(12)

port A(8) + port C (Upper)(4) (PC4 - PC7)

port C (lower)(4) (PC0 - PC3)

→ 8255 operated in two modes.

→ BSR mode → Bit set-Reset.

→ IO mode.

BSR → used to enable (or) disable the (8 pins) of Pins

IO mode → port A / port B / port C can be programmed as i/p, o/p.

→ Mode 0 (Simple I/O) (PA, PB, PC)

→ Mode 1 (Handshaked I/O) (PA, PB), (PC → used as Hand

→ Mode 2 (Bidirectional I/O)(PA), (PB & PC → used as Handshak

→ Control port decides BSR mode (or) IO mode.

8255 - PPI Architecture:

→ Read/write Control logic block generates control signals to a the blocks

→ Group A → Port A
→ Port C (upper)

→ Group B → port C (lower)
→ port B

→ Data bus (8-bit) from D0-D7 generates control signals for GrpA & Grp B.

Scanned by CamScanner

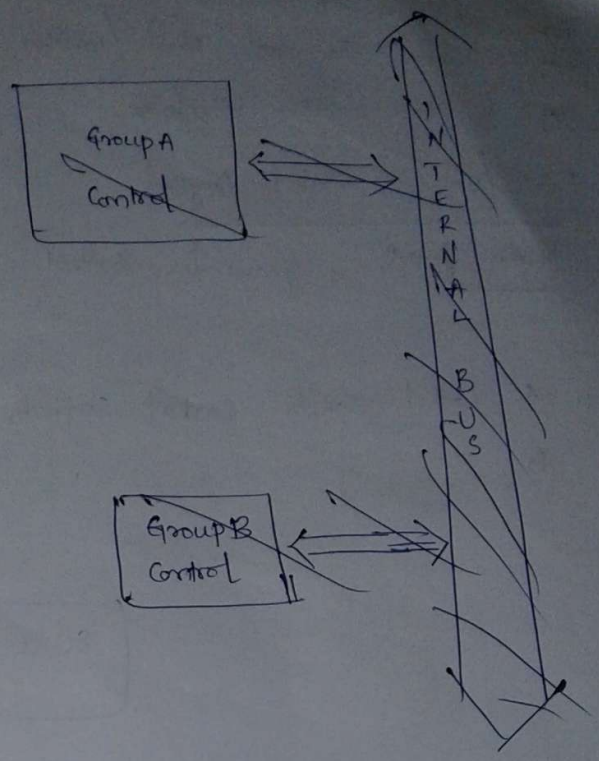
pins
pins) of Port'c'

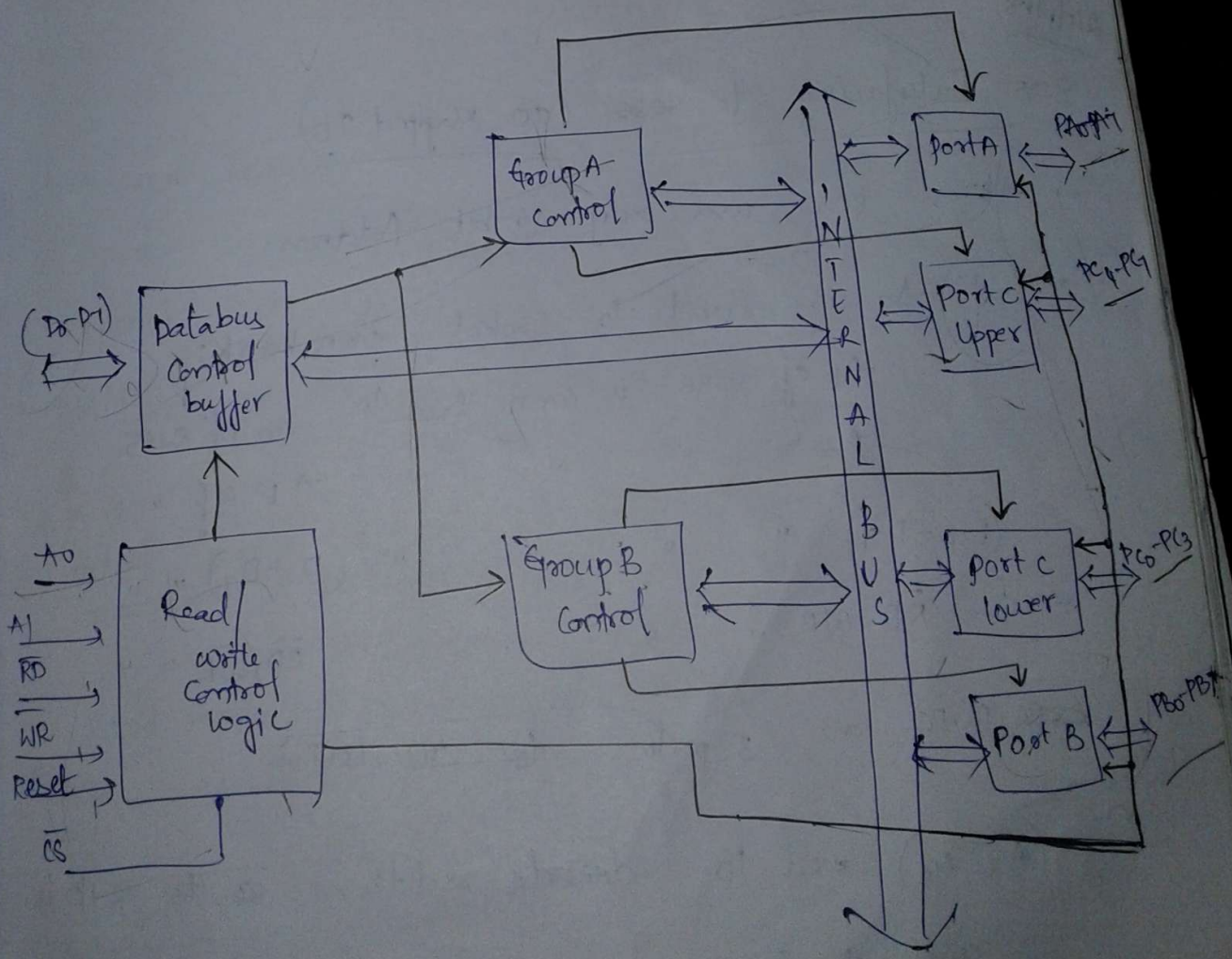
xed as handshaked signals)

→ used as handshaked)

als to all

c (lower)
B





Scanned by CamScanner

→ There are 8 blocks

→ The data to be read / written must be placed on D_{0-7}

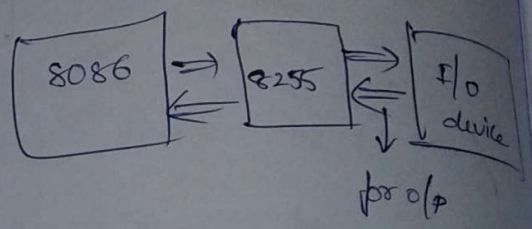
→ It is an 8-bit interface.

Read / write Control logic:

Main func^n: generates control logic to read/write

Group A Control sends Control signals to PA & PC (upper)

Grp B " " " " " PB & PC (Lower)



24/1/18

8255 Interfacing to 8086: (IO Mapped IO)

→ Here; 8086 uses only 16-bit Address.

→ ∵ A_0 is used to select even bank;

A_1 of 8086 is connected to A_0 of 8255

A_2 " " " " A_1 of "

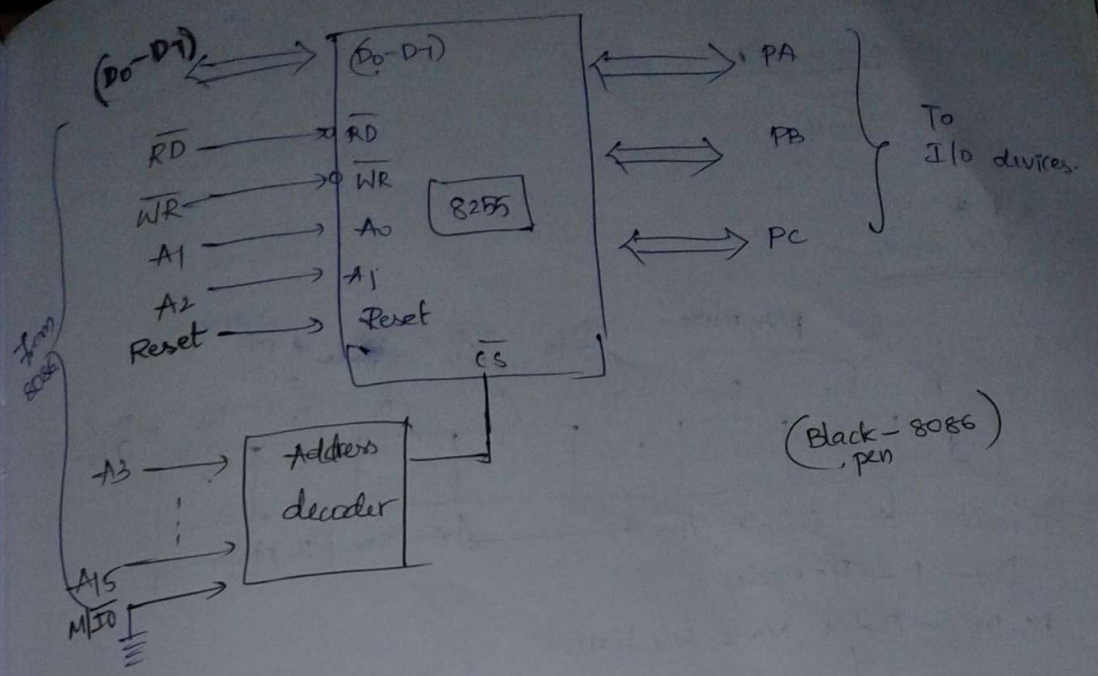
(D_0-D_7) " " " " (D_0-D_7) " "

$\overline{RD}, \overline{WR}$ " " " " $\overline{RD}, \overline{WR}$ " "

→ PA, PB, PC are 3 ports to IO devices.

→ (A_3-A_{15}) used to decode address & it is given to $\overline{CS}$.

Scanned by CamScanner



→ Address Decoder is used to enable the chipselect ($\overline{CS}$) signal.

then, Data on ports are transferred to (D_0-D_7) of 8086.

→ $M/\overline{IO} = 0$ ⇒ IO operatn. (I/O devices)

*** Control word formats of 8255:

- BSR mode
- IO mode

(i) BSR :- Bitset-Reset

D7							D0
0	x	x	x				

BSR Mode — Not used — PC pin selection (3-bits) (from 000 - PC0 to 111 - PC7) — Set/Reset.

; if $D_0 = 1$ → Set
if $D_0 = 0$ → Reset

Ex:

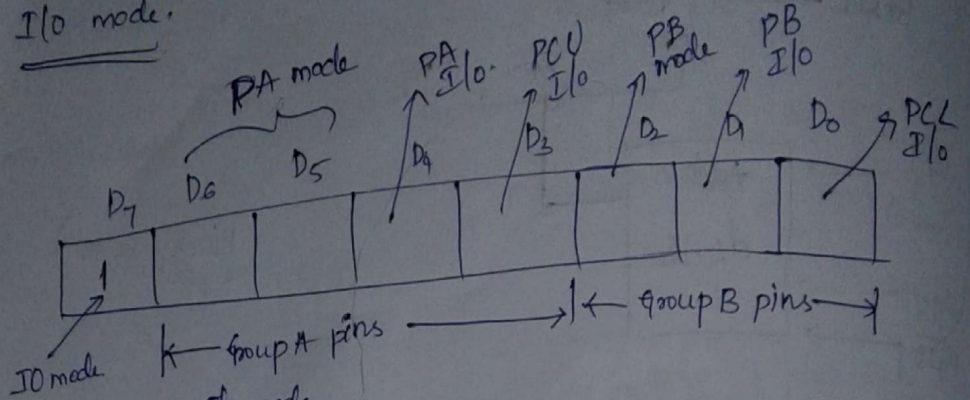
if Set PC3

$\Rightarrow \frac{0000}{0} \frac{0111}{7}$

$= 07H$

***

(ii) I/O mode.



$D_7 - 1 \rightarrow$ I/O mode.

D_6, D_5 — Port A Mode selection.

00 → 00 — mode 0
01 → 01 — mode 1
10 → 1 X — mode 2.
11

D_4 – port A – I/O selection.

→ 0 – o/p.
→ 1 – i/p

(∵ Port C is operated on only mode-0)

D_3 – Port C (upper) – I/O selection

→ 0 – o/p
→ 1 – i/p

D_2 – Port B mode selection.

→ 0 – mode 0
→ 1 – mode 1

D_1 – port B IO Selection

→ 1 – i/p
→ 0 – o/p

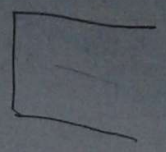
D_0 – Port C (lower) IO Selection

→ 1 – i/p
→ 0 – o/p.

Scanned by CamScanner

Address

Ex: PA – 8000H
PB – 8002H
PC – 8004H
Control port – 8006H.



⇒ Ex: PA – 8884H
PB – 8886H
PC – 8888H
CP – 888AH

Write a program to set PC_5 & Reset after some delay, continously and the address of
PA – F000H.
PB – F002H
PC – F004H
CP – F006H

Set PC_5

0	x	x	x	1	0	1	1

⇒ $\underbrace{0000}_{0}$ $\underbrace{1011}_{B}$

⇒ 0BH.

25/1/18

```
Up: MOV  DX, F006H
    MOV  AL, 0BH      } set PC5
    OUT  DX, AL

    CALL DELAY.
```

0	x	x	x	1	0	1	0

⇒ $\underbrace{0000}_{0}$ $\underbrace{1010}_{A}$

⇒ 0AH.

```
    MOV  DX, F006H
    MOV  AL, 0AH      } Reset PC5
    OUT  DX, AL

    CALL DELAY
    JUMP UP
    Int 03H.
```

DELAY

```
up: MOV  CX, FF00H
    DEC  CX           } (or) loop up
    JNZ  up
    RET
```

Write (Give the CWR format) Program to configure three I/O ports of 8255 as mentioned below.

PA – 8801H
PB – 8803H
PC – 8805H
CP – 8807H

(i) PA – mode 1 as i/p
(ii) PB – mode 0 as o/p
(iii) PC as i/p. (PCU & PCL as i/p)

Scanned by CamScanner

I/O

D7	D6	D5	D4	D3	D2	D1	D0
1	0	1	1	1	0	0	1
PA mode			PA I/O	PCU I/O	PB mode	PB I/O	PCL I/O.

1011 1001

B 9

⇒ B9H.

```
MOV DX, 8807H.
MOV AL, B9H.
OUT DX, AL.
Int 03H.
```

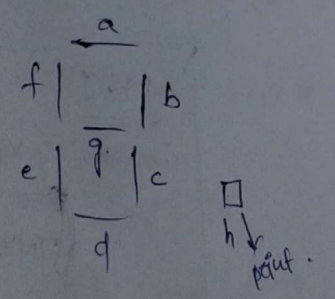
I/O Devices Interfacing:

I/O Devices are interfaced to 8086 through 8255.

(Keyboard, ADC, Stepper Motor, DAC, Displays)

* Displays:

7-segment Display:



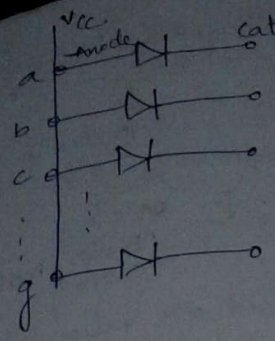
PA – 8800H	8801H
PB – 8802H	8803H
PC – 8804H	8805H
CP – 8806H	8807H

- Common Anode Method → if 7 LED's anodes are shorted.
- Common Cathode Method → if 7 LED's cathodes are shorted.

Common Anode Methodes:

If all Anodes are shorted.

Scanned by CamScanner



Common Anode.

	g	f	e	d	c	b	a	Hexa
0 →	0	1	1	1	1	1	1	3FH
1 →	0	0	0	0	1	1	0	06H
2 →	1	0	1	1	0	1	1	5BH
3 →	1	0	0	1	1	1	1	4FH
4 →	1	1	0	0	1	1	0	66H
5 →	1	1	0	1	1	0	1	6DH
6 →	1	1	1	1	1	0	1	7DH
7 →	0	0	0	0	1	1	1	07H
8 →	1	1	1	1	1	1	1	7FH
9 →	1	1	0	1	1	1	1	6FH

→ Common Cathode is complement of the truth table.

Common Cathode.

	g	f	e	d	c	b	a
0 →	1	0	0	0	0	0	0
1 →	1	1	1	1	0	0	1
2 →	0	1	0	0	1	0	0
3 →	0	1	1	0	0	0	0
4 →	0	0	1	1	0	0	1
5 →	0	0	1	0	0	1	0
6 →	0	0	0	0	0	1	0
7 →	1	1	1	1	0	0	0
8 →	0	0	0	0	0	0	0
9 →	0	0	1	0	0	0	0

Scanned by CamScanner

Write a program to interface 7-segment Display using Common anode method.

O/P device

(i) o/p device

(ii) CWR – for PA o/p, PB o/p, PC o/p } mode 0

send CWR value → CWR

(iii) send 3FH → PA.

delay

send 06H → PA

delay

⋮

PA – 8800H
PB – 8802H
PC – 8804H
CP – 8806H } o/p ports.

27/1/18

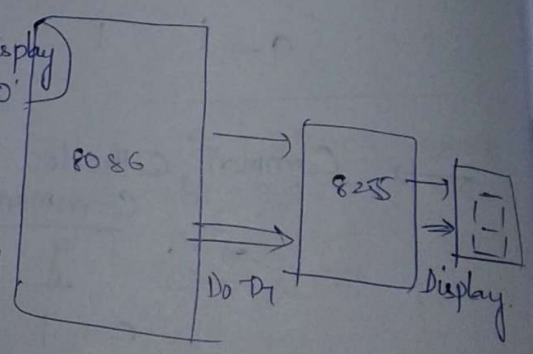
CWR → IO.

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

8 ⇒ 80H

```
MOV   DX, 8806H
MOV   AL, 80H      } 8255 act as o/p.
OUT   DX, AL
```

```
up: MOV  DX, 8800H   → Port A
    MOV  AL, 3FH     } Display '0'
    OUT  DX, AL
    CALL Delay
    MOV  AL, 06H     } Display '1'
    OUT  DX, AL
    CALL delay
    MOV  AL, 5BH     } Display '2'
    OUT  DX, AL
    CALL delay
    ⋮
    up to '9'
```



DELAY:

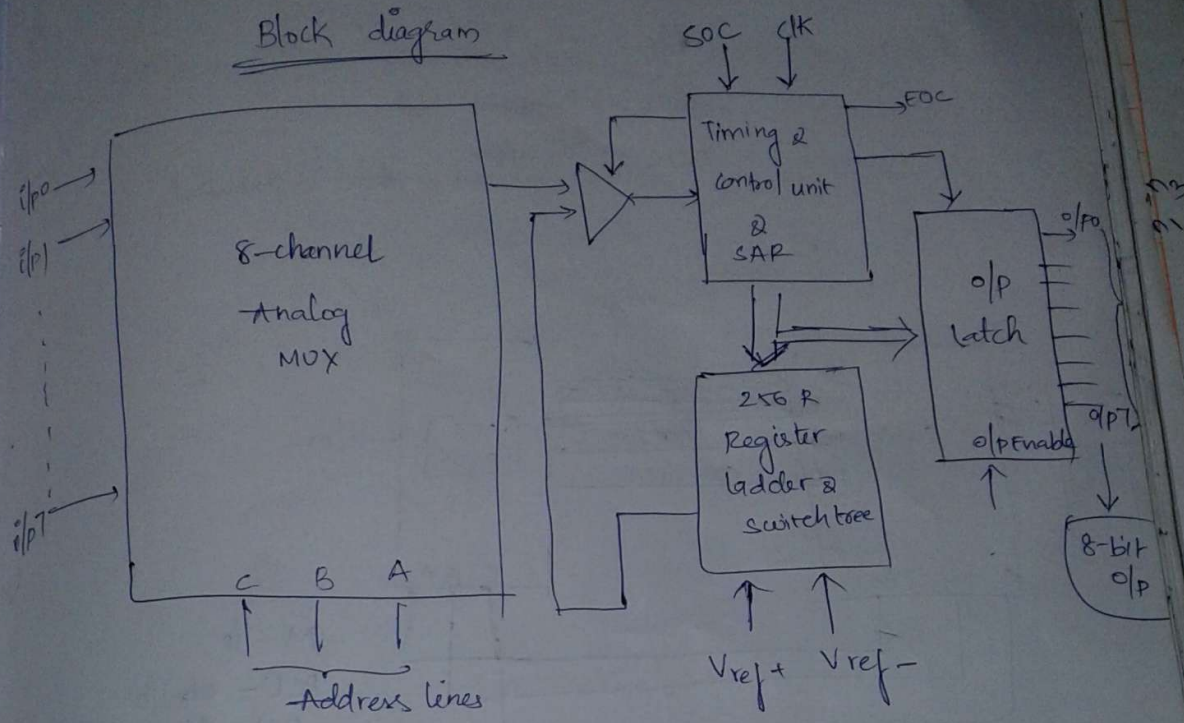
```
up: MOV  CX, FF00H
    Loop up
    RET
```

Jump up
Int 03H

ADC Interfacing:

ADC 0808/0809:

Block diagram

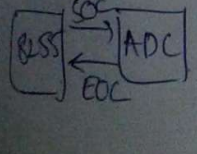


ADC is i/p device.
(∵ As 8086 understands in digital form)

ADC — i/p device
↳ IC – ADC 0808/0809
→ 8-bit ADC.

it has 7-i/p's; ABC are address lines to select the i/p. [Ex: if i/p 4; CBA – 100]

→ SAR – Successful Approximation Register
→ SOC – start of conversion; EOC – End of Conversion.
(8255 sets a signal to ADC i.e, SOC)
Then, after conversion ADC gives EOC signal to 8255)



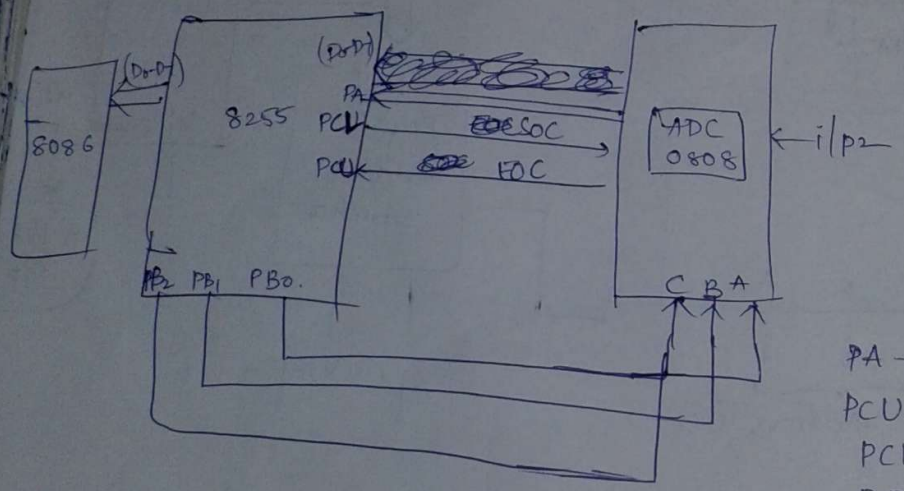
→ Interface ADC 0808 with 8086, using 8255 ; i/p 2 of ADC is used ; PC(upper) acts as i/p port to receive EOC signal while PC(lower) acts as o/p port to send SOC to ADC. PA acts as 8-bit i/p data port to receive digital o/p from ADC.

Program

i/p 2 ⇒ CBA = 010.

SOC → PC_{lower}

EOC → PC_{upper} → 1 ; EOC is completed.



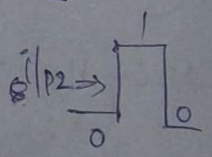
PA – i/p
PCU – i/p
PCL – o/p
PB – o/p.

CWR

1	0	0	1	1	0	0	0

9 8

⇒ 98H.



(i) Initialize 8255 as i/p.
(ii) Send SOC signal to ADC 0808
(iii) check EOC signal using RCL.
(iv) Data is read from PA.

7/2/18
DAC
Memory
Inter
DA
2^8 =
0–2
Maximu

Scanned by CamScanner

1/2/18

DAC Interfacing:

Memory:

DAC

$2^8 = 256$

0-255

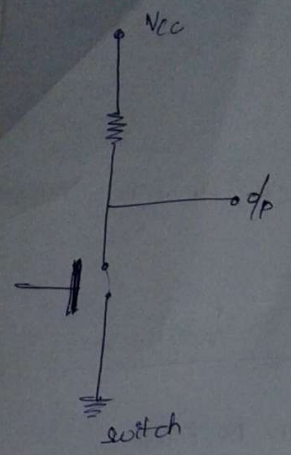
Maximum value using DAC is 00H FFH (255)

To generate → Sawtooth wave

```
MOV DX, 8800H
MOV AL, 80H
OUT DX, AL
MOV DX, 8800H
up2: MOV AL, 00H
     OUT DX, AL
up1: INC AL
     CMP AL, FFH
     JNE up1
     OUT DX, AL
     JMP up2
     int 03h
```

Scanned by CamScanner

Keyboard Interfacing:



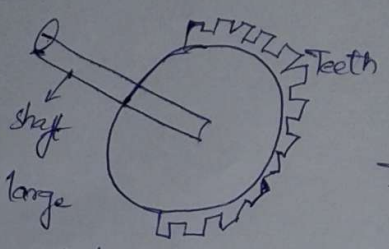
LED's → o/p → 80h

0808/0809 ← ADC – i/p – 90h

0808/0809 ← DAC – o/p – 80h

Stepper Motor – o/p – 80h

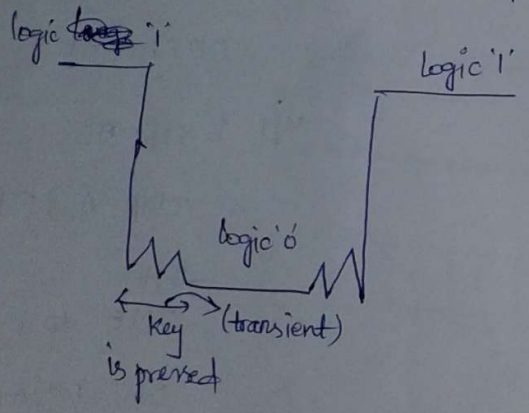
Keyboard – o/p – 90h



$\frac{360^\circ}{\text{teeth}} = 1.8^\circ$

when key is pressed, large '1' changes to large '0' with fluctuations

(fluctuations are due to spring)

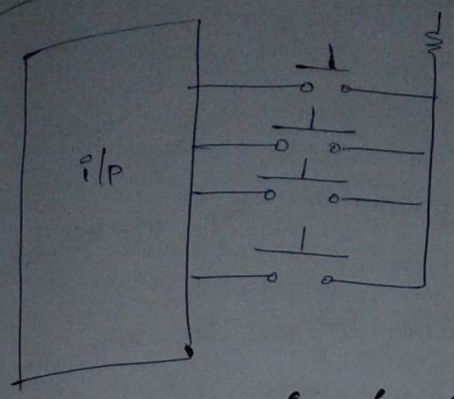


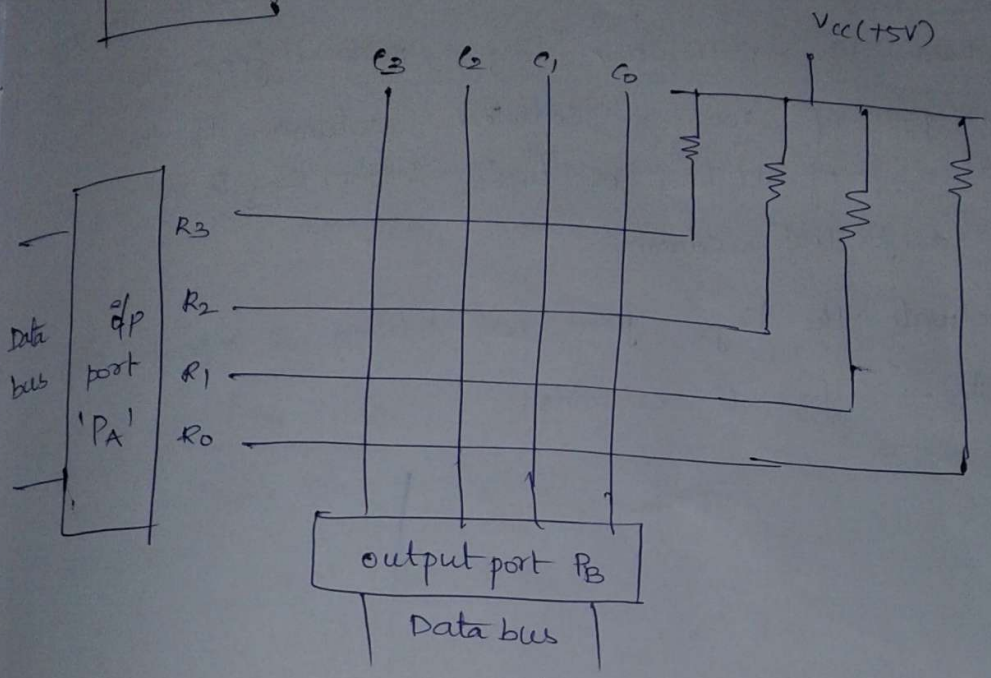
This can be eliminated by debouncing using hardware & software

Key debouncing

- H/W → using flip flop
- S/W → using Delay program.

4×4 matrix Keyboard:





R_3, R_2, R_1, R_0 → Return lines

C_3, C_2, C_1, C_0 → Scan lines

when Key at C_0 is pressed, R_0 is also zero.

Steps to identify the pressed key:

Check-1 (whether any Key is pressed (or) not)

1. Make all the column lines '0' by sending low all o/p lines. This activates all Keys in the Keyboard Matrix.
2. Read the status of return lines. If the status of all lines is logic 'high'. then the Key is not Pressed otherwise Key is pressed.

9/12/18

Memory / Interfacing.

check-2.

1. Activate keys from any column by making any column line zero.

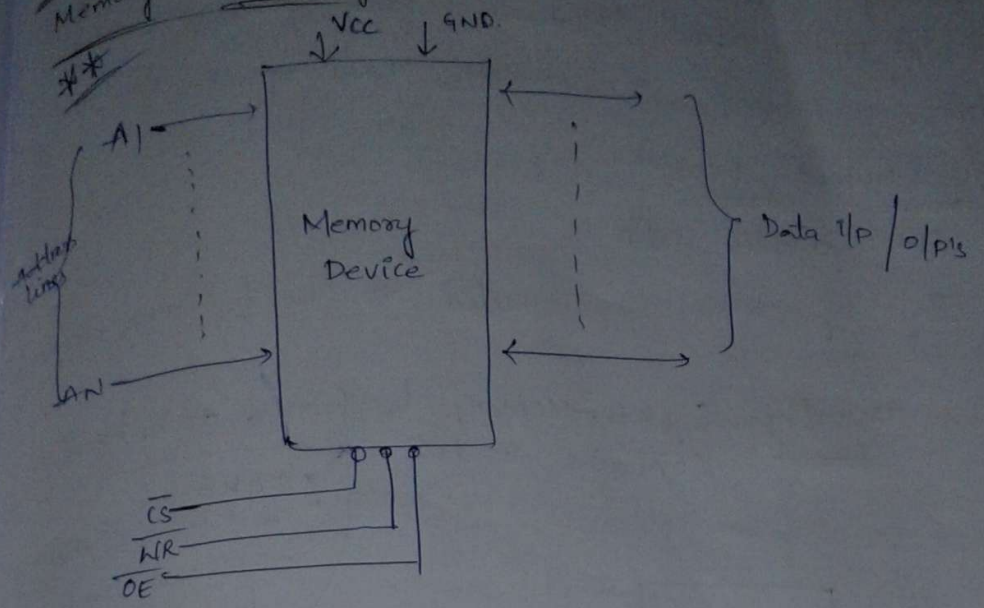
2. Read the status of return lines. The zero in any return line indicates, "Key is pressed" from the corresponding row & selected column. If the status of all lines is logic high, key is not pressed for that column.

3. Activate the keys from next column & repeat step-2 & step-3 for all columns.

Scanned by CamScanner

9/12/18

Memory Interfacing:



Address lines ($A_1 \cdots A_N$):

From address lines; Capacity of Memory device can be known.

Ex: $A_1 \cdots A_{12} \Rightarrow 2^{12} = 2^{10} \cdot 2^2 = 4kB$ → no. of Memory locations are available

Data lines:

In each Memory location; no. of bits stored is known by Data lines.

⇒ Read/write.

It indicates size of data on which read/write is performed.

Chip select: It is used to select the memory device

$\overline{CS} = 0 \Rightarrow$ Memory device is enabled.

$\overline{CS} = 1 \Rightarrow$ " " " disabled.

Scanned by CamScanner

$\overline{WR}$: ~~Write is~~ It is connected to $\overline{WR}$ of memory device.

→ ($\overline{WR}$) It is used to enable write operation

→ $\overline{OE}$ is used to enable read operation.

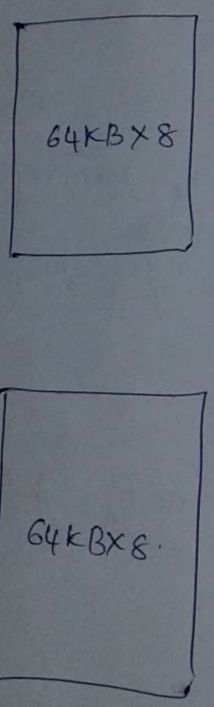
$\overline{OE}$ (output Enable):

It enables $\overline{RD}$.

$\overline{RD}$ of 8086 is connected to $\overline{OE}$ of Memory device

Address Decoding: 64KB-Memory location & each location contains 8 bits.

64KB
$= 2^6 \cdot 2^{10} = 2^{16}$
⇒ 16 address lines to access each Memory device



The address range of 1st memory ~~location is~~ device is 00000H to 0FFFFH

and 2nd memory device address range is 10000H to 1FFFFH

To achieve this generate select lin

→ Address Decoding

(i) Using logic

If

and

A14
A15
A16
A17
A18
A19

(ii) Using

A
B
C

G1
G2
G3

Scanned by CamScanner

To achieve this, CPU address has to be decoded to generate select lines for the memory device.

→ Address Decoding can be done using

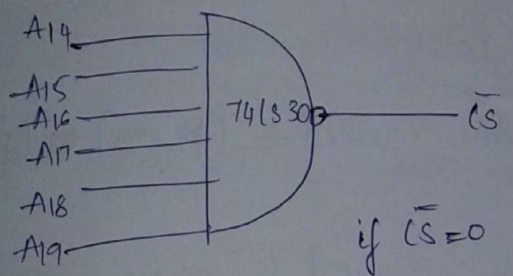
(i) Using logic Gates:

If 16KB
↳ $2^4 \times 2^{10} = 2^{14}$
⇒ 14 address lines
$A_{13} \ldots A_0$ are decoded.

and 16KB X8 is the memory device;

$\underbrace{A_{14} \ldots A_{19}}$
↓
used to select $\overline{CS}$.

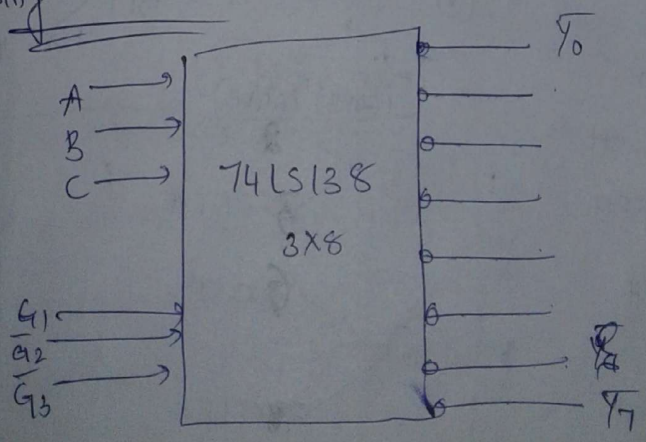
(∵ 8086 → consists of 20 address lines.)



if $\overline{CS} = 0$ ⇒ then memory device is enabled.

⇒ $A_{14} \ldots A_{19}$ → '0'

(ii) Using Decoder:



Scanned by CamScanner

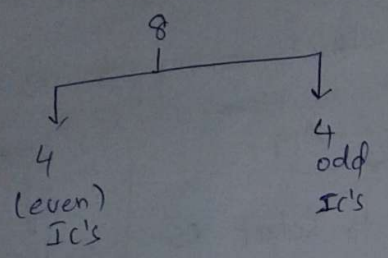
*** S-RAM Interfacing:

Q: Interface 512 kB of SRAM using 64kB × 8. Memory is to 8086 Min^m mode configuration. Starting address is-

(i) Total No. of Memory IC's = $\frac{512K}{64K} = 8$.

⇒ 8 IC's are required.
(4 are even & 4 are odd)

(ii)



(iii) Address Decoding is done by using Decoder.

∵ More no. of IC's.

⇒ No. of Decoders required = 2 (for even & odd)

(iv) No. of address lines are required to access 64kB is

$= 2^6 \cdot 2^{10} = 2^{16}$ ⇒ 16 address lines are required.

S.A = 0000H

F.A = FFFFH

A_{19} A_{18} A_{17} A_{16} A_{15} ---------- A_0

(v) $128kB = 2^7 \cdot 2^{10} \Rightarrow 17$

S.A 00000H

F.A 1FFFFH

even (64kB) odd (64kB)

(vi) Address map:

I Device: S.A = 00000H ; F.A = 1FFFFH

II Device: 20000H ; 3FFFFH

III Device: 40000H ; 5FFFFH

IV Device: 60000H ; 7FFFFH

Scanned by CamScanner

(VI) 50000H — 5FFFFH ; (VII) 60000H — 6FFFFH ;

(VII) 70000H — 7FFFFH ; (VIII) 80000H — 8FFFFH

⇒ Total → 4 S-RAM's (each contains even & odd)
⇒ 4 × 2 = 8 IC's.

12/2/18

→ Address map for even bank:

		A_{19} A_{18} A_{17} A_{16}	A_{15} A_{14} A_{13} A_{12}	A_{11} A_{10} A_9 A_8	A_7 A_6 A_5 A_4	A_3 A_2 A_1 A_0
SRAM I →	00000H →	0000	0000	0000	0000	0000
	1FFFEH →	0001	1111	1111	1111	1110
SRAM II →	20000H →	0010	0000	0000	0000	0000
	3FFFEH →	0011	1111	1111	1111	1110
SRAM III →	40000H →	0100	0000	0000	0000	0000
	5FFFEH →	0101	1111	1111	1111	1110
SRAM IV →	60000H →	0110	0000	0000	0000	0000
	7FFFEH →	0111	1111	1111	1111	1110

$A_0 = 0$ always

→ A_0 is always '0' — to select even Bank.
→ $\overline{BHE}$ " " '0' - to " odd ".

→ A_1 to A_{16} are 16 address lines used.

→ A_{17}, A_{18}, A_{19} are decoder i/p's.

where; A_{17}, A_{18} - i/p's
& A_{19} - GND.

→ Address map for odd bank:

(I) → 00001H to 1FFFFH

(II) → 20001H to 3FFFFH

(III) → 40001H to 5FFFFH

(IV) → 60001H to 7FFFFH

Scanned by CamScanner

						$A_{13}A_{12}A_{11}A_{10}$
SRAM I → 00001H →	0000	0000	0000	0000		0001
1FFFFH →	0001	1111	1111	1111		1111
II → 20001H →	0010	0000	0000	0000		0001
3FFFFH →	0011	1111	1111	1111		1111
III → 40001H →	0100	0000	0000	0000		0001
5FFFFH →	0101	1111	1111	1111		1111
IV → 60001H →	0110	0000	0000	0000		0001
7FFFFH →	0111	1111	1111	1111		1111

Here; $A_0 = 1$ is always '1' (odd bank).

A_1 to A_{16} – address lines.

A_{17}, A_{18}, A_{19} – i/p's for decoder.
↓
GND.

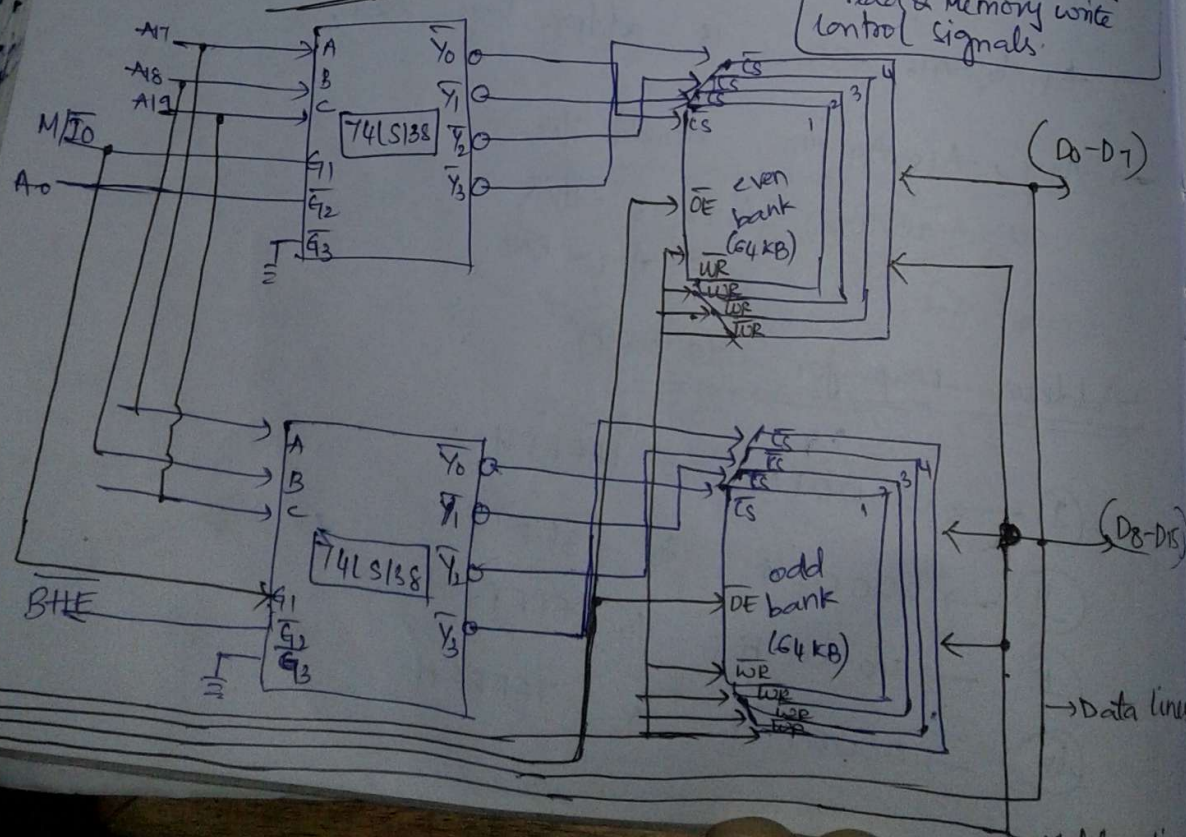
** RAM Interfacing of 8086:

Converting 3X8 decoder into 2X4 Decoder:

From Min^m mode configuration;

For Max^m mode; $\overline{WR}$ is absent & $M/\overline{IO}$ is also absent. there will be Memory read & Memory write control signals.

From 8086; Min^m mode configuration:



Programs are ends at FFFFH of SRAM is 8KB. show the configuration.

Scanned by CamScanner

Programs are stored in EPROM that starts from 8000H & ends at FFFFH. The size of each EPROM is 4KB. The capacity of SRAM is 16KB. Interface at an address of 5000H & also shows the complete memory interface in MMIO mode configuration.

Scanned by CamScanner

15/12/18

(Q) Write a program to interface 8086 which uses EPROM of size 32KB & total size is 128KB

The address range is E0000H to FFFFFH

(i) $\frac{128KB}{32KB} = 4$

(ii) 2 even, 2 odd

(iii) No. of decoders = 1+1 = 2

(iv) $32KB = 2^5 \cdot 2^{10} = 2^{15} \Rightarrow 15$ address lines

(v) $A_0 \rightarrow$ select Even bank

$\overline{BHE} \rightarrow$ select odd bank

$64KB = 2^6 \cdot 2^{10} \Rightarrow 16$

Address map:

(I) E0000H to FFFFFH

even: E0000H to FFFFEH

Odd: E0001H to FFFFFH

I) E0000H to FFFFFH

II) F0000H to FFFFFH

Even bank:

	A_{19}	A_{18}	A_{17}	A_{16}	A_{15}	A_{12}	A_{11}	A_8	A_7	A_4	A_3	A_2	A_1	A_0
EPROM I → E0000H →	1	1	1	0	0	0	0	0	0	0	0	0	0	0
FFFFEH →	1	1	1	0	1	1	1	1	1	1	1	1	1	0
EPROM II → F0000H →	1	1	1	1	0	0	0	0	0	0	0	0	0	0
FFFFFH →	1	1	1	1	1	1	1	1	1	1	1	1	1	1

$A_{16} \rightarrow$ E/P

$A_{16}, \overline{A_{17}}, \overline{A_{18}}$ are E/P's for decoder (3×8)

& A_{19} is used

Odd bank:

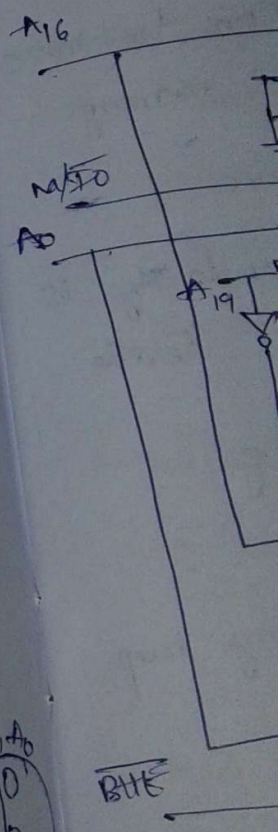
EPROM 3 → E0001H
FFFFFH

EPROM 4 → F0000
FFFFF

$A_1 - A_{15} \rightarrow$

$\overline{A_{18}}, \overline{A_{19}}, \ldots$

(For ROM →



Scanned by CamScanner

& A_{19} is used to select Enable ($\overline{G}$)

odd banks:

EPROM 3 → F0001H → 1110 0000 0000 0000 0001

FFFFFH → 1110 1111 1111 1111 1111

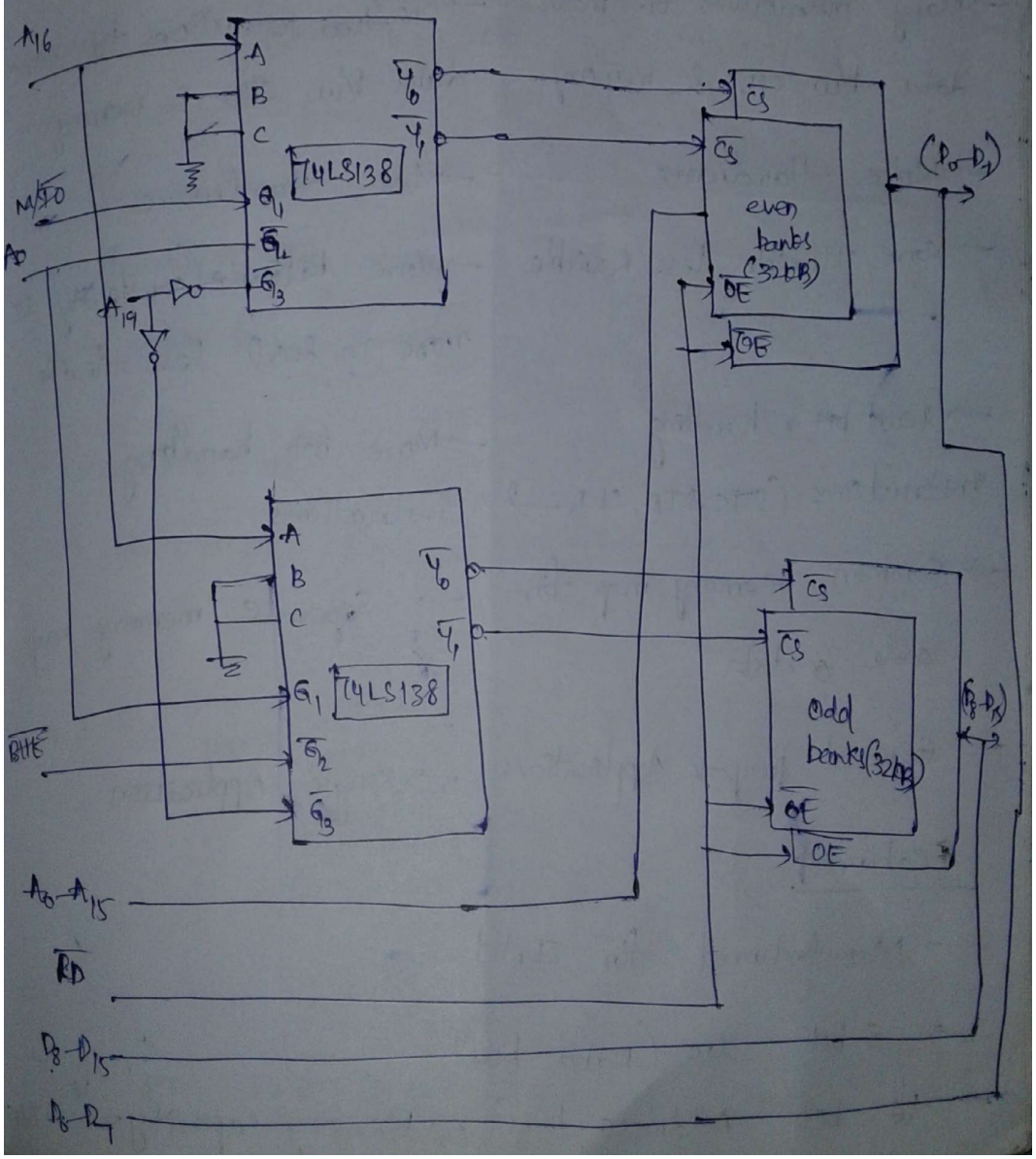
EPROM 4 → F0001H → 1111 0000 0000 0000 0001

FFFFFH → 1111 1111 1111 1111 11111

A_1–A_{15} → Address lines

$\overline{A_{18}}$, $\overline{A_{19}}$, A_{16} – i/p's , A_{17}/A_{18}/A_{19} → $\overline{G_3}$

(For ROM → only $\overline{RD}$ control line)



Scanned by CamScanner

15/2/18

UNIT - IV

8051 MICROCONTROLLER

Microprocessor	Microcontroller
→ contains ALU, control unit Registers	→ contains µprocesor, Memory, Peripheral devices, (RAM/ROM) (ADC/DAC), common devices, Timers/counters, interrupts
→ Access time for memory & IO devices is more	→ less ~~ate~~ access time (∵ they are inbuilt)
→ Many instructions to move data b/w CPU & memory	→ one/two instructions to move data b/w CPU & memory
→ More Hardware	→ less Hardware
→ More Flexible, less Reliable	→ More Reliable (complexity is more in 8051) less flexible
→ Few bit handling instructions (STC, STD, CLD, →)	→ More bit handling instructions.
→ Common Memory map for code & data	→ Seperate memory map
→ General Purpose Applications	→ Single Application

Features:

→ Manufactured for Intel

→ 8 bit µc (Data bus)

→ 16 bit Address bus ⇒ Memory capacity = 2^{16} = 64KB

→ Registers: A, B, ...
All are 8-bit

→ SP (8 bit), PSW (P, CY, ...)
↓
stack Pointer

→ 128 Bytes of
↓
32 Bytes for Register Banks

→ 4KB of

→ 2 Timers (Counters)

→ 3 interrupts
2 enter

→ Operate

→ Int

→ 4 I/

→ 40

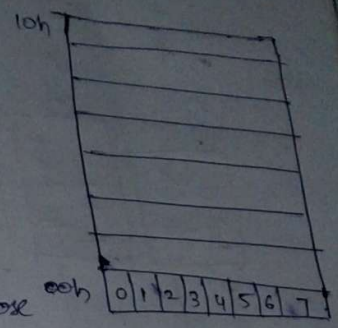
→ Registers: A, B, General Purpose ($R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7$)

All are 8-bit registers except PC (16 bit), DPTR (16 bit)

→ SP (8 bit), PSW (Program Status Word)
(Flag Register)

↓
Stack Pointer

→ 128 Bytes of RAM



- 32 Bytes for Register Banks
- 16 Bytes for Bit Addressable
- 80 Bytes (General Purpose Registers)

→ 4 kB of internal EPROM.

→ 2 Timers (Timer 0 & Timer 1)
(counters)

→ 3 internal, 2 external } interrupts

→ SCON – Serial Control Register
TCON – Timer " "
TMOD → Timer mode Register.

→ Operated at 20MHz

→ Interrupt Enable & Interrupt Priority → External interrupts

→ 4 I/O ports – port 0 (8 pins)
– port 1 (8 pins)
– port 2 (8 pins)
port 3 (8 pins)

→ 40 pin DIP but, simpler than 8086.

Scanned by CamScanner

Architecture of 8051:

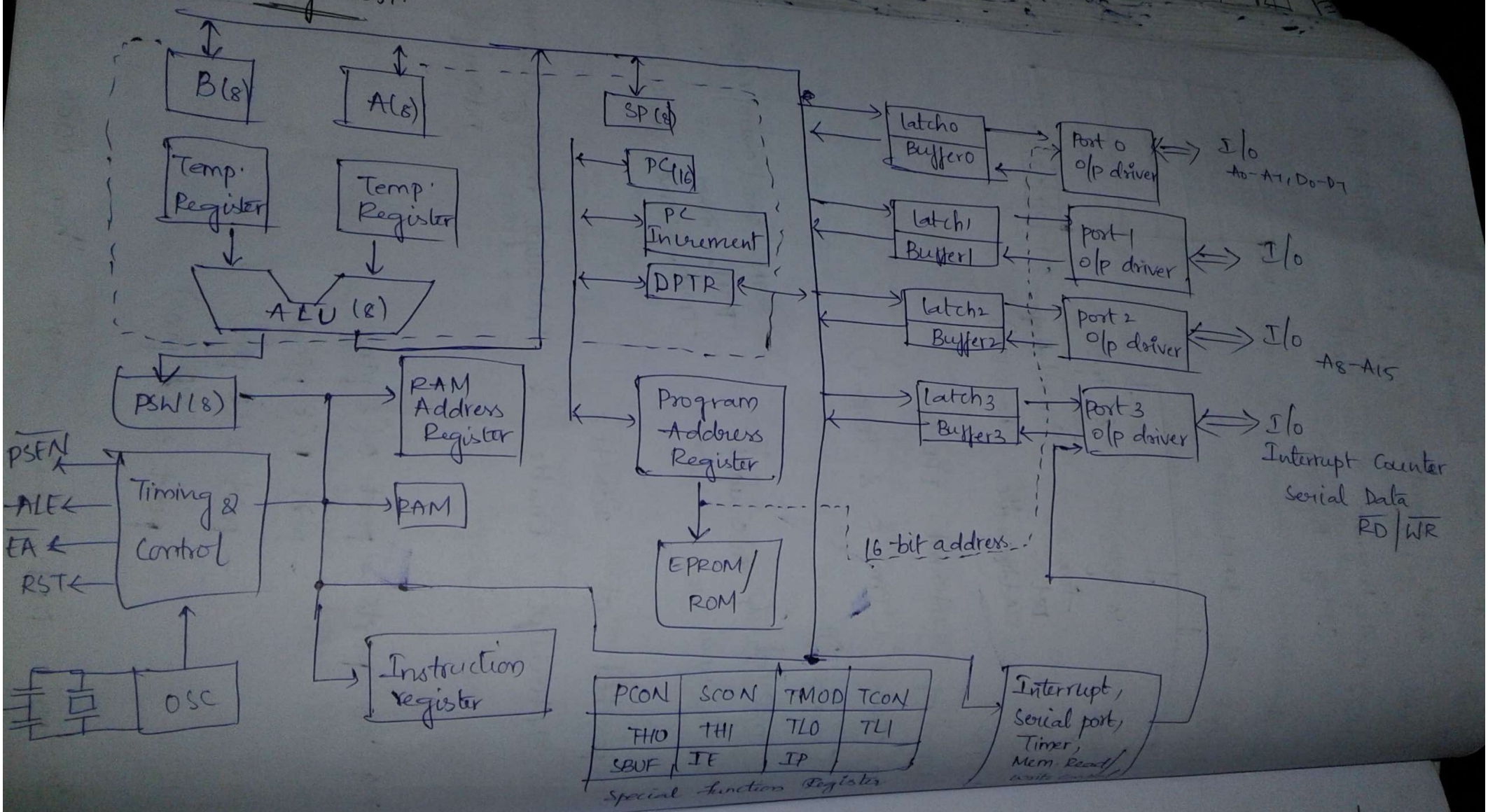


Fig diagram

Oscillator signal

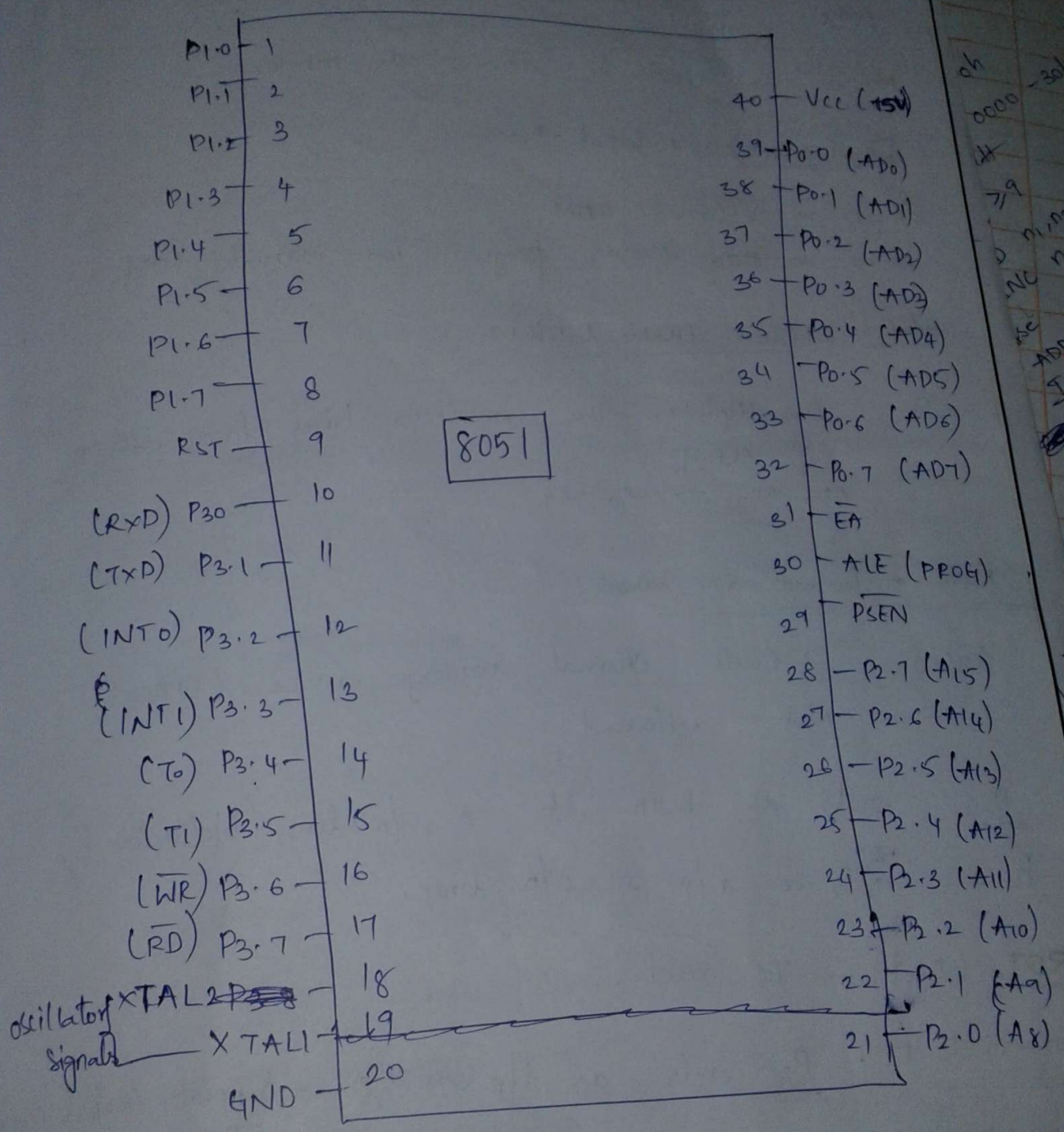
(INT0

(INT1

(TxD

(RxD

Pin diagram of 8051:



16/2/18.

Pin diagram of 8051:

ports : P0, P1, P2, P3 acts as i/p or o/p ports.

port 0 is used to access the memory.

$\overline{EA}$ - (31 pin) : External Access.

$\overline{EA} = 0 \Rightarrow$ external ROM

$\overline{EA} = 1 \Rightarrow$ Access program from internal ROM.

ALE : (Address Enable latch) :

It demultiplexes the address lines from data line if ALE=1

$AD_0 - AD_7 \Rightarrow A_0 - A_7$

$\overline{PSEN}$ - Program store enable.

$\overline{PSEN} = 0 \Rightarrow$ Activate external memory device (EPROM)

$= 1 \Rightarrow$ Not activated.

P_2 — acts as both i/p & o/p lines (bidirectional)

P_1 — acts as i/p or o/p lines.

RST : (Reset) — To Reset.

pin — 10 to 17 : P_3 acts as i/p (or) o/p → Supports serial commⁿ Timers & Interrupts.

→ TxD — Transmit data (serial (data) commⁿ) → pin-10

→ RxD — Receive data (" " ") → pin 11

→ 8051 has UART — Universal Asynchronous Receiver Transmitter

→ pin 12 — INT0 } Interrupts (Interrupt 0 & Interrupt 1)
pin 13 — INT1 }

pin 14 —
pin 15 —
pin 16 — $\overline{WR}$
pin 17 — $\overline{RD}$
pin-18 — } CR
pin-19 —
crystal oscillat

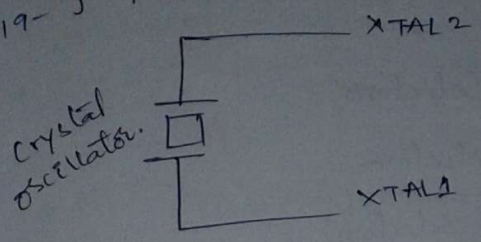
Registers :
A - (8)
B - (8)
DPTR —
PC
PSW

Scanned by CamScanner

pin 14 — T_0
pin 15 — T_1 } Timers

pin 16 — $\overline{WR}$
pin 17 — $\overline{RD}$

pin-18 — } crystal inputs. ⟶ for frequency of 8051
pin-19 —



Registers:

A-(8) — Accumulator — to store results during Arithmetic & logical operations

B-(8) — Base — to store " " Multiplication, & Division.

Remainder — B
Quotient — A

DPTR — (16) — Data Pointer ⟶ high, low.

DPH (8) | DPL (8)

→ used to store 16-bit address.
→ used as index Register.

PC — (16) — Program Counter
→ stores address of next instruction to be fetched.

PSW — (8) — Program status word.
— Flag Register.

7	6	5	4	3	2	1	0
CY	AC	F0	RS1	RS0	O	—	P

P – parity flag – 0 → result contains ~~even~~ odd no. of 1's – odd parity
↳ 1 → " " even " " – even "

O – overflow → 0 – ~~no sig.~~
→ 1 – sign changes
if Sign of result ~~is~~ changes.

RS1 & RS0 – Register Bank Selection
→ For 4 Bank Selections

RS1	RS0	Bank Selection
0	0	Bank0
0	1	Bank1
1	0	Bank2
1	1	Bank3

→ Each bank contains 8 registers (r_0 to r_7)

~~SP – Stack Poin~~

F0 → User flag – not used

AC → Auxiliary flag – 0 → No carry
→ 1 → Carry from 3rd bit

CY – Carry flag – 0 – No carry.
→ 1 – Carry from MSB

→ SP – Stack Pointer: PUSH → SP ← SP + 1
POP → SP ← SP − 1

On Reset ; SP ← 07H.
& All 4 ports acts as i/p.

→ (r_0 – r_7) → General Purpose Registers.

→ each 8051 Register has some internal RAM address.

Special funct
* A – E0H
* B – F0H
* PSW – D0H
SP – 81H
DPH – 83H
DPL – 8
* P0 – 80
P1 –
* P2 –
* P3 – B
* IP –
* IE –
→
add
→
→ SFR's
RAM
I/O P
→ B
→

Scanned by CamScanner

Special Function Registers (SFR's):

* A – E0H
* B – F0H
* PSW – D0H
  SP – 81H
  DPH – 83H
  DPL – 82H
* P0 – 80H
  P1 – 90H
* P2 – A0H
* P3 – B0H
* IP – B8H
* IE – A8H

Internal RAM Address.

TMOD – 89H
* TCON – 88H
  TH0 – 8CH
  TL0 – 8AH } – Timer 0
  TH1 – 8DH
  TL1 – 8BH } – Timer 1
* SCON – 98H – Serial control
  SBUF – 99H – Serial Buffer
  PCON – 87H – Power control

→ only PC doesnot have internal RAM address.

→ * → indicates Bit Addressable.

~~in address every bit can be controlled~~

Bit Addressable: each & every bit can be controlled in an address.

→ SFR's are located immediately above the 128 bytes of RAM.

I/O ports: → Each port has i/p buffer.

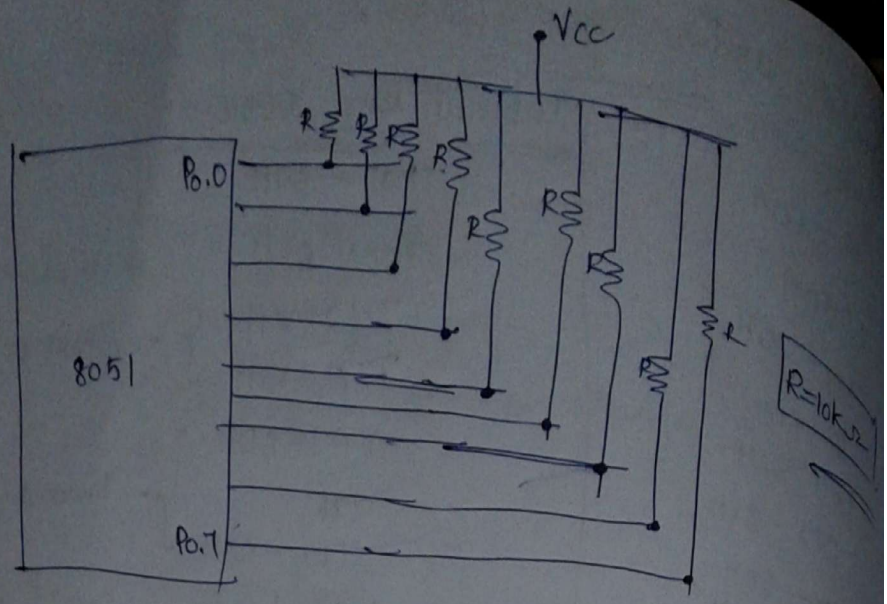
→ P0, P1, P2, P3

→ Bidirectional I/O ports

→ P0 – open drain o/p's
  – use external pull-up ~~registers~~ resistors

→ P1, P2, P3 → have internal pull-up ~~registers~~ resistors.

Scanned by CamScanner



→ on Reset; P_0, P_1, P_2, P_3 → ~~i/p~~ acts as i/p ports

17/2/18

Memory Organization:

Internal RAM (128 B) → working Register (32 B)
→ Bit addressable (16 B)
→ General purpose (80 B)

↓

2^7 → 7 lines are requires
→ In hexadecimal ⇒ 00h to 7Fh

→ we can use only one register bank at a time

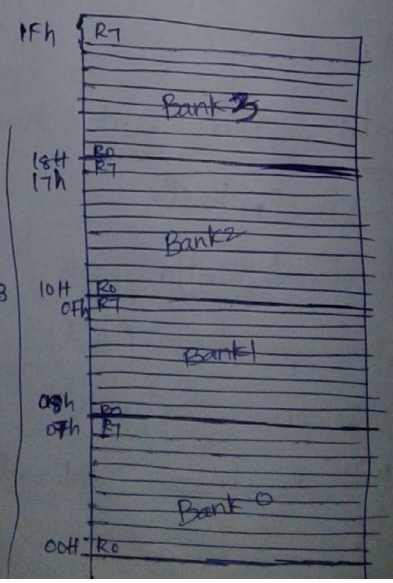
→ on reset bank 0 is selected

→ It is byte addressable

1. Working Register:

It contains 4 Banks

Bank 0, Bank 1, Bank 2, Bank 3



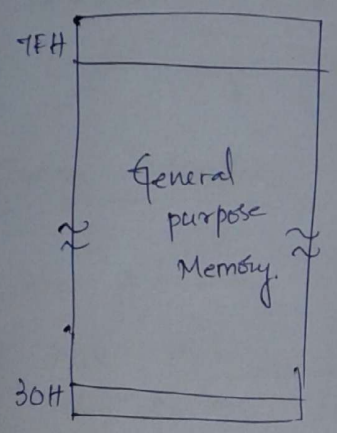
Scanned by CamScanner

2. Bit Addressable:

This will be above working Register.

2F	7F	78H
2E	77	70H
2D	6F	68H
2C	67	60H
2B	5F	58H
2A	57	50H
29	4F	48H
28	47	40H
27	3F	38H
26	37	30H
25	2F	28H
24	27	20H
23	1F	18H
22	17	10H
21	0F	08H
20	0F	00H

3. General Purpose:

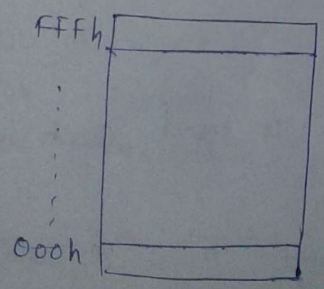


This will be above working register.

Internal ROM: $4kB = 2^2 \cdot 2^{10} = 2^{12}$

000h to FFFh

→ Program is stored in ROM because it is permanent



Scanned by CamScanner

Addressing Modes:

1. Immediate
2. Direct
3. Register
4. Register indirect
5. Register specific
6. Index
7. stack

① Ex: **Immediate:**

MOV A, #05H (Data)

MOV B, #07H

MOV Rn, #08H

$\boxed{n \to 0 \text{ to } 7}$

② Ex: **Direct:**

MOV A, 05H

~~MOV Rn, A~~

~~MOV A, Rn~~

③ **Register:**

Ex: MOV A, B

MOV Rn, A

MOV A, Rn

④ **Register indirect:**

Ex: MOV A, @Ri where i=0 & 1 / MOV R0, #07H MOV A, @R0

R0, R1 is used as index/pointer register.

⑤ **Register Specific:**

Specific register is used such as accumulator

MOV A, #67h

Ex: SWAP A ; A=76h

(6) Index:
DPTR / PC are used as index registers

Ex: MOVC A, @ A+DPTR
MOVC A, @ A+PC

if A=50
DPTR =1234
@ 1284

(7) Stack:
Ex: PUSH 04H ; R4 Contents into stack
POP 03H ; Top of stack are poped into R3.

21/2/18

Instruction set of 8051:

1. Data Transfer Instructions
2. Arithmetic "
3. logical "
4. Bit-level "
5. Control Transfer "

Data Transfer Instructions:

*MOV: MOV Dest, Source

Ex: MOV A, Rn ; n=0 to 7
MOV Rn, A

Source can be register, Mem. location, immediate operand.
Destination can be Register, A, ~~data~~

EX: MOV A, #data
MOV A, direct
MOV Rn, direct
MOV Rn, #data

→ Instructions to access external Data Memory.

Scanned by CamScanner

→ Ex: MOVX A, @Ri

MOVX – to ~~move~~ copy the Contents of external address from Ri. into Accumulator.

i → 0 & 1

n → 0 to 7

→ Ex: MOVX A, @DPTR

copy the contents of external address from DPTR into Accumulator:

DPTR – 16 bit address,

but it Contains 8-bit Data

→ MOVX @Ri, A

Copy data from A to the external address into Ri.

→ MOV @DPTR, A

Copy data from A to the external address in to DPTR

[MOVX → to move data b/w 8051 & external memory]

Instructions to Access External RAM/ROM:

To move data b/w 8051 & external RAM/ROM.

→ MOVC A, @ A+DPTR,

copy the contents of A+DPTR in to A.

→ MOVC A, @ A+PC

Copy the contents of A+PC into A.

*PUSH: PUSH direct

Ex: PUSH 07H

Contents from ~~Top of stack~~ nth location moved to Top of stack ~~location~~

Scanned by CamScanner

* **POP:** POP direct

Ex: POP 07H

Contents from Top of stack are moved to th location

Data Exchange Instructions:

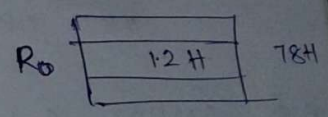
* **XCH:** A, B → exchanges A, B.

Ex:
XCH A, B
XCH A, Rn
XCH A, @Ri
XCHD A, @Ri i = 0 & 1.

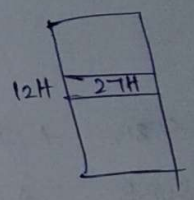
↳ It exchanges lower nibble (4-bits) b/w A & Address in Ri. (lower 4-bits).

Program:

```
MOV  A, #56H
MOV  R0, #78H
XCH  A, R0      // A - 12, R0 - 56
XCHD A, R0      // A - 56, R0 - 16
```



Ex:
```
MOV  R0, #12h (or) R0, 12h
MOV  A, #38H
MOV  B, @R0     // B = 27H
ADD  A, B
```



@ R0 → address

Arithmetic Instructions:

ADD
INC
SUBB
DEC
MUL
DIV
ADDC

DA – Decimal Adjustment.

ADD:

ADD A, Rn
ADD A, direct
ADD A, B
ADD A, #data
ADDC A, #data
ADDC A, direct.

* ADDC:

ADDC – Add with Carry.

↓

A ← A + [ ] + CF
data

* Contents of Accumulator, data & Carry is added & stored in Accumulator

Ex: ADDC A, #data
ADDC A, direct

* INC: Incremented by '1'

INC A
INC Rn
INC direct
INC @Ri
INC DPTR

* SUBB: Subtract with Borrow.

* SUBB A, Rn ; A ← A − Rn − CF

Contents of Accumulator are Subtracted from Rn & Carry.

SUBB A, #data
SUBB A, direct
SUBB A, @Ri
SUBB A, B

* DEC: Decrement

DEC A
DEC Rn
DEC direct
DEC @Ri

Scanned by CamScanner

c:

- Add with Carry.

-A + [ ] + CF
data

ontents of Accumulator
ata & Carry is
added & Stored in
-Accumulator

EX: ADDC A, #data
ADDC A, direct

mulator are
from Rn & Carry.

* MUL: MUL AB

(B)(A) ← (A) × (B)
8bit 8bit

(B) → USB Upper bits
(A) → LSB lower 8-bits

* DIV: DIV AB

B) A (Q → A
(R) → B.

Quotient - A (Accumulator)
Remainder - B (Base)

Write 8051 Program, Subtract R_1 Contents of Register Bank 0 from R_2 Contents of Register Bank 2.

Scanned by CamScanner

III. Logical Instructions:

* **ANL (AND):**

ANL dest, Source.

EX: ANL A, #data
ANL A, R0
ANL A, direct
ANL A, @direct

* **ORL (OR Operation):**

Syntax: ORL dest, source

Ex: ORL A, # data
ORL A, R0
ORL A, direct
ORL A, @direct

* **XRL:**

XRL dest, Source

Ex: XRL A, #data
XRL A, R0
XRL A, direct
XRL A, @direct

* **CPL (Complement):**

Syntax:
CPL A Complement Accumulator

* **CLR: (clear)**

CLR A clear Accumulator.

IV. Bit-level Instructions:

* CLR C

clear CF

Scanned by CamScanner

* CLR bit:
  CLR bit
  (clearing particular bit)
  Ex: CLR P0.2

* SETB C:
  set Carry flag

* SETB bit:
  set particular bit

* CPL C: Complement CF

* ANL C, bit: bit-level AND operatn b/w CF & specified bit
* ORL C, bit: " " OR " " " "
* MOV C, bit: " " MOV " " " "
* RL A: Rotate left on Accumulator
* RR A: Rotate right accumulator
* RLC A: Rotate left through carry on Accumulator
* RRC A: " right " " "
* SWAP A: Xchanges the upper & lower 4 bits of accumulator.

Control Transfer Instructions:

- Jump
  - Conditional
    - JZ
    - JNZ
    - JC
    - JNC
    - JB
    - JBC
    - JNB
    - CJNE
    - DJNZ
    - NOP
  - Unconditional
    - SJMP (short)
    - AJMP (Absolute)
    - LJMP (long)
    - JMP @ A+DPTR
- CALL
  - ACALL (absolute)
  - LCALL (long)
  - RET
  - RETI.

Scanned by CamScanner

SJMP: If address is within −127 to +128 locations (Relative JMP) SJMP address.

AJMP: If range is 0000h to 7FFFh. within 2K15 AJMP address.

LJMP: 0000h to FFFFh

JMP @ A+DPTR: External JMP

JZ: JZ relative address; Jump if A=0

JNZ: JNZ " " ; " " A≠0

JC: JC " " ; " " CF=1

JNC: JNC " " ; " " CF=0

JB: JB bit, relative address; Jump if bit is set.

***

LCALL: LCALL address (16 bit address)

RET: Return from Subprogram

RETI: Return from ISR.

Write 8051 program, sub R_1 contents of register bank 0 from R_2 contents of Register bank 2.

```
MOV   PSW, #10h ; bank2 is selected
MOV   A, R2
MOV   PSW, #00h ; bank 0 is selected
MOV   B, R1
SUBB  A, B
end.
```

C	A	F	RS1	RS0	O	–	P

00 – bank 0
01 – bank 1
10 – bank 2
11 – bank 3

0001 0000 — 10h

Write 8051 Program to perform 16 bit addition.

```
MOV   A, #14h
ADD   A, #77h ; A=0Bh, CF=1
MOV   R0, A   ; R0=0Bh, CF=1
MOV   A, #12h
ADDC  A, #11h ; A=23h + CF = 24h
MOV   R1, A
end
```

12 14 h
+ 11 77 h

ADDC ADD

24 0B h

R_1 R_2

Sum of 'n' natural no's:

```
MOV   A, #00h
MOV   R0, #05h
up: ADD   A, R0
Djnz  R0, up
End.
```

if n=5;

Average.

```
MOV   A, #00h
MOV   R0, #05h
MOV   B, R0
up: ADD   A, R0
Djnz  R0, up
DIV   AB
end.
```

Factorial:

```
MOV   A, #01h
MOV   R0, #05h
up: MOV   B, R0
MUL   AB
Djnz  R0, up
end.
```

Biggest / Smallest:

Simple programs from T.B.

Scanned by CamScanner

24/2/18

UNIT- V

8051 REAL TIME CONTROL.

[TMOD, TCON, SCON, IE, IP → 12 Marks.]

* Timer/Counter Operation:

→ Timer used to generate accurate time delays.
↳ Used to count internal pulses.

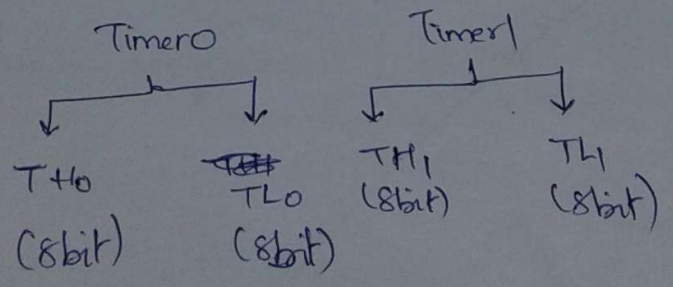
→ Counter counts external pulses (or) events.

→ Source for, Timer: Crystal Oscillator (XTAL1 & XTAL2)
, Counter: from external ~~at~~ T_0 & T_1 pins.

→ TMOD register decides Timer (or) Counter.

→ Timer is similar to a watch. ~~It counts from 0 to 59 secs~~

→ Two 16-bit Timers in 8051 → Timer 0 (16)
↳ Timer 1 (16)



→ ~~Counter~~ Timer counts from 0 to $2^{16} = 65535$.

→ The duration of pulse is decided by ~~crystal~~ crystal oscillator.

1 Machine Cycle = 12 clock cycles

Scanned by CamScanner

→ Crystal oscillator freq → 4MHz to 40MHz

→ for 8051, Crystal Oscillator freq → 11.0592 MHz

$$XTAL\ freq = \frac{11.0592\ MHz}{12}$$

$$= 0.9216\ MHz$$

$$\text{clock period} = \frac{1}{XTAL_{freq}} = \frac{1}{0.9216 MHz} = 1.085\ \mu s$$

→ Duration of each pulse depends on Crystal oscillator freq.

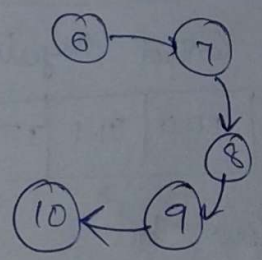
Ex: if XTAL freq → 11.0592 MHz;
Duratⁿ on each pulse is 1.085 μs.

→ Timer Operation:

Ex: If 4 ~~eyet~~ events are to be counted;
Subtract it from maxᵐ value
Here; max = 10

⇒ count = 4

10 − 4 = 6

⇒ 

⇒ Count = 4
6 → 7, 7 → 8, 8 → 9, 9 → 10

** TMOD Register (8): Timer Mode Register.

7	6	5	4	3	2	1	0
Gate	C/T̄	M1	M0	Gate	C/T̄	M1	M0
← Timer 1 →				← Timer 0 →			

Gate:

Gate → 0 → Controlling of timer is done using software
→ 1 → " " " " " hardware
(Default — software)

Controlling of timer → start the timer
→ Run " "
→ Stop " "

C/$\bar{T}$: Counter/Timer

C/$\bar{T}$ = 0 → Timer (16-bit)
= 1 → Counter

M1 M0:

M1M0 — Modes of operation

M1	M0		
0	0	—	Mode 0 → 13 bit Timer
0	1	—	Mode 1 → 16 bit Timer
1	0	—	Mode 2 → 8-bit Timer (TL/TH)
1	1	—	Mode 3 → Split Timer.

Split Timer: (3 timers)

TL0, TH0, T1 as independent timers.

(2M)
Ex: for Timer1 — mode2 — (software default) → TMOD = 0010 0000 = 20H.
⇒ MOV TMOD, #20h.

TCON Register (8 bit):

Timer Control Register.

TF1	TR1	TF0	TR0	IE1	IT1	IE0	IT0
7	6	5	4	3	2	1	0

TF1 — Timer1 Overflow bit

when, timer reaches its max^m value; TF1 = 1.

→ It is set by hardware on timer/Counter overflow

→ clear when interrupt processed.

TR1 — Timer1 Run bit

→ It is set/Reset by software to turn timer/Counter

Scanned by CamScanner

on/off

TF0 – Timer 0 Overflow bit

when timer reaches its maxm value, TF0=1

same as TF1.

TR0 – Timer 0 Run bit

same as TR1

→ 'setb' is used to set the particular bit

setb TR1 / TR0.

↳ start the Timer

→ 'clr' → to stop the timer.

clr TR1/TR0

IE1 – Interrupt 1 Edge flag.

→ It is set by hardware when external interrupt edge is detected.

→ when interrupt is processed, it is cleared. (IE1)

IT1 – Interrupt 1 Type control bit.

→ set / cleared by software to specify -ve edge (or) low level triggered external interrupts.

IE0 – Interrupt 0 Edge flag.

→ set / cleared by Hardware when external interrupt edge is detected.

→ cleared when interrupt is processed.

IT0 – Interrupt 0 Type control bit

same as IT1.

Scanned by CamScanner

Modes of Operations:

→ Mode 1 Programming: ~~Steps~~ characteristics:

→ It is a 16-bit Timer,

∴ It allows values of 0000h to FFFFh

when it reaches max^m value (FFFF); it rolls up to starting value (0000)

to be loaded into the Timer Registers (TL&TH)

then, timer is started.

→ This is done by 'setb' (TR1/TR0)

Timer 0 → setb TR0

Timer 1 → setb TR1

→ After Timer is started, it starts to count ~~the~~ up until it reaches its limit of FFFFh, when it rolls over from FFFFh to 0000h, it sets high the Timer Overflow flag (TF ~~high~~). ⇒ (TF1/TF0 = 1)

→ The timer flag can be monitored. when it is raised, we must stop the timer; using the instructions,

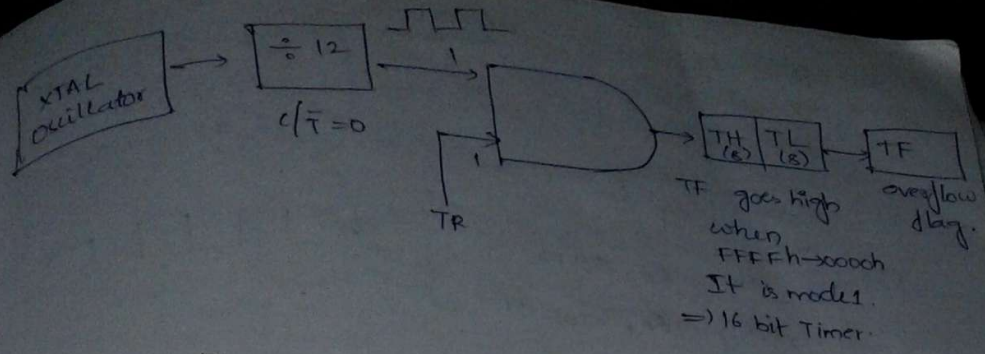
Clr TR1 → Timer 1

Clr TR0 → Timer 0

→ After the timer reaches its limit, & rolls over in order to repeat the process, the registers TL& TH must be reloaded with the original value & 'TF' must be reset.

→ The steps can be diagramatically represented as;

Scanned by CamScanner



~~Mode 1~~ ~~Progr~~ Steps to program Mode-1: (Common to any mode)

→ load TMOD Register indicating which timer is to be used & which mode is selected.

→ load registers TL & TH with initial count values

→ start the Timer.

→ Keep monitoring the TF (Timer overflow) flag with instruction JNB TFx, Target.

x → 0(or)1
Target → Label.
JNB → Jump if no bit

~~Keep~~ to see if it is set get out of the loop when TF=1 (high)

→ stop the timer.

→ clear TF flag for the next round, go to step-2 to load TL & TH again.

Write 8051 Program to generate a Square wave at P1.3 without Timers.

~~Set P1.3~~

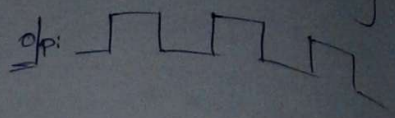
Program:

without Timers

SJMP → short Jump
∵ code is with the memory

```
up: SETB  P1.3
    ACALL Delay
    CLR   P1.3
    ACALL Delay
    SJMP  up
    end
Delay: MOV  R0, #0FFh
d1: DJNZ  R0, d1
    RET
```

o/p:



with Timers: 10ms pulse width at P1.3 using T0 - Mode1

TMOD → 0000 0001
⇒ 01H.

for step-2:

(i) XTAL freq = 11.0592 MHz ⇒ $\frac{11.0592\,MHz}{12} = 921.6\,kHz = f_{clk}$

$\frac{1}{f_{clk}}$ = clk period = 1.085 μs

(ii) $\boxed{\text{Count} = \frac{\text{Required Delay}}{T_{clk}}} = \frac{10ms}{1.085\mu s} = 9216$ pulses are counted.

(iii) $65536 - 9216 = (56320)_{10}$

= DC00h

(iv) ⇒ TH → DCh, TL → 00h

Program:

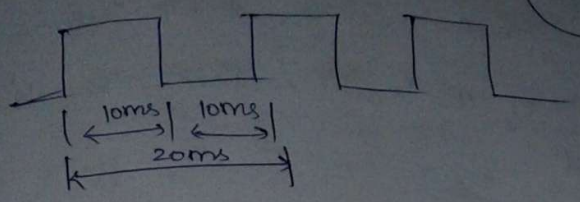
```
~~setb P1.3~~
~~MOV TMOD, 01h~~
~~MOV TL, 00h~~
~~MOV TH, DCh~~
```

```
up: setb P1.3
    ACALL Delay
    Clr   P1.3
    ACALL Delay
    SJMP  up
    end
Delay: MOV TMOD, #01h
up2: MOV TH0, #0DCH
     MOV TL0, #00h
```

Scanned by CamScanner

here: JNZ TF0, here
clr TR0
clr TF0
SJMP up2

from 56321 to 65535
TF0 = 1
at 65536, TF0 → 0
⇒ cleared



Generate a square wave with ON time of 3ms & OFF time of 10ms on P0, Timer 1 - Mode 1.

TMOD — 0001 0000 ⇒ 10h

XTAL freq = 11.0592 MHz.

clk = 1.085 μs

$$\text{count} = \frac{10\text{ms}}{1.085\mu\text{s}} = 9216$$

$$\text{Count} = \frac{3\text{ms}}{1.085\mu\text{s}} = 2765$$

Scanned by CamScanner

Mode-2 Programming:

→ It is 8-bit Timer, It allows only values of 00h to FFh. to be ~~after~~ loaded in to TH Register.

→ After TH is loaded with 8-bit Value, 8051 gives a copy of it to TL, Then the timer must be started.

setb TR1/TR0.

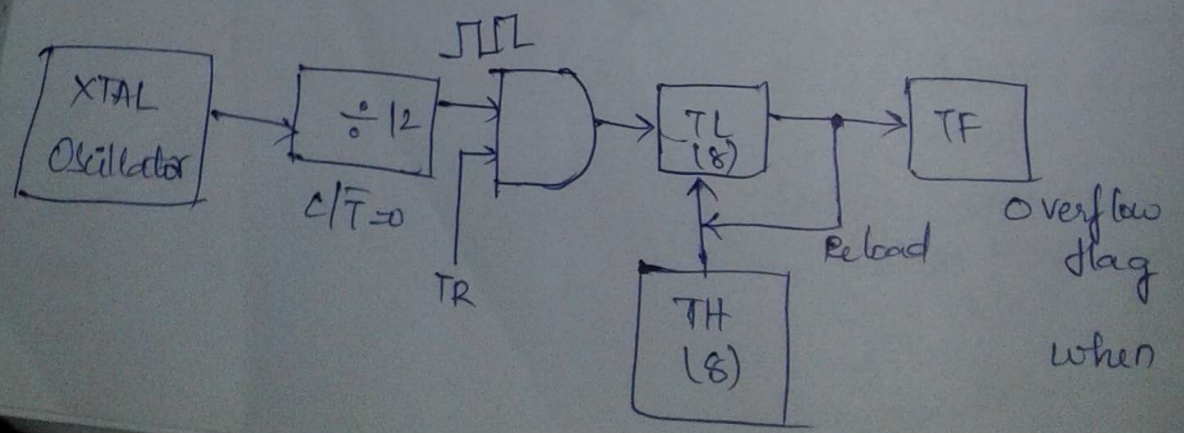
→ After timer is started, it starts count up by incrementing TL Register. It counts up until it reaches the max^m value of FFh, when it rolls over from FF to 00h, it sets high the TF flag.

→ When the TL rolls from FF to 00, & TF is set to '1'; the TL is reloaded automatically with the original value kept by the TH Register.

→ This mode is also called "Auto-Reload Mode."

→ Continue to repeat the process, we must simply clear TF. & let it go without any need by the programmer to reload the original value.

Diagramatically:



when FF→00h
TF=1

Steps: for mode 2
→ load TM
to be us
→ load reg
→ start
→ keep
to see
clear
→ to
→
26/2/18
write
P1.0 us

for mode 2 Programming:

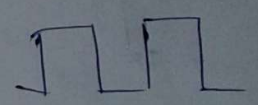
steps same as mode-1.

- load TMOD Registers indicating which timer is to be used & which mode is selected.
- load register TL with initial count value
- start the Timer.
- Keep monitoring the TF flag with instruction JNB TFx, Target. to see if it is set get out of the loop when TF=1 (high).
- clear the TF flag for the next round.
- go to step-4, ∵ Mode 2 is Auto-Reload.

26/2/18

write a 8051 program to generate a square wave on P1.0 using Timer-1 mode-2 (using Polling)

```
MOV  TMOD, #20h
     MOV  TH1, #05h
     SETB TR1
* Here: JNZ  TF1, Here
       CPL  P1.0
       clr  TF1
       SJMP Here
       end.
```



$T = (56-5) \times 1.085\,\mu s = 0.27\,ms$

$T_{ON} = 0.27\,ms$ (∵ Dutycycle = 50%)

$T_{OFF} = 0.27\,ms$

$T = 0.54\,ms$

$f = 1.85\,kHz$.

Scanned by CamScanner

Counter Programming:

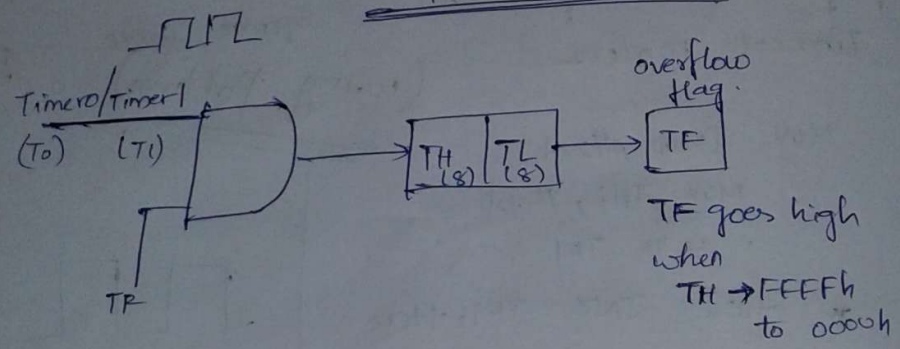
→ $C/\bar{T} = 1$ ⇒ Counter

⇒ To count the external events (pulses)

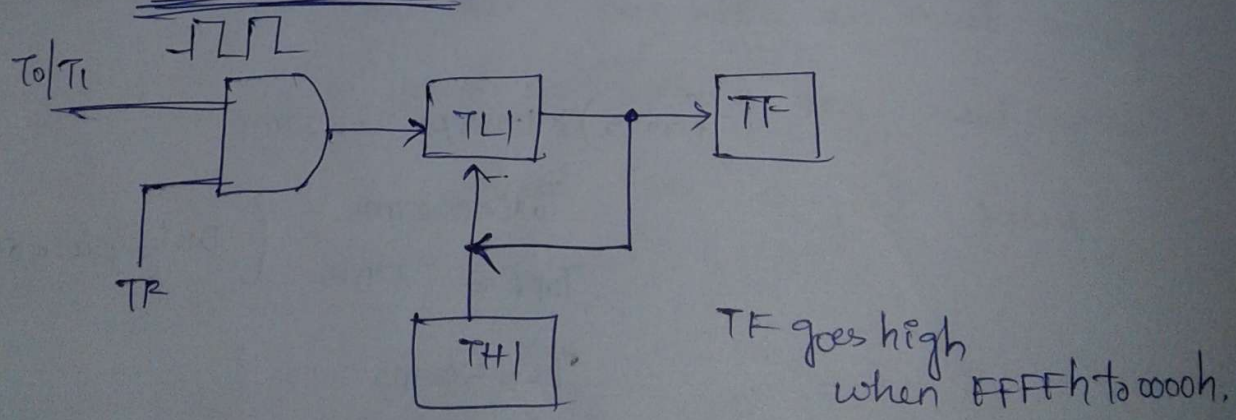
→ pin(14) – T_0
pin(15) – T_1] – sources.

→ TMOD, TH, TL } same as timers.

Counter in mode 1.



Counter in mode 2.



Scanned by CamScanner

Serial Communication:

Serial	Parallel
→ bit by bit transmission of data	→ byte by byte data transmission
→ for long distance	→ for short distance
→ less expensive	→ Costly.
→ More error	→ less error
→ less efficiency	→ More efficiency.

→ Types of serial Commⁿ:

1. Simplex — one way commⁿ
2. Half-Duplex — two way commⁿ, but not simultaneously
3. Full-Duplex — " " " , simultaneous.

→ synchronous Serial Data Transfer
→ Asynchronous " " "

synchronous: used for Block transfer
→ A Block can be sent at a time.

Asynchronous: character oriented.
→ At a time a byte can be sent

Format:

Synchronous:

Sync1	Sync2	Data

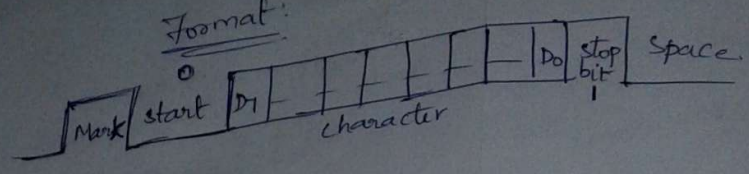
→ 2 Synchronized bits
→

Scanned by CamScanner

-Asynchronous

Framing – start bit + Data + stop bits

Format:



~~character~~

1 start bit + Data (D7–D0) + 1 (or) 1½ (or) 2 stop bits

Now-a-days only 1 stop bit is used.

space – end of the data

~~start~~

→ In 8051 (supports) → UART → Universal Asynchronous Receiver Transmitter

Data Transfer Rate:

Baud Rate :- No. of signal changes in one second.

units : bits/sec.

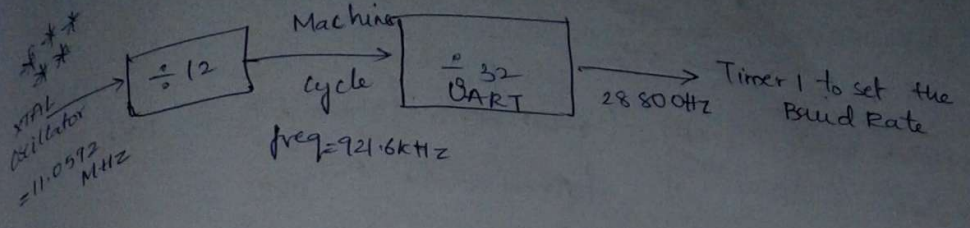
[For coaxial cable ; Bit rate = Baud Rate]

BR – 110 (Hz)
– 150
– 300
– 600
– 1200
– 2400
– 4800
– 9600
– 19200

Scanned by CamScanner

→ To allow Data transfer b/w 8051 & PC; Baud Rates must be matched.

→ To match BR's of 8051 & PC;



→ 8051 internal UART divides freq. of machine cycle to match the BR of PC.

→ Timer 1, TH1 Register values for Various baud rates.

Baud rate	TH1	TH1 in Hexa
9600	-3	FDh
4800	-6	FAh
2400	-12	F4h
1200	-24	F8h

Serial Communication Programming:

SBUF - Serial buffer
- 8-bit registers
- used for serial comm.

Sending data: SBUF → TXD → ---

Receiving data: SBUR → RXD ← ---

SCON: (Serial Control Register)

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

SM0, SM1

Serial Mode

SM1	SM0	
0	0	→ SM0 → shift register, Baudrate = f/12
0	1	→ SM1 ⇒ is used
1	0	→ SM2
1	1	→ SM3

SM1 ⇒ 8-bit UART (8-bit data, 1 start bit & 1 stop bit)

SM2 ⇒ 9-bit " (fixed data Rate)

SM3 ⇒ 9-bit " (variable data Rate)

Serial Mode-2:

Used in Multi-processing system

REN: Receive Enable

TB8 & RB8: Not used

TI: Transmit Interrupt

1 → Data present in SBUF is Completely transmitted

0 → " " " " " transmitting

RI: Received Interrupt

1 → Valid serial data is placed in SBUF before, it is lost

Algorithm:

Sending serial data (8-bit):

1. load TMOD Registers with 20h
2. load TH1 with one of the values to set the baud rate for serial data transfer.
3. load SCON register with 50h (indicates serial mode 1)
4. start the timer
5. character byte to be transferred serially is written in to SBUF.

Scanned by CamScanner

Monitor TI flag using JNB TI, here to see whether the character bytes has been transferred completely.
transfer next byte, go to step-5

To write a Program to send a letter or character 'A' continously using 4800 baud rate, 8-bit data, 1-start bit & 1-stop bit.

```
MOV   TMOD, #20h
MOV   TH1, #FAh
MOV   SCON, #50h
SETB  TR1
up:   MOV  SBUF, #'A'
here: JNB  TI, here
      CLR  TI
      SJMP up
      END
```

Write 8051 Program to display the name continously using 7600 baud rate, 8-bit data, 1-start bit, 1-stop bit.

```
MOV   TMOD, #20h
MOV   TH1, #0FDh
MOV   SCON, #50h
SETB  TR1
up:   MOV  A, #'S'
      ACALL Delay
      MOV  A, #'A'
      ACALL Delay
      MOV  A, #'H'
      ACALL Delay
      MOV  A, #'I'
      ACALL Delay
      MOV  A, #'T'
      ACALL Delay
      MOV  A, #'H'
      ACALL Delay
      MOV  A, #'I'
      ACALL Delay
      SJMP up
```

Scanned by CamScanner

```
Delay: MOV SBUF, A
here: JNB TI, here
      Clr TI
      RET.
```

RI

28/2/14

8051 Serial

Algorithm to

→ load

→ load

→ load s

→ start

→ RI

see

→ when

are

→ To

Write

serial

rate

80

Scanned by CamScanner

27/6/14

8051 Serial Communication:

Algorithm to receive a char. Serially:

→ load TMOD with 20h
→ load TH1 with initial constant.
→ load SCON with 50h
→ start the timer.
→ RI flag is monitored with JNB RI, here
see if entire character has been received yet.

→ when 'RI' is raised, Sbuf has a byte, its contents are moved to safe place.
→ To receive next character, go to step-5.

Write 8051 program to receive a character serially & put it in port-1 using 2400 baud rate.

```
MOV  TMOD, #20h
MOV  TH1, #-12
MOV  SCON, #50h
SETB TR1
here: JNB  RI, here
      MOV  A, SBUF
      MOV  P1, A
      clr  RI
      SJMP here
      end.
```

8051 Interrupts:

8051 serves the peripheral devices in 2 ways.

1. Polling
2. Interrupts

Scanned by CamScanner

Polling:

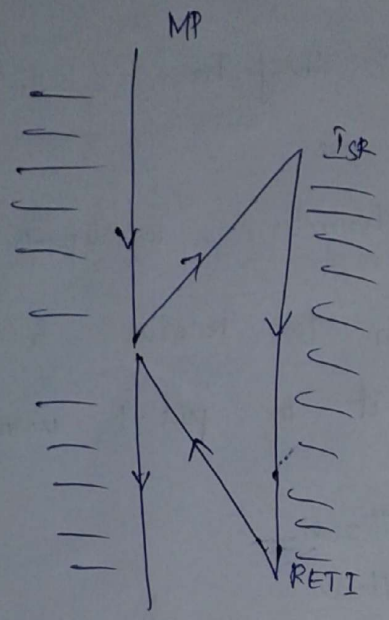
8051 Continously monitors all the external peripheral devices. this is called polling.

(Disadvantage, wastage of time)

Ex: here: JNB RI, here.

Interrupt:

Interrupts are handled at Interrupt handler and ISR



→ Interrupt is better than Polling.

8051 (6)

- 2 External (H/w)
  - INT0
  - INT1
- 3 Internal.
  - TF0
  - TF1
  - RI / TI → (Received / Interrupt)
  - RESET (H/w)

Scanned by CamScanner

→ IVT (Interrupt Vector Table) → stores addresses of ISR

(IVT)

	ROM location (Hex)
Reset	0000h
External H/w (INT0)	0003h
Timer 0 (TF0)	000Bh
External H/w (INT1)	0013h
Timer 1 (TF1)	001Bh
Serial Com (RI / TI)	0023h

→ To enable / disable the interrupts; SFR (8-bit registers)

Special Func^n Registers: TMOD, TCON, SCON, IE, IP

IE : - Interrupt Enable:

7							0
EA	–	ET2	ES	ET1	EX1	ET0	EX0

(7) EA – Enable / Disable all interrupts except Reset.

[Enable-1, Disable-0]

EA – 1 – enable
– 0 – disable.

(5) ET2 – Enable / Disable Timer 2 → (used for 8052) (∵ in 8051, Timer0, Timer1.)

(4) ES – Enable / Disable Serial Com. Interrupts (RI / TI)

(3) ET1 – Enable / disable Timer1 interrupt (TF1)

Scanned by CamScanner

(2) EX1 – ~~External int~~ Enable / disable external interrupt 1

(1) ET0 – Enable / disable Timer 0 interrupt (TF0)

(0) EX0 – Enable / disable external interrupt 0.

Ex: → for enabling Serial commⁿ & Timer 0 interrupts
→ enabling ~~ETO~~ ES & ET0.

⇒ 10010010
⇒ 92h.

MOV IE, #92h.

then, ES & ET0 → enabled.

Priority Interrupts:

Default priority:

- INT0 (high priority)
- TF0
- INT1
- TF1
- RI/TI (low priority)

IP: Interrupt Priority Register

7	6	5	4	3	2	1	0
–	–	–	PS	PT1	PX1	PT0	PX0

↓

7, 6, 5 → are reserved.

PS – priority for serial commⁿ

PT1 – " " Timer 1

PX1 – " " INT1 (external interrupt 1)

PT0 – " " " Timer 0

PX0 – " " INT0 (external interrupt 0)

Scanned by CamScanner

8051 Program to generate square wave on pin P1.2 if it uses Interrupt for timer 0.

```
        ORG     0000h
        LJMP    MAIN

        ORG     000Bh
        CPL     P1.2
        RETI
        ORG     0030h
MAIN:   MOV     TMOD, #01h
        MOV     TH0, #DCH
        MOV     TL0, #00h
        MOV     IE, #82h.
        SETB    TR0.
up:     SJMP    up
        end.
```

IE = 1000·0010
=> 82h.

de external interrupt 1
interrupt (TF0)
interrupt 0.
Timer 0 interrupts.

0 → enabled.

rupt 1)

Scanned by CamScanner

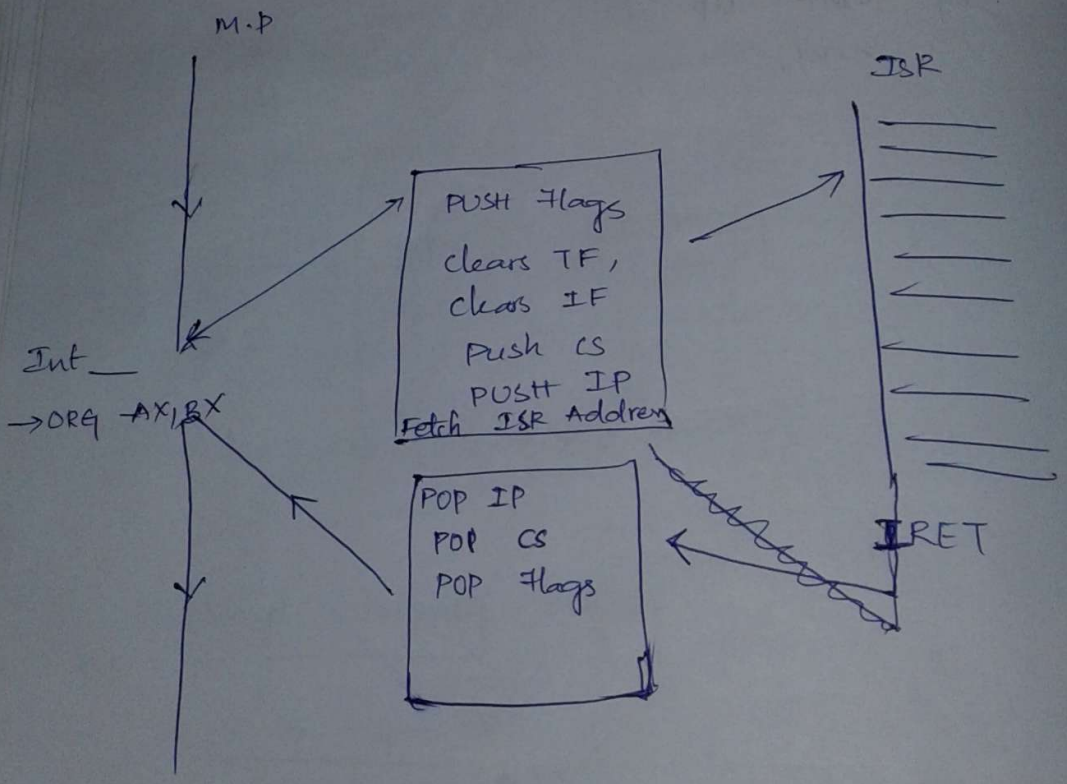
2/3/18

UNIT - III

Interrupts — 8259 PIC
DMA Controller — 8257 DMA
Serial Commn. — 8251 USART.

Interrupt structure : (of 8086)

In 8086, for push operation;
SP → decremented by '2' &
flags are placed on to the stack.
It clears the IF & TF.
(Interrupt enable flag &
Trap flag)



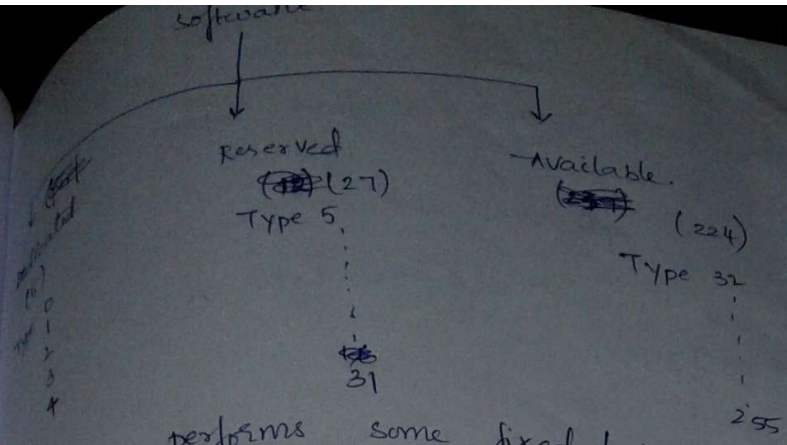
8086 Interrupts

- Hardware (2)
  - NMI
  - INTR
- Software (256)
  - INT 0 to INT 255
  - (or) Type 0 to Type 255

Dedicated (6)
Type 0
1
2
3
4

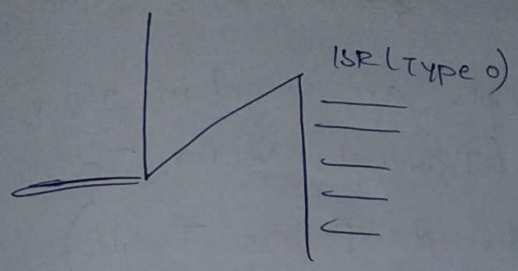
Dedicated :-
1. Type -
2.

Scanned by CamScanner



Dedicated :- performs some fixed / dedicated operations.

1. Type-0 – Divide by zero Interrupt: (INT 0)



ISR – displays the register contents of each & every line.

→ It has highest priority.

2. Type-1 – Single stepping Interrupt: (INT 1)

to check result step by step ; then TF=1 (Trap Flag)

It is executed.

if TF = 1 ; Type-1 interrupt.

3. Type 2 – Non-Maskable Interrupt: (INT 2)

we cannot ignore the interrupt.

4. Type-3 – Break pt. Interrupt: (INT 3)

To display results at the end of program.

Scanned by CamScanner

5. Type-4 (INT 4) - Overflow Interrupt:

This is executed when overflow flag is set.

→ Reserved Interrupts are not in use (Type-5 to 31)

→ Available " are available for the user (Type 32 to 255)

→ 4-bytes are required for starting address of Int.

⇒ Each interrupt requires 4 bytes.

⇒ $256 \times 4 = 1024$ bytes $= 2^{10}$ ⇒ 10 address lines

→ Software interrupts are stored in IVT (Interrupt Vector Table).

(or) Interrupt Pointer Table)

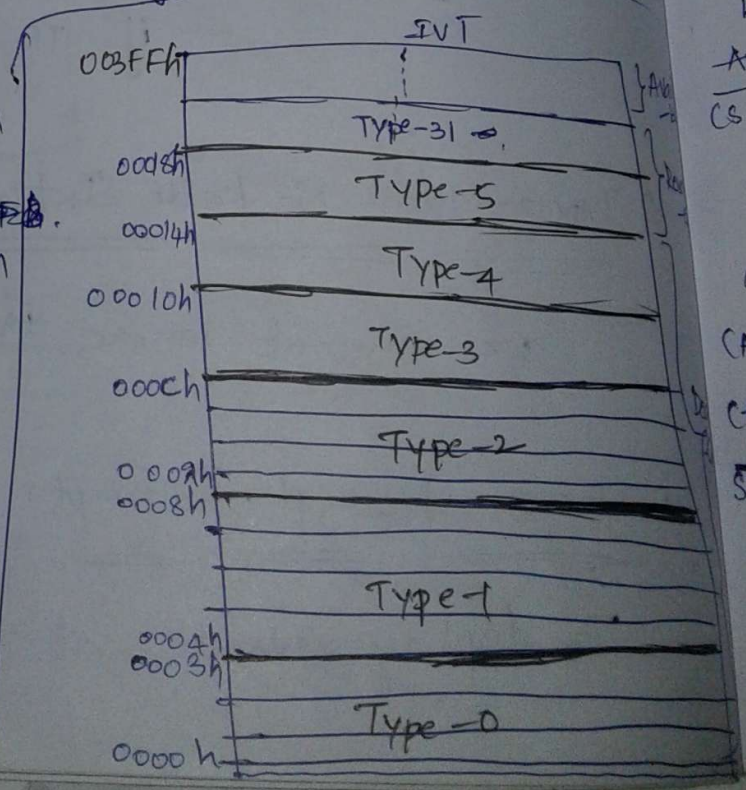
→ First ~~24~~ bytes of 8086 are allocated to software Interrupts.

→ if Type 'N' interrupt;

Starting address ⇒ $4 * N$

2^{10} — S.A → 000h

F.A → 3FFh



Priorities:

— INTO

— NMI

— INT

— INT

→ 8259

5/3/18

8259

RD

WR

A0

CS

CAS2

CAS1

CAS2

SP/EN

Scanned by CamScanner

Priorities :

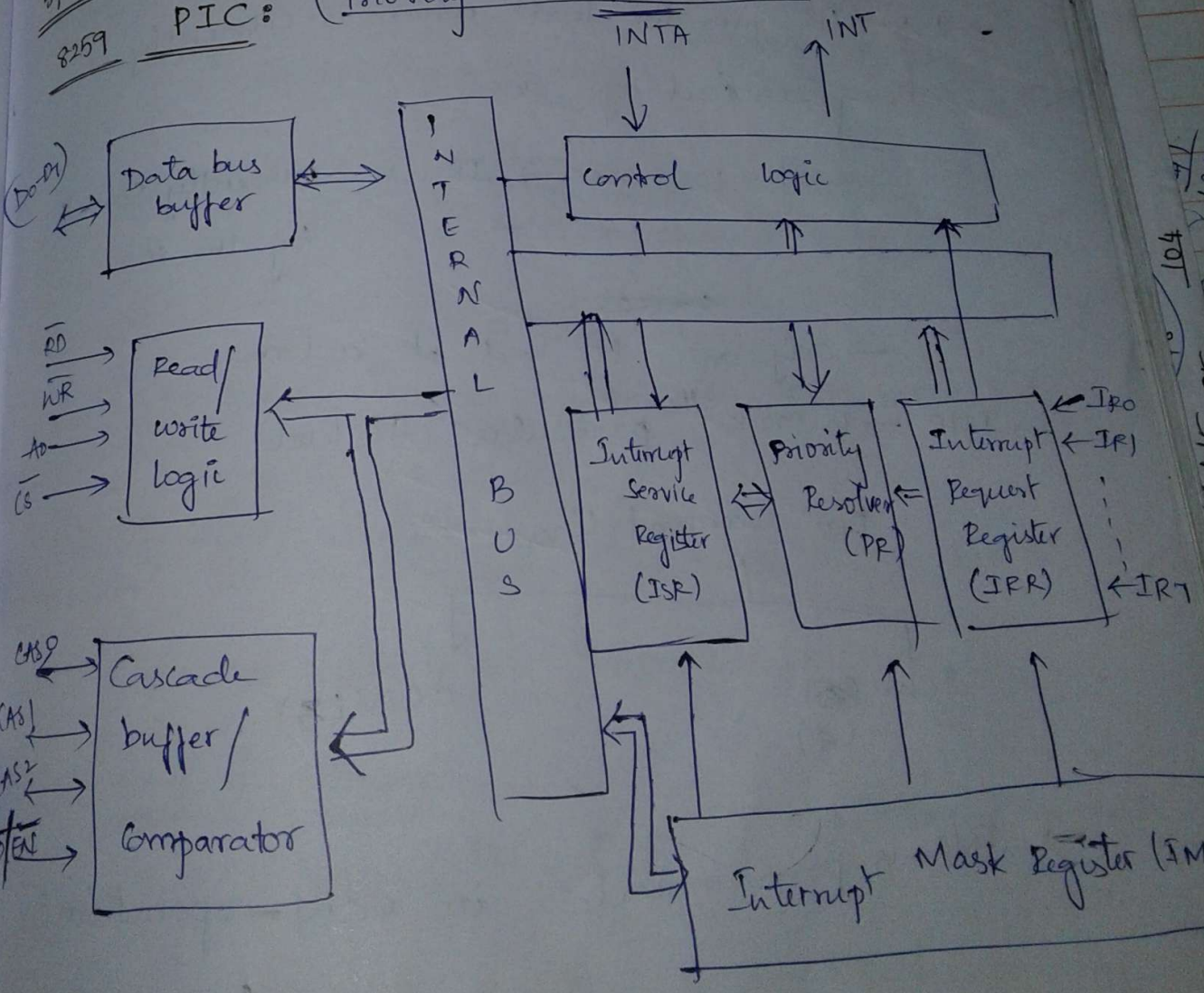
- INT0 , INT (3 - 255) — HP (High Priority)
- NMI (INT2)
- INTR
- INT1 ——— Low Priority.

→ 8259 – PIC (Priority Interrupt Controller).

(8 – 64) I/O devices

5/3/18

8259 PIC: (Priority Interrupt Controller)



Scanned by CamScanner

→ IRR, ISR, PR, IMR are 8-bit registers.

→ IRR gives interrupt request level which requests the service.

IR levels – IR0 to IR7

if i/p is connected to IR0, IR1 & o/p to IR7,

IR7	6	5	4	3	2	1	0
1	0	0	0	0	0	1	1

→ PR – if one I/o device requests the service, PR is not used.

– if more than one I/o devices requests the service, PR decides which has the highest priority.

→ ISR – It gives the level which is currently being serviced by 8086.

if 1000 0000 ⇒ IR7 is currently served by the 8086

→ only one bit is '1' at a time

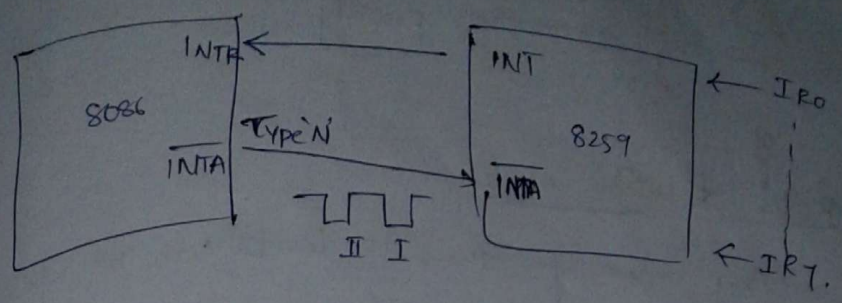
→ IMR – To mask the particular IR level

two command lines

ICW (4) OCW (3)

→ IMR is programmed in OCW – operational command word

setting the priority. IR (request) is sent to 8086 from 8259 through INT &. After then 8086 sends $\overline{INTA}$ (acknowledge) to 8259



→ Data bus buffer → Sends data b/w 8086 & 8259 & also used to ~~sen~~ check the status of 8086.

→ Read/Write logic → $\overline{CS}=0$.

→ Cascade buffer / Comparator → 3 I/O Cascade lines CAS0, CAS1, CAS2 (i/p's)

$\overline{SP}/\overline{EN}$ = Active low

$\overline{SP}=0$ ⇒ slave mode is enabled. (Single mode)

$\overline{EN}=0$ ⇒ 8259 is operated in Master mode. (Cascade mode)

→ 8259 is interrupted with 8086 from 8 devices up to 64 devices.

During I (1st clk cycle):

→ when it is servicing, particular ISR bit is '1' & corresponding IRR bit is '0'.

During II clk cycle:

→ it gives the type of interrupt in 8086.

→ Default; IR0 — high priority & IR7 — low priority.

Modes for Priority:

1. Fully Nested mode (or) Default mode:

IR0 – high priority
⋮
IR7 – low priority

2. Specific Rotation mode:

Any one of the IR level can be specified by assigning the low priority. Hence, the priorities of other IR levels get fixed in cyclic order.

Ex: If low priority is assigned to IR2, other priorities are as shown below (i.e., cyclic order).

IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
4	3	2	1	0	7	6	5

(High priority) (low priority)

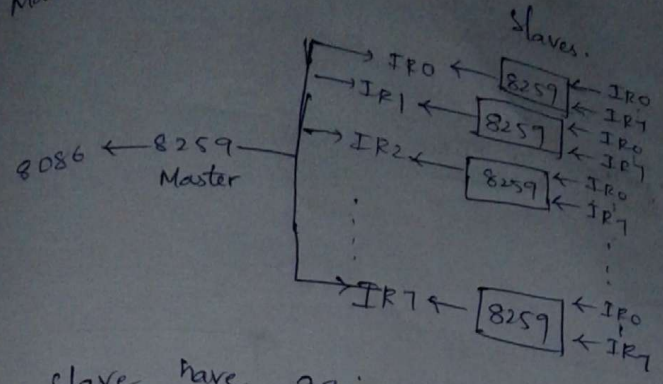
3. Automatic Rotation mode:

If IR4 → low priority
IR3
IR2
IR1
IR0
IR7
IR6
IR5 → high priority.

Scanned by CamScanner

1. Special Fully Nested mode (default mode):

Master mode



each slave have again 8 IR levels.

Here; IR0 — H.P (high priority)
IR7 — L.P (low priority)

Ex: slaves

IR1 ← IR3 (I priority)
IR1 ← IR6 (II ")

IR3 ← IR0 (III ")

IR7 ← IR4 (IV ")

18/18

8259 PIC:

→ End of Interrupt (EOI) Command: EOI resets the set bit in ISR after the interrupt is served.

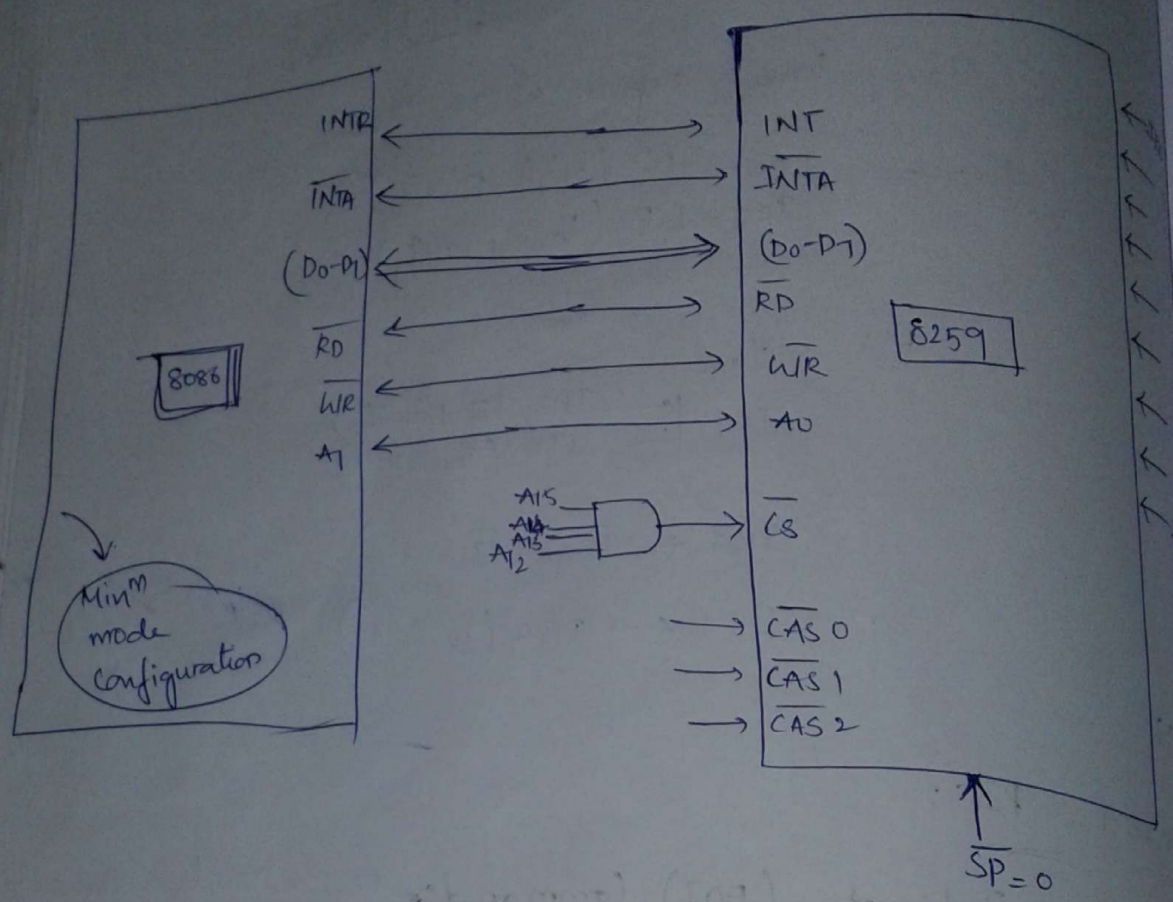
1. Non-Specific EOI: when 8259 receives this command it will reset the high priority IR level setbit of ISR.

2. Specific EOI: 8259 have 8 IR levels, each IR level can be specified by 3-bit address. when this command is given to 8259, it will specify the specified IR level of ISR

3. Automatic EOI: In response to first $\overline{INTA}$ pulse 8259 will set the corresponding IR level of ISR

and in response to 2[illegible] reset the corresponding set bit of ISR by giving command.

8259 Interfacing to 8086:



Command words: (CW) (7)

- Initializational Command words (ICW) (4)
  - ICW1
  - 2
  - 3
  - 4
- Operational Command words (OCW) (3)
  - OCW1
  - 2
  - 3

Scanned by CamScanner

→ 1 to 4 ICW are written (required) to initialize 8259 (i.e) to act as i/p & o/p)

ICW1:

7	6	5	4	3	2	1	0
x	x	x	1	LTM	x	SNGL	ICW4

if ICW4 is required → 1
not " → 0.

SNGL – single mode → 1 (only 1 8259 as slave mode)
cascade mode → 0 (master mode)

LTM – level Triggered → 1
edge " → 0

ICW2:

T7	T6	T5	T4	T3	0	0	0

Interrupt Vector Address.

ICW3: → slave
→ Master

slave

0	0	0	0	0	ID2	ID1	ID0

Identification number of slaves

Master

SL7	SL6	SL5	SL4	SL3	SL2	SL1	SL0

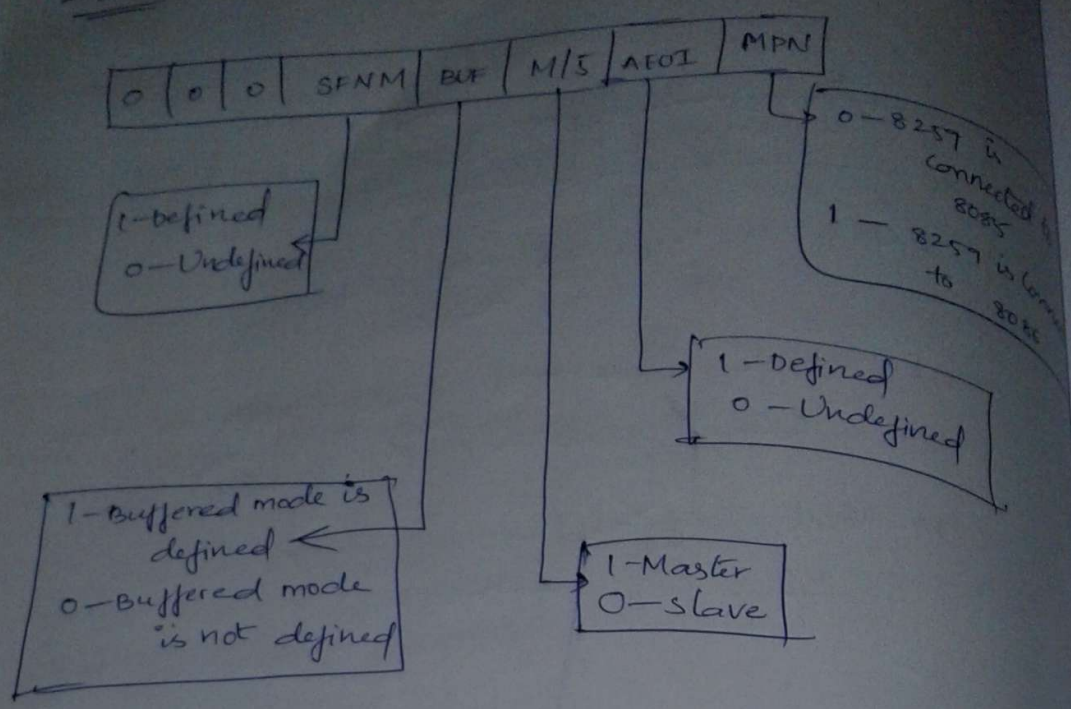
SL0 → 1 → slave 0 is connected
→ 0 → " " not "

⇒ SLn → 1 → slave n is connected
→ 0 → slave n is not connected.

if all
'1', all
are co

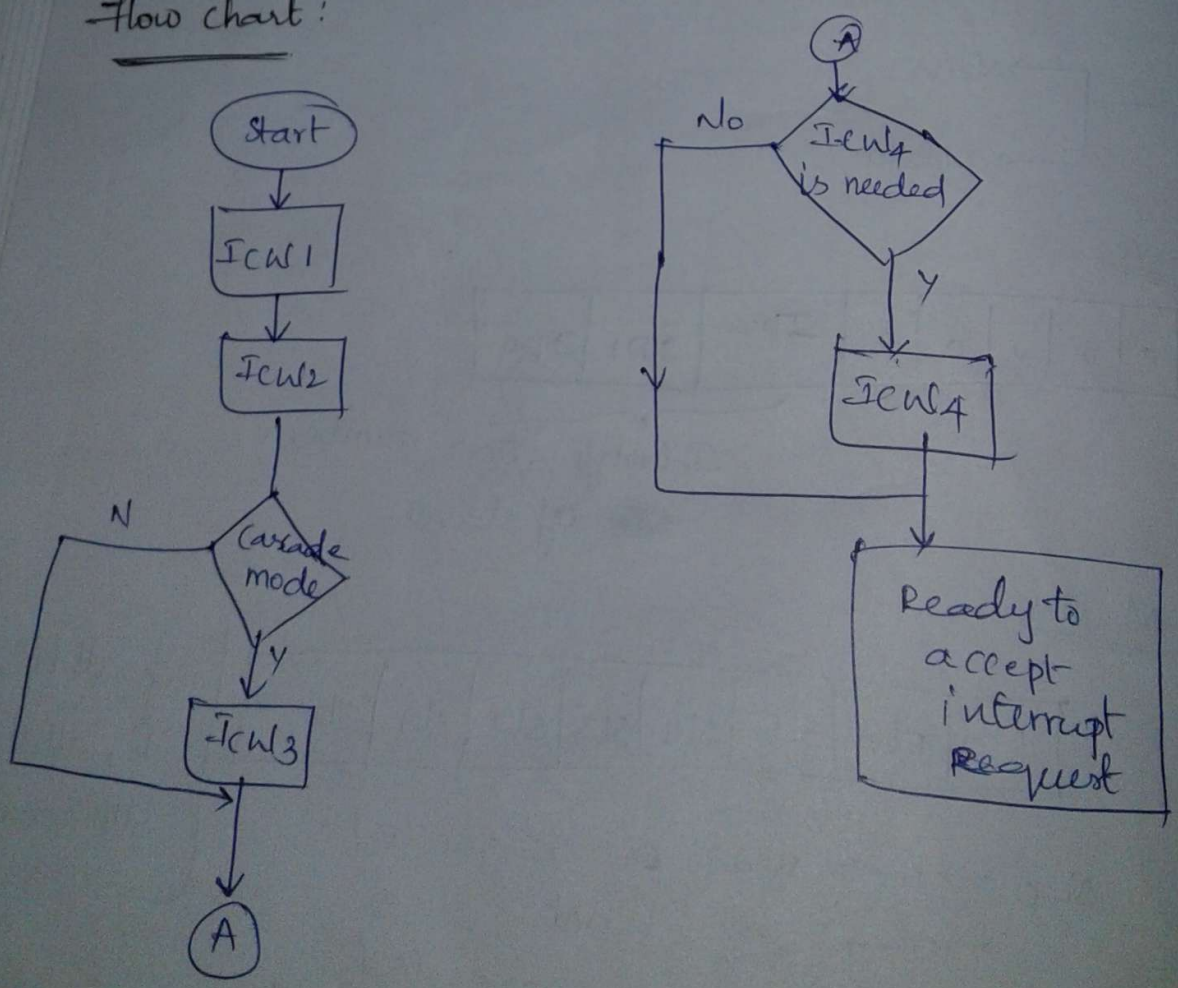
Scanned by CamScanner

ICW4:



→ if 8259 is operated in cascade (master) mode; AEOI is required.

Flow chart:



Scanned by CamScanner

OCW1 It is programmed to mask the particular IR level

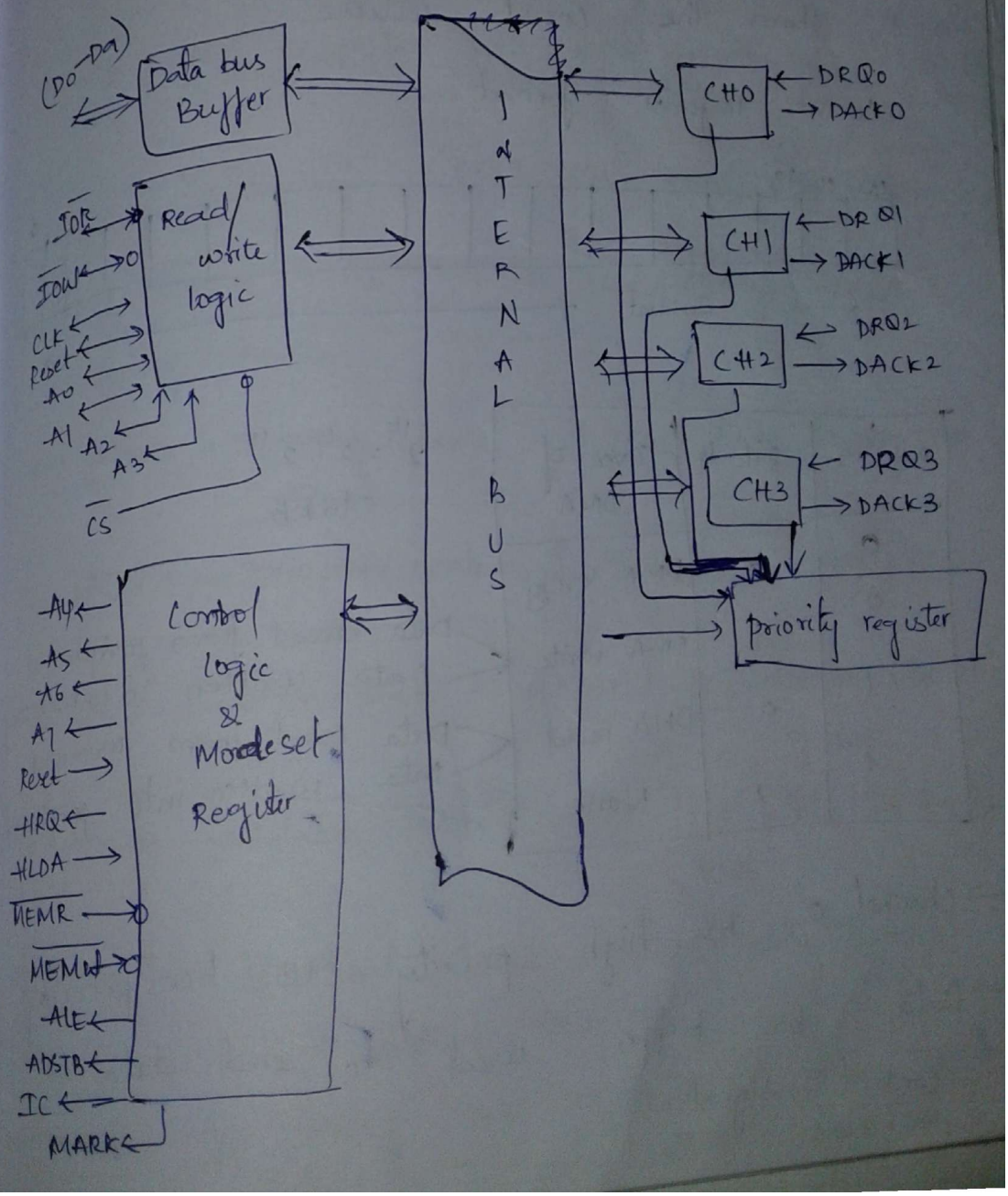
M7	M6	M5	M4	M3	M2	M1	M0

M7 → 1 → IR7 is masked
0 → IR7 is not masked.

Mn → 1 → IRn is masked
0 → IRn is not masked.

12/3/17

DMA Controller (8257)



Scanned by CamScanner

→ HRQ of DMA is connected to HOLD of 8086

→ If 8086 wants to allow DMA. Then 8086 send ackno- from HLDA (8086) to HLDA (8257)

→ Then DMA sends DACK to peripheral & DRQ is received by DMA from peripheral then data transmission takes place.

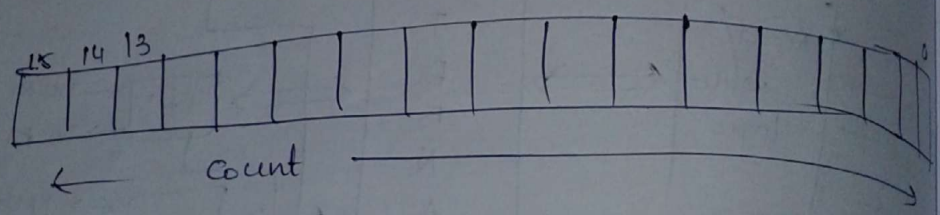
→ Each channel is capable of transmitting 16 bit

→ each channel has 2 16-bit registers. Add Reg & Count Register

→ A.R stores starting address of memory from where data transfer should begin

→ C.R. stores the count value

Count register format



Bit 15	Bit 14	Type of DMA
0	0	DMA verify
0	1	DMA write
1	0	DMA read
1	1	None

$2^{14} = 2^4 . 2^{10}$
$= 16 kB$

Data read from peripheral
Data written into memory

Data read from memory
Data written into peripheral

→ channel 0 has high priority CH3 low priority

→ Data bus buffer used to send data or control signals.

Scanned by CamScanner

select registers.

A_3	A_2	A_1	A_0	
0	0	0	0	— CH0—AR
0	0	0	1	— CH0—CR
0	0	1	0	— CH1—AR
0	0	1	1	— CH1—CR
0	1	0	0	— CH2—AR
0	1	0	1	— CH2—CR
0	1	1	0	— CH3—AR
0	1	1	1	— CH3—CR
1	0	0	0	— Control / status word.

→ 8257 is operated in 3 states

1. Idle state — performs nothing
2. slave " — MP will write CW to 8257 / reads SW from 8257
3. Master " — 8257 will control the system bus

→ $\overline{IOR}$, $\overline{IOW}$, A_0-A_3 used as i/p's during slave & o/p's during master

→ $\overline{MEMR}$ — memory Read. it is active during DMA read

→ $\overline{MEMW}$ — memory write. it is active during DMA write

→ AEN - Address Enable = 1 ⇒ used to disable (D_0-D_7) of 8086

→ ADSTB - Address Strobe - used to latch D_0-D_7 of 8086

→ TC - Terminal Count - when data transfer is completed then TC=1

→ MARK - for every 128 cycles it is going to be high. It indicates present DMA cycle is

128th from previous mark o/p. Mark always occurs at 128 cycles from the end of data block.

Types of DMA transfer.

Byte - Byte by Byte transfer

Block - Block by Block transfer

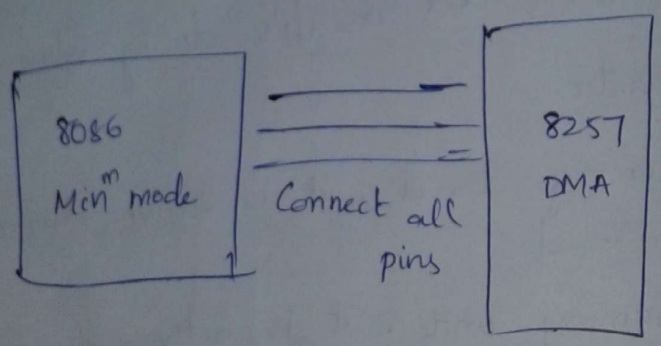
Ex: Total data 10kB

Block size 1kB

no. of cycles required - 10

Demand - untill 10kB of data transferred 8257 does nothing.

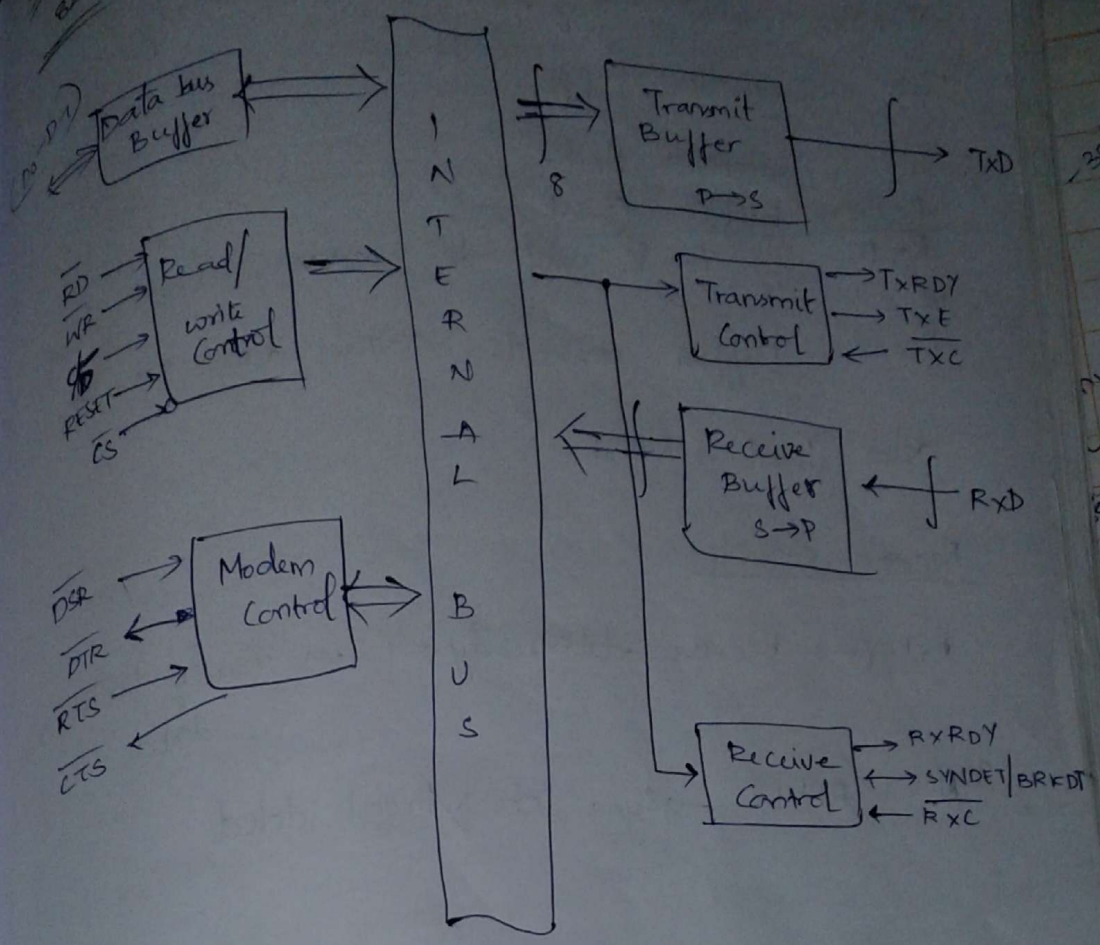
Interfacing 8257 with 8086:



Scanned by CamScanner

14/3/10

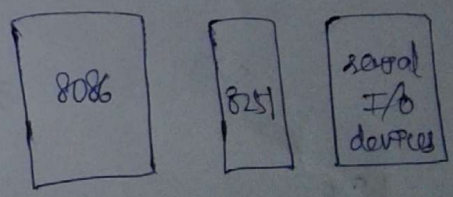
8251 USART:



8251 is connected to 8086.

It supports Universal Synchronous Asynchronous Receiver Transmitter.

B/w serial I/O devices and 8086, 8251 is connected



There are 7 blocks.

Transmit buffer

It is used convert parallel to serial data (8 bit → 1 bit)

Transmit control:

TxRDY - Transmit ready - when data is ready.

TxE - Transmitter empty.

$\overline{Tx}$ - Transmitter clock

Receive Buffer (S→P)

RxD - It is used to receive serial data.

Receive buffer converts serial data into 4bit data (1 bit → 8 bits)

Receive control:

RxRDY → Receive ~~data~~ ready = 1 ⇒ ready to receive data.

SYNDET/BRKDT → Sync Detect/Break detect

C/D = 0 ⇒ Data is transferred b/w 8251 & ~~8051~~ 8086.

C/D = 1 → control lines are transferred b/w 8251 & 8086.

$\overline{DSR}$ → Data set ready

$\overline{DTR}$ → Data ~~Tra~~ Terminated

$\overline{RTS}$ → Request to send

$\overline{CTS}$ → clear to send

Scanned by CamScanner

is checked by CPU using status read operation.

DSR is used to check if the data set is ready when communicating with a modem. (o/p signal)

DTR → It is used to check that the device is ready to access data when 8251 is communicating with a modem.

RTS → o/p signal

=0 → indicates Modem that receiver is ready to receive a data byte from modem

CTS → o/p signal

=0, 8251 is enabled to transmit the serial data provided the enable bit in the command byte is set to 1.

Command words:

RS 232 C:

→ Serial communicat^n standard or protocol which is connected b/w 8251 and I/O devices.

→ EIA (Electronics Industrial Association) in 1960 introduced this connector.

Scanned by CamScanner

$\overline{DSR}$ is checked by CPU using status read operation. This is used to check if the data set is ready when communicating with a modem. (o/p signal)

$\overline{DTR}$ → It is used to check that the device is ready to access data when 8251 is communicating with a modem.

$\overline{RTS}$ → o/p signal

=0 ⇒ indicates Modem that receiver is ready to receive a data byte from modem

$\overline{CTS}$ → o/p signal

=0, 8251 is enabled to transmit the serial data provided the enable bit in the command byte is set to 1.

Command words:

RS 232 C:

→ Serial communicat^n standard or protocol which is connected b/w 8251 and I/O devices.

→ EIA (Electronics Industrial Association) in 1960 introduced this connector.

Scanned by CamScanner

RS 232 C

9-pin DIP | 25 pin DIP

TTL → +5V → logic 1
0 → logic 0

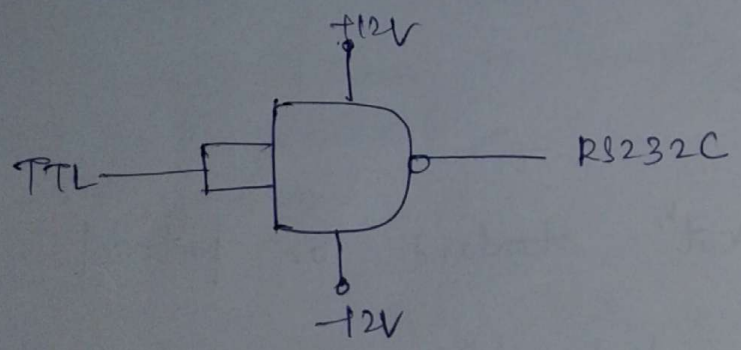
voltage levels are diff from TTL voltage levels.

+3V to +15V → logic '0'
-3V to -15V → logic '1'

8086 → supports TTL
8251 → " "
I/O → " "

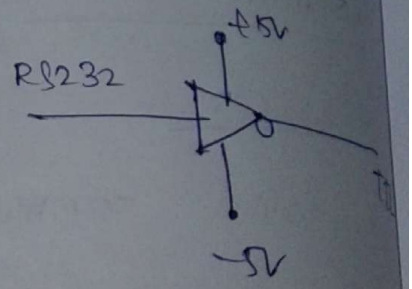
Line driver is used to convert TTL to RS232

Line Driver:



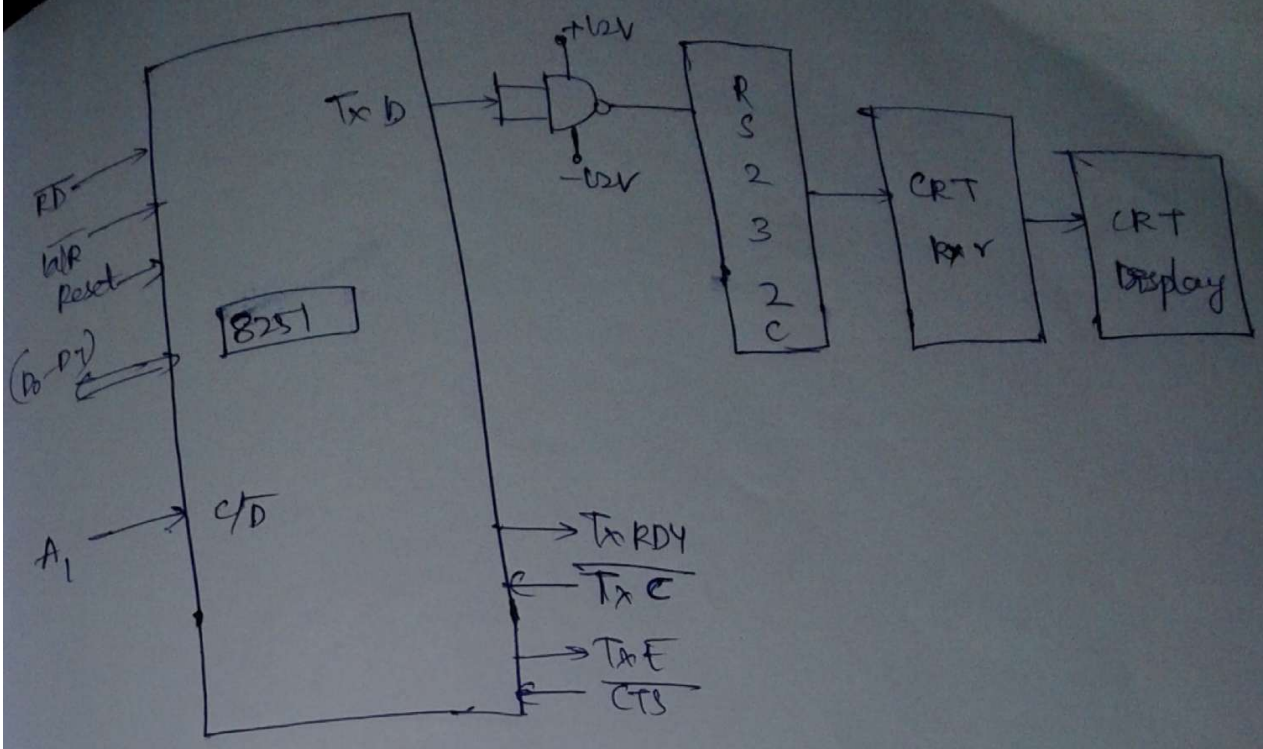
IC 1488

Line Receiver:



IC 1489

Interfacing serial I/O device to 8251:



Scanned by CamScanner