicates tristate signal.

Exclusive means non-multiplexed. $A_{16} - A_{19}$ are exclusive

T_W :

Sometimes, reading a memory (1 byte) doesn't get completed in 4 T-states.

T_W indicates wait state. (Not necessarily 1T)



→ Who gives the T-states? Who gives clock pulses.

For worksheet:

requirements:

2 x 2764

2x 6264

During bus cycle if reading 2bytes of data, 2 locations accessed consequtively.

Instruction byte	Data bytes	
1D	32	14
+	12	24

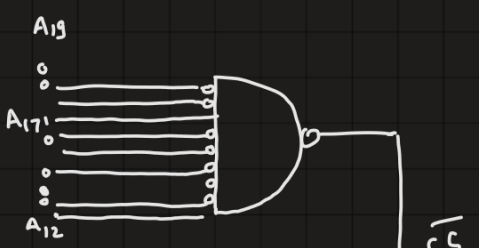
1D 32 16

Instruction

Interface 4 kb ram to 8086, assuming address lines are demultiplexed. Starting address of RAM is 20000H

Address Map:

20000H	0010	0000	0000 0000 0000
20FFFH	001	0000	



→ Active Low when i/p == 0 it works

For 16 bit 2 x 4Kbyte reading

BHE	A0	
0	0	Read Word (16 bits)
0	1	Read byte at Even address
1	0	Read Byte at Odd Address
1	1	Invalid

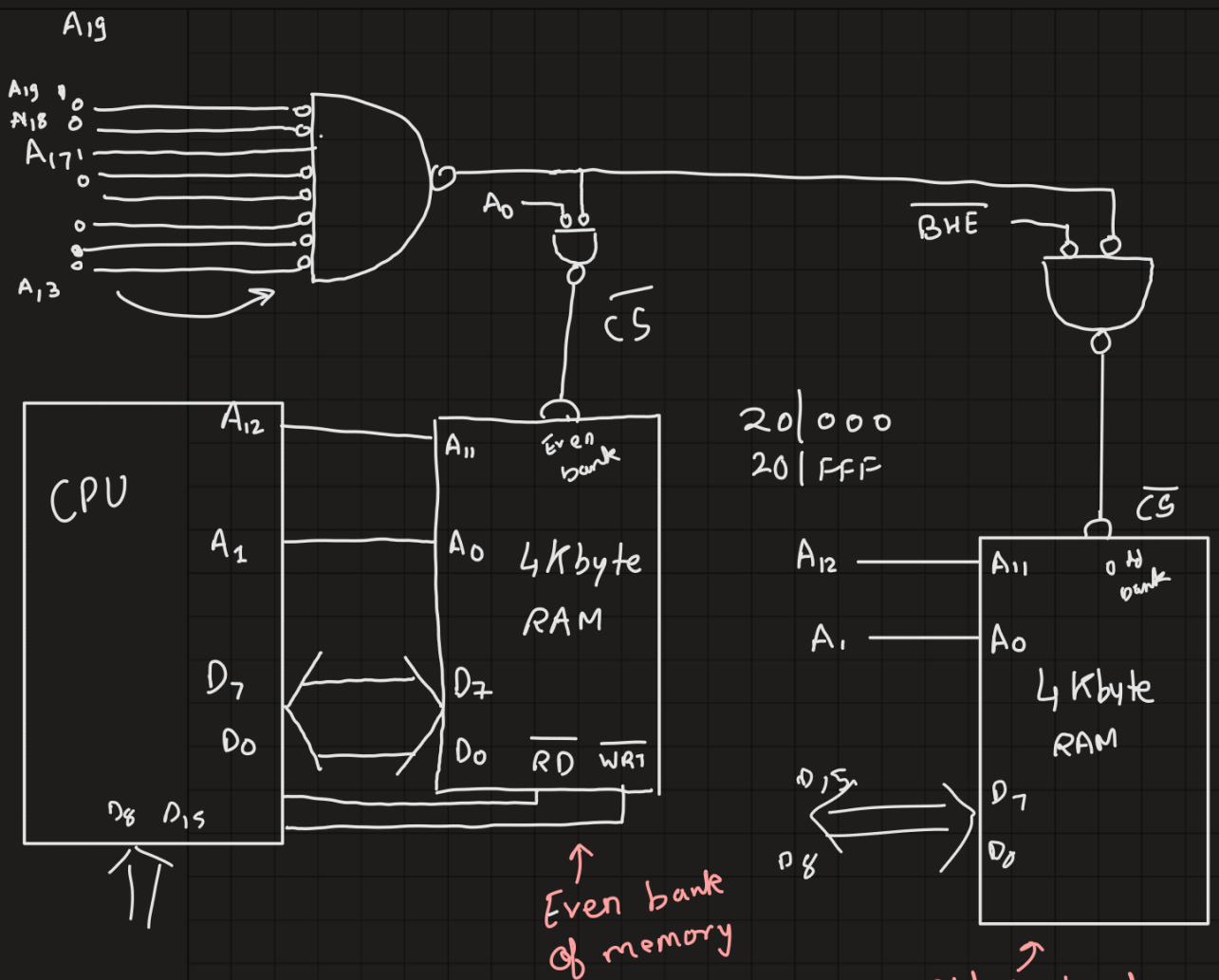
Instruction example:

MOV AX, <1000H>

16 bit register with AH → and → AL

AH ↓ Higher D8-15

AL ↓ D0-D7



Processor makes $\overline{BHE}$ 0 in order to read 16 bits.
→ BHE is a o/p signal of CPU & i/p signal to gating logic of memory

" Because processor knows its going its reading 16 bits, the contents on bus cycle are increased by 2."

- - Chip selected - - -

BHE	A_0	Odd Bank	Even bank	
0	0	✓	✓	read 1 word
0	1	✓	×	1 byte from odd bank.
1	0	×	✓	1 byte from even bank
1	1	×	×	Invalid.

Odd bank → All odd addresses fetched from that memory
Even bank → -11- Even address -11-

1st T-state - ALE goes high, address is latched
2nd T-state - rd, $\overline{wr}$ activated.
3rd T-state - Time required to access memory
4th T-state - we wait for $\overline{RD}$ to go trailing (falling) edge.

* First bus cycle starts when CPU is started.
→ first bus cycle is read pulse, to read a program from memory.

Programming Model of the CPU:

A set of named registers with some intricacy on inside which makes an efficient CPU.

→ Register that are named are altogether called programming model.

Register

General purpose
- Ax
- Bx
- Cx
- Dx

→ 16 bit register
also accessible as 8 bit registers
AL AH
BL BH ...etc.

L H
↓
low → high

② Segment registers:

CS	Code
DS	Data
ES	Extra data Segment
SS	Stack Segment

* 8284 Clock Generator:
Generates Clock Pulses.

Program Status Word

what is PSW?
what is semaphores?.

SI	Source Index
DI	Destination Index
SP	Stack Pointer
DP	Base Pointer
IP	Instruction Pointer
Flags/PSW	

- Anything available on IP it is dumped on bus cycle.

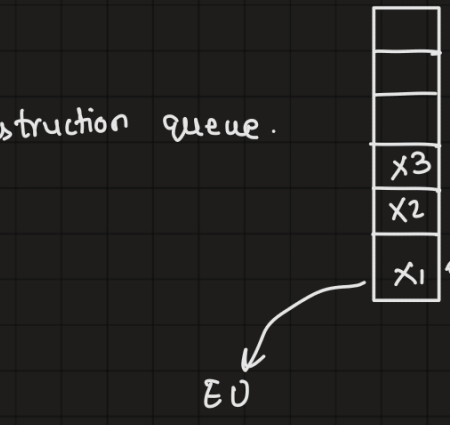
8086 → BIU (Bus Interface Unit)
EU (Execution Unit)

** Segment registers makes multiprocessing possible.

BIU → fetch from memory i.e. Job is to run bus cycles

6 byte instruction queue

EU → fetches from instruction queue.



IP points to next instruction
All registers are 16 bit.

Conversion of 16 bit address to 20 bit Address:
CS x 16 + IP.
Anything binary multiplied by 16 == Shifting left by 4 bits.

denoted using CS:IP ⇔ CSx16 + IP

1000 x16 + 0
10000 + 0

Relocatable program module is a feature of multiprogrammed CPU.

Direct Addressing Mode:

Direct Addressing,
Indirect Addressing,
Indexed Addressing,
Based Addressing,

MOV [1000], BX direct, as we are specifying the address of operand.
MOV AX, BX
MOV [BX], BX Indirect

Find bus cycles required, assume:
- opcode is single byte

How much size will the instruction take to execute.

let's say BX = 1000
MOV [BX] BX would move data of BX to address [BX] i.e. 1000.

1st Bus cycle : OP Code fetch happens
2nd -11- : T_1 : DS : BX ← Value of BX comes on bus
T_2 : $\overline{WR}$ ← write bar signal activation
T_3 : Access memory (Time taken to access the memory).
T_4 : Processed to memory state. (Data is moved to memory location).

Default Segment Assignment
→ Whenever BP register is used, Stack Segment
→ For all memory locations segment assignment is Data Segment, except BP.

Segment Override Prefix
→ DS: MOV AX, [BP] ← Overriding

Indexed Addressing

```
MOV AX, [SI]
MOV AX, [SI + 5]
```

Based Addressing

```
MOV AX, [BX]
```

String Addressing Mode

- Most instructions are single byte
- defines source string & destination string.

Source string → Defined by → DS: SI → Data Segment

Destination string → Defined by → ES: DI → Extra Segment

MOVSB

↑ Move single byte

→ Reads source string & transfer to destination.

MOVSW ← word

Prefix: → Together 2 bytes

REP : MOVSB

↑ Repeat prefix does moving job for CX number of times

REP : MOVSB

SI = 3000 ES=0200 CX = 03FF

DI = 1000 DS = 0300

REP : MOVSB DS:SI ES: DI

DS:SI → offset

↳ converts to physical address

0300 x 16 + 3000

3000 + 3000

6000

DF – Directional Flag.

1 → read string from end to start

Error of Bus contention:

→ when two masters try to write on same Bus.

(BUS master is CPU).

DMA → Program driven data transfer.
transfer

Addressing Modes in 8086

1) Immediate Addressing Mode:
   → Data operand is part of instruction itself.
   e.g. MOV CX, 4929H
   ↑ directly giving data instead of memory location

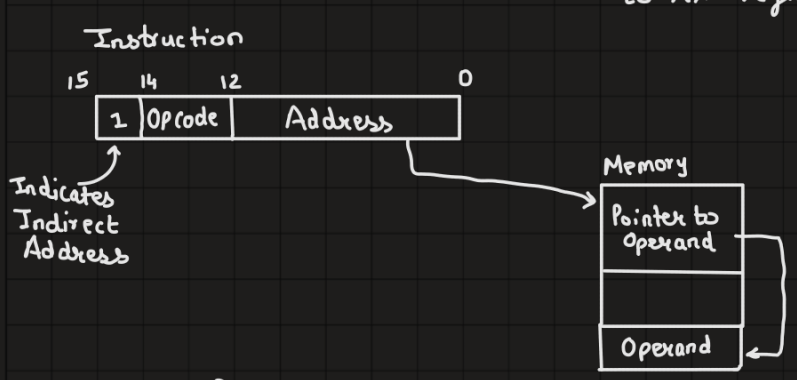
2) Register Addressing Mode:
   → Register is source of operand.
   MOV CX, AX
   ↳ Copies contents of 16-bit AX register to 16-bit CX register.

3) Direct Addressing Mode:
   → Effective address of memory is written directly in instruction.

   MOV AX, [1592H]
   ↑ Effective Address

4) Indirect Addressing Mode:
   → Allows data to be addressed at any memory location through an offset held in any of the following:
   1) BP 2) BX 3) DI and 4) SI

   MOV AX, [BX] # Moving the contents from the address' contents of BX' to AX register.



Indirect Addressing

5) Based Addressing Mode:

Offset Address of operand is given by sum of BX/BP contents register and 8/16 bit displacement.

```
MOV  DX , [BX +04]
```

BX → Base Register
BP → Base Pointer

6) Indexed Addressing Mode:

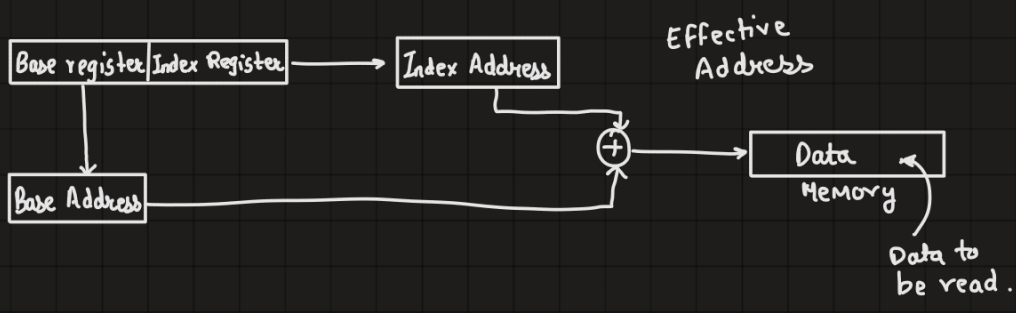
→ Operand Address found by adding contents of source index/destination index register and 8 bit/ 16 bit displacement.

```
MOV  BX, [SI + 16 ]
```

7) Based Indexed Addressing Mode:

→ Offset address computed by adding base register to content of indexed register.

```
MOV  CX, [ AX + SI ]
```



Based Indexed with Displacement Addressing Mode:

→ Operand offset computed by adding contents of Based register, indexed register, and 8/16 bit displacement.

```
MOV  AX  [BX + DI + 08]
```

16/03/23

Flags of 8086

x - Don't care bit.

x	x	x	x	OF	DF	IF	TF	SF	ZF	x	AF	x	PF	x	CF

} Flag register

PSW - Processor Status Word
↳ Flags
↳ Purpose → To indicate a signal

* Every bit of flag register has a signal value.

→ Parity Flag
1 → No. of bits = Even
0 → -11- Odd

AF - For 8 bit addition, 4→5 bit carry
For 16 bit address, 8→9 bit carry

CF - For 16→17 bit in 16 bit register addition carry.

SF - In last arithmetic operation, if its signed arithmetic was used if MSB is one SF is set to 1.

(1)0101 → -5
↳ SF ← Signed flag will be set to 1 here.

ZF - Zero Flag

OF - Overflow Flag ← If operand (resulting) exceeds number space

DF - Directional Flag (string addressing)

TF - When TF is made 1, (both are kind of interrupt registers bro!)
↳ Trap Flag

IF - Interrupt Flag
↳ hardware interrupt also

• Single step is a part of interrupt process
↳ Instructions are executed one at a time.

how to make TF = 1
↳ Trap Flag

Single stepping → processor initiated interrupt (under user control?)

* Processor has hardware interrupts, it has an interrupt pin.

✳ Everything under programming module is present inside processor.

Instruction: JMP

JMP: IP new, code segment same.

JMP 1000H

Program control transferred to 1000H, nxp IP will be 1000H.

↓ Instruction Pointer

JMP 2000 : 1000H

↑ CS ↑ IP

Contents of CS & IP both changed, thus a long jump/ far jump.

* JMP is a unconditional jump

Conditional Jump: if zero flag is set to true.

(Zero Jump) JZ +09H ⟹ If zero is

JNZ ← No zero

JC → Carry

JNC → No Carry

Q Write a program for transferring contents of a series of 8 bit Numbers. The source series is stored from 1000:1000 destination series stored from 1000:2000
Length of series is stored at 1000:4000 and 1000:4001

; → comments } → comments

```
MOV   AX, 1000H ; } Initialize data segment, ∵ we're accessing
MOV   DS, AX    ; } data segment as SI, BX, CX need directly Data Segment
MOV   SI, 1000H ; } SI pointer for source series
MOV   BX, 2000H ; } BX pointer for destination series
MOV   CX, [4000H] ; } Count in CX register (No. of bytes to be transferred)

BACK: MOV AL,[SI] ; } Get number in AL from source series (∵ 8 bit reading)
      MOV [BX], AL : } Transfer to destination (AH,AL) for AX register
      INC SI ;
      INC BX ; } Increment pointers to next no.
      DEC CX ; } decrement count.
      JNZ BACK ; } Test for zero, repeat if non-zero.
RET ;
```

Q. WAP to find number of negative elements in the given 8-bit series

```
MOV AX, 1000H
MOV DS, AX
MOV SI, 100DH
MOV BX, [4000H]
MOV CX, 0

BACK:
      MOV AL [SI]
      ADD AL, 00
      JNS AHEAD;
        INC CX
      AHEAD:
      INC SI;
      DEC BX;
      JNZ BACK;
ret;
```

odd

Q. WAP to Find number of elements in a 16 bit cell series. Series starts from 1000:0000. Number of elements in the series are stored in CX register. Store answer in 1000:1000 and 1000:1001

```
MOV  AX, 1000H  ; } DS Initialization
MOV  DS, AX     ; }
MOV  SI, 0000   ;
MOV  BX, 0H

REDO:
     AND  SI, 1 ;
     JNZ SKIPPED
       INC BX
     SKIPPED: INC SI   } INC twice because reading 16-bit number.
              INC SI
INC SI       DEC  CX
             JNZ  REDO

MOV  [1000], BX
```

Alternate Approach:

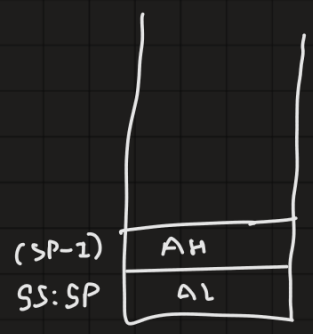
```
MOV AX,1000
MOV DS, AX
MOV BX,0000
MOV DX,0000

BACK:
    MOV AX,[BX]
    AND AX, 0002H
    JNZ NEXT
    INC DX
NEXT:
    INC BX
    INC BX
    DCX CX
    JNZ BACK
```

} Incrementing twice because 16 bit no.

DCX CX ——→ ?

STACK:

	AH	AL
PUSH AX	SP − 1	SP
POP AX	SP+1 ↓ AH	SP+2 ↓ AL



1000:0000

CALL 3000H ← Calling sub-routine from memory 3000

- Present contents of IP pushed onto stack
- <SS:SP> ← IPL
- <SS:SP-1> ← IPH

Instruction Pointer High, Low

→ Near Call (Intra Segment Call)
-> Contents of IP doesn't change

CALL 3000:4000 → Far call
↳ Contents of CS and IP both change.

* Conditional calls and jumps are always short calls and short jumps.

0 - 00	80 - 128
127 - 7F	FF - 1

ret → near return ↓ returns contents of ip

Far return ↓ returns contents of CS & IP.

CALL can be said to be a combination of 'PUSH IP + JMP'

Q. Write a sub-routine for adding two 16 bit numbers and WAP to add two 64 bit Numbers using the subroutine written by you.

```
ADD64   PROC   NEAR
        ADD    AX,BX      ; LSB 32 bits
        ADC    DX,CX      ; MSB 32 bits
        RET
ADD64   ENDP


Start:

MOV DX, 01234SH
MOV AX, 678901H
MOV CX,345678H
MOV BX, 901234H

CALL ADD64

; result stored in - DX:AX
```

Interrupts:

1) Servicable Interrupt Inservicable interrupt

INT 'n'
↓
type number of interrupt

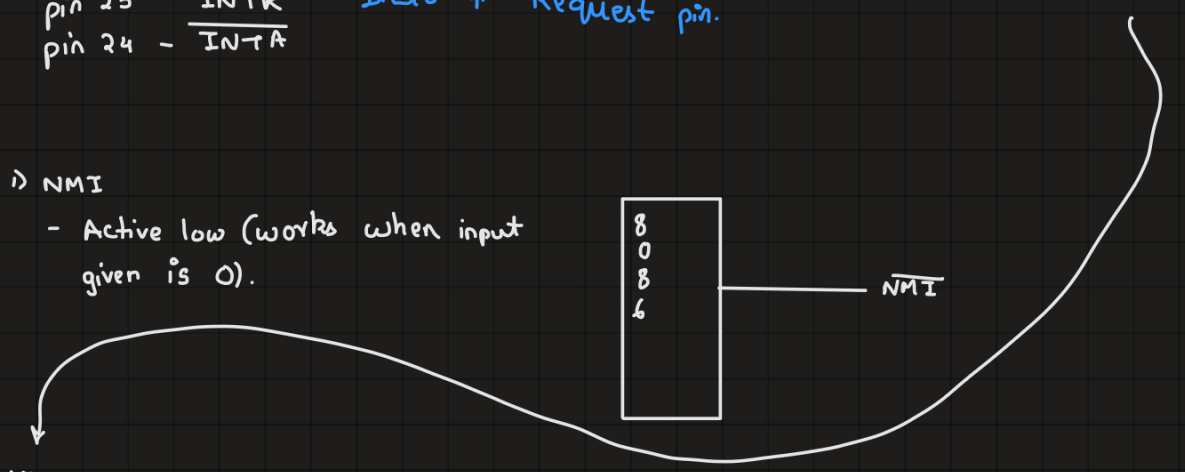
INT 00 } dedicated interrupts
⋮
INT 04 }

INT 05 } Reserved interrupts
INT 0F }

INT 10 } Available interrupts.
INT FF }

Hardware interrupts
pin 17 - $\overline{NMI}$ Non maskable interrupt
pin 23 - INTR Interrupt Request pin.
pin 24 - $\overline{INTA}$

type no 1 INT 01



1000:0000 xx yy 00 10 MOV AX, [1000]
The processor pushes the contents of IP and CS onto the stack.

[SS:SP] ← IPL
[SS:SP] ← IPH
00004 IPL
00005 IPH
[SS:SP-2] ← CSH
[SS:SP-3] ← CSL

00006 - CSL
00007 - CSH
[SS:SP-4] FL
[SS:SP-5] FH
} Flag register, low, high

TF - Trap Flag

Single Step - Type 2 interrupt
- IRET

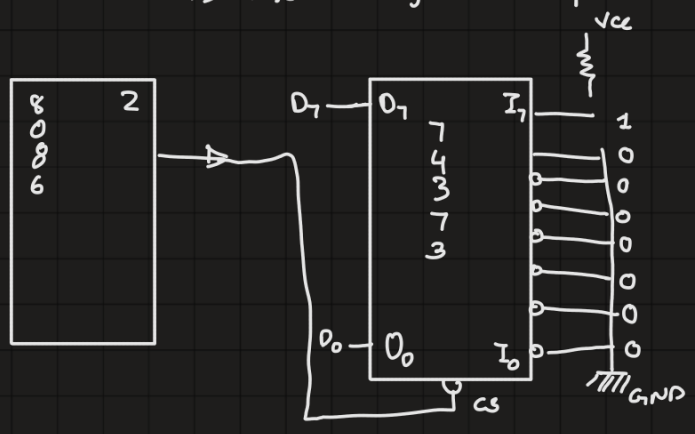
* NMI sampled during T2 of bus cycle.

Dedicated interrupts:

1) Divide by Zero error
2) Non Maskable Interrupt
3) Single Step interrupt
4) INT 3
5) INTO

Opcode for INT * INT 'n' → Opcode - CD.

↓ INTR → Active High
→ Executes last bus cycle of process and turns $\overline{IA}$ ba



In response to first $\overline{INTA}$ cycle, processor doesn't expect anything. In the 2nd $\overline{INTA}$ cycle whatever is present on databus is taken in as type number.

- Size of vector table in 8086 = 1KByte

27/03/2023

$\overline{INTA}$ cycles timing is almost equal to that of $\overline{RD}$.

1. First $\overline{INTA}$ is wasted, nothing happens
2. Second $\overline{INTA}$ processor takes data on data bus (prolly?)
3. SS:SP
   SS:SP-1 } stack cycle (I)
4. 80H x4
5. SS:SP-2
   SS:SP-3 } stack cycle CS pushed on stack.
6. (80 x4) +2,3 ← New CS.
7. PUSHF ← Pushing Flags, Clears TF Flag along with all other flags.

IRET brings back contents of CS as well as previous fla

Types:

0 ÷ → Divide by 0, no instruction
1 NMI → "vectored" hardware interrupt
2 SS → Subsequent interrupt
3 INT3
4 INTO

NMI:
Used for important events.
e.g. Power goes off → there is electricity left for 5-10 ns.
↳ That time is used to store content of register to some memory before everything dies.

NOP instruction - 0C - opcode
↳ does literally Nothing.

INT3:

→ To add breakpoints.

→ Opcode is single byte (CC)

INTO:

- Interrupt overflow:

→ IF an overflow occurs interrupt overflow will occur

→ If overflow flag not set INTO is set to NOP.

Q. WAP to set trap flag.

```
PUSHF
POP BX
OR BX, 0100H
PUSH BX
POPF
```