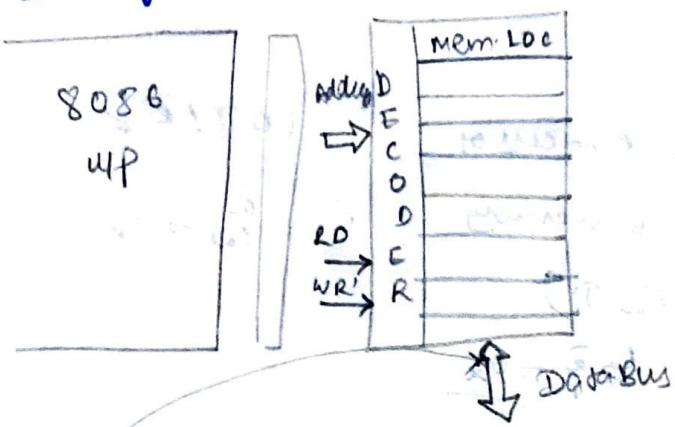
11-04-23

By default → if not specified: Memory is byte organized (8 bit)

Memory Interface

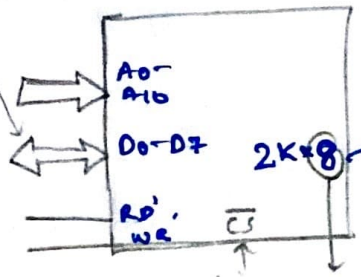


2K – memory chip

$1K = 1024 = 2^{10}$

$\{2K \Rightarrow 2048 = 2^{11}\}$

$\therefore$ 11 address lines (A0–A10)



2K × 8 bits = 16Kb ↔ 16K bits

= 2KB → 2K Bytes

Cluster:

000 ⋮ 299 → but can only name 99 chips in 1 cluster

000 → 099 | 100 – 199 | 200–299

0th Cluster | 1st Cluster | 2nd Cluster

vary from 00–99

will only check specific cluster (not all) 145 → in 1st cluster

(∵ only check 0, 1, 2 to see in which cluster)

[remaining: unused state]

Q. Use 2K Byte memory to interface 8K bytes of memory in 8088

→ will have to use 4 2K chips

↓

$2048 = 2^{11}$

2K chip → 11 address lines

- In previous example: 100 in each is decimal: $10^{2} \rightarrow$ 00 – 99 (varying)

∴ $2^{11} \rightarrow$ 11 bits in each cluster: (Binary)

000 0000 0000 → 000

⋮

111 1111 1111 → 7FF

just to write for Hexa

{like 00-99}

8K → 8×1024

$= 8 \cdot 2^{3} \cdot 2^{10}$

$= 2^{13} \rightarrow$ 13 address lines → 11 will go to each cluster

→ 13, 12 will help to identify which chip (cluster) = which column

{4 chips needed}

RAM1: $00000_H \rightarrow 007FF_H$ 0th → 00 → $A_{11}-A_{12}$

RAM2: $00800_H \rightarrow 00FFF_H$ 1st 01

RAM3: $01000_H \rightarrow 017FF_H$ 10 1st cluster

RAM4: $01800_H \rightarrow 01FFF_H$ 11

$007F9_H$ ↳ RAM2

800

$A_{11}A_{12}$

01000 0000 0000

11 bits

FFF

01 111 1111 1111

11 bits → This varies

This decides cluster

A_0-A_{10} $A_{13}-A_{19}$ → 0 for all

2K 2K 2K 2K

D_0 D_7

$A_{11}-A_{19}$ → can be used differentiate b/w chip / to Decoder

Decoder

18-04-23

Q. Design a memory system for 8088 (8) processor

ROM section using 2732 chip for an address 00000H - 02FFFH

RAM section using 4016 chip for an address 03000H - 03FFFH

Design memory interface using absolute address

ROM size: $\frac{32}{8}$ K = 4K → $2^2 \cdot 2^{11} = 2^{13}$ ↳ 13 0's - 13 1's

↳ 2-2K

RAM size: $\frac{16}{8}$ = 2K = 2^{11}

Address

ROM1 : 00000H - 00FFFH (4K)

ROM2: 01000H - 01FFFH (4K)

ROM3: 02000H - 02FFFH (4K)

RAM1: 03000H - 037FFH (2K)

RAM2: 03800H - 03FFFH (2K)

MSB A13 A12 ... A11 LSB

0 0 0 (4K)

0 0 1

0 1 0

0 1 1

1 0 0

1 0 1

1 1 0 (2K)

1 1 1 (2K)

0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 → 00000

0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 → 01000

0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 → 02000

0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 → 03000

0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 → 03800

A11, 12, 13

4K → 2^{12} → A0 - A11

2K → 2^{11} → A0 - A10

A10: common

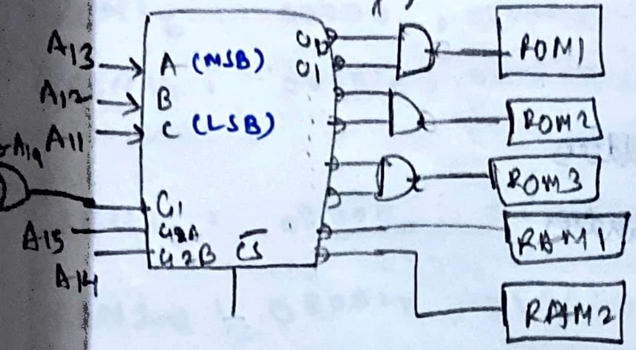
* A0 - A10 → Address lines

11, 12, 13

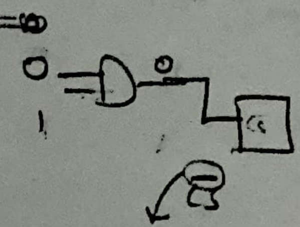
17-04-23

can only go O_0 | O_1 at a time

if O_0 ~~high~~ → low [$A_{13}A_{12}A_{11}$ 000] → All rest ~~low~~ high (off)



(AND → 001 2
↓
O_1 low → 0
O_2 → 1



should take lower

8086 → 20bit Address Bus , 1MB

80286 → 24bit Address Bus , 16MB

16 bit data transfer → (in one cycle)

Q. MOV [437A$_H$] , BX

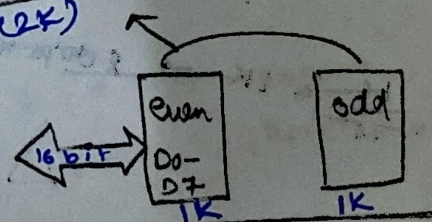
BL → 437A (8b)

BH → 437B (8b)

(8088)
previously → 1 address over 1 chip

now → 1 address over 2 chips (bank)

1 address (2K) [even should be with D0-D7 else with odd it will need 2 cycles]



must Divide address to 2 banks (chips)

8 bit → D0 - D7

8bit → D8 - D15 → $\overline{BHE}$ high
or Bus High Enable (for higher bits)
8bit → D0 - D15 → 8-15 or full 0-15

in 8088 → continuous address

8086 → we split

$A_1 - A_{19}$ | $A_1 - A_{19}$

00004
00002
00000
even ↑
A_0

00003
00001
odd ↑
$\overline{BHE}$

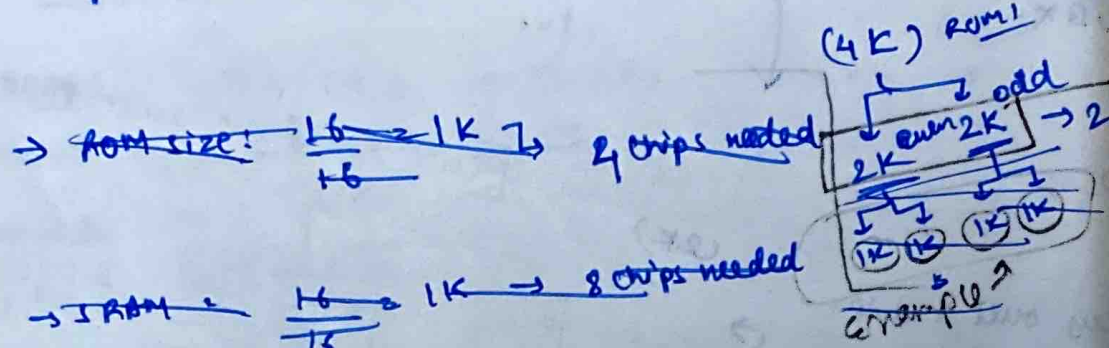
A_0 → Control signal (to choose b/w even/odd)
↳ cannot use A_0 for both [as it will activate 1 at a time only]

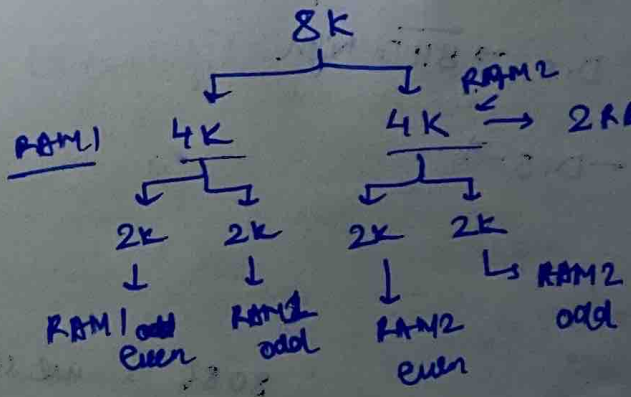
for 16 bit data ~~transfer~~ transfer (we need to activate both) at same time

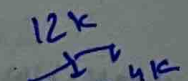
Example

memory chips 2KB each,

Interface 4K 2716 (ROM) starting at 00000 H , 8K 6116 (SRAM) starting at 08000







*(Address space = double the size of chip)

4K → 2 × 2K

18-04-23

ROM1 E : 00000H, 00002H, 00004H, ... 00FFEH → even

4K { ROM1 O : 00001H, 00003H, 00005H, ... 00FFFH

[Although, 4K split into 2K, address is continuous. $2 \cdot 2K = 2^{12}$ ⇒ 0000 - 00FFF]

00000 - 00FFF

08000 - 08FFF

RAM1 E : 08000H, 08002H, 08004H, ... 08FFEH

RAM1 O : 08001H, 08003H, 08005H, ... 08FFFH

09000 - 09FFF

RAM2 E : 09000H, 09002H, 09004H, ... 09FFEH
(A0)

RAM2 O : 09001H, 09003H, 09005H, ... 09FFFH
(BHE)

(A_{15} - A_{19} : 0's)

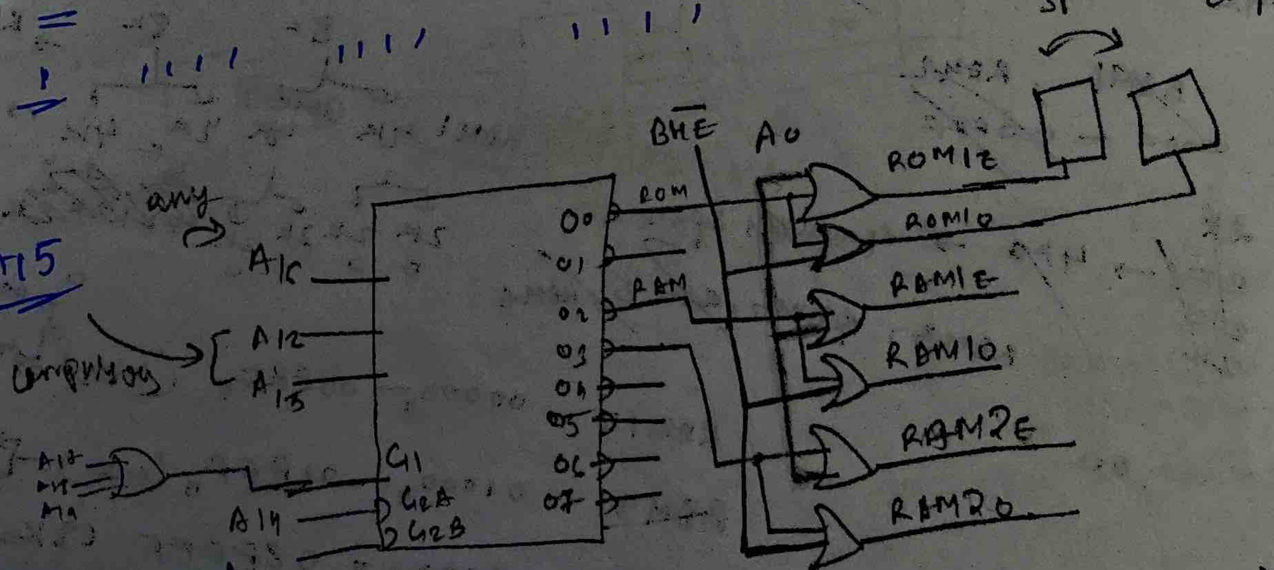
A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1

ROM → A0 for decoding; Address bus

ROM1 0FFF

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 — 08000
1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 — 0FFFF RAM1

1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 — RAM2
1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1



instead of OR gate ↵

∴ all odd (ROM, RAM1, RAM2) → to one decoder (BHE to G2B)
all even → to another decoder (A0 to G2B)

(more capacity)

2^8 ↳ 8 address line

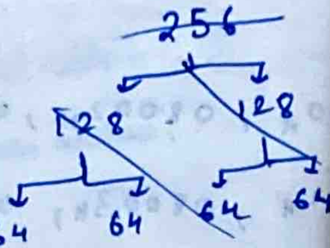
Q1 Interface 1M of SRAM to 8086, chips of size 256 K

Starting 00000 H
Ending FFFFF H

↓ All varying

10^6

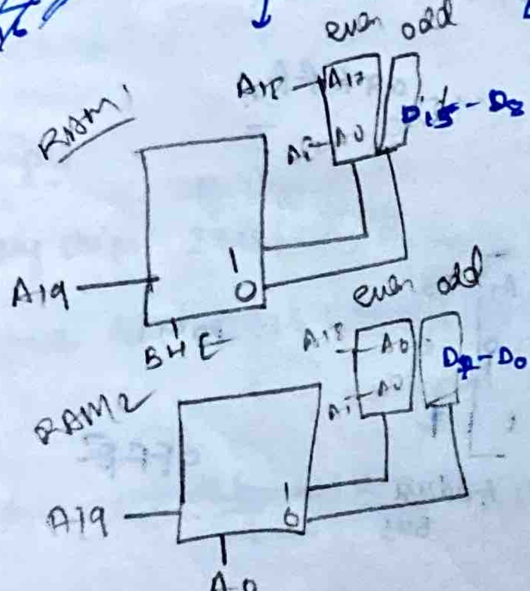
$2^6 \to 64$

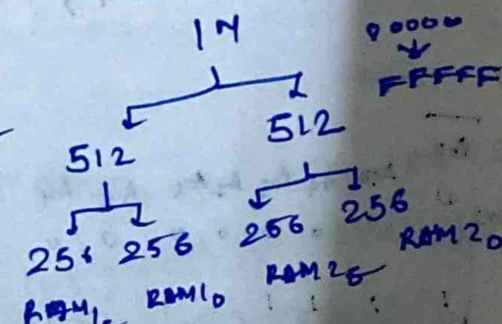


~~256/16~~

~~256/16~~

4 chips
↳ 2 address spaces





Q2 Interface 16K RAM to 8086, starting 00000

Chips ~~~~ 2K (4 chips) and 4K (2 chips)

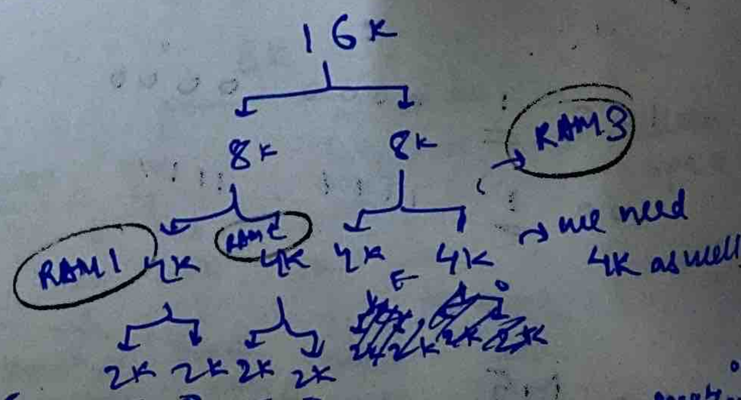
4K 2^{12}
4K → 8K → 2 address space

RAM1, RAM2
00000 — ~~00FFF~~

2K
2K → 4K → 4 address space
2K
2K

RAM3, ~~RAM4, RAM5, RAM6~~

00



E → 00000, 00002, 00004 ... 00FFE
O → 00001, 00003, 00005, ... 00FFF

RAM1: 00000_H — $00FFF_H$ (4K)

RAM2: 01000_H — $01FFF_H$ (4K) → 01000, 01002, ... / 01001, 01003, ... 01FFF

RAM3: 02000_H → 03FFF (8K)
↳ 1FFF
→ 02000, 02002, ... 03FFE
→ 02001, 02003, ... 03FFF

19-04-23

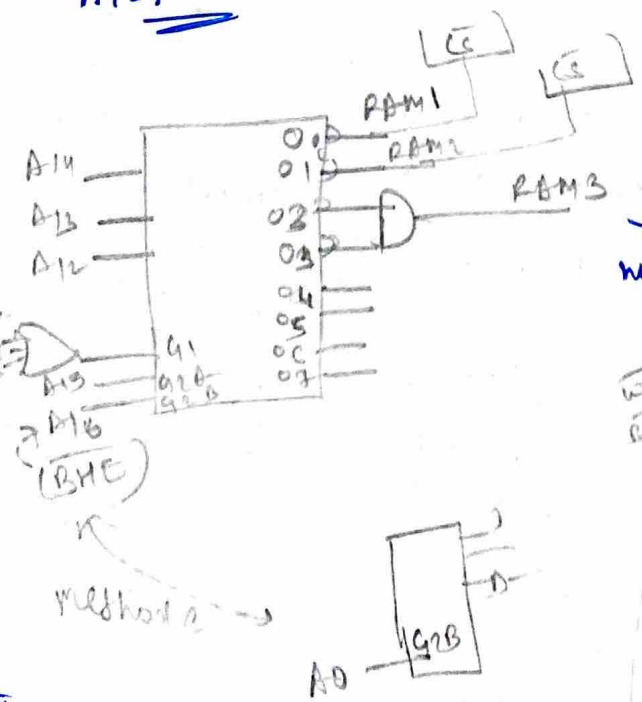
A19 A18 A17 A16 A15 A14 A13 A12 A11 A10 A9 A8 A7 A6 A5 A4 A3 A2 A1 A0

RAM1

RAM2

RAM3

A12, A13



Any next Address Cycle = 3rd decoder combination

→ A14 A13 A12

0 0 0 RAM1

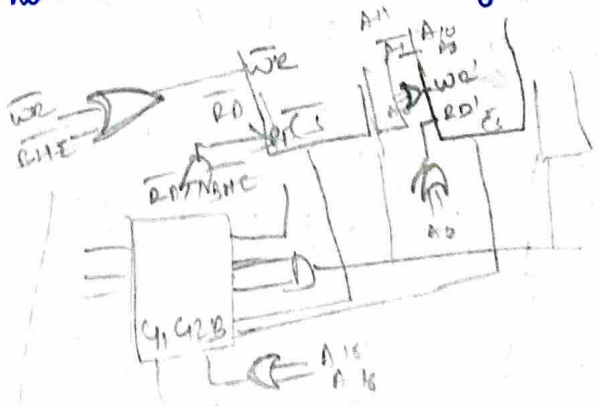
0 0 1 RAM2

0 1 0 } RAM3

0 1 1

→ now and →

Hence AND gate is used



(used for Address lines)

Address = of memory space

~~19 04 83~~

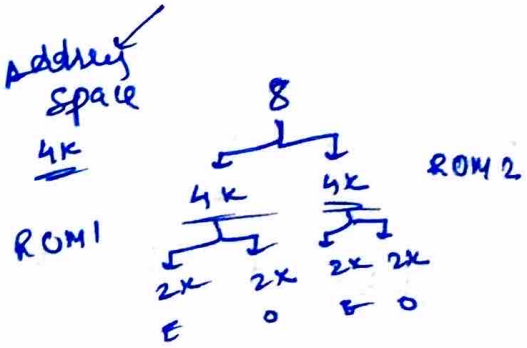
Memory interfacing using 80286

→ 24 address lines

Q. 8K of ROM to 80286 starting 080000H . 2K chips available

ROM1 : 080000 – 08FFFF

ROM2 : 080000 – 08FFFF

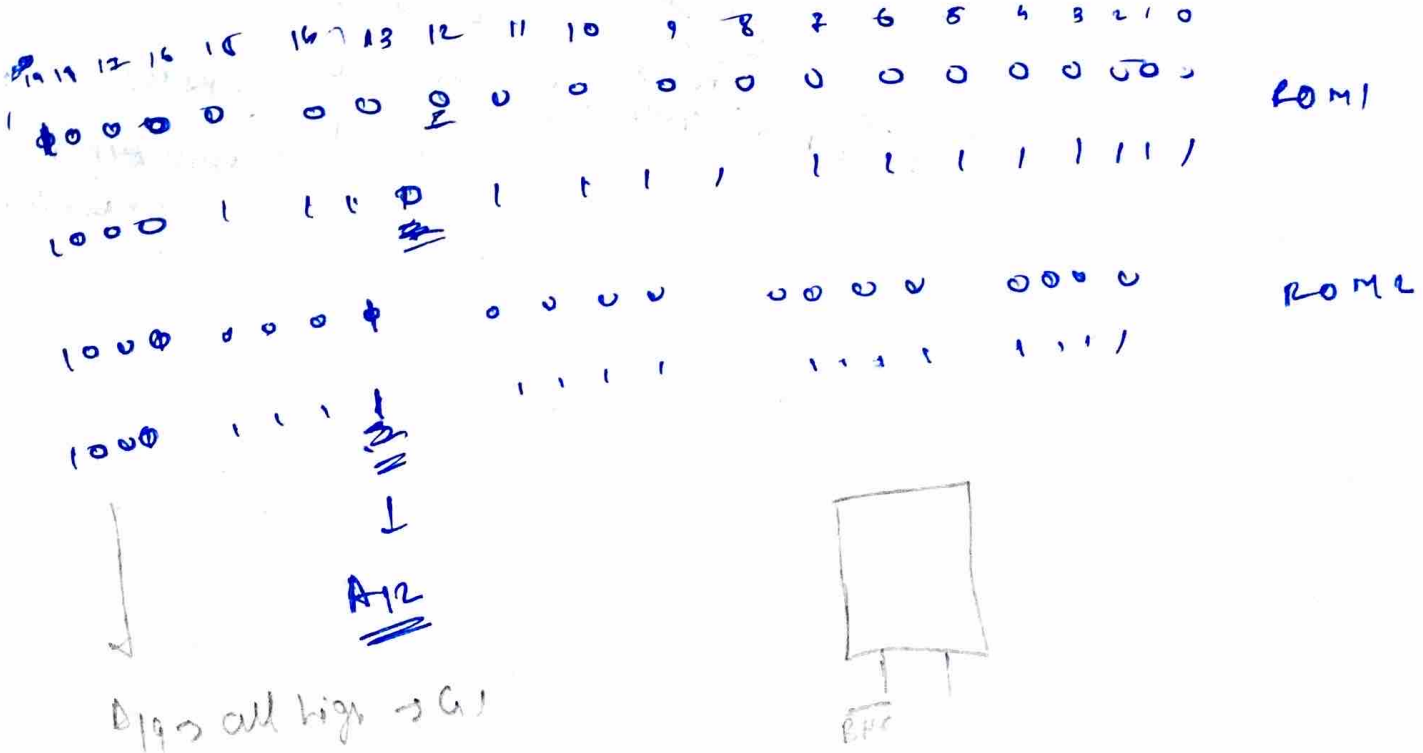


ROM1E : 080000, 080002, 080004 --- 08FFFE

ROM1O : 080001, 080003, 08005, --- 08FFFF

ROM2E : 090000, 090002, 090004, ... 09FFFE

ROM2O : 090001, 090003, 090005, ... 09FFFF



25-04-23

Tutorial Ques-7

[if diff sizes → eg: 32K : A_1-A_{15}
64K : A_1-A_{16} → 16 for address/decoding

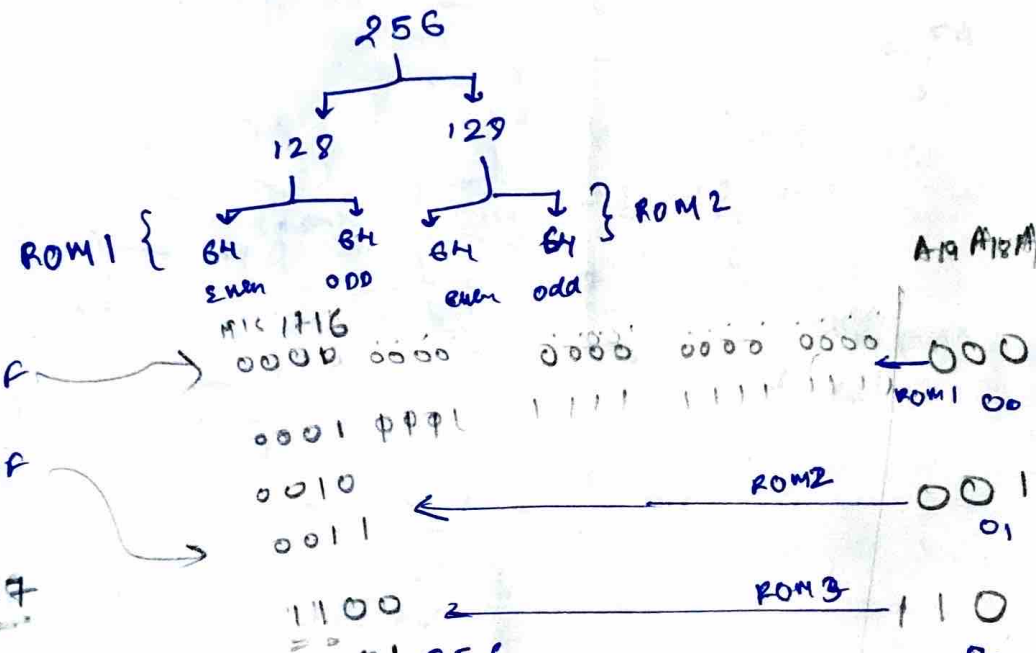
8086

Q.1 256K ROM, ~~1000~~ 00000
256K ROM, 40000
256K RAM 60000

64K
2^{16} → A_0 - A_{15} → for chip addressing

128K
2^{17} → 17 address lines

Chips: 64K,



ROM 256(00000)

ROM1: 00000 — 1FFFF
ROM2: 20000 — 3FFFF

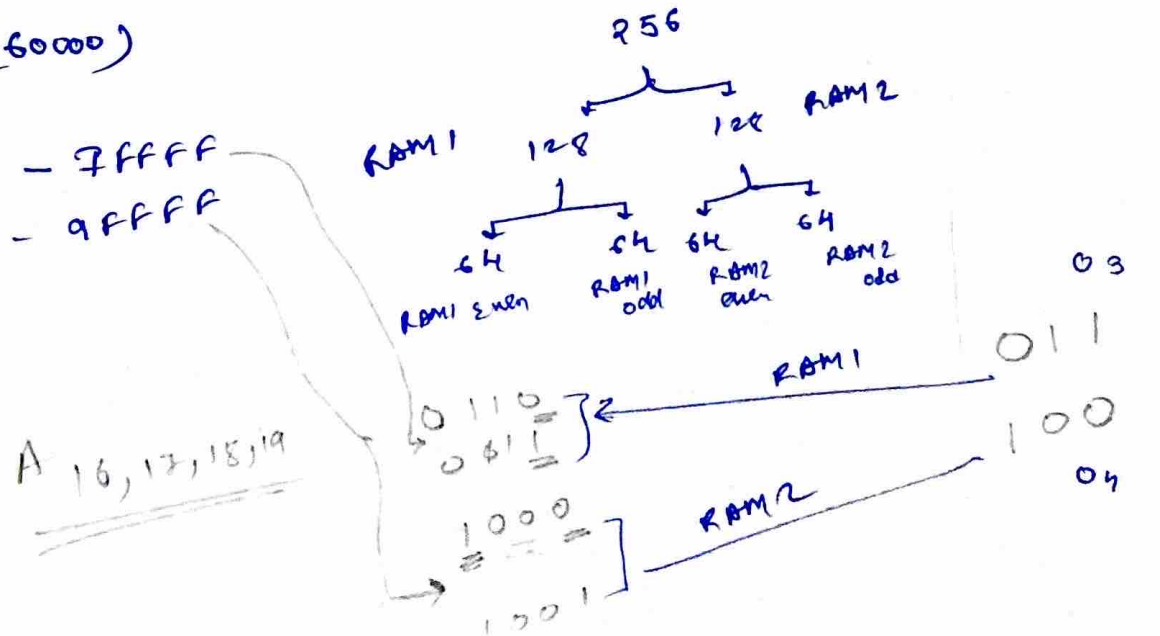
A16, A17

ROM 256(40000)

ROM3: 40000 — DFFFF
ROM4: E0000 — FFFFF

RAM 256 (60000)

RAM1: 60000 — 7FFFF
RAM2: 80000 — 9FFFF



27-04-23

Interrupt flag should be high

Interrupts

↳ If high → checks for cond^n before responding

Hardware interrupt:

- INTR → Interrupt pin {Keyboard connect to processor here} (if connected, will respond through INTA)
- $\overline{INTA}$ → Interrupt Acknowledge pin
- NMI → Non-maskable interrupt → If Active → no cond^n checked ↳ will respond
  ↳ If ~~can be set trace~~ not checked ~~flag checked~~

↳ Processor checks status of NMI / INTR AFTER END of every instruction (checking for any interrupts)

software interrupt [execution of INT instruction

→ TRAP : INT 01 (→ vector number) {Interrupts program after each instruction {halts {waits for next command}

→ Exception (error) in program

→ works similar to subroutines → ~~service~~ interrupt service routine (ISR)

① device sends INTR signal
→ Pushes content of flag into stack

② μP sends INTA signal back to device (acknowledge)
→ Trap Flag, Interrupt Flag cleared
→ Pushes current CS to stack
→ Pushes current IP to stack

{POP IP → one, CS}

③ device sends back vector number
→ Branches to ISR ↳ under INT 03 commands

INT __ __ ↳ 0-15 ↳ 0-15
→ 2^8 → 256 possible (0 - 255)

[INT 00H → INT FFH]

First 1K memory space (00000 - 003FFH) allotted for interrupt
[CS → 16b
IP → 16b]
4 memory space
↳ 4 × 256 = 1024 (1K memory)

- INT 00_H → 00000 00001 IP 0 — {Each ~~segm~~ interrupts require 4 memory location to CS, IP}

  00002 00003 CS 0

- INT 03 → 0000C 0000D IP 3 — (3×4=12 ↳ C)

  0000E 0000F CS 3

- INT0 → Divide by zero error interrupt
  ↳ quotient too large for AL/AX

- INT 1 → Single step interrupt
  when TF high / press F7 → Program executed line by line

- INT 2 → NMI
- INT 3 → Breakpoint [Emit
- INT 4 → Into {overflow flow flag checked}

  MOV AX, ——
  ADD AX, BX
  INTO / NOP

  OF=1 ↙ Into executed | OF=0 ↓ → To Skip → NO operation

- INT 5 → BOUND
  BOUND AX, [SI]
- INT 6 → Invalid opcode
- INT7 → Co-processor not available
  (eg: to do float operation: upto 386 co-processor)
  → ESX prefix to show use of co-processor
- INT8 → Double fault (more than 1 fault)
- INT9 → co processor segment overrun
  ~~Real mode~~ (8086)
  ~~co-process~~
  Displacement cannot be more than 64K ($FFFF_H$)
  (B → NO segment, C → stack overrun)
- INTA → Invalid Task segment

01-05-23

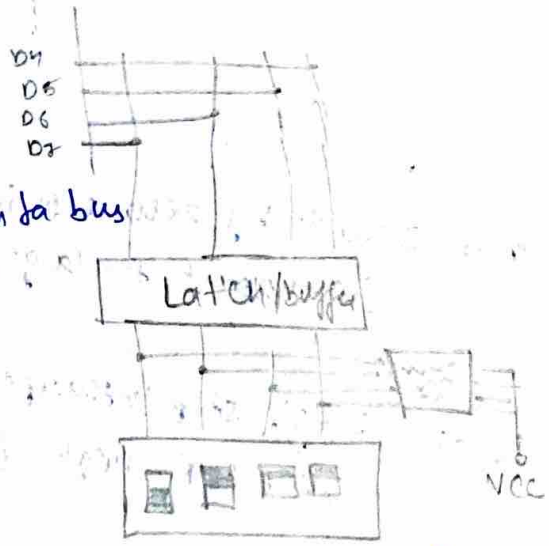
INT 11 → Alignment error { starting address not even }
(486 interrupt)

↓
Devices connected to INTR pin (eg: Keyboard)
↳ generates its own vector number

CLI → clear IF → 0
STI → set IF → 1

When resetting system, IF = 0

vectoring of INTR

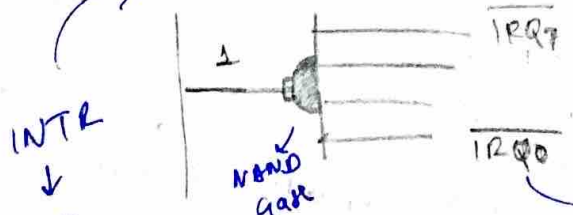


- Keyboard sends vector signal to data bus via ↳ Latch
  ↳ eg: 7F vector no.

Similarly, mouse
↳ eg: FF vector no.

How to connect more than one device with INTR ←

vector no. ← sends INTA back



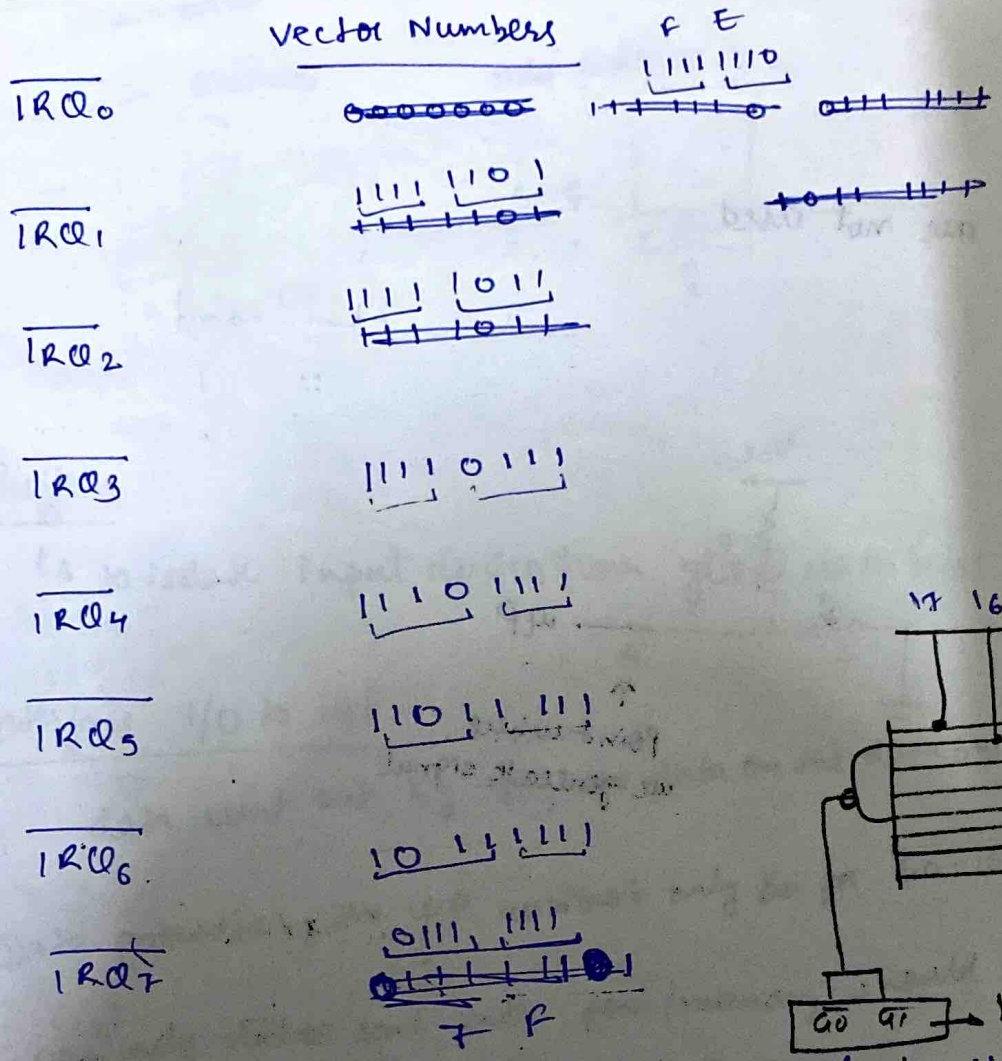
} n no. of diff. devices

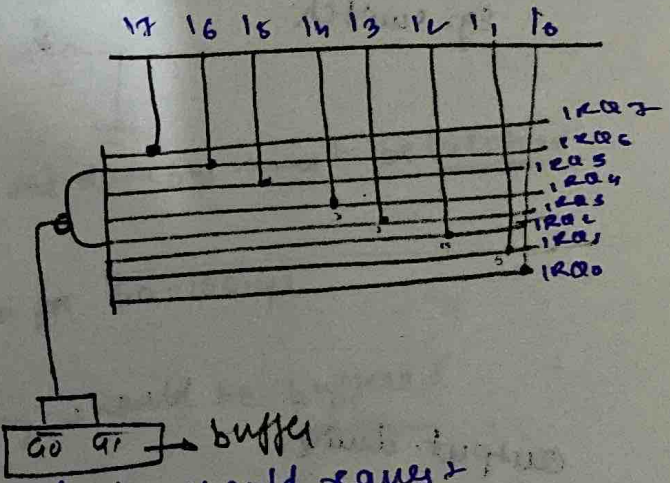
→ interrupt request

(any one is low, NAND → High)
↓
INTR is High

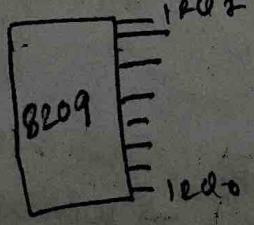
~~Left all high~~

(devices active low





↳ only one device should raise



→ Priority — IRQ7 → 1st

IRQ6 → in queue

02-05-23

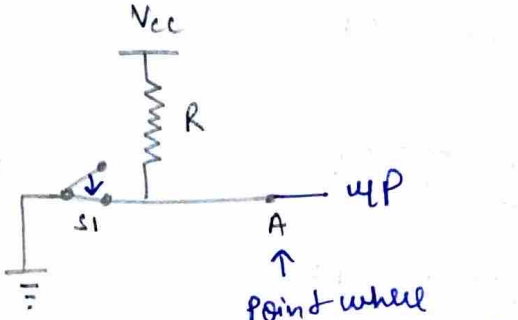
Memory mapped I/O

IN and OUT instructions are not used

I/O interfacing

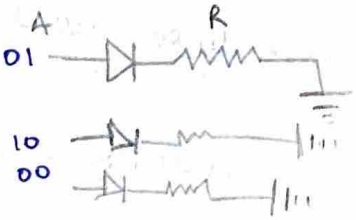
Input device

eg: switch



output device

eg: LED



IN instruction (from I/O mapped I/O) → Dedicated address for input/output
→ separate address for memory

IN Destination, Source (accumulator) (port → address where device is connected (I/O)

IN EAX, 04_H MOV DX, 2000_H (MOV not for device, only memory)

IN AL, 00_H

IN AX, 20_H → Fixed Port address: 8 bit ($2^8 = 256$) devices
→ variable address: 16 bit (> 256 devices)

~~DX, AX~~

16 bit device ← AX, DX

8 bit device ← AL, DX (variable addressing)

32 bit device ← EAX, DX

8	16
8	8
16	16
16	8

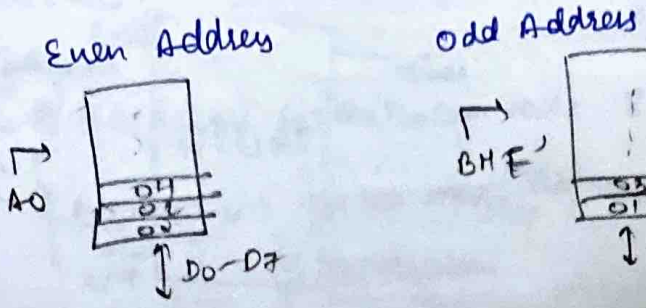
→ similarly for OUT

OUT 05_H, AL

OUT DX, AL

04-05-23

~~Even Addr~~



Buffers

Is to isolate input devices from global data bus

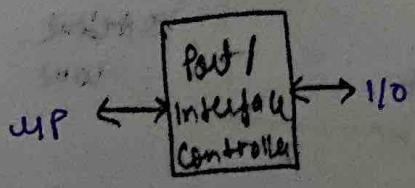
[Output devices → Latch]

Interface I/O to µP

data sent out by µP, the data on the data bus must be latched

Data provided by the µP available only ~~for~~ for 50-100µs

Similarly data sent into port/memory should be buffered



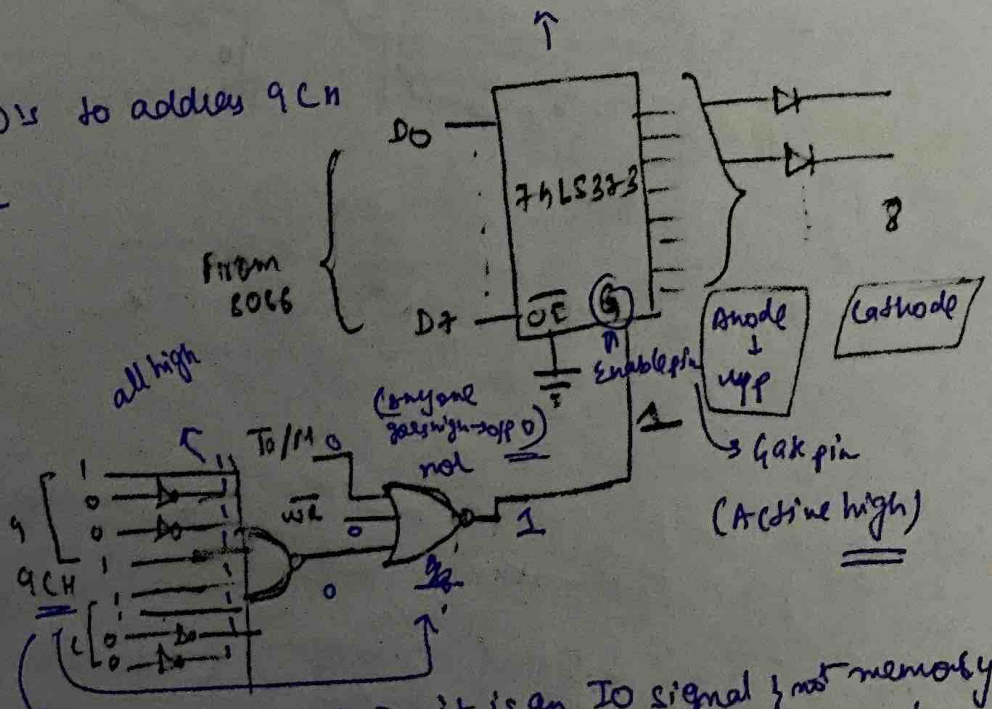
Latch → always active high enable

Connect set of 8 LED's to address 9CH

OUT 9CH, AL

∴ To turn on odd LED's

1 3 5
1 0 1 0 1 - -



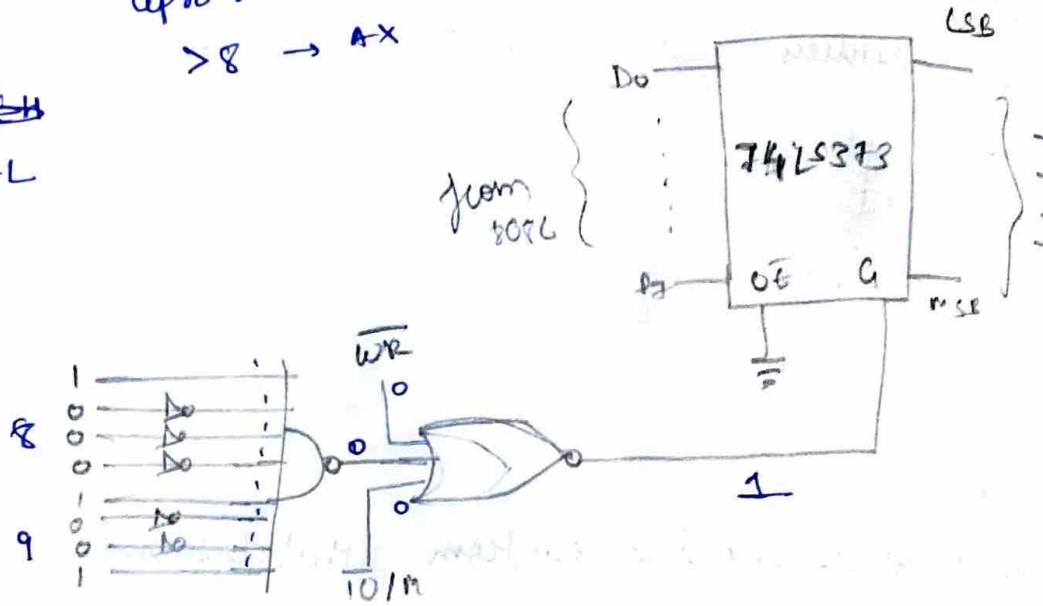
How will it recognize it is an IO signal & not memory loc

⇒ ⇒ M/IO → should be 0

Design ~~AL 89H~~

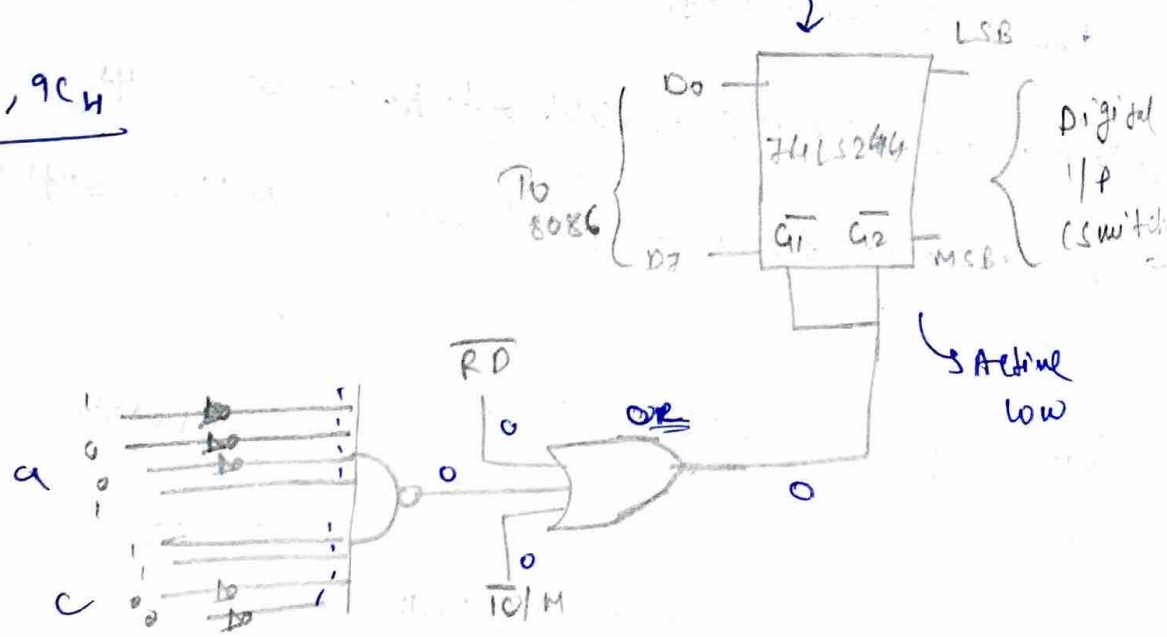
OUT ~~A 89H~~
89H, AL

upto 8 LED's → AL
>8 → AX



IN AL, 9CH

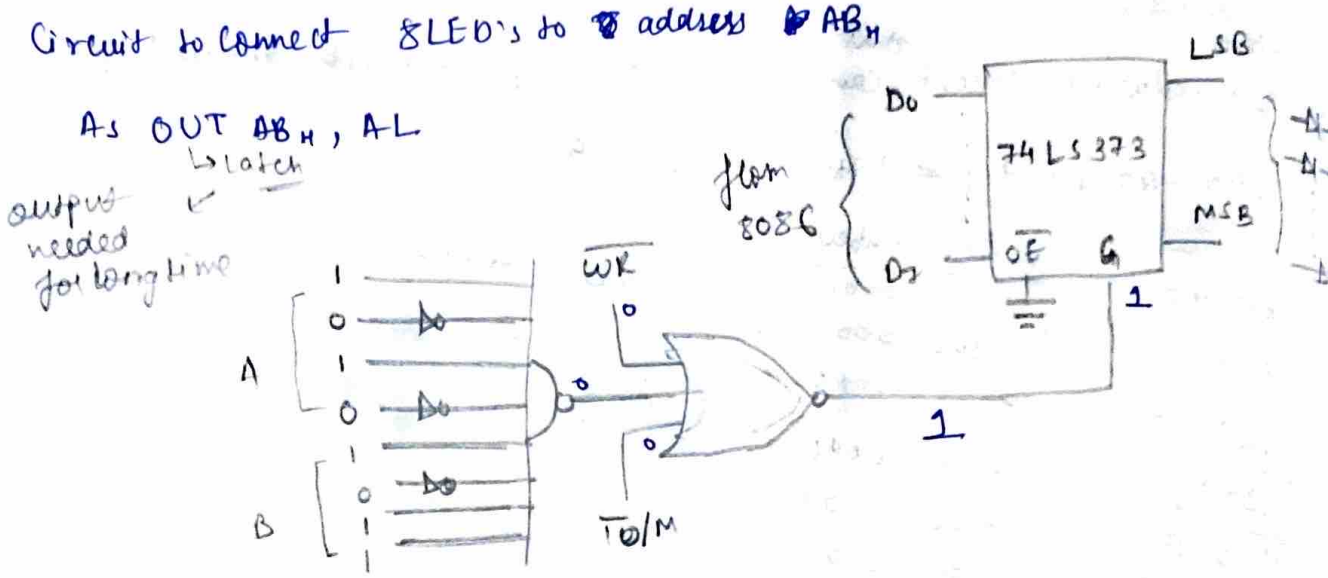
Buffer



08-05-23

Q-1 Circuit to connect 8 LED's to address AB_H

As OUT AB_H, AL
↳ Latch
output needed for long time



Q-2 ↳ Port address (8 bit)

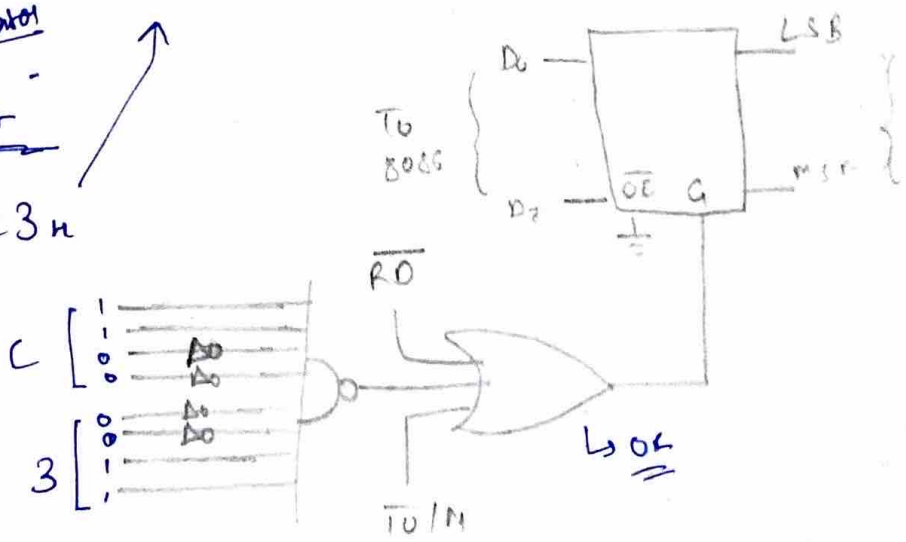
OUT PA, ACC
↳ Accumulator

OUT $6A_H$, AL
0110 ↳ Fixed addressing
1010

Q-3 ↳ Accumulator

IN AL, ~~PA~~

IN AL, $C3_H$



Q-4 An 8 bit output device is connected using variable addressing mode. The format of the instruction will be:

MOV DX, 2000H
OUT DX, AL → variable addressing (generally DX used)

~~OUT DX, AX~~
~~out DX, EAX~~

Programmable Peripheral Interface

8255 ppi

$\mu P \leftrightarrow$ [8255 ppi] $\leftrightarrow$ device

↓

system side of interface

↳ 3-8bit ports (A, B, C)

- A → 8b ⟷ $PA_0 - PA_7$
- Upper C (4b) → 8b → $PC_4 - PC_7$
- Lower C (4b) → $PC_0 - PC_3$ (can use a sub device / relay)
- B → 8b → $PB_0 - PB_7$

(from μp) $D_0 - D_7$ → (A_0) → [8255 Data bus]

∴ Even address

(A_0 → Low)

A_1 →, A_2 →, reset → [R/W control] ← CS

of $D_8 - D_{15}$ → [8255 Data bus]

↓

odd address

A_0 → high

A_1 →, A_2 →, reset → [R/W control] ← CS

Types of I/O Transfer

Simple

- Just to data transfer

Handshake

- check for device readiness
- Then transfer

C → Mode 0

B → Mode 0, Mode 1

A → mode 0, mode 1, Mode 2

D_7

- 0 – BSR (Bit set reset)
- 1 – I/O
  - Mode 0: Simple I/O (A, B, C)
  - Mode 1: Handshake I/O (A, B); C – Handshake
  - Mode 2: Bidirectional Handshaking I/O (A); C – Handshake; B – Mode 0, 1

(Eg: Harddrive – both input & output)

09-05-23

Control Word

D_7	D_6 D_5	D_4	D_3	D_2	D_1	D_0
1 - I/O Mode	Port A - Mode 00 - Mode 0 01 - Mode 1 1X - Mode 2	Port A 1 - i/p 0 - o/p	Port C upper 1 - i/p 0 - o/p	Port B mode 0 - Mode 0 1 - Mode 1	Port B 1 - i/p 0 - o/p	Port C lower 1 - i/p 0 - o/p
	Group A			Group B		

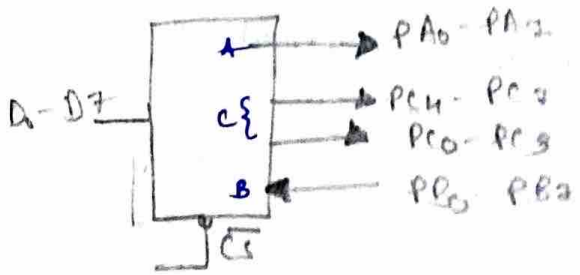
[Control word never less than 80]

Eg: What is mode & I/O configuration for ports A, B & C of 8255A after register is loaded with 82H

82

1000 0010

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	0	0	0	1	0
I/O		A → Mode 0	A_L o/p	C_H - o/p	B Mode 0	B → i/p	C_L → o/p



(if nothing mentioned)
simple mode of operation

Eg: Find the control word

PA = out

PB = in

PC0 – PC3 = in (CL)

PC4 – PC7 = out (CU)

If nothing mentioned
↓
(~~assume~~ simple mode of operation)
↓
mode 0

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

83H

10-05-23

Eg: Read 12 switches & display switch condition on 12 LED's with 8255H & Base address 00H

PA = 00H (Base address always to port A)
PB = 02
PC = 04H
CR = 08H

~~D7 — D6 — D~~

control word 83H (opposite 98H)

B → CL → input
A → CU → output

12 input
12 outputs

I/P [PA – 8 bit, LPC – 4 bit]

O/P [UPC, PB]

LPC, PB } I/P

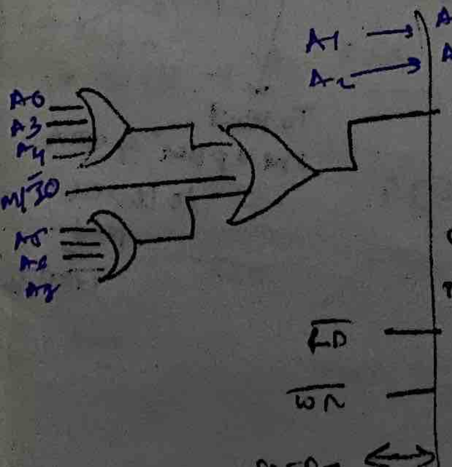
UPC, PA } O/P

PA PB LPC
1000 0011

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

& All 3 ports in mode 0 operation }

Base address 00



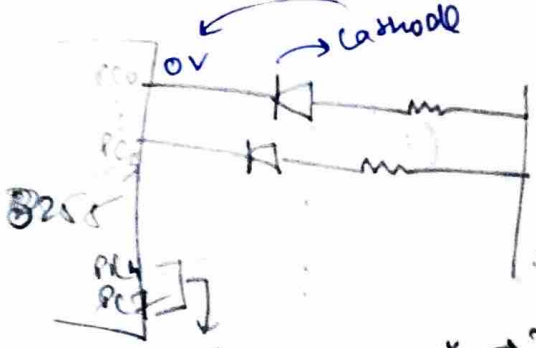
8255

A7	A6	A5	A4	A3	A2	A1	A0	
0	0	0	0	0	0	0	0	PA
0	0	0	0	0	0	1	0	PB
0	0	0	0	0	1	0	0	PC
0	0	0	0	0	1	1	0	CR → Control Register

00H → first row

A0 → Enables Even bank (even address)

→ A1, A2 must show all combinations to generate 4 addresses
→ All rest same



98h

Data needed → mask → rotate to lower port the

```
creg  equ 06h
porta equ 00h
portb equ 02h
portc equ 04h
```

Port Declaration

~~IN AL, 00h~~ → or 98h

MOV AL, 10011000b

OUT creg, AL

↳ store in control register

control word (98h)

```
IN  AL, porta
OUT PORTB, AL
IN  AL, PORTC
AND AL, 0F0h
MOV CL, 04h
ROR AL, CL
OUT PORTC, AL
```

PC4–PC0

[xx' garbage LSLC-D]

[we only need upper 4bits]

we mask by the (F0)

then rotate (half)

store in port C

To send the upper 4 data to lower 4 port (PORTC)

Bitset reset (BSR) (when less than 8 LEDs)

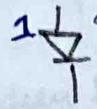
MSB							LSB
D7	D6	D5	D4	D3	D2	D1	D0
0 – BSR	x	x	x	Bit2	Bit1	Bit0	Bit set/reset

D6 D5 D4 → Don't Care

D3 D2 D1 → select the bit; D0: 1 – set, 0 – reset

PC →	0	1	2	3	4	5	6	7
B0	0	1	0	0	0	1	0	1
B1	0	0	1	1	0	0	1	1
B2	0	0	0	0	1	1	1	1

PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0
1	0	1	0	1	0	1	0



→ To turn on PC1

D0 ↳ set / Reset ↳ only one port of PC

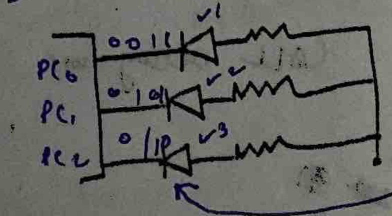
∴ To turn on PC0
↳ D0 → 1 (set)
D1 – 0
D2 – 0 → which port bit
D3 – 0

↳ BSR Word: 01

Eg: Connect 3 LEDs to Port C. Blink one LED after the other after regular intervals of 1ms

8255 Base address 00H

PC7 PC6 PC5 PC4 PC3 PC2 PC1 PC0



here 0 → to glow the LED (as cathode)

Eg: PC0 → PC2

1st cycle → on PC0 (creg equ 03H)

PC0	PC1	PC2
00H	03H	05H

1ms

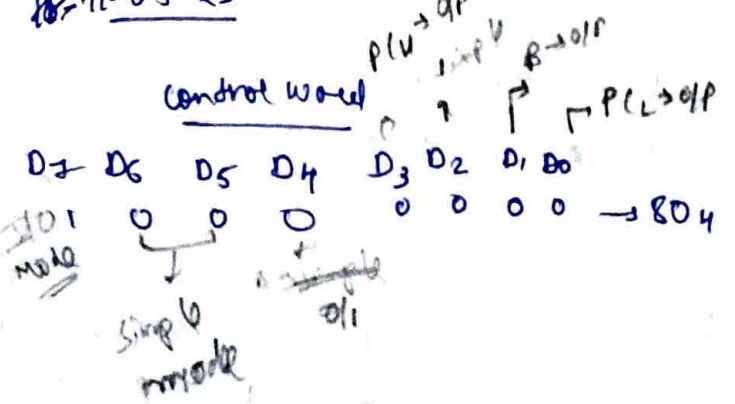
~~MOV AL, 00H~~
~~OUT CREG, AL~~
~~LX AL, Port A~~

0 0 0 1 → 01H
0 0 1 1 → 03H
0 1 0 0 → 04H

	D3	D2	D1	D0
PC0 →	0	0	0	0
PC1	0	0	1	1 → 03
PC2	0	1	0	1 → 05H
PC1	0	0	1	0

→ PC0

PC0	0	0	0	1 → 01H
PC1	0	0	1	0 → 02H
PC2	0	1	0	1 → 05H




```
CREG  EQU  0BH
      MOV  AL, 80H
      OUT  CREG, AL      ↳ Control word to control Register
```

TURN ON LED's

```
X1:   MOV  AL, 00H
      OUT  CREG, AL      ↳ BSR word (00) to control Register
      MOV  AL, 09H
      OUT  CREG, AL
      MOV  AL, 05H
      OUT  CREG, AL
      CALL Delay 1ms
```

This time put 0 for PC1

```
      MOV  AL, 01H
      OUT  CREG, AL
      MOV  AL, 02H
      OUT  CREG, AL
      MOV  AL, 05H
      OUT  CREG, AL
      CALL Delay 1ms
```

0 for PC0

```
      MOV  AL, 01H
      OUT  CREG, AL
      MOV  AL, 03H
      OUT  CREG, AL
      MOV  AL, 04H
      OUT  CREG, AL
      JMP  X1
```

There are set of 8 LEDs connected to Port C of 8255 - a logic high turns on the LEDs

↳ write a code snippet that will turn on LEDs connected to even bits of Port C using BSR

↳ 8255 is mapped to address 80H

↳

→ 8255

→ Result

		A7	A6	A5	A4	A3	A2	A1	A0
Base address:	80H → PA	1	0	0	0	0	0	0	0
	82H → PB	1	0	0	0	0	0	1	0
	84H → PC	1	0	0	0	0	1	0	0
	86H → CR	1	0	0	0	0	1	1	0

~~MOV AL, 80H~~

MOV AL, 80H
OUT CREG, AL

← Logic

PC0 000 ↓ (01H) LED on

PC2 0 0 0 ↓ (05H)

PC4 1 0 0 ↓ (09H)

PC6 1 1 0 ↓ (0DH)

MOV AL, 01H
OUT CREG, AL

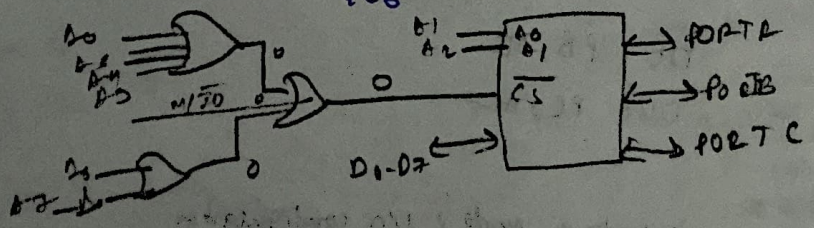
MOV AL, 03H
OUT CREG, AL

MOV AL, 05H
OUT CREG, AL

MOV AL 09H
OUT CREG, AL

MOV AL, 0DH
OUT CREG, AL

HLT



15-05-23

Tutorial on 8255

Q.1 A → Input + 00H (assume 8 bit)

B, C → output CR +06H

+02H +04H

Base address 50H

(Assume simple mode of operation)

90H

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	0	0	1	0	0	0	0
9				0H			

PA = EQU 50H

PB = 52H

PC = 54H

CR = 56H

MOV AL, 90H

OUT CR, AL [First always CW (control word) (control Register)]

IN AL, PA [Read data]

OUT PB, AL

OUT PC, AL

Q.2 What is mode & I/O configuration for ports A, B, C in 82C55A after CR is loaded with 82H → CW → 1000 0010

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	
1	0	0	0	0	0	1	0	PC (upper) → output
								PB → input
simple mode I/O	A mode 0 simple		PA output	PC (lower) output	B mode 0 simple			

824H − 04H = 7FH
(8085 μP)

Q.3 Write BSR word subroutine to set PC7 & PC3 → Port C address 82H (8085 μP)

BSR				Bits			set		
D7	D6	D5	D4	D3	D2	D1	D0		
0	x	x	x	0	1	1	1	→ PC3	→ 07H
0	x	x	x	1	1	1	1	→ PC7	→ 0FH

BSR word → BSR word written on ~~Port C~~ CR

```
MOV AL,0FH (PC7)
OUT 84,AL
MOV AL,07H (PC3)
OUT 84,AL
```

PA	80	7E
PB	82	80
PC	84	82
CR	86	84

~~7EH + 06H~~ (8085) = ~~84H~~

Q.4 Find the control word

PA → out
PB → in
PC0−PC3 → in (C Lower)
PC4−PC7 → out (C upper)

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

8 3H

→ Write code to send data from B → A & PCL → PCU

use starting address (300h) 303h should be 306h for 8086

```
CR EQU 306h
PA EQU 300h
PB EQU 302h
PC EQU 304H
~~CR EQ~~

MOV AL, 83H
OUT CR, AL
IN AL, PB
OUT PA, AL
IN AL, PCL
~~OUT PCU, AL~~
AND AL, 0FH → mask upper (data) values
ROR AL, BL → Rotate data to upper bits
OUT PC, AL
```

MOV BL,04H ← (can't do directly)

AL AND 0000 1111 → 0x ← ignore values; x = data (4 bit) → ROR 4 times (BL ← 04H) → x0 → PC (upper)

17-05-23

Q.6 Design 8086 based circuit using 8255 IC

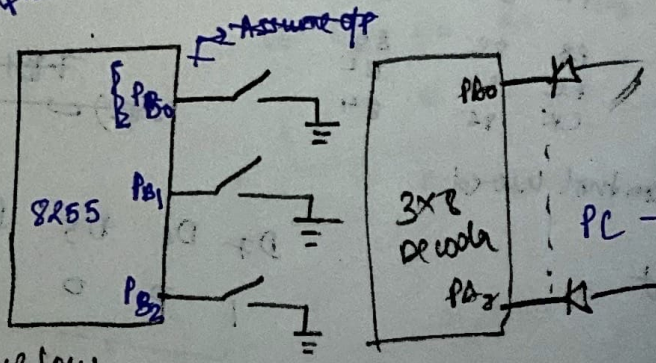
To check → compare with 8 bit data → AND
↳ OUT → FE
eg: 000

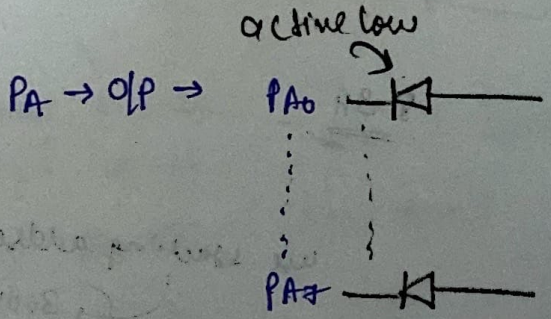
- Design 3×8 decoder — active low
- 3 switches to set input of decoder
- 8 LEDs

PB_2 PB_1 PB_0	PA7 ..... PA0 D0 ..... D7
0 0 0	1 1 1 1 1 1 1 0 = FE

eg: switch 000 → LED 0 glows
010 → LED 2 glows

PB_0 → input





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

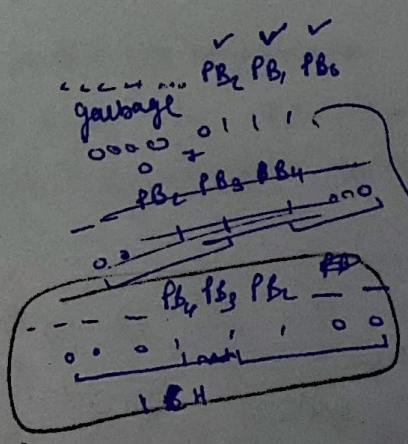
= 82H

Base Address: 00H

PA: 00H
PB: 02H
PC: 04H
CR: 06H

↳ 8086 (either even/odd)

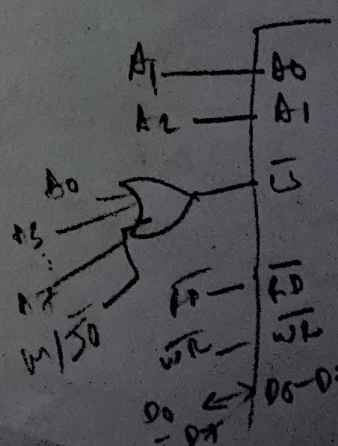
PB1

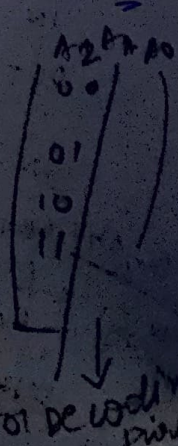


MOV AL, 82H
OUT CR, AL
IN AL, PB_2
AND AL, 07H → Just draw table of this

(write the logic you will use)

The switch eg: → to connect to → AND with 1
{all others to 0}





Have all to be 1 → To activate

For decoding purpose

Q-7 Write a subroutine that would set the odd bits of Port C of 8255 PPI using BSR. Assume address of control register is 06H.

D7	D6	D5	D4	D3	D2	D1	D0	
				Bit			Set	→ set odd bits of Port C
0	x	x	x	0	0	1	1	PC0 = 03H
0				0	1	1	1	PC3 = 07H
0				1	0	1	1	PC5 = 0BH
0				1	1	1	1	PC7 = 0FH

Base Address = 06H

PA: 06H

PB: 0EH

PC: 0AH

CR: 0CH