02/08/17

Microprocessors

1st generation (1971 - 1973)

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

2nd generation

→ NMOS
→ Compatible with TTL
→ Sub-routine nesting.

3rd generation

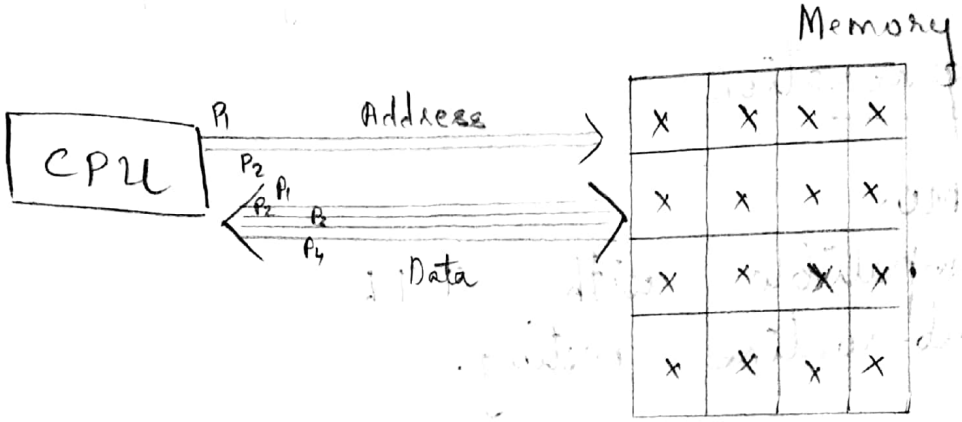
→ HMOS
→ high speed
→ program relocation
→ virtual memory
→ segmented memory
⟹ 8086.

* Bit - A unit of information.
* Using 2 bit positions, we can generate 4 addresses.

$2 \Rightarrow 4$
$3 \Rightarrow 8$
$\vdots$
$n \Rightarrow 2^n$

* 4 bits ⇒ nibble
* 8 bits ⇒ byte
* 16 bits ⇒ word
* 1024 bits = 1kB = 2^{10}
* 1024 × 1024 = 1MB = 2^{20}

1GB = 2^{30}



8086

* Address bus ⇒ 20 bit ⇒ 2^{20} addresses ⇒ 1MB
* Data bus ⇒ 16 bit

8086 is called a 16-bit processor.
It means we can read or write 16 bits at a time.

Total ⇒ 40 pins.

7/8/17

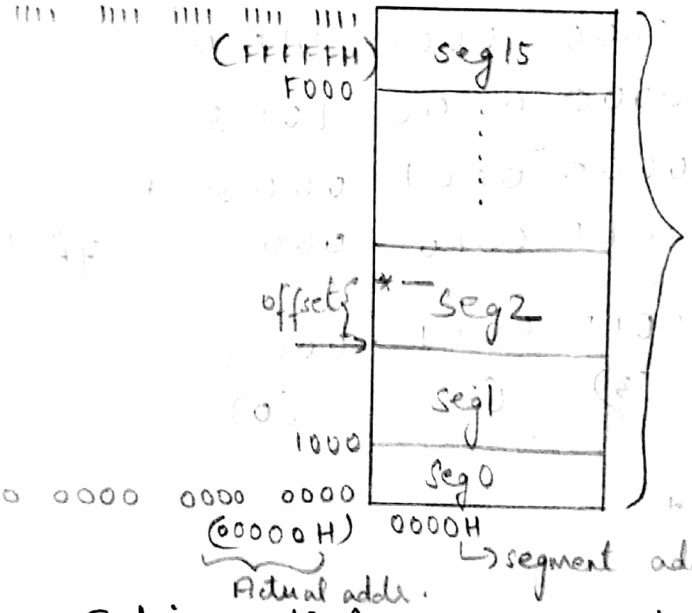
8086 Architecture

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

It has 2 portions:

1) Bus Interface Unit (BIU)
   - → Interface with peripherals
   - → Memory Address Calculator (ion)
   - → Pipelining

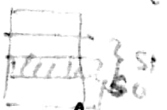
2) Execution Unit (EU)
   - → Execute Instructions.



Logical Memory

Non overlapping, maximum number of segments = 16

Overlapped: We can't predict logically



* Entire 1MB memory is divided into equal parts ⟹ segments

  1 segment = 64 kB (16 bit segment reg)

  No of parts = $\frac{1 \times 1024 \times 1024}{64 \times 1024}$ = 16 parts

* To find a data, 1st find the segment in which it lies, followed by the

offset.

Actual physical address (20 bit)
= segment address (16 bits) + offset (16 bit

⟹ 20 bit physical address:
1) shift segment address 4 bit, left
2) add offset

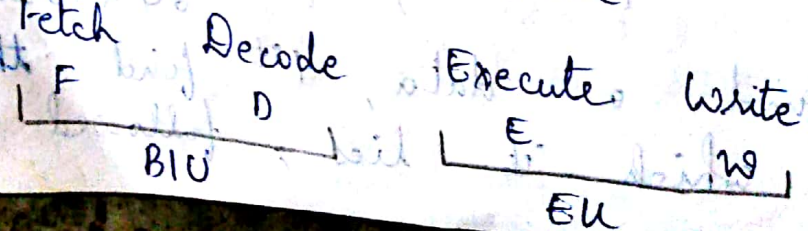
Eg: segment addr. = 2001H
offset = A320H

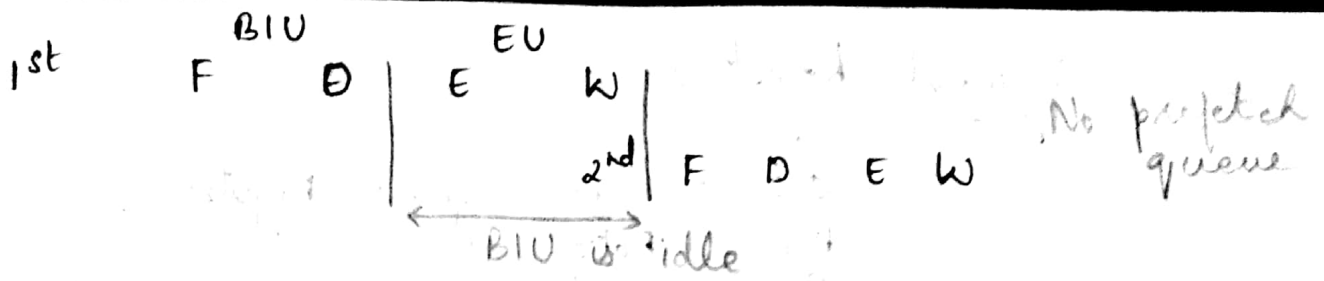
	(2)	(0)	(0)	(1)	
	0010	0000	0000	0001	
1 shift	00100	0000	0000	0010	
2 shift	001000	0000	0000	0100	
3 shift	0010000	0000	0000	1000	
4 shift	00100000	0000	0001	0000	+
	0000	1010	0011	0010	0000	offset
	0010	1010	0011	0011	0000	
	(2)	(A)	(3)	(3)	(0)	

⟹ 20010 +
A320
2A330

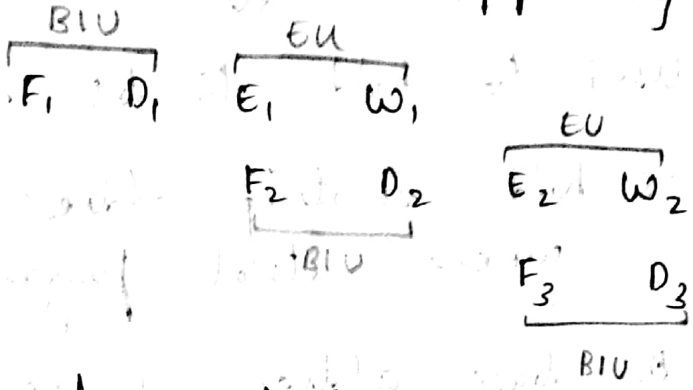
Pipelining

For an instruction, we have

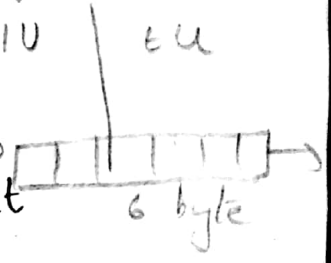




To avoid this, we use pipelining (overlapped execution of BIU & EU)



Here we need a temporary storage to hold F-D instructions ⇒ 6 byte pre decoded Queue (pre fetch)



8086 Registers

1) General data Registers (16) - Execution unit

2) Segment Registers (16) - BIU

3) Index & Pointer Register (16) - EU

4) Flags (16) - EU

GDR

	8' AH	8' AL	
AX			Accumulator
BX	BH	BL	: Address
CX	CH	CL	Counter
DX	DH	DL	Data

Segment Registers

CS : Code Segment Register
DS : Data
SS : Stack
ES : Extra

- Used to hold 16 bit segment address.

CS: hold starting address of segment where actual program

DS: base address of register where data is stored

Index & Pointer Registers

SP: Stack Pointer
: hold address of stack top

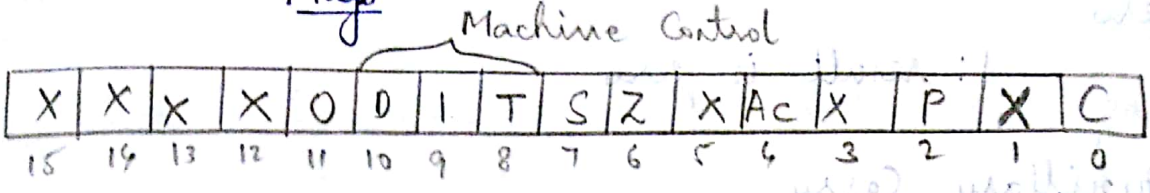
BP: Base Pointer
: Addressing modes

SI: Source Index
: Offset of source data is stored

DI : Destination Index
: Stores offset of destination data.

IP : Instruction Pointer
: Address of next instruction to be executed.

Flags



1) Status flag / Conditional code — states of arithmetic operations
2) Machine control
   - shows statistics of machine operation.
   - DIT
   - Direction, Interrupt, Trap.

- Overflow flag: If an arithmetic operation results in a carry from 16th bit position, this flag is set to 1.

- Direction Flag
  - Indicates direction of string processing
  0: Auto increment
  1: Auto decrement

- Interrupt Flag
  0: Ignore interrupts
  1: Enable interrupts

- Trap — for debuggers mode enabling
  1: single step execution.
  : there is an interrupt after each instruction.
  : used in debugging.

- Sign
  0: +ve number
  1: -ve number

- Zero

1: result is zero

- Auxillary Carry

1: When there is a carry in to lower nibble.

- Parity

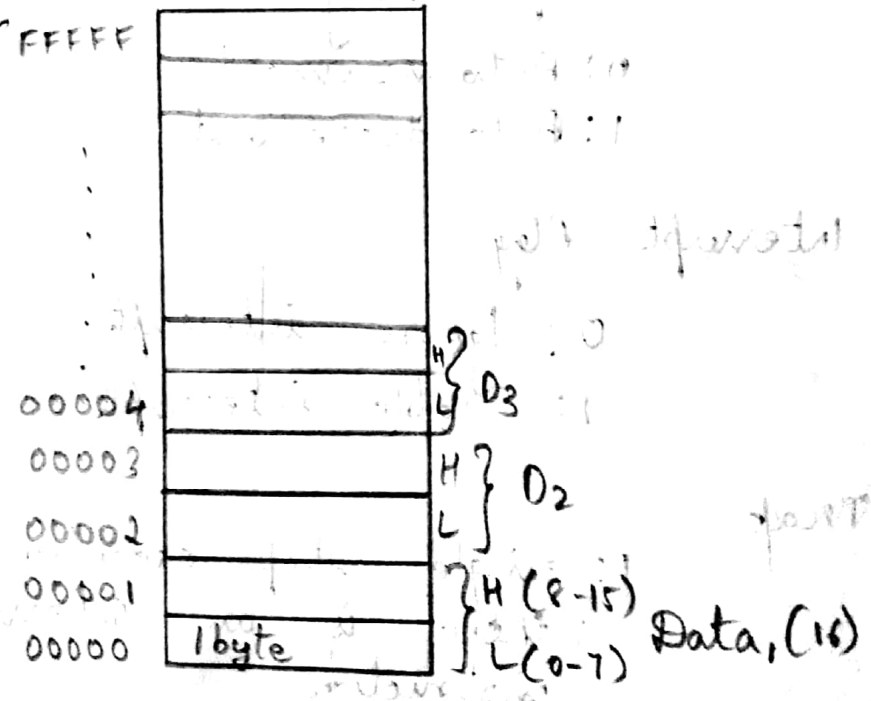
1: even parity
0: odd parity

- Carry

1: If there is a carry.

9/8/17

Physical Memory Organization



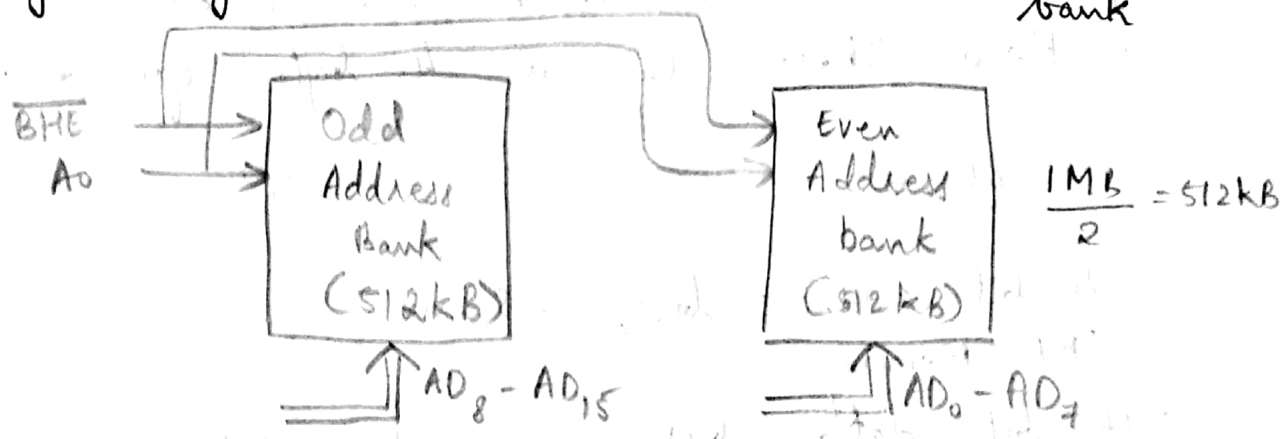
Data bus = $D_0 - D_{15}$

$D_0 - D_7$ lower byte

$D_8 - D_{15}$ higher byte

Lower byte (0-7) in Even add = Even address bank

Higher byte (8-15) in Odd add = Odd address bank



BHE → To select bank

A_0 → To select address.

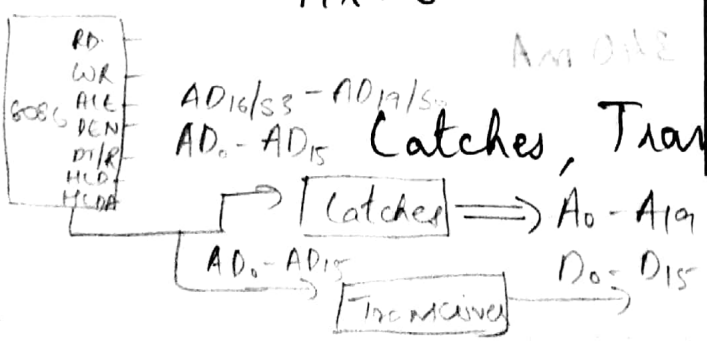
Modes of Operation of 8086

1) Minimum Mode

- Only one processor (8086)
- All signals (RD, WR, HALT..) generated by processor.
- MN/$\overline{MX}$ = 1

  MN = 1

  MX = 0

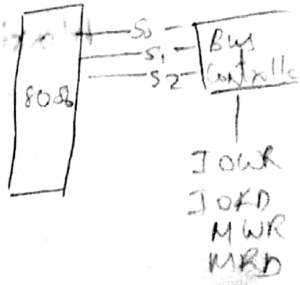


2) Maximum Mode

- More than 1.
- 8086 generate only status signal S_0, S_1, S_2. Then bus controller generate all other signals.
- MN/$\overline{MX}$ = 0

  MN = 0

  MX = 1



Catches, Transceivers, Memory

8088/8086

- It can work with 8085 & 8086 ⇒ 8088
- Here there is a 4 byte queue.

8086	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 (∵ we address only 8 bit)
- $AD_0 - AD_{15}$	- $AD_0 - AD_7$ & $A_8 - A_{15}$
- $\overline{BHE}$ s/g used	- Not used
- 6 byte queue	- 4 byte Queue
- BIU fetch next instruction when 2 byte is free in Queue	- 1 byte.
- $IO/\overline{M}$	- $IO/\overline{M}$ (To work with 8085, it is inverted)
- Maximum Current 360 mA	- 340 mA

14/08/17

Pin Diagram of 8086 (Study)

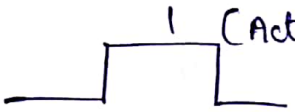
It is classified into 3 types:

1) Common Pins
   - It has common function in both minimum & maximum mode.
   - Pins except 24 to 31.
2) Maximum mode pins
3) Minimum mode pins.

⟹ Pins AD0 – AD15 (Pin 16-2, 39)

Address Pins
- Active high.

└ These lines are multiplexed bidirectional address/data bus.



⟹ A19/S6 – A16/S3

These lines are multiplexed unidirectional address & status bus.

S3, S4 – represents currently active s/gs.

S4, S5 – interrupt status

S6 always 0.

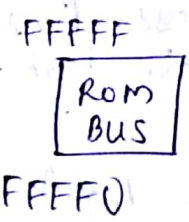
⟹ $\overline{BHE}$ / S7 (Pin 34)

- Bus high enable.
- If data is present in the higher bytes (AD8 – AD15).

⇒ $\overline{RD}$ (Read)
Pin 32

⇒ READY (Pin 22) Input
- Used to synchronize with low speed devices.
- It indicates that the device is ready to transfer data.

⇒ RESET (Pin 21) Input
- It is set to FFFF0



⇒ NMI
- It is a non-maskable interrupt sig.
- High priority interrupts.

⇒ INTR
- To consider low priority interrupts.

⇒ $\overline{TEST}$ (Pin 23)
- To include time delay.

⇒ CLK

⇒ V_{cc} / V_{ss}

⇒ MN/$\overline{MX}$

In minimum mode:

* $\overline{INTA}$ (Pin 24) o/p.
  - Interrupt acknowledgement s/g.
* ALE (Pin 25) o/p.
  - Address Latch Enable s/g.
  - It indicates that a valid address is available on bus $AD_0 - AD_{15}$.
    If it is low – data is present.
* $\overline{DEN}$ (Pin 26) o/p.
  - Data Enable s/g.
  - If data is present on multiplexed address/data bus, DEN is set to low.
* DT/$\overline{R}$
  - Data Transmit / Receive s/g.
  - Shows direction of data flow.
* M/$\overline{IO}$
  - M/M or IO operations.
* $\overline{WR}$
  - Enable write operation.
* HOLD
  - Interface with DMA
* HLDA
  - Hold Acknowledgment s/g.

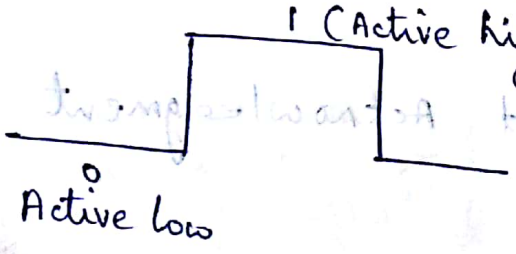
In maximum mode:

* $\overline{QS_1}$ & $\overline{QS_0}$
  - Status of instruction queue.
* $\overline{S_0}$, $\overline{S_1}$, $\overline{S_2}$
  - Indicates operation being done by microprocessor.
* $\overline{LOCK}$ (Pin 29)
  - Lock the bus access.
* $\overline{RQ}/\overline{GT_1}$ and $\overline{RQ}/\overline{GT_0}$
  - Bus request & grant s/g.

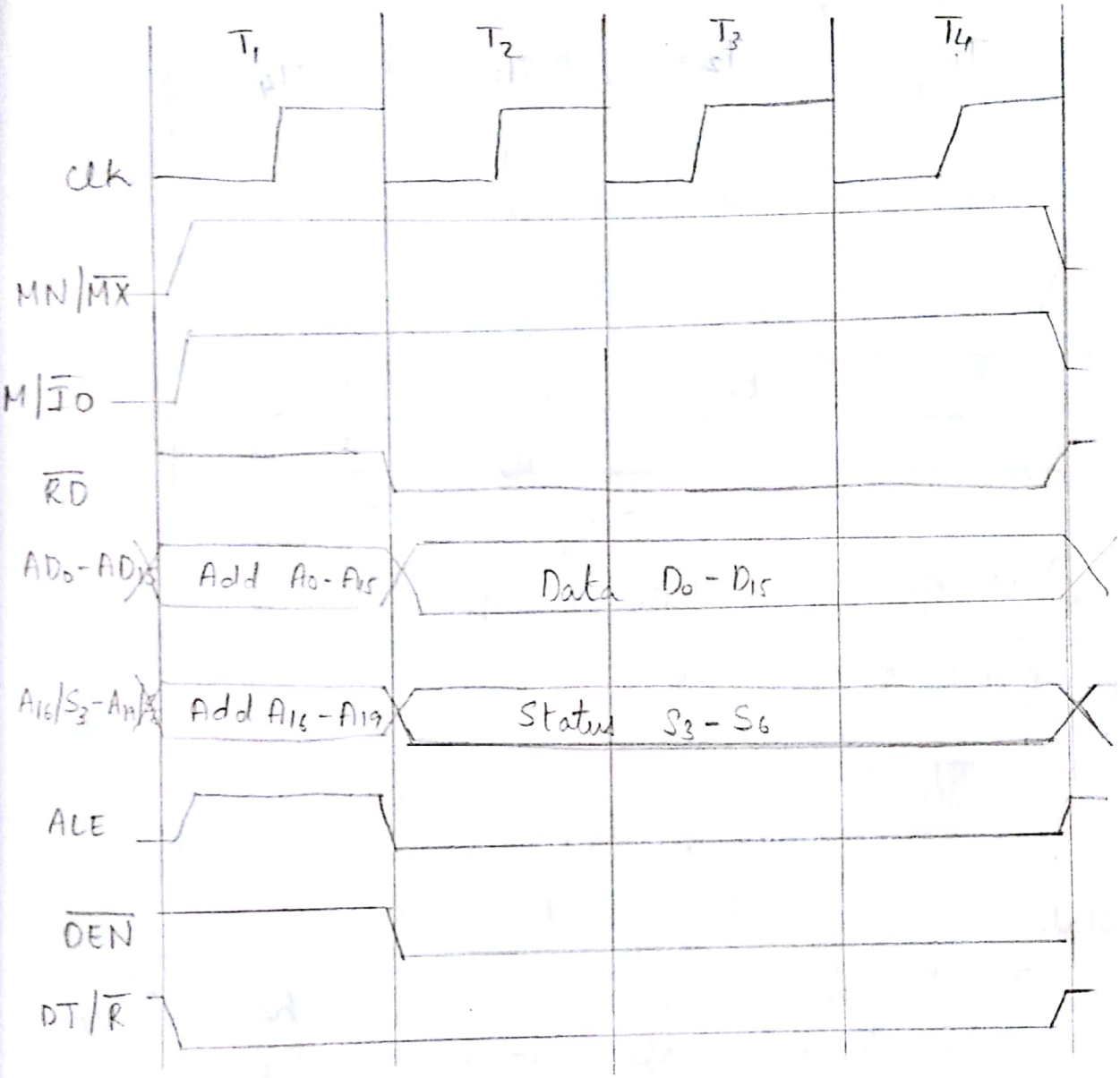
16/08/17

Timing Diagram:

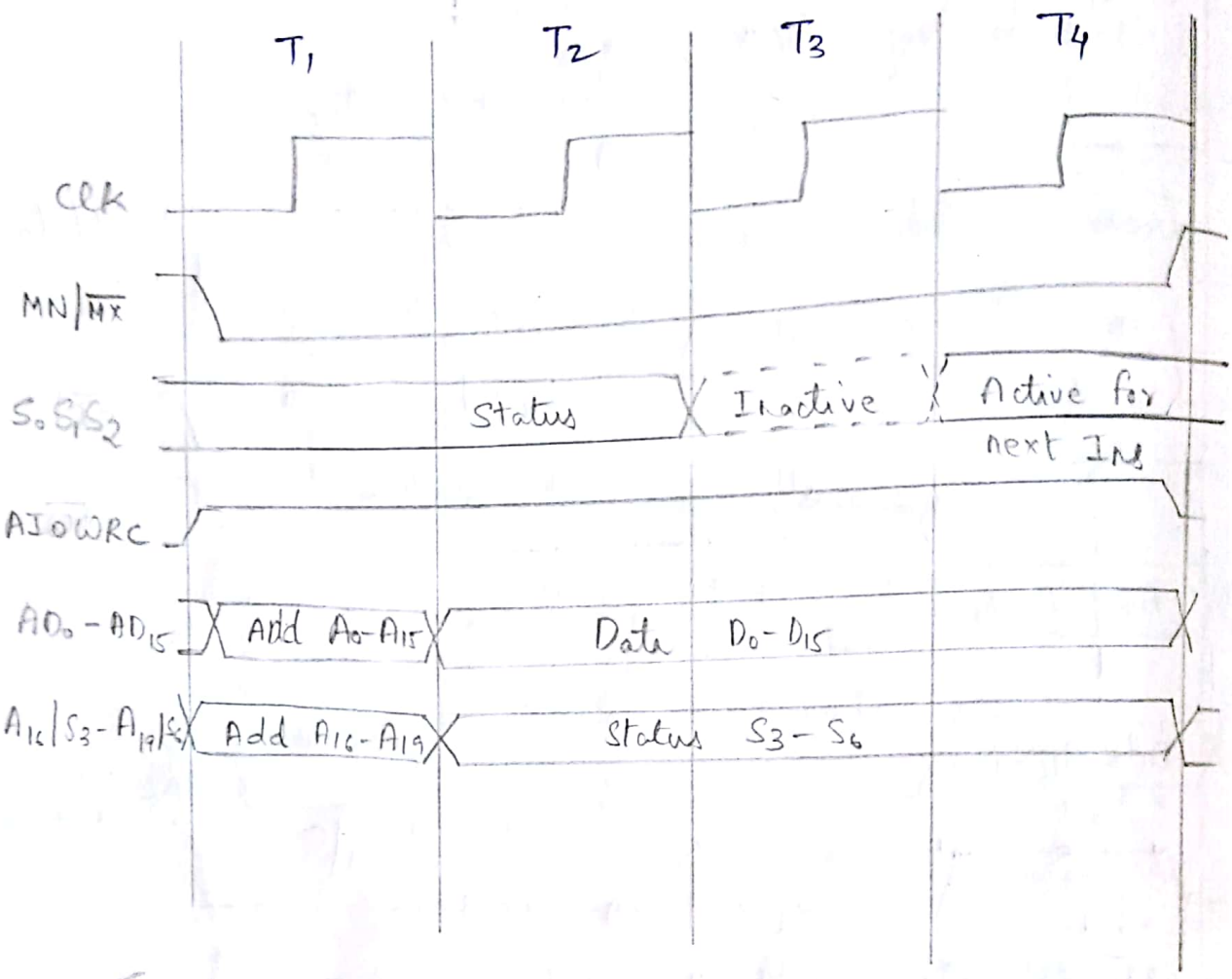
* It shows the enabling & disabling of various signals.
* Minimum Mode/Maximum Mode
  Memory/IO
  Write/Read
* We need 4 clock cycle (4T states) for Memory WR/RD
  IO WR/RD



Minimum Mode Memory Read:



Maximum Mode IO Write:



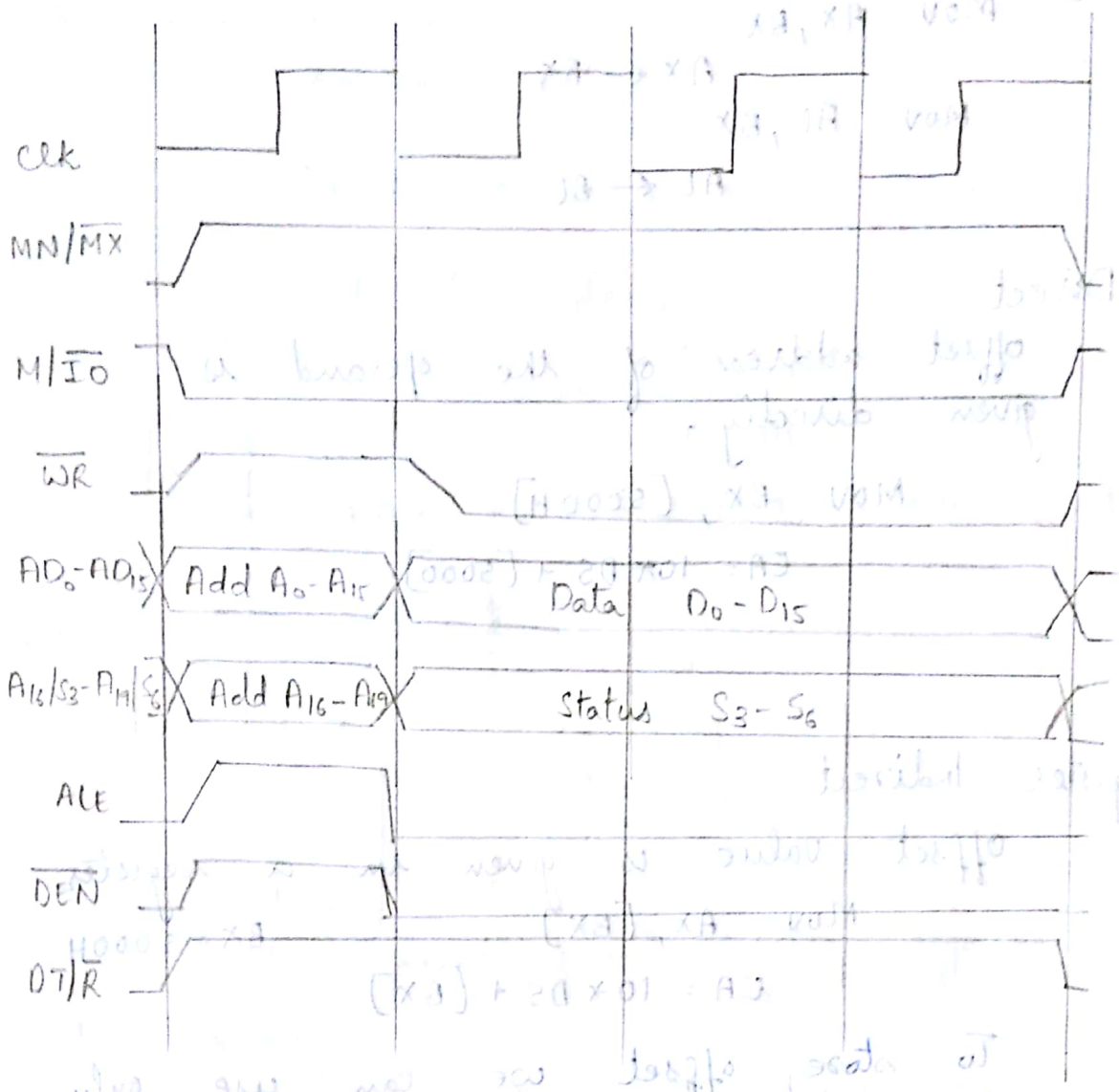
(Status s/g active from previous cycle's T_4).

To differentiate read/write, we have

AMRC, AMWRC, AIORC, AIOWRC } - bus controller

- only for maximum mode.

Minimum Mode IO Write :



21/8/'17

Addressing Modes :

Sequential :

1. Implied :
   - HALT
   - RESET
   - CLR
2. Immediate

   Data is given directly to registers.

   MOV AL, 02H AL ← 02

   MOV AX, 0102H AH ← 01, AL ← 02

3. Register

MOV AX, BX
AX ← BX

MOV AL, BX
AL ← BL

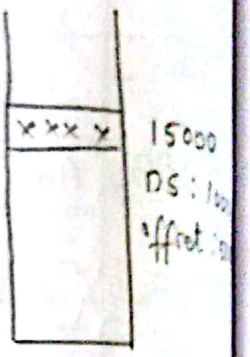
22/8/17

4. Direct

offset address of the operand is given directly.

MOV BX, [5000H]

EA = 10*DS + [5000]



5. Register Indirect

offset value is given in a register

MOV AX, [BX]

BX = 5000H

EA = 10 * DS + [BX]

To store offset we can use only BX | SI | DI

6. Indexed

MOV AX, [SI]

* When we use SI | DI

EA = 10 * DS + SI

OR

EA = 10 * ES + DI

* Used for string operations.

7. Relative Based

MOV AX, 50H [BX]
[BP]

BX: Base register

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

8. Based Indexed

MOV AX, [BX][SI]
BP DI

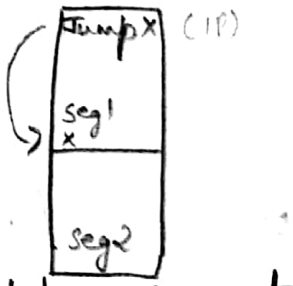
EA = 10 * DS + [BX|BP] + [SI|DI]

9. Relative Based Indexed

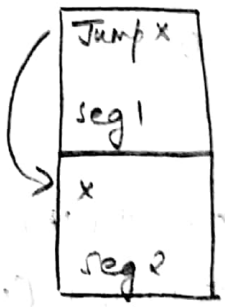
MOV AX, 20H [BX][SI]
BP DI

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

Note: Non-sequential:



Intra segment



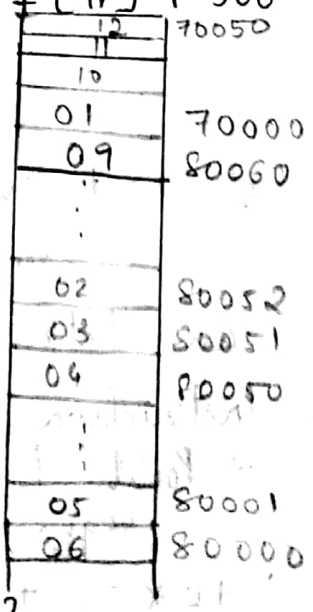
Inter segment

DS: data segment
CS: code segment

10) Relative

Jump 500H

EA = 10 * CS + [IP] + 500



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

1) MOV AX, BX

AX ← 0052
(Register)

2) MOV AX, [0001] (Direct)

$10 * DS + 0001 = 80,0001$

$AX \leftarrow 05$

3) MOV AX, [BX] (Register Indirect)

$EA = 80000 + 0052$

$AX \leftarrow 02$

4) MOV AX, 08 [BX] (Relative based)

$EA = 10 * 8000 + 0052 + 08$

$= 80060$

$AX \leftarrow 09$

5) CALL 50H

$EA = 10 * CS + [IP] + 50$ (Relative)

$= 10 * 7000 + [0000] + 50$

$= 70050$

Q.

70020 : Jump 05H
70021 :
70022 :
⋮

Address of next instruction

CS : 7000H

It is an intrasegment

IP : Instruction Points

→ Points to next instruction in sequence

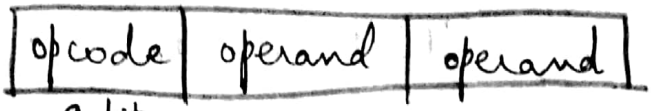
$\therefore 10 * CS + IP + 05$

$= 70000 + 0021 + 05$

$= 70026$

23/08/17 # Instruction Set of 8086

Instruction format:



1) RESET - 1 byte instruction
   - only opcode (8 bits)
2) MOV AX, BX - 1 byte instruction
3) MOV AL, 02H - 2 byte instruction
   (8) (8)
4) MOV AX, 0001H - 3 byte
   (8) (16)

Classification:

- Data transfer
- Arithmetic
- Logical
- Branch
- String
- Flag Manipulation
- Processor control

Data Transfer Instructions

- used to transfer data from source to destination.

MOV Des, Src

Src: register, m/m location or immediate operand

Des: register or m/m operand.

* MOV DS, 5000H (wrong)
* MOV AL, BL AL ← [BL]

- Push operand
  It pushes the word into top of stack

  PUSH BX
  (First BH then BL)

- POP des:
  - Pop operand from top of stack to des.

- XCHG Des, Src
  This exchanges Src with Des
  - Can't exchange 2 m/m locations directly.

  XCHG DX, AX
  [AX] ⇌ [BX]

- XLAT
  Equals MOV AL, [AL][BX]
  AL ← 10 * DS + [AL] + [BX]

- IN Accumulator, Port Address
  Eg: IN AX, 0028H : AL ← [0028]
  16 8
  AH ← [0029]

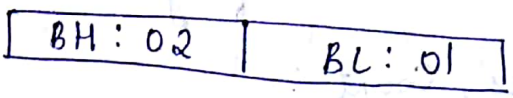
- OUT Port Address, Accumulator
  Eg: OUT 0028H, AX

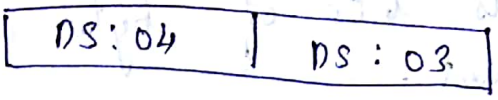
- LEA Register, Src:
  Loads 16-bit register with the offset address of the data specified by the src.

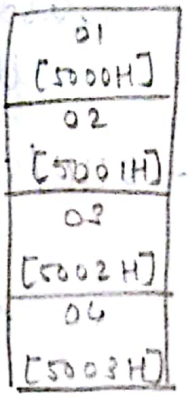
Eg: LEA BX, [DI]

- LDS

LDS BX, [5000H]







- LES

load extra segment

LES BX, [0301H]

- LAHF
  - Copy lower byte of flag register to AH.
- SAHF
  - Copy content of AH to lower byte of flag register.
- PUSHF
  - push flag register to top of stack.
- POPF
  - Pops the stack top to flag register.

BIT MANIPULATION OR LOGICAL

- NOT Src:
  - Complements each bit of src to produce 1's complement
- AND
- OR
- XOR

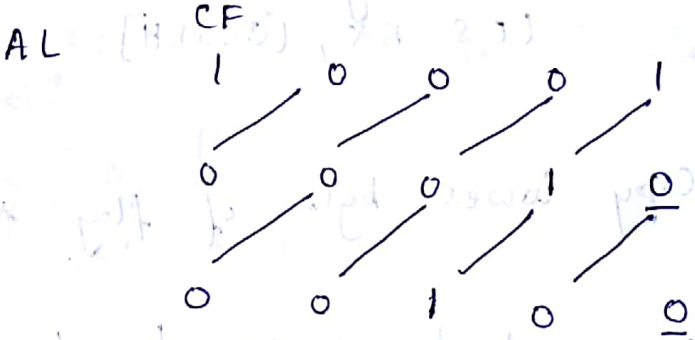
- TEST Des, Src

Perform bit by bit AND of operands, result is not stored any where
Used to check flag

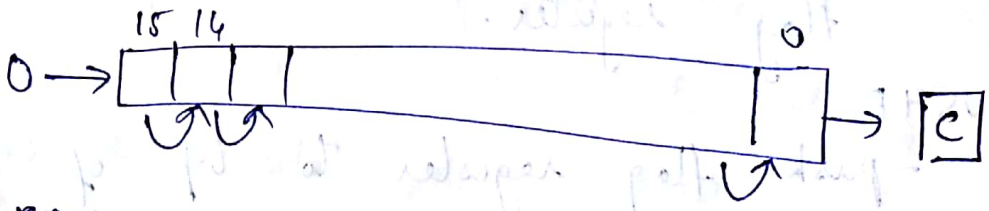
- SHL/SAL Des, Count:

It shifts bits of byte or word left by count.
It puts zero(s) in LSBs.
MSB shifted to carry flag.

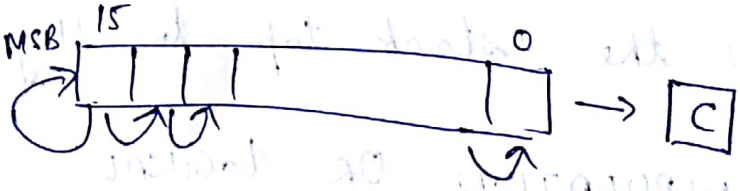
SHL AL, 02



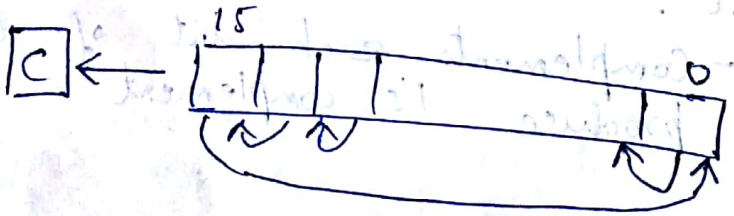
- SHR Des, Count



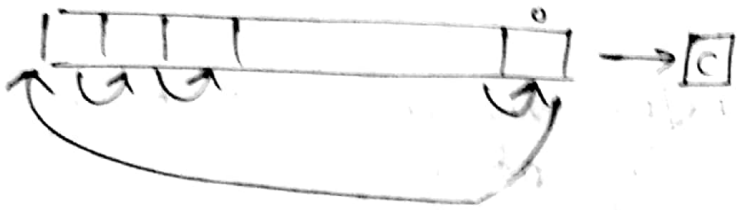
- SAR Des, Count



- ROL Des, Count (without carry)

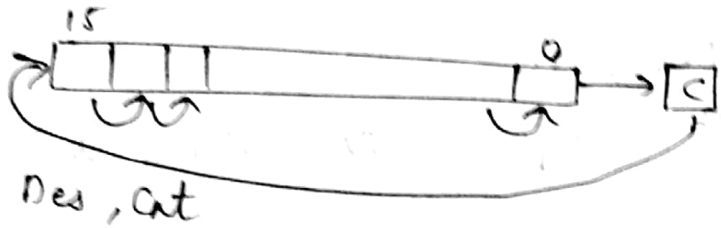


- ROR Des, Count : (without carry)



To check if a no. is odd or even

- RCR Des, Cat:



- RCL Des, Cat

BRANCH

- CALL
  - Call procedure or functions
- RET
- JUMP (JMP)
  - Unconditional jump

FLAG MANIPULATION

- STC
  It sets carry flag to 1.
- CLC
- CMC
- STD
  sets direction flag to 1
- CLD

MACHINE CONTROL

- WAIT
- HLT
- NOP
- LOCK
- ESC

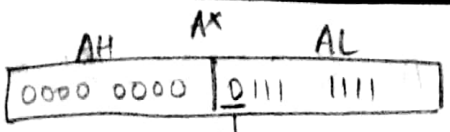
29/08/17

String Instructions

- LOADSB / LOADSW
  Load AL/AX by the content of string pointed by DS:SI
- STOSB/ STOSW
- CMPS Desc, Src
  It compares the string bytes or words
- SCAS string
  - Scan the string
- MOVS/MOVSB/MOVSW

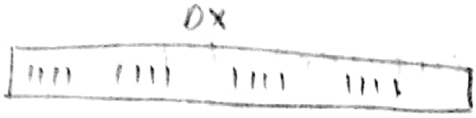
Arithmetic Instructions

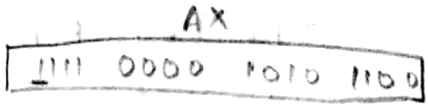
- ADD Desc, Src:
- ADC Des, Src:
  It adds 2 operands with CF
- SUB Des, Src:
- SBB Des, Src:
  Subtract with borrow
- Inc Src:
- Dec Src:
- CMP Des, Src:
- MUL Src
- IMUL Src:
- DIV Src:
- IDIV Src:
- CBW (Convert byte to word)
  Convert byte in AL to word in AX



↓ Extend sign bit left

- CWD (Convert word to double)





- AAA (ASCII Adjust After Addition)

* 4 types of data

1) ASCII (Unpacked)
   - Entering from keyboard.
   Eg: 15

31	35

(Representations)

2) Unpacked BCD

01	05

3) Packed BCD
   15

4) Hex 0F

Rules for AAA:

1) Clear AH.
2) Move the value to be converted to AL register.

Case 1: If the lower nibble of AL (1st digit in AL) < 9 and 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: 1) 02 + 03 → 05
unpacked BCD → unpacked BCD

```
0000  0010
0000  0011
-----------
0000  0101
```

AL

0000	0101

2) 09 + 05

```
0000  1001
0000  0101
-----------
0000  1110   (0E)   E > 9
0000  0110
-----------
0001  0100
```

AH

0000	0001

AL

0000	0100

3) 05 + 05

```
0000  0101
0000  0101
-----------
0000  1010   (0A)
0000  0110
-----------
0001  0000
```

AH

0000	0001

AL

0000	0000

4) 08 + 08

```
      1(AC)
0000  1000
0000  1000
-----------
0001  0000
0000  0110
-----------
0001  0110
```

AH

0000	0001

AL

0000	0110

- AAS
- AAM
- AAD

- Decimal Adjust after Addition (DAA)

1) Eg: 25+25

1) The value to be converted must be in AL.
2) If the lower nibble of AL > 9, add 06. Then check the upper nibble of AL. If it's greater than 9, add 60 otherwise do nothing.

1)
```
  0010   0101
  0010   0101   +
-------------------
  0100   1011   → (4A)
  0000   0110
-------------------
  0101   0000
   (50)
```

2) 49+23 (= 72)
```
  0100   1001   +
  0010   0011
-------------------
  0110   1100
  0000   0110
-------------------
  0111   0010
   (72)
```

- DAS

Q. Assume that the capacity of AL is infinite. The value in AL is 9C4A. What is the result after DAA instruction.

29)

```
1001   1100   0100   1010
0000   0000   0000   0110
-----------------------------
1001   1100   0101   0000
0000   0110   0000   0000
-----------------------------
1010   0010   0101   0000
0110
-------
1 0000
```

9C4A → 10250

Q. For addition operation, the user entered 2 values 09 and 05 through keyboard. What will be the result in hex format, unpacked BCD, packed decimal?

```
  0011  1001
  0011  0101
-------------
  0110  1110   (6E)
        0110
-------------
  0111  0100
```

```
  1
 39
 35
----
 74
```

AH AL

0000 0001 0000 0100

```
  0110  1110
        0110
-------------
  0111  0100
```

(74)

30/08/'17

8086 Programs

```
ASSUME   CS: Code   DS: Data
```

Note * In assembly level program, we have 2 parts
→ Assembler directives
→ Instructions

```
DATA  SEGMENT
      Data  DB  01H
      Msg   DB  "programming $"
      Xxx   MACRO parameter
            .....
            ENDM
DATA  ENDS
CODE  SEGMENT
Start : mov  ax, data
        Mov  ds, ax
        Yyy  PROC param
             ....
             ENDP
CODE  ENDS
END.
```

Q. Write an assembly level program to print a string "hello world".

```
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
```

o/p register - DX
AL ← 09h - print string
01h - read
02h - write

* INT - interrupt

Q. Write a program to print 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
```

Q. Write a program to display string with sub-routine.

```
ASSUME CS: Code DS: Data
DATA SEGMENT
    MES DB "HELLO WORLD! $"
```

```
ENDS
CODE  SEGMENT
    START: MOV AX, Data
           MOV DS, AX
           Display PROC YYY
              (Push all registers)
               LEA DX, YYY
               MOV AH, 09H
               INT 21H
              (Pop all registers)
           RET
           ENDP
           CALL display MES
           MOV AH, 4CH
           INT 21H
    CODE  ENDS
    END  START
```

Q. Write a program to add two immediate values 03 & 02.

```
ASSUME  CS: Code  ~~DS: Data~~
~~DATA~~  S
CODE  SEGMENT
    START: MOV  bl, 02
           MOV  cl, 03
           ADD  bl, cl
CODE  ENDS
END  START
```

mov bl, 02
add bl, 03

Q. Write a program to add 2 numbers located at offset address 5000 & 6000. Store the result to offset address 7000.

```
ASSUME  CS: Code

CODE  SEGMENT
    Start: MOV bl, [5000]
           add bl, [6000]
           mov [7000], bl
CODE  ENDS
END
```

Q. Write a program 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      } si is used when we
           MOV di, 6000      } need to inc. address.
    XX:    MOV bl, [si]
           mov [di], bl
           INC si
           INC di
           DEC cl
           JNZ XX
CODE  ENDS
END
```

Q. Write a program to find 1+2+3+4..+10.

```
ASSUME  CS: Code  DS: Data
Data  Segment
      SUM  DB  00H
Data Ends
```

```
Code Segment
Start : MOV CL, 0AH
XX : ADD SUM, CL
     DEC CL
     JNZ XX
CODE ENDS
END
```

OK

```
mov bl, 00h
mov cl, 01h
xx: add bl, cl
    inc cl
    cmp cl, 0ah
    JNZ xx
```

Q. Write a program to check whether the initialised variable number is even or odd.

```
ASSUME CS: Code DS: Data
Data Segment
     num DB 05h
     odd DB "No. is odd"
     even DB "No. is even"
Data 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
```

Q. Write a program to count the no. of 1s in an eight bit number.

```
ASSUME    CS: Code    DS: Data
Data Segment
        num    DB    07H    decr    DB    08H
        count  DB    00H
Data Ends
Code Segment
        Start : MOV   ax, Data    mov cl, count
                Mov   DS, ax      mov bL, decr
                mov al, num
        yy:     SHR
                JNC   xx
                inc   cl
        xx:     dec   bl
                JNZ   yy
CODE ENDS
END
```

11/09/'17

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

1) Identify the mode of operation of 8086
2) Identify the type of operation
3) Draw the timing diagram.

1) Maximum Mode
2) Memory Write

Q. Write an assembly level program to find the factorial of a number.

```
ASSUME  CS: Code  DS: Data
Data  Segment
        num   DB  05H
        p     DB  01H

Data  Ends
Code  Segment

        Start :  mov  ax, Data
                 mov  DS, ax
                 mov  al, p
                 mov  cl, num
        yy :     MUL  cl
                 Dec  cl
                 JNZ  yy

Code  Ends
End
```

OR

```
ASSUME  CS: Code
Code  Segment
        Start:  mov  al, 04h
                mov  cl, 01h
                mov  bl, 01h
        Loop :  mul  bl, cl
                inc  cl
                cmp  cl, al
                JLE  loop
```

```
mov dl, bl   } (print bl)
mov al, 02h
int 21h

Code Ends
End
```

Q. Write a program to print the Fibonacci series, first 6 numbers.

```
ASSUME CS: Code
Data Segment    sum DB 00h
Code Ends       Segment
    Start: mov al, 00h
           mov bl, 01h
           print al
           print bl
           mov cl, 04h
    LOOP:  ADD sum, bl

           mov al, bl
           mov bl, sum
           print bl
           dec cl
           JNZ LOOP
```

0 1 1 2 3

al 0
sum = 0
sum = 1
al = 1
bl = 1
2
3
3 →

13/09/17

MODULE - III

INTERRUPT

A signal which interrupt the normal sequential execution of a processor.

Classification

* Internal - Signal produced inside the processor. (from internal circuit).
* External - S/g produced outside the processor. (from external circuit).

---

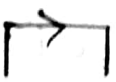
* Software - Signal produced by a software part (program).
* Hardware - produced by hardware devices. Eg: Keyboard

---

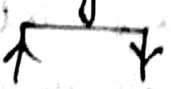
* Maskable (INTR) - It can be ignored.
* Non-maskable (NMI) - It cannot be ignored.

---

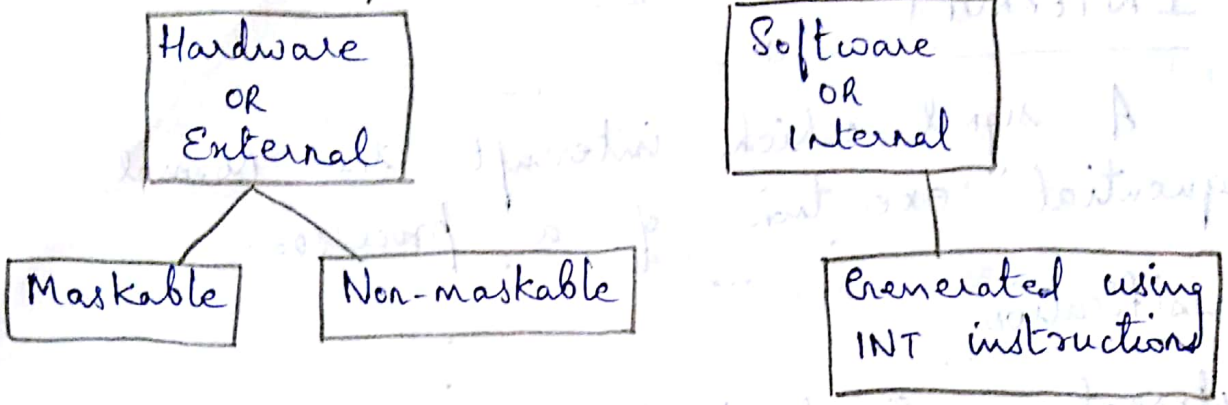
* Level triggered : Interrupt triggered during the level.



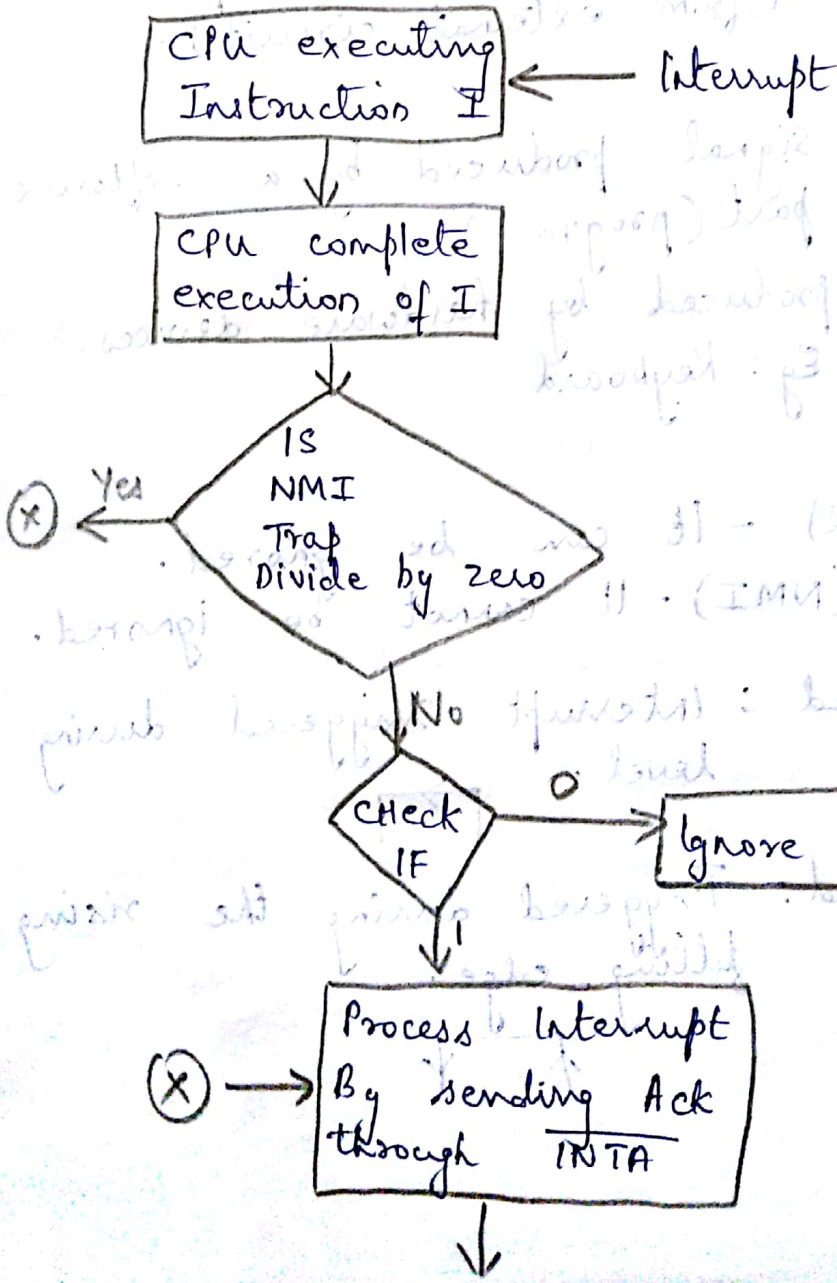
* Edge triggered : Triggered during the rising or falling edge.



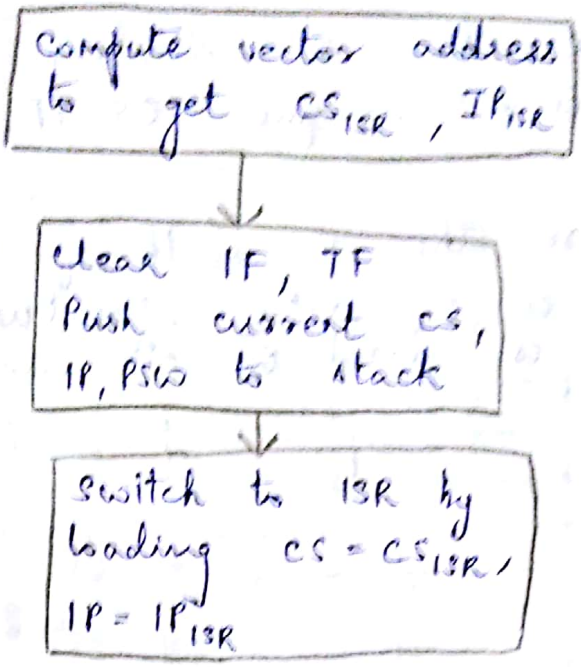
Interrupts



Interrupt Processing Cycle



* An interrupt is processed only after completing the current instruction



IVT: Interrupt vector table

PSW All the flags CF, IF, TF

Interrupt Vector Table (IVT)

→ Software interrupts are generated using INT n, where n is type of interrupt

→ There are 256 interrupt in 8086

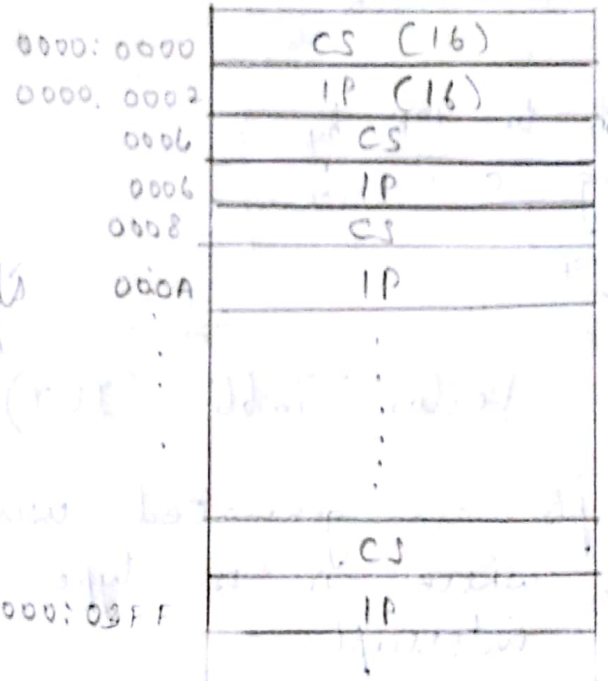
0 - FF

Type 0 - Divide by zero
1 - Trap
2 - NMI
3 - INT
4 - INTO
⋮
FF

→ ISR - These are functions used to handle interrupts.
Each interrupt has an ISR.
∴ There are 256 ISRs in total.

→ Each ISR has a starting address as well as is a unique CS:IP

→ IVT - Stores the CS & IP of each ISR
- It is a data structure
- It is stored in 0th code segment



Total size = 256 × 4
= 2^{10}
= 1kB

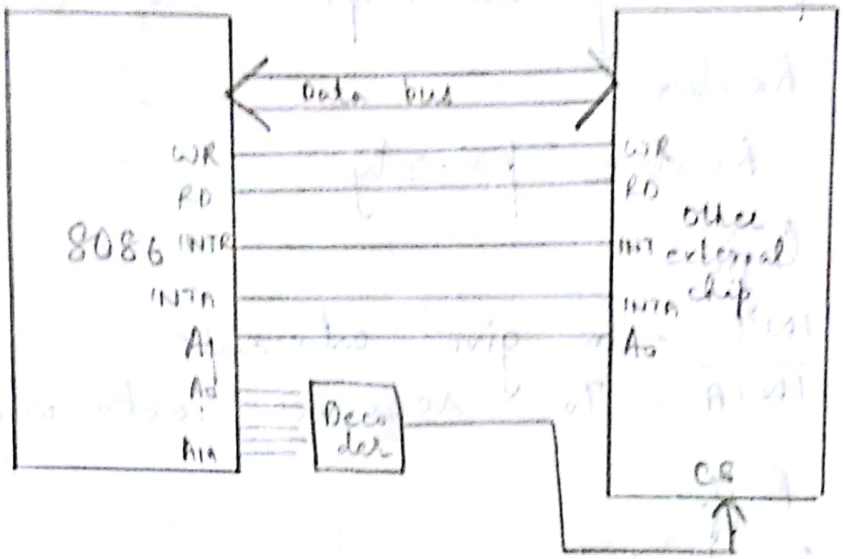
Eg: INT 2

1st step : 2 × 4 = 8
Go to 0008 to get CS
Add 2 bytes ⇒ 000A to get IP

27/09/17

Interfacing

General diagram



CS: Chip Select

8259 Programmable Interrupt Controller

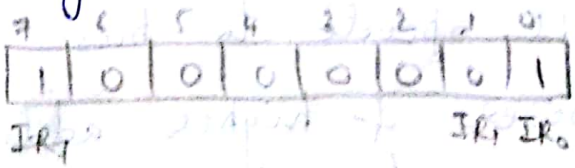
- It helps to process more than 2 interrupts.
- Deals with multiple interrupts.
- Priority Resolver.

8086 can handle only 2 interrupts using INTR & NMI

8259 block diagram

IR0 - IR7 → 8 interrupt requests possible

To store the condition of IRs, we have 'interrupt request register' (IRR). 8 bit register.



Interrupt Mask Register : To enable & disable interrupts.

1 - disable
0 - enable

In - service Register
- Stores the details of currently processing interrupts.

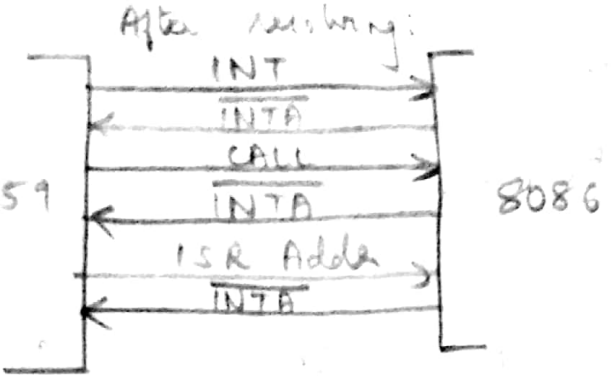
Priority Resolver
- Resolves priority

Control Logic
INT - To give interrupt
$\overline{INTA}$ - To receive acknowledgement

Cascade Buffer
- To handle more than 8 interrupts
- Connect more 8259

Operation Cycle

1. Interrupt receive through IR0 - IR7.
2. Set the corresponding bit of IRR.
3. Priority resolver resolve priority & send INT signal to 8086.
4. 8086 acknowledges through INTA signal
5. Set corresponding bit in ISR, reset IRR bit
6. Place CALL opcode to bus (to enable interrupt SR)
7. Send lower & higher address of ISR in response to subsequent INTA
8. Reset ISR register.



Operating Modes of 8259

1. Fully Nested Modes
   - Default mode
   - IR0 - highest priority
     IR7 - least priority
2. Automatic Rotation
   - IR0 to IR7 all have equal priority initially.
   - After getting interrupt in each line, set it to lower priority.
3. Edge & Level Triggered Interrupts
4. Reading 8259 Status
   - Used to read the status of the internal registers of 8259.
5. Special Fully Nested Mode
   - More than one 8259 used
   - Master/Slave arrangement
   - Slave give INT to Master, master to 8086.

6. Buffered Mode
   - Enable intermediate storage using buffer
7. Cascade Mode
   - Used with more than 1 8259
8. End of Interrupt
   - To reset ISR bit
9. Specific Rotation
   - Some interrupts fixed, others are rotated
10. Specific Mask Mode

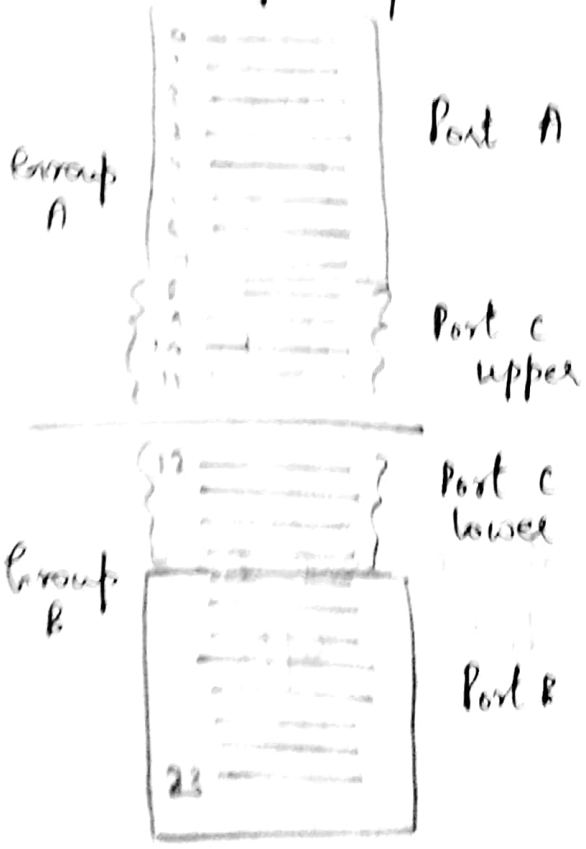
MODULE - IV

8255

Programmable peripheral interface
To handle many input/output ports.

Architecture

- 24 input pins

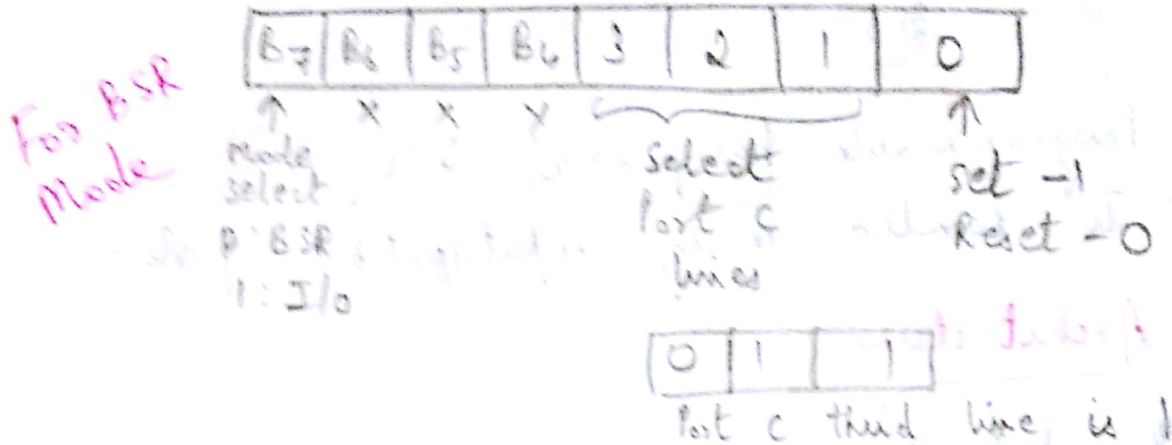


Operation Modes

1) Bit set/Reset Mode: (BSR)
   - To set/reset Port C pins

2) I/O Mode

Bit set/reset control word format:



I/O Modes:

To enable i/p - o/p operations

Three modes:

Mode 0:

- Default mode
- Simple input/output
- Use two 8-bit ports A and B, two 4-bit ports Port C upper & lower
- No handshake (All ports can be used for I/O operation)

Mode 1:

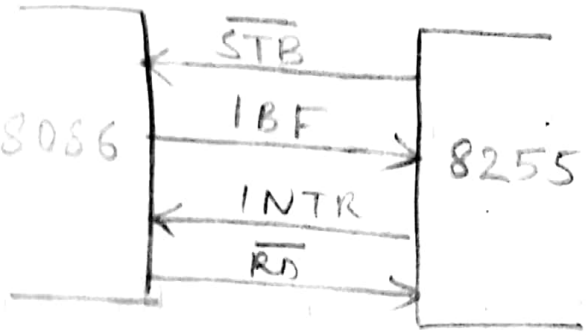
- Input/output with Hand shake
- Only Port A and B available
- Port C: generate handshake signals
- PC0 - PC2: generate control signals for Port B
- PC3 - PC5 generate control signal for Port A
- PC6, PC7 independent data lines

Mode 2:

- Bidirectional I/O data transfer.
- Only group A is enabled.
- Port A works as 8-bit bidirectional

Mode 1

i/p

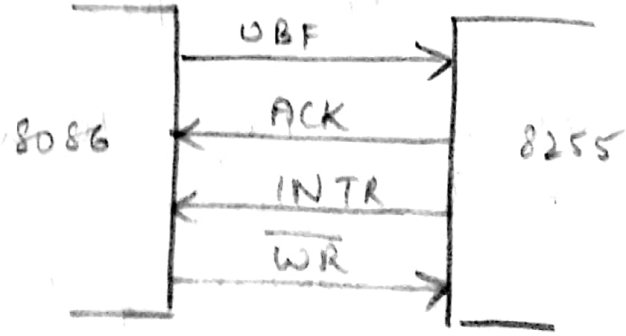


$\overline{STB}$: store
: data ready to be transmitted

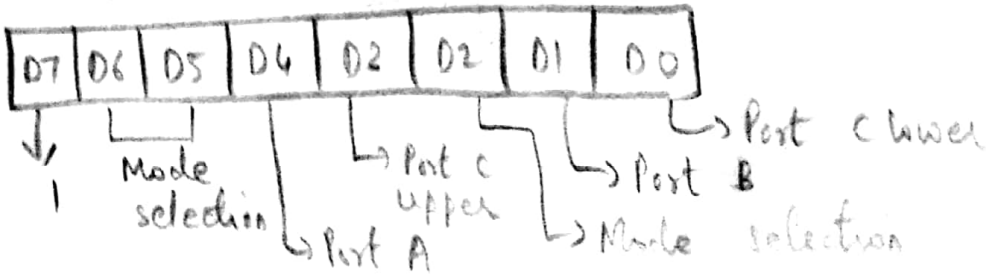
IBF : Input buffer full

Data first stored to i/p port, then send to 8086

o/p



For I/O Mode



Q. Interface 8255 with 8086 to work as I/O port. Configure Port A as output port, Port B as input Port, Port C as o/p port. What is the control word value.

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0
8				2			

Value is 82H.

03/10/17

Interfacing I/O Ports

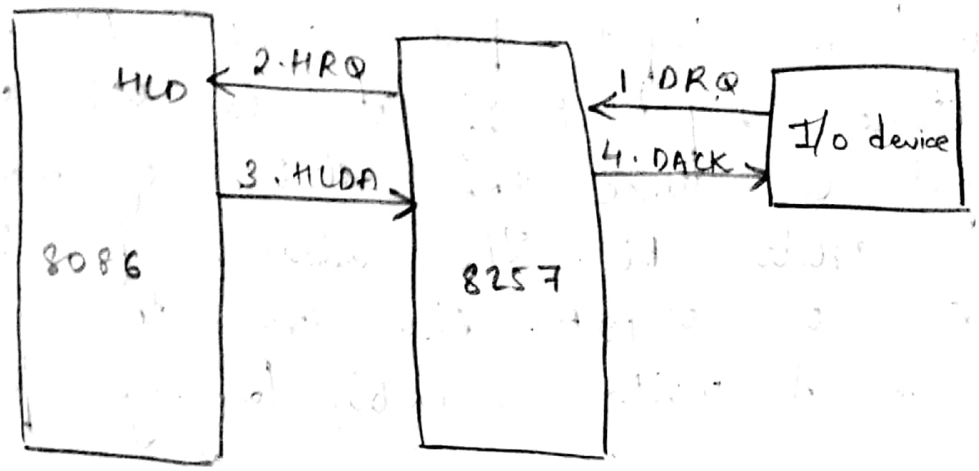
1) Memory - Mapped I/O
   Memory & I/o share same address space.
2) I/O - Mapped I/O

8257 DMA Controller

- We can connect 4 requesting devices
- We have DRQ and $\overline{DACK}$ for each device.
- For each channel, we have 16-bit address and count registers
  - Address - starting address of memory
  - Count - No. of data (bytes) to be read/written.

* 8257 has 10 registers in total.

Operation Cycle



8279 (Programmable Keyboard/Display Interface)

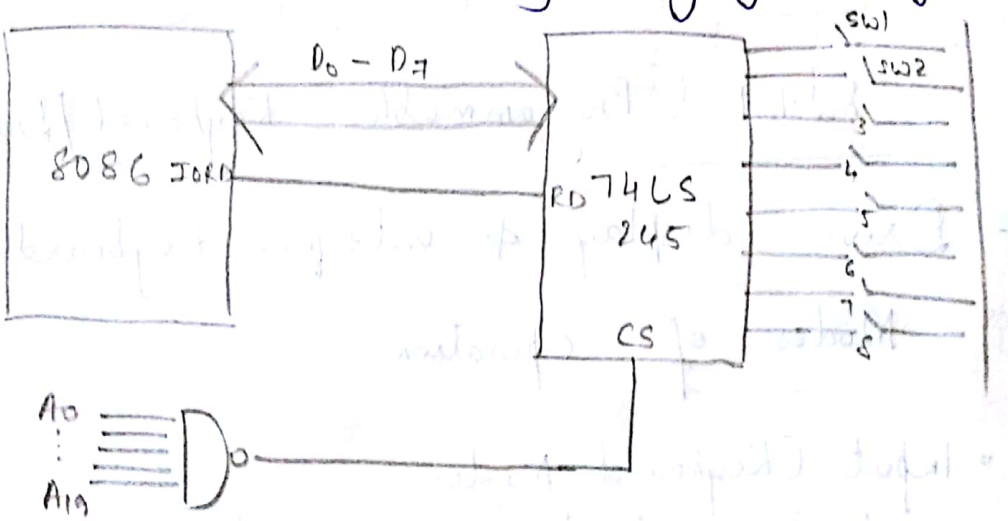
- Drive display & interface keyboard simultaneously

Modes of Operation

- Input (Keyboard) Modes
  1. Scanned Keyboard Mode:
     We can connect 8x8 keyboard or 4x8 Keyboard
     * Value is stored
     - 2 key lockout
       - The last released is stored in FIFO
     - N key rollover
       - All are stored in FIFO.
  2. Scanned Sensor Matrix.
     64 bit corresponding to each key
  3. Strobed Input.
- Output (Display) modes
  1. Left Entry Mode (Notebook)
  2. Right Entry Mode (Calculator)

04/10/17

Q.a) Interface 8 switches 1, 2, ... 8 (chip name) with 8086 using an input port 74LS245. The switches input 1 if it is ON otherwise input 0. Store the status of switches in register BL. The address for the port is 0740H. Also draw the interfacing diagram & write assembly language program.



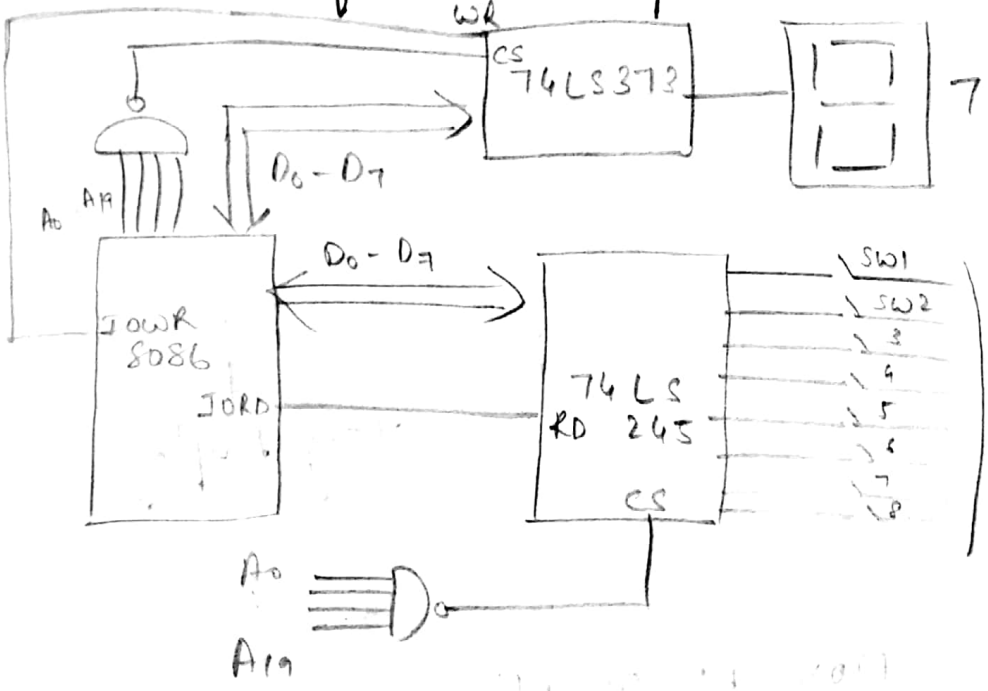
Port 0740

① Initialize / Activate Port
② Read Port

```
MOV  BL, 00H
MOV  DX, 0740H
IN   AL, DX
MOV  BL, AL
```

contin

Q. b) Interface an output port 74LS373 to the port number 000A and using that output port, connect a 7 segment display that displays the number 1 to 8 based on the number of switches pressed.

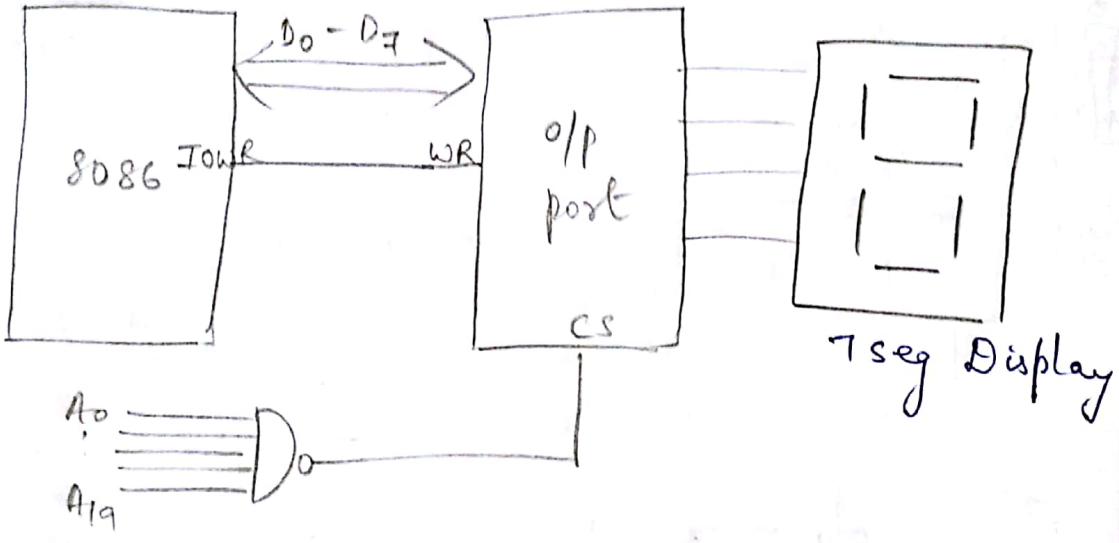



```
MOV  BL, 00H
MOV  DX, 0740H
IN   AL, DX
MOV  BL, AL
MOV  CL, 01H
MOV  DX, 000AH
XX: RCR  BL
    JC   display
    INC  CL
    CMP  CL, 08h
    JNZ  XX
    JMP  exit
display: MOV AL, CL
    OUT  DX, AL
    JMP  XX
Exit:
```

Eg. If BL = 00100100 then we need to display 3 & 6.

```
XX: RCR  BL
    JNC  XY
    MOV  AL, CL
    OUT  DX, AL
XY: INC  CL
    CMP  CL, 08h
    JNZ  XX
```

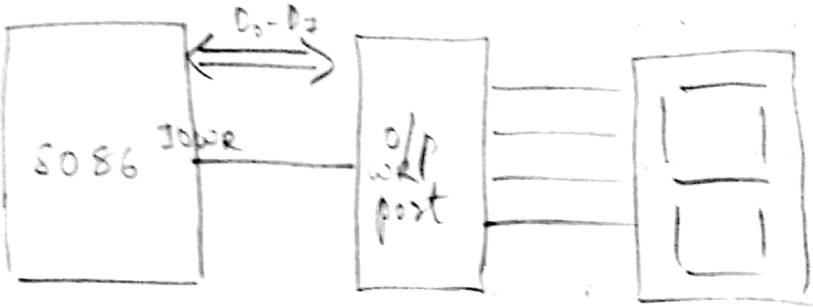
Q. Using an o/p port & 7 segment display design a counter that displays 0 to 9 with 1sec delay between the display. Select a port address suitable.




```
        MOV  DX, 000BH
        MOV  CL, 00H
XX:     MOV  AL, CL
        OUT  DX, AL
        CALL delay for 1ms
        INC  CL
        CMP  CL, 0AH
        JNZ  XX
```

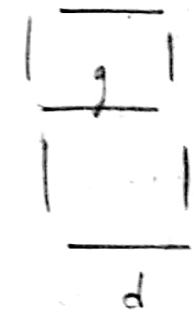
```
Delay MACRO
      MOV CX, count
L1:   NOP
      DECCX
      JNZ L1
ENDM
```

8. Through a hardware circuit for interfacing 7 segment display with 8086 using o/p port. that display the numbers 1 to 5. The 7 segment codes corresponding to these numbers are stored at 2000:0000H onwards (Here data is in memory)



7 seg. display

2000:0000 code for 1
2000:0001 code for 2
:0002 3
:0003 4
:0004 5



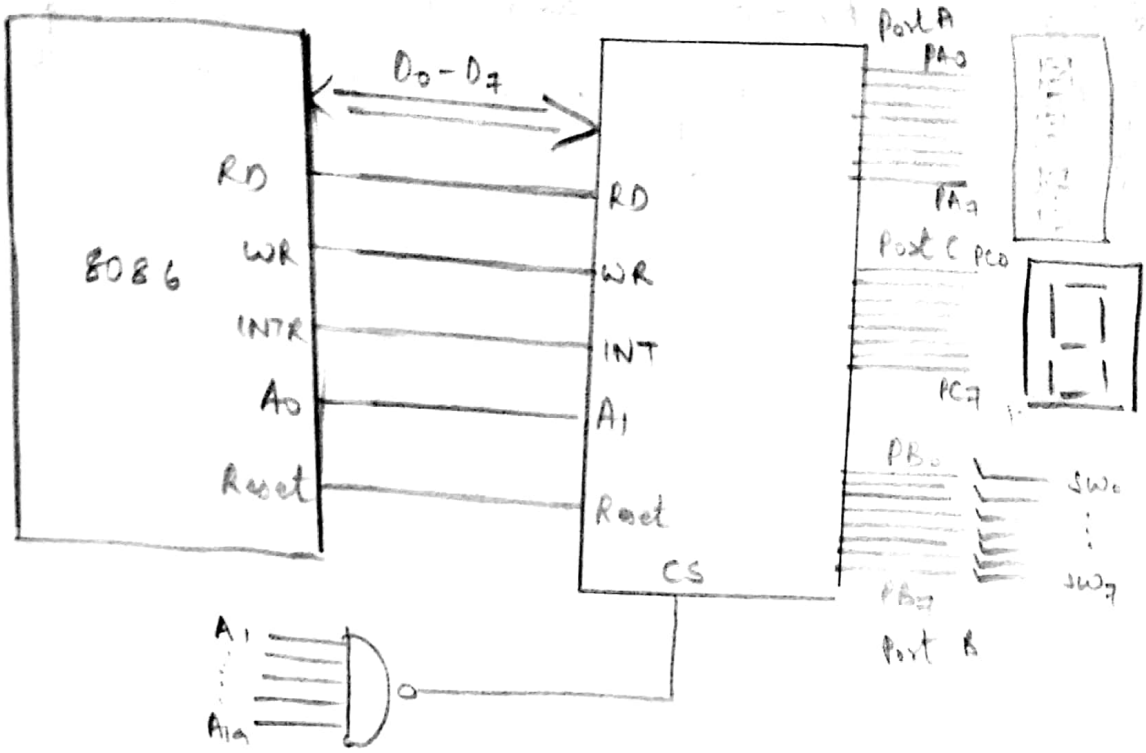
	a	b	c	d	e	f	g	ground	
(1)	0	0	0	0	1	1	0	0	(0C)
(2)	1	1	0	1	1	0	1	0	(
(3)	1	1	1	1	0	0	1	0	
(4)	0	1	1	0	0	1	1	0	
(5)	1	0	1	1	0	1	1	0	

07/10/17

```
        MOV   AX, 2000
        MOV   DS, AX
        MOV   BX, 0000
        MOV   CL, 05H
        MOV   DX, Port Address
XX:     MOV   AX, [BX]
        OUT   DX, AX
        INC   BX
        DEC   CL
        JNZ   XX
```

Interfacing 8255

Q. Interface 8255 to 8086 to work as I/O port. Initialize A as o/p port, port B as i/p port & port C as o/p port. The Port A address is 0740, port B is 0742 and Port C 0744. Write a program to sense the switch positions S0-S7 connected to port B. And the sense pattern to be displayed on port A. to which 8 LEDs are connected. While the port C display, the number of ON switches. Draw the hardware. The port address of 8255 is 0746.



Control word

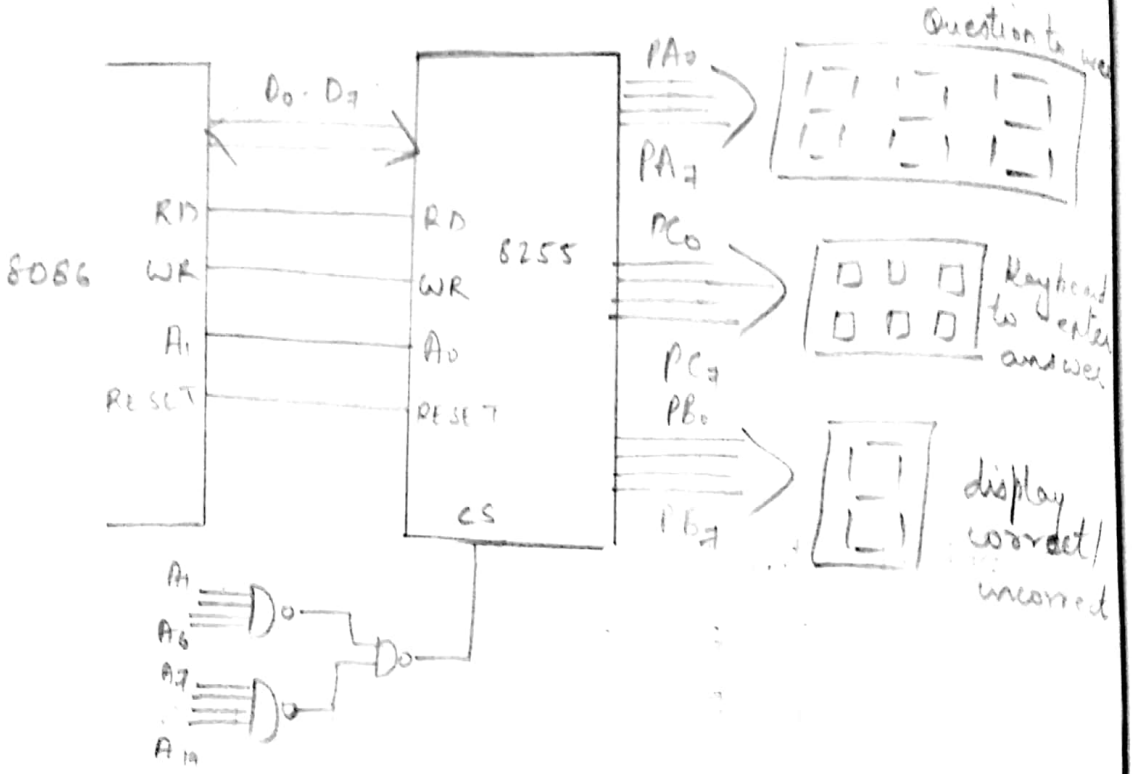
7	6	5	4	3	2	1	0	
1	0	0	0	0	0	1	0	= 82

```
MOV   DX, 0746
MOV   AL, 82H      } 8255 Initialization
OUT   DX, AL

MOV   DX, 0742     } Read port B
IN    AL, DX

MOV   DX, 0740     } Write to port A
OUT   DX, AL
MOV   CL, 08H      MOV BL, 00H
yy:  RCR   AL
     JNC   XX
     INC   BL
XX:  DEC   CL
     JNZ   yy
     MOV   DX, 0746
     MOV   AL, BL
     OUT   DX, AL
```

Q. Design a hardware and write ALP for a question-answering system for 8086 using 8255.



Assume CS: Code DS: Data

Data Segment

Question DB "What is the lowest prime no.? $"

Correct DB "Your answer is correct $"

Incorrect DB "Your answer is incorrect $"

Data Ends

Code Segment

Start : MOV AX, Data

MOV DS, AX

MOV DX, 8255 port

MOV AL, 81

OUT DX, AL

MOV DX, port A address

LEA AX, offset Question

OUT DX, AX

```
MOV DX, Port C Address
IN AL, DX
MOV on port C address
CMP AL, 02H
JNZ xx
MOV AX, offset Correct
OUT DX, AX
JMP xx
xx: MOV AX, offset of Incorrect
    OUT DX, AX
yy:
```

Write a program to generate delay of 100 ms using 8086 system that runs on 10 MHz frequency.

```
                    Clock Cycle
MOV CX, Count  —— 4
Loop: NOP      —— 3
DEC CX         —— 2
JNZ loop       —— 16
```

Loop instructions = 16 + 2 + 3
= 21

(T_d) Required delay = 100 ms

(1 iteration) Total loop duration = 21 × Clock period

(n iteration) Total time = n × 21 × clock period

$$n \times 21 \times T = T_d$$

$$n \times 21 \times \frac{1}{f} = 100\ ms$$

$$n \times 21 \times \frac{1}{10 \times 10^6} = 100 \times 10^{-3}$$

$$n = \frac{10^6}{21} = 47619$$

∴ Here count = 47619

To reduce count:

```
        MOV  CX, count
XX:     NOP        — 3
        NOP        — 3
        NOP        — 3
        NOP        — 3
        DEC CX     — 2
        JN2 XX     — 16
```

09/10/17

Module - V

Microcontroller

- A system on a small chip.
- It is a processor with memory and I/O.

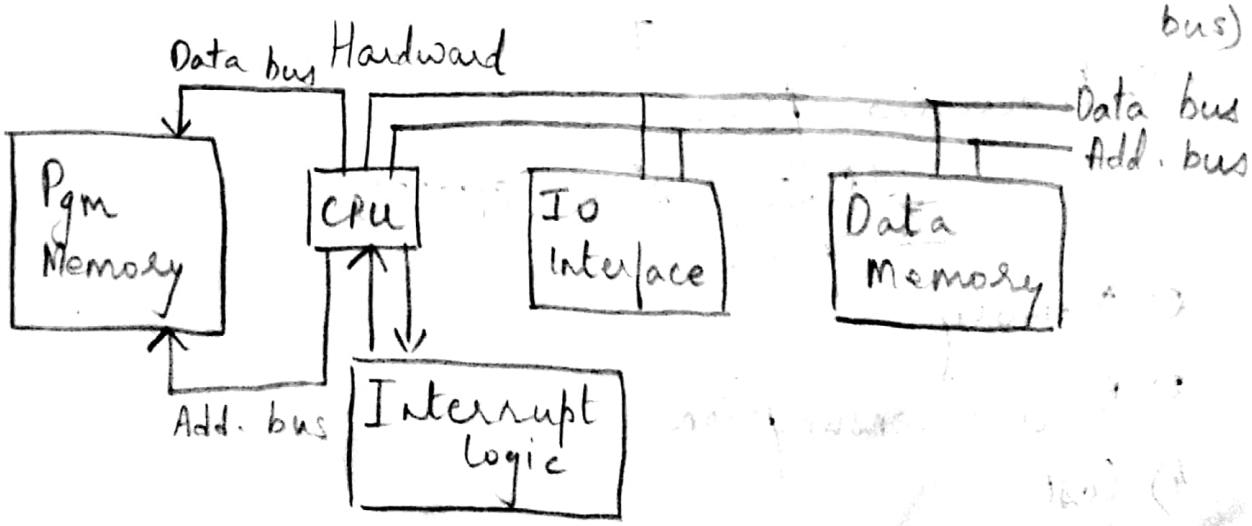
Type / Classification.

1) Bit
   - → 8 bit
   - → 16 bit
   - → 32 bit

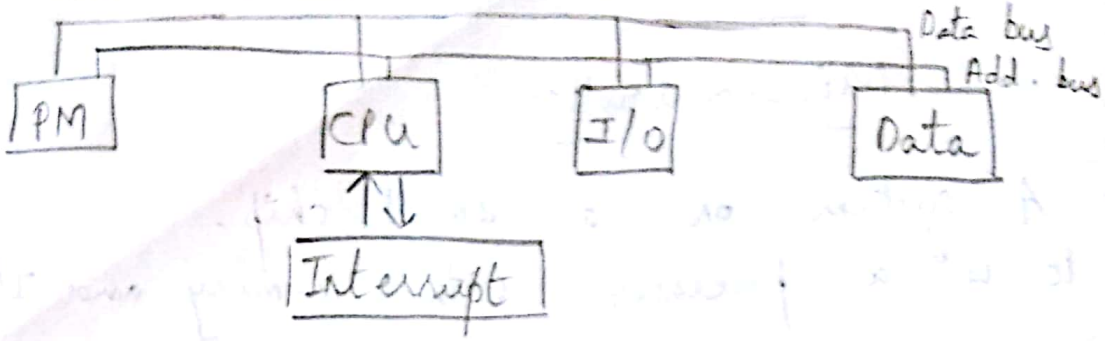
2) Memory Position
   - → External memory
   - → Embedded memory

3) Instruction format
   - → CISC
   - → RISC

4) Architecture
   - → Hardward
   - → Von NewMan / Princeton (common Add. & Data bus)



Von NeuMann



Applications

Raj Kamal

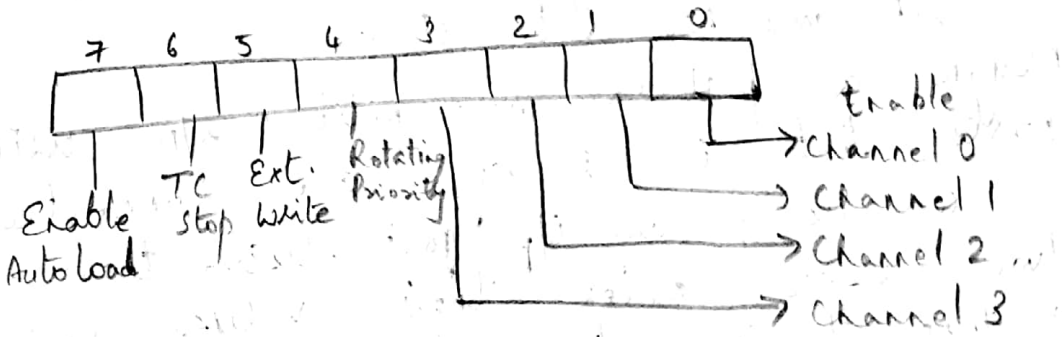
1) Alternate Energy
   Eg: Solar cells
2) Electronic Devices
   Eg: TV, AC, remote etc.
3) Industrial
   Eg: Temp. sensor, pressure sensor
4) Medical
   Eg: ECG, blood preasure meter, Ventilator
5) Smart grid

Selection of Microcontroller

1) Processing power
   → Data size (8/16/32)
   → speed
2) Memory
3) Power consumption
4) Cost
5) Availability

6) Online Support
7) I/O capability
- How many I/O devices can be connected to MC.

9. Interface a DMA Controller 8257 with 8086 so that the channel 0 DMA address register has I/O address 80. The mode set register has the address 88H. Initialize 8257 with normal priority, TC stop & non-extended write. The transfer takes place using channel 0. Write an ALP to move 2kB of data from a peripheral device to m/m address 2000:5000



Here we get 0100 0001
4 1

41

```
MOV  AL, 41H
MOV  DX, 88H
OUT  DX, AL
MOV  AX, 2000
MOV  DS, AX
MOV  DX, 80H
MOV  AL, 50
OUT  DX, AL
```

Channel 0 -
80 Address Register
81 Count Register

Channel 1
82
83

Port Address : 8bit at a time

```
MOV  AL, 00        } DX ← 5000..
OUT  DX, AL        }
MOV  DX, 81H       find hexadecimal
MOV  AL, upper(2kB)   of 2kB.
OUT  DX, AL
MOV  AL, lower(2kB)
OUT  DX, AL
```

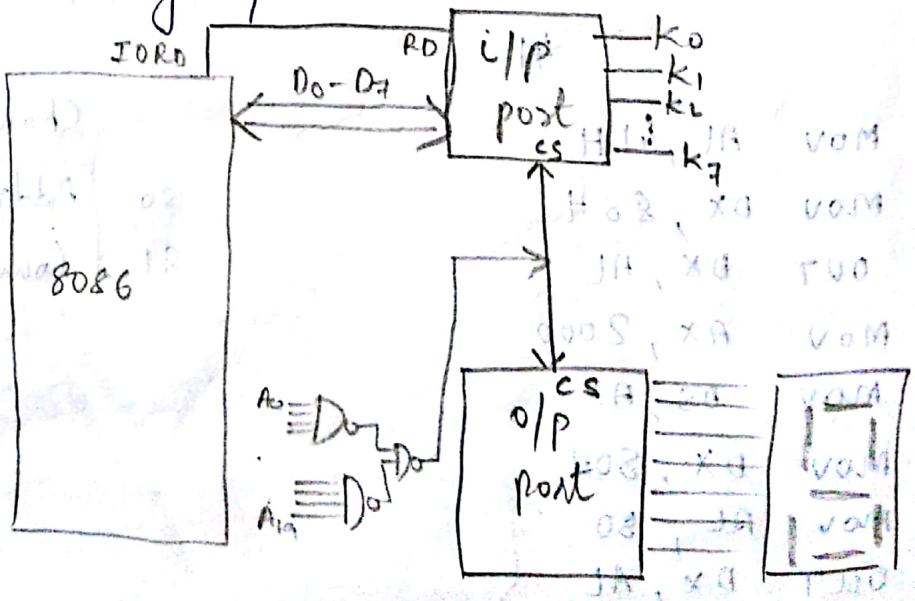
(Only initialization needed. No read/write)

10/10/17
mod 3.

Q. Explain the mechanism used in 8086 to return the program status after an interrupt execution.

Before executing the ISR, we save the CS, IP and PSW to stack.

Q. Design a pattern recognition s/m using I/O ports. That is 8 keys are connected to 8086 using an i/p port. Using these keys, two users enter their values one after another. If they have the same pattern, the 7-seg display displays the same pattern otherwise wrong pattern.



```
ASSUME CS: Code DS: Data
Data Segment
     wrong db "wrong"
     same db "same"
Data ends
Code Segment
     mov DX, #input port
     IN AL, DX
     MOV BL, AL
     IN AL, DX
     XOR AL, BL
     CMP AL, 00
     JNZ displaySame
     MOV DX, #outport
     mov AX, offset wrong
     OUT DX, AX
     JMP XX
displaySame: MOV DX, #outport
     MOV AX, offset same
     OUT DX, AX
XX:
```

20/10/17

Resources (different parts) of Microcontroller

1) Memory
   - Program Memory
     - Instructions that are to be executed
     - ROM
   - Data Memory
     - Both read & write allowed
     - RAM

2) I/O ports
   To communicate with external devices.

3) Serial Ports

4) Interrupt Control

5) Oscillator
   - Generate clock signals.

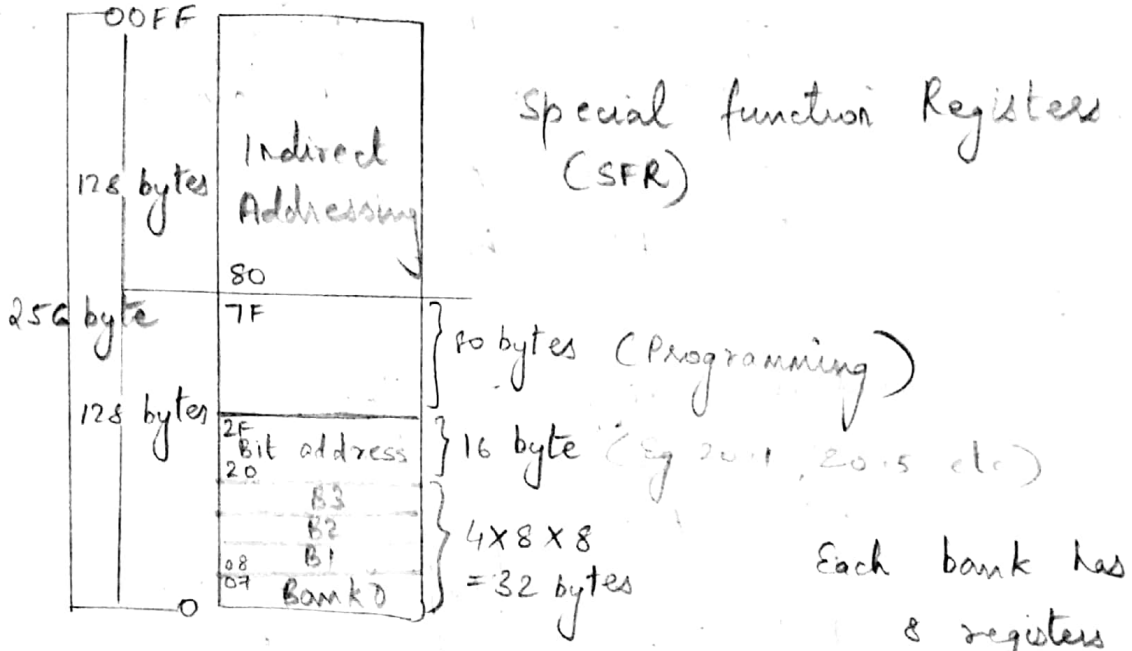
6) Timer/Counter.

8051

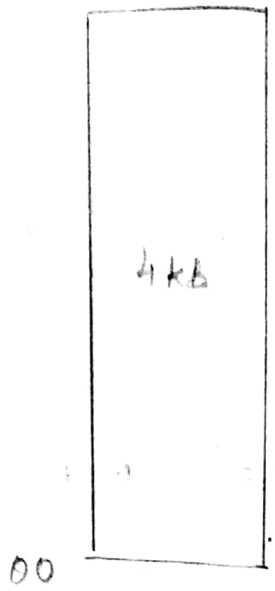
— 40 pin
— 8 bit MC
   → Data bus (8 bit)
   → Address bus (16 bit)
— Hardward Architecture
   - Pgm. memory (ROM) – 4kB
   - Data Memory (RAM) – 256 bytes
     But for programming only 128 bytes available.
- 4 I/O port, each 8 bit.
   - 32 I/O lines
- Two 16 bit up Counters
   T_0, T_1
— Expandable Memory
   - up to 64 kB
— 5 interrupts.

8051 Memory

(a) RAM



(b)



31/10/17

Special Function Registers

To control I/O ports, serial ports, stack etc.

1> ACC (8 bit register)
To hold result & values of operands for different arithmetic & logic operations.

2> B register

- Alternative accumulator.
- Used for multiplication & division along with accumulator.

Eg: MUL AB
DIV AB.

3) PSW

Contains different flags.

CY - Carry
AC - Auxiliary carry
F0 - general purpose register
RS1, RS0 } To select the banks
OV - overflow
P - Parity

4> Stack Pointer

- Register that points to stack top.
- Two operations: Inc & dec.

5> DPTR

- 16 Bit register
- Used to access external memory (data/code).

6> Latches

- To program the I/O ports
- One for each I/O port.

7> Ports

All are bidirectional

PORT 3

- It has 8 extra functions

8> Interrupts
- 5 interrupts in 8051
External Interrupt 0
Timer Interrupt 0
External Interrupt 1
Timer Interrupt 1
Serial Communication

9) Interrupt Priority (IP)
- Set priority of 5 priorities.
1 - high priority
0 - low priority

10> Interrupt Enable (IE)
- Enable & disable interrupts
1 - enabled
0 - disabled.

To control Timers & Counters:

11> TCON

12> TMOD

TCON

7	6	5	4	3	2	1	0
TF_1	TR_1	TF_0	TR_0	IE_1	IT_1	IE_0	IT_0

TR: Timer Run
Enable & disable timer

TF: Timer Overflow
If count goes beyond 2^{16}, we set TF.

IE:
INT1 → IE1
INT0 → IE0

IT (Interrupt Type):
0 - level triggered
1 - edge triggered.

TMOD

7	6	5	4	3	2	1	0
Gate 1	C/T1	M11	M10	Gate 0	C/T0	M11	M10

Timer 1 configuration

Gate 1: for TR1
Clear to enable timer 1 whenever TR1 bit is set.

Gate 0: for TR0

C = Counting mode
T = timing mode

13) SCON (Serial Communication) 14 > SBUF

14) ~~PMOD~~ PCON (Power control)

15) PMOD (Power mode)

01/11/17 SCON (Serial Port Control Register)

Serial Communication
- Bit by bit communication
- To communicate with 8086.

1> SCON
2> SBUF

7	6	5	4	3	2	1	0
FE/SM0	SM1	SM2	REN	TB8	RB8	TI	RI

Enable / Disable

FE/SM0 - To indicate errors in communication.

SM_0	SM_1	Mode	
0	0	0	shift register
0	1	1	8 bit UART
1	0	2	9 bit UART
1	1	3	9 bit UART

Universal Asynchronous transmission.

To set the 9th bit,

TB_8

0 - 9th bit 0

1 - 9th bit 1

RB

If 9th bit = 0, RB = 0

bit = 1, RB = 1

T_1 : After transmitting 8th bit $T_1 = 1$

R_1 : After 8th bit is received $R_1 = 1$

REN : Enable serial ports.

SBUF

Buffer to store data to be send or received.

PCON

7	6	5	4	3	2	1	0
SMOD1	SMOD0	X	POF	GF1	GF0	PD	IDL

$SMOD_1$: To double the mode rate of serial comm^n.

$SMOD_0$ 0 - FE (To select the 7th bit in SCON).

1 - SM_0

POF : Power off flag

To ON & OFF the MC

GF1 } General purpose flags.
GF0

PD : Power down mode

IDL : Idle mode

1 - MC remains idle.

01/11/17

Module - VI

Addressing Modes

1) Implied
   - No operands
   Eg: RLC

2) Register
   MOV R_0, R_1
   $R_0 \leftarrow R_1$

3) Direct
   MOV R_0, 5000
   $R_0 \leftarrow [5000]$

4) Immediate
   MOV R_0, #50
   $R_0 = 50$

5) Indirect
   MOV R_0, @R_1
   $R_1 = 5000$
   $R_0 \leftarrow [5000]$

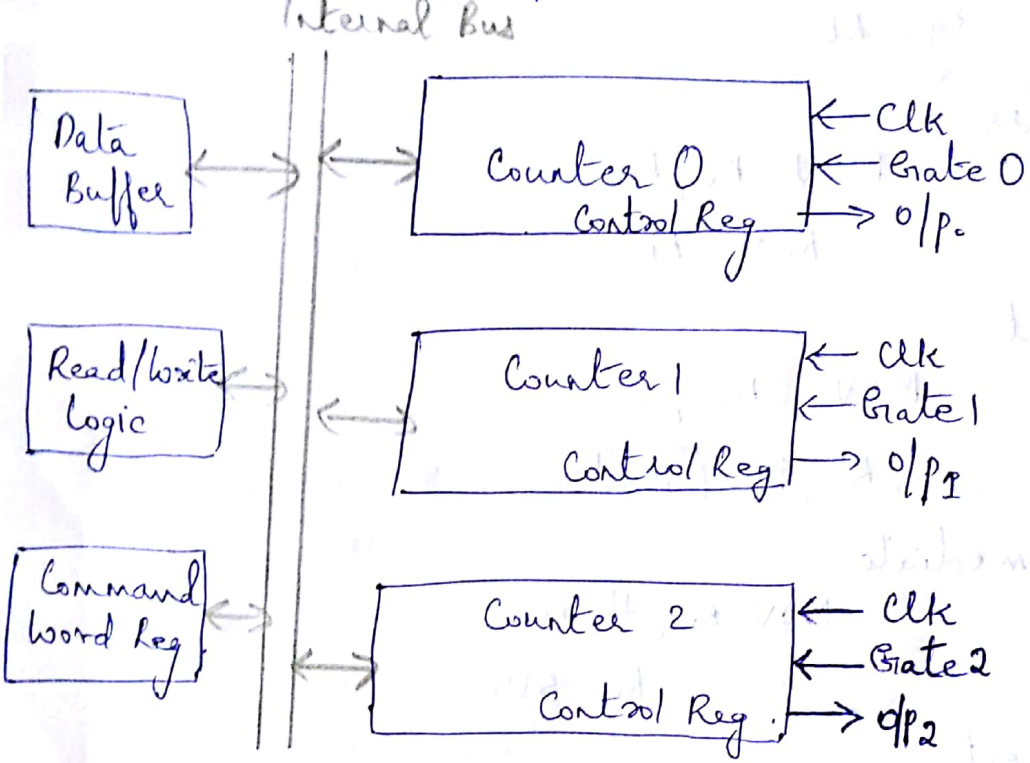
03/11/17

Programmable Interval Timer 8254/8253

- To introduce delays in 8086, we used to write programs but this was not accurate.
- 8254 can include delays which are accurate and can also count.

Architecture

- 3 counters.
- 16 bit down counters.
- BCD/Hex
- No interdependence between each.



Gate: indicate external triggers/ interrupts or for latching the counters

- All counters have local control registers.
- ~~Control~~ Command Word Register:
  Control all operations of the 3 counters.

7	6	5	4	3	2	1	0
SC_1	SC_0	RL_1	RL_0	M_2	M_1	M_0	BCD

00 - C_0
01 - C_1
10 - C_2

0 - BCD
1 - HEX

RL: To read internal registers/counters.

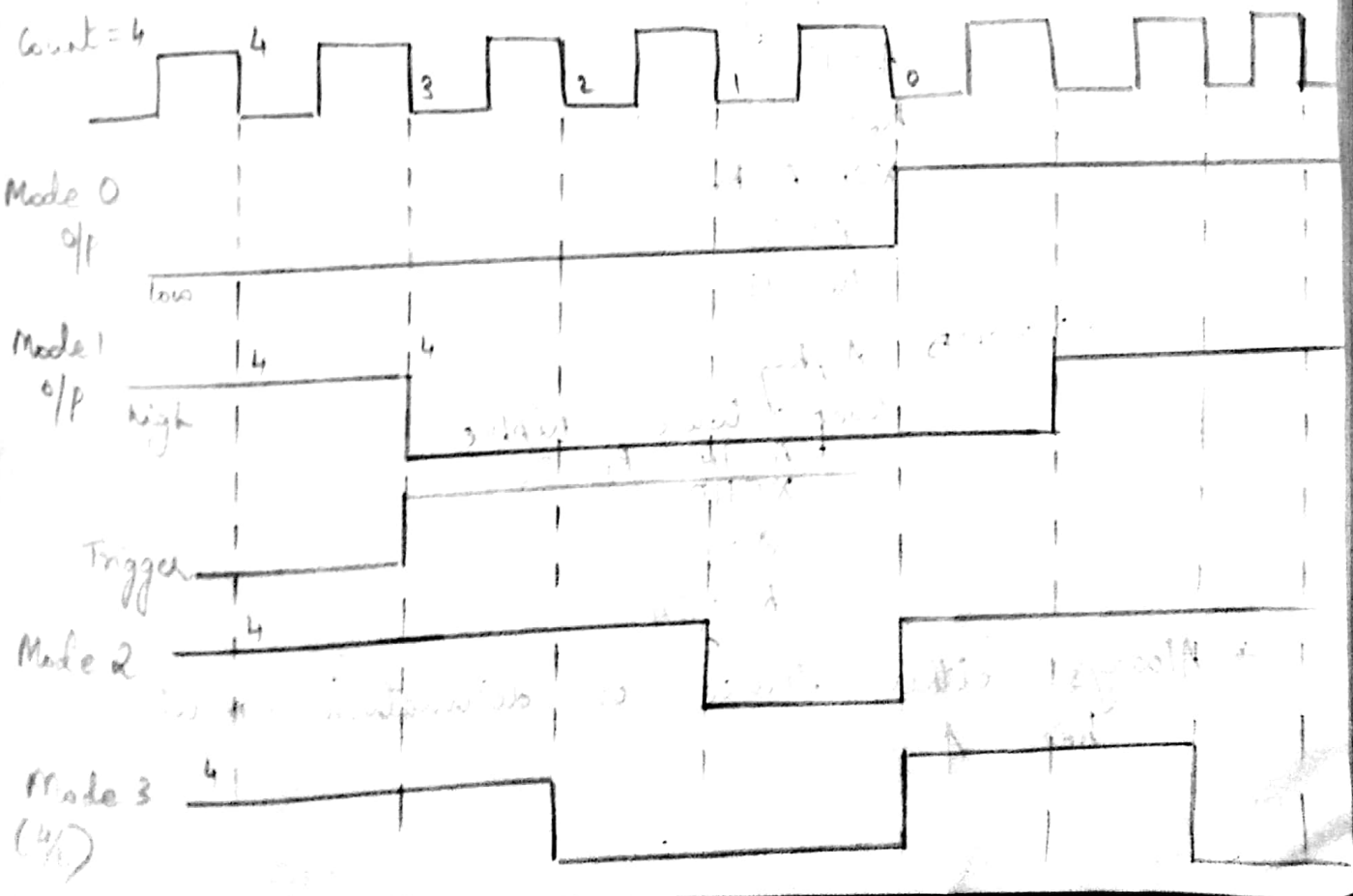
→ RL_1 RL_0

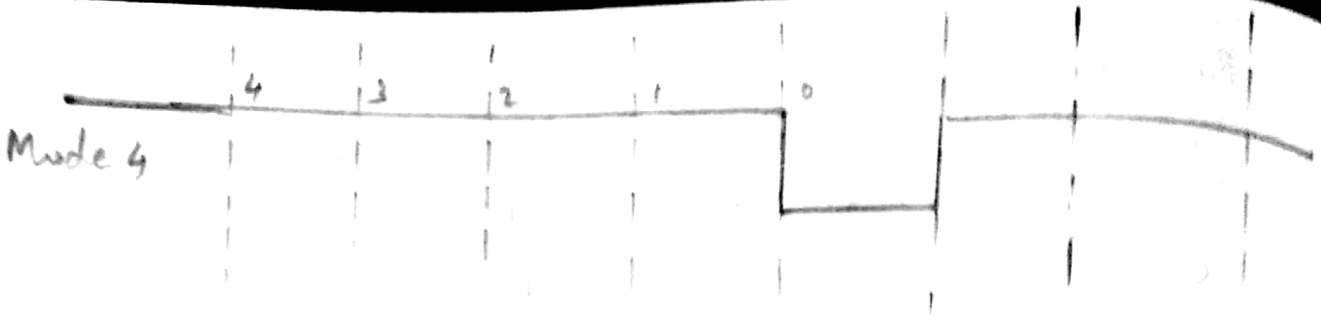
0	0	: Latch Counter "ON THE FLY Read"
0	1	: Read/Load LSB only
1	0	: Read/load MSB only
1	1	: Read/Load LSB then MSB

We can interface using a latch when L is selected counter continues

→ M_2 M_1 M_0

0	0	0	: Interruption TC
0	0	1	: Monostable Multivibrator
0	1	0	: Divide by N counter
0	1	1	: Square wave generator
1	0	0	: s/w trigger
1	0	1	: H/w trigger





06/11/17

8051 Instructions

1. Data Transfer

→ MOV dest, source

→ MOVC

To read/write from external m/m.

→ MOVX

Move from internal to external m/m.

→ PUSH

→ POP

→ XCH A, Reg

A = 11

R_0 = 33

XCH A, R_0

A = 33

R_0 = 11

→ XCHD A, Reg

Swap lower nibble

A = 14 R_0 = 35

XCHD

A = 15

R_0 = 34

* Always either source or destination must be A.

2 Arithmetic

- ADD
- ADC
- SUB
- SUBB
- INC
- DEC
- MUL AB
- DIV AB
- DA A

3. logical

- ANL
- ORL
- XRL
- CLR A
- CPL A
- RL A
- RLC A
- RR
- RRC
- SWAP A

  Swap the nibbles

  4c → c4

4. Bit Manipulation / Boolean

- Destination must be C register
- ANL C, bit
- ORL
- XRL
- CLR c/bit
- CPL c/bit

- SETB c/bit

5. Branch

- JMP
  - SJMP 8 bit address
  - AJMP 11 bit
  - LJMP 16 bit
- CALL
- RET
- IRET
- JC/JNC
- JB/JNB bit address
- JZ/JNZ
- DJNZ/DJZ Reg, location
  - For comparison
  - Decrement Reg & jump to the location based on Z flag value.
- CJNE/CJE Src, dest, location
  - Compare the value of src & destination and jump.

Programs

Q1. Write an 8051 program to add 2 and 3 and store the result in register R_7 and external m/m location FF00

```
MOV  a, #2
MOV  R0, #3
add  a, R0
MOV  R7, a
```

```
MOV  dptr, #FF00
MOVX @dptr, a
```

Q2. Write a program to exchange the content of m/m location FF and FF00.

```
MOV  dptr, #FF00
MOVX a, @dptr
MOV  R0, a
MOV  a, FF
MOV  FF, R0
MOVX @dptr, a
```

Internal m/m upto FF.

a = 22
R0 = 22
a = 11
FF ← 22
dptr ← 11

Q3. Divide the content of R0 by R1 and store the result in R2 and R3. Restore the original content of R0.

```
MOV  A, R0
MOV  B, R1
DIV  AB
MOV  R2, A
MOV  R3, B
MOV  b, R1
MUL  ab
ADD  R3
```

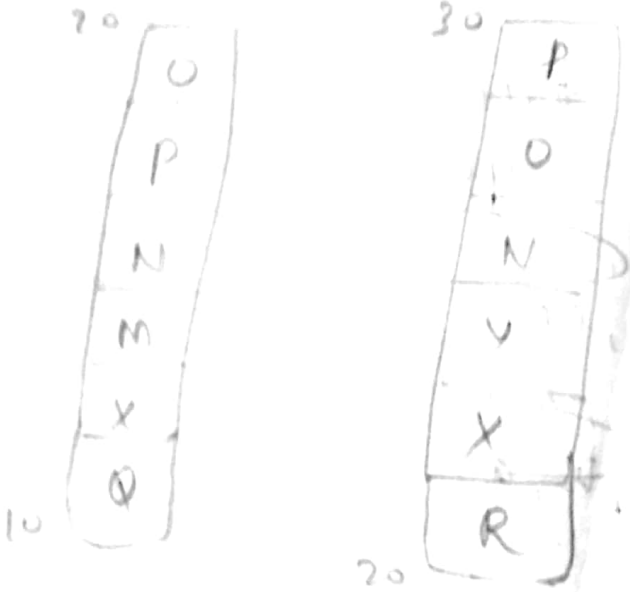
R0 = a * R1 + R3

Q4. Store the higher nibble of R_7 to both the nibbles of R_6.

Q5. Transfer a block of data stored in location 20H to 30H to an external location 1020H.

1030H.

Q6. Find out the number of locations that contains equal values between two blocks 10H - 20H and 20H - 30H.



4.

MOV a, R7.