MICROPROCESSOR

RAM → SRAM → based on flipflops
→ DRAM → CMOS + Cap.

SRAM → static
expensive

DRAM → dynamic.
cheaper.

Capacitor can leak the charge so there is a refreshing circuit for capacitor.
discharge → speed goes down.

Memory in processor → level 1 catche. → for smaller calculations.
RAM → level 2 catche.
Hard drive → level 3 catche

RAM → used in servers

8085 → the first processor used as part of computer. (8 bit).

CPU → Central processing unit.
MPU → Micro processing unit
MCU →

Anything that has microprocessor as its fundamental unit is called embedded system.

What does a processor do?

Basic arithmetic operations.

Fundamentally, digital world is based on AND, OR, NOT.

8085 → 8 bit 8086 → 16 bit

80286 → 16 bit 80386 → 32 bit

80486 32 bit

Pentium 1 → 32 bit

Pentium 2 "

Pent 3 "

Pent 4 "

Pent 4 → 64 bit ⇒ speed 3 GHz

One 64 bit operation takes 2^{32} steps on a 32 bit processor.

Dual core Processor → two processors running in para

Core 2 Duo → two processors on a single chip & share process at the same chip at the same speed.

Core 2 quad

Core i3

Core i5

core i7

two architecture structures of microprocessor;

i) Van Neu Man. → ALU one instruction at a time.

ii) Harvard.

GPU → Graphics processing unit → MAC

There is no speed of processor → it is of the oscillatory source outside the chip.

BIOS → knows about the architecture and address of everything on the motherboard.

Ports → Parallel → LPT1 LPT2
Ports → Serial → com 1, 2 12

Microprocessors by Goankar.
Microcontroller by Mazidi.

LECTURE 2:

All digital ICs require fundamentally a source because it process data in the form of voltage.

Sequential circuits → states → require a sense of time → provided by providing a clock source → in this case a crystal oscillator. → square wave form ⊓⊔⊓

⊣⊢ → AF capacitors → Audio frequency. → has polarity used in rectification

⊣⊢ → RF → radio frequency capacitors →
in series it blocks DC
in parallel it blocks AC.

Clock source connected → sequential.

Instruction → Activity 2^n

Program? Set of instructions

Processor executes an instructions. → 8 bit instruc

Where does instruction come from?

There is no instruction / memory on the processo

The programs are always written outside the processor

When you have memory, you need an address.

Memory mapping → we assign addresses to memory locations.

How to access any address?

No. of address lines.

eg 64KB of memory → 2^{16} → 16 address line

1TB memory → 2^{40} → 40 address lines

All the processors for microprocessor must be
8 bit in size.

Memory parameters → Address lines, data lines.

How processor accesses the instructions?

Program counter is a 16 bit register.
↳ It points towards the next instruction to be executed.

When you start the PC, the program counter generates the first memory location

16 address lines with the processor.
first 8 → $AD_0 \cdots AD_7$ →
next 8 → $AD_8 \cdots AD_{15}$ →

AD → Address and Data.

How do we know whether there is address or data at that time?
ALE → Address Latch enable.
1 → Address 0 → Data

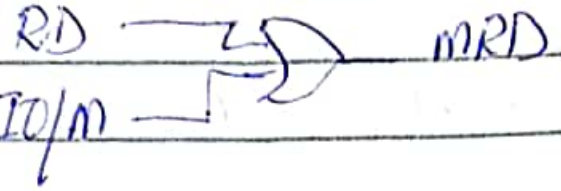
At the instant we start the processor → there are only addresses.

How does a processor know what to do?
ie read, write, etc...

Pins → READ
→ WRITE
→ IO/M → IO → Input Output
M → memory (ports)

Memory, Ports → on the motherboard.

Memory read → MRD



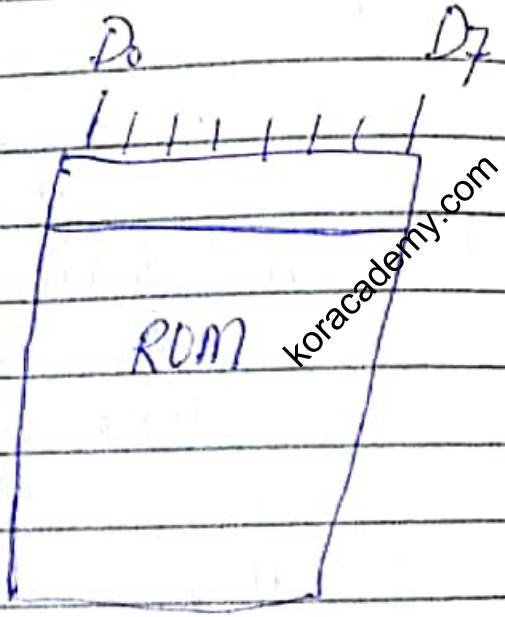
TTL → 5V

74373 → 20 pin IC

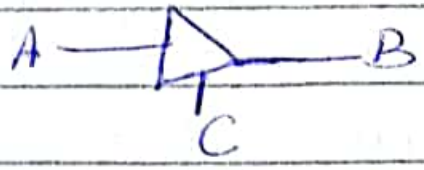
8 tristates and 8 f.f's.

output Enable → OE

Latch enable → $\overline{LE}$

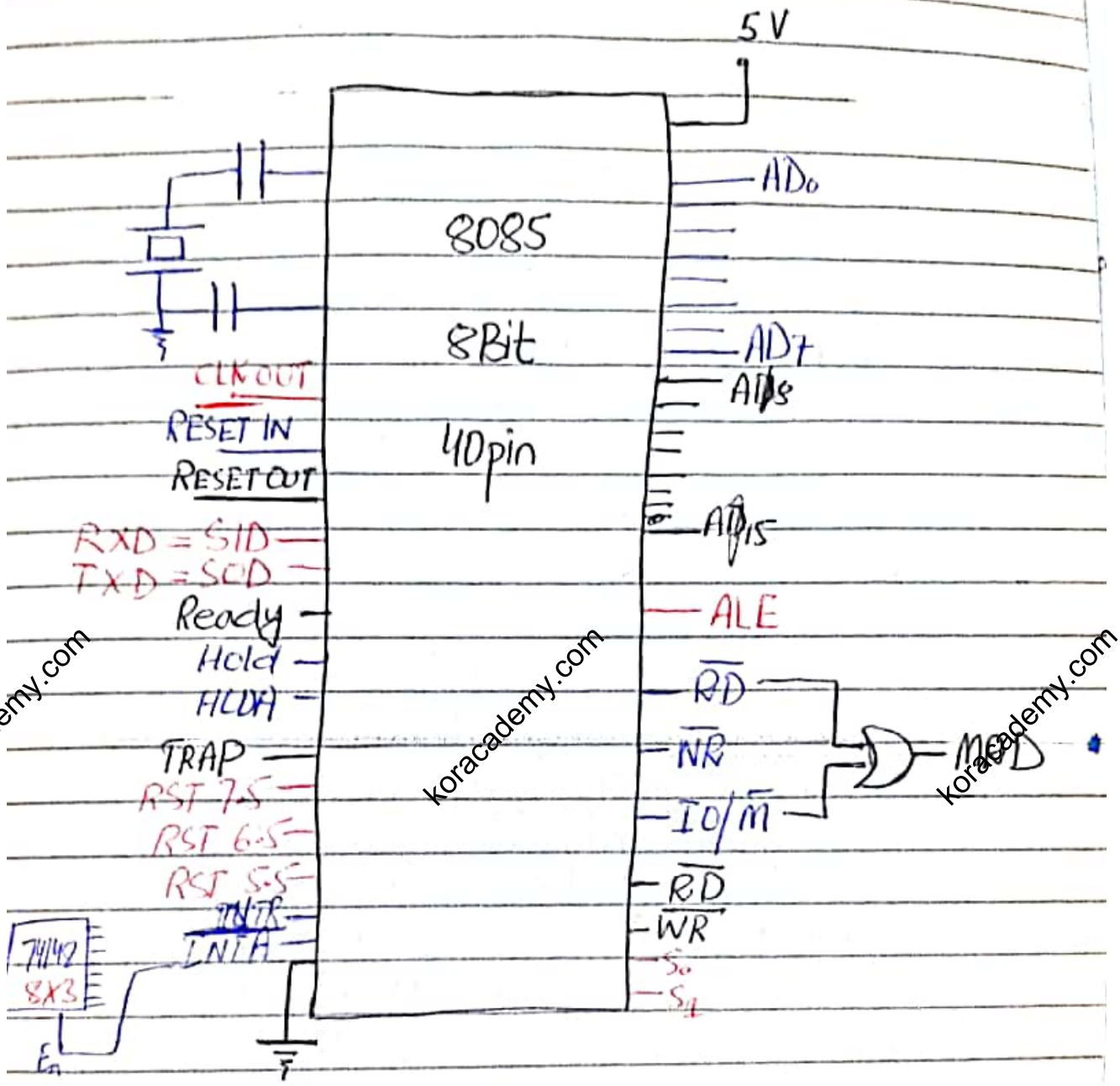


Tristate buffer →



If C=0 A — B

If C=1 A ~ B



First cycle

Demultiplex the address → Demultiplexing address latch data lines.

→ process → Use a temporary memory.

Seund cycle

Instruction gets transfered from external memory to the instruction register.

ALE = 0

Program counter increments.

Third cycle

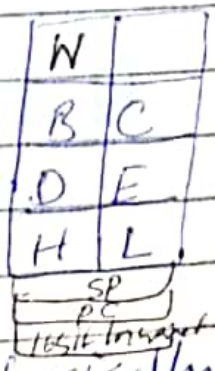
The output of instruction register is enabled and it goes into the instruction decoder.

Decoder decodes the instruction.
And now the instruction goes into the control and timing unit.

The process of fetching instruction → fetch cycle
→ first 3 cycles (3T states)

Control and timing unit actually translates your instruction to activity.

ACC → Accumulator.
These can act as a pair and can store 16 bits of data.

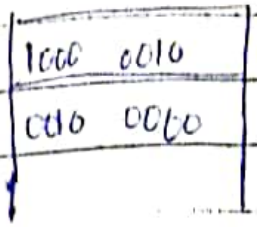


8 bit register

Basically (16 bit)

Stack Pointer is a counter
Stack memory is in the RAM
Every time something is stored stack pointer increments and vice versa

let an instruction MVI A, 20H
operational code
let it be B2



CLKOUT → to provide clock pulse to other devices on the motherboard.

Reset in → Changes the value of program counter to zero.

Reset out → To tell other things on the board that I have restarted.

Lecture 3

How to troubleshoot Microprocessor?

Remove crystal oscillator → manual button

Two status pins → tells us about 4 possible statuses
→ used for troubleshooting.

debug → software troubleshooting → hardware

Synchronous → speed of transmission = speed of recieving
for such communication we only need 2 lines.

If we are not sure about the speed, the slowest one decides the speed.

The ready pin checks if the prevous has been recieved or not → when yo can send the next bit.

DMA → direct memory access → used for copying
8257

Processor is not used for copying.

P

Hold → to ask µ processor for using buses.
Timing calculated based on how much time buses will be available for DMA.

HLDA → Hold acknowledge → response of the µp

Interupts → changes the value of program counter and the current address is stored in stack

Polling → eg I am looking at the door (monitoring) and someone enters.

Interupting → eg I am busy with my work and someone comes in and I look at the door.

Interupt

Software
Pre programmed

Hardware
Due to signals. (we don't know when it will be)

Interupt

Maskable
We can stop/ignore them through programs. (priority)

Non maskable
Can't do anything with
Trap → they can change your normal pro

5 Mkrupt pins.

(1) Trap non maskable → hardwired
(Reset is not an Mkrupt because for comeback from it)
Highest priority

Priority RST 7.5 > RST 6.5 > RST 5.5 (Maskable)

INTR and Interrupt acknowledge.
Lowest priority of all interrupts.
8 different interrupts.

If it is not masked and you execute it → it activates the interrupt acknowledge signal which goes to the priority encoder (PIC) enable signal. The encoder generates the code ---

Interrupt controller → to control the interrupts → what to do, where to go.

Serial controller To communicate externally, it converts serial to parallel and parallel to serial.

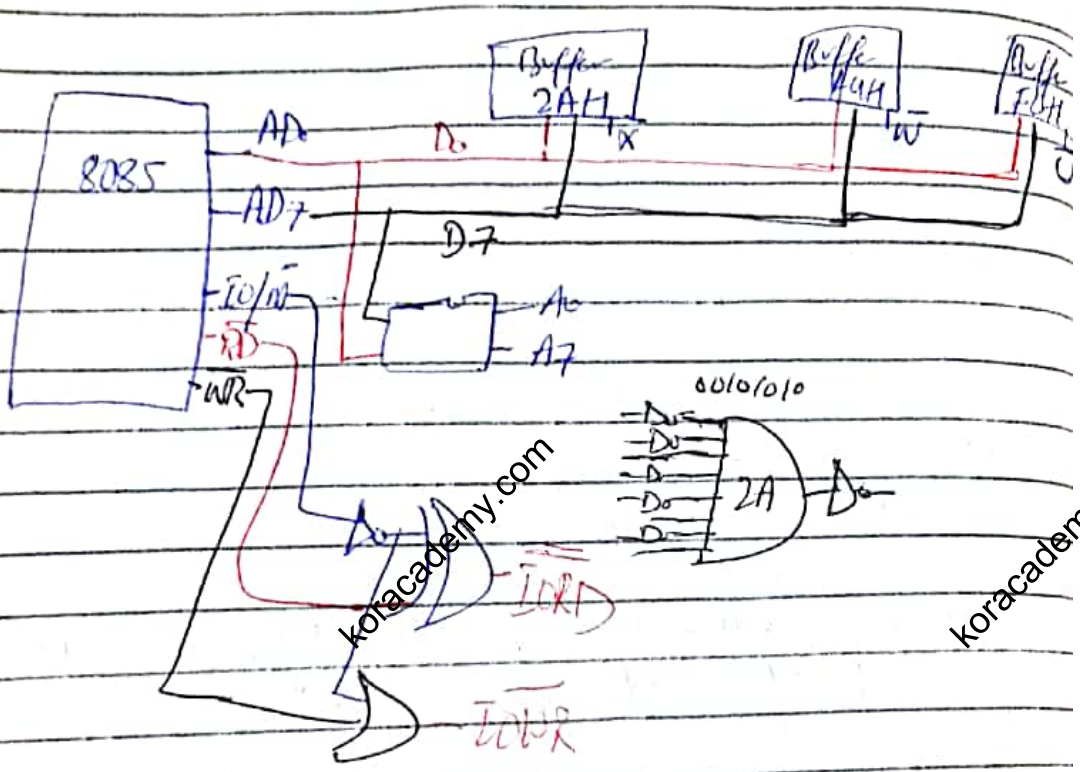
INPUT OUTPUT MAPPING (I/O)

Mapping → Assigning addresses to locations.

Connecting two different types of entities → Interface
Ports can not be of more than 8 bits.

Port → transition b/w two different entities.
Tristate switches with both input and output ports.
74373 → output port
74245 → input port → 20 pin IC → 8 input 8 output 1 Vcc 1 ground 1 DIR (direction) 1 enable → used to back buffers.

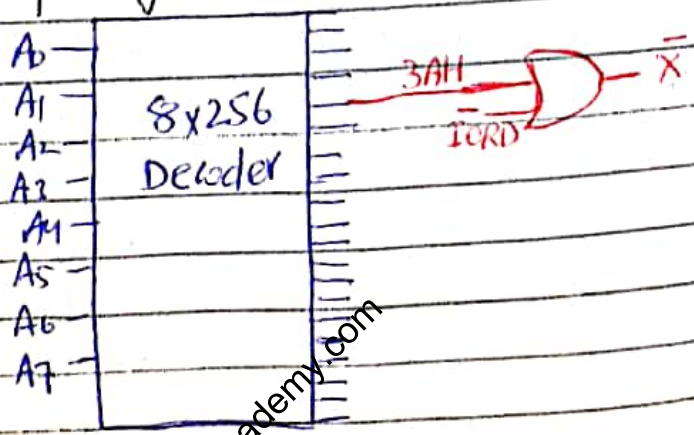
eg Q. Interface 3 i/p ports at addresses
2AH, 4AH, FCH
3 o/p ~~ports~~ ports at addresses
4AH, AAH, FCH

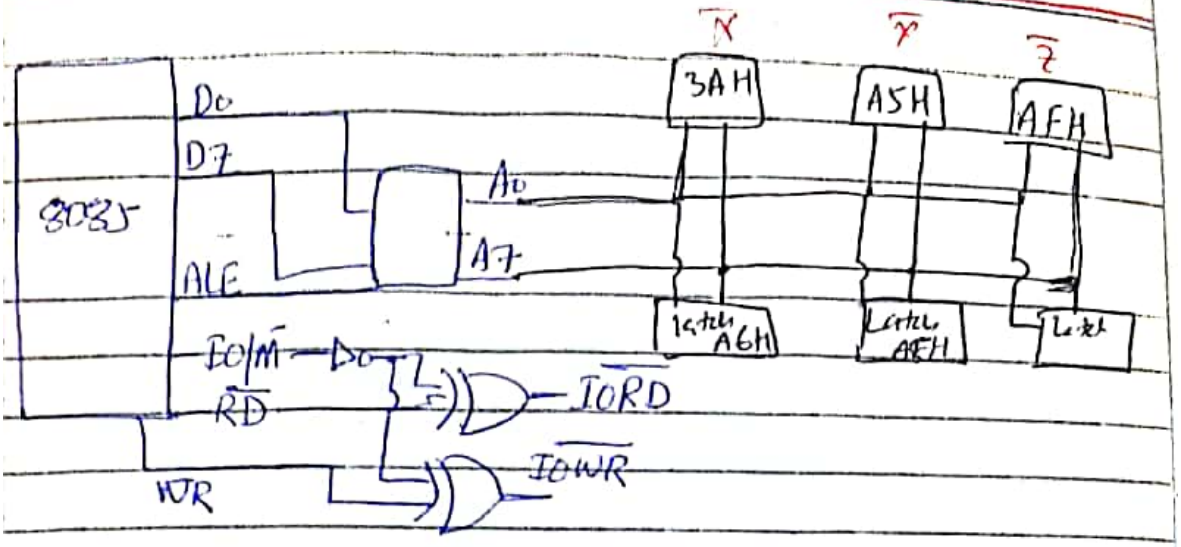


↑ when combine with IOWR
output → latch

Lecture 4

Interfacing Ports Using Decoder





We dont have a 8x256 decoder Maximum 4x16

4x16 decoders to have one 8x256

(Assignment) Proteus 74154 IC

Memory Interfacing

Memory cannot be interfaced by using gates because we have a range of address to memory not a specific address.

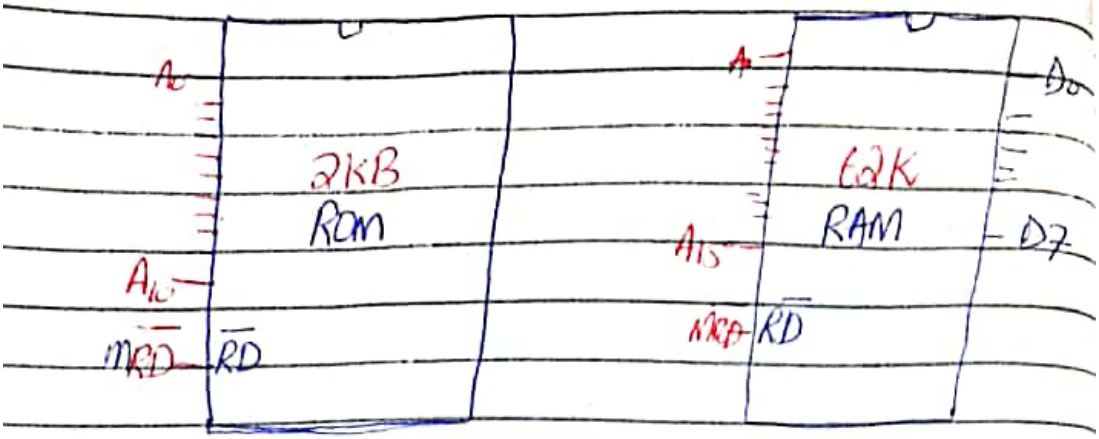
First locations → ROM because as the Microprocessor starts it has to read instructions and the instructions are stored in the ROM

16 address lines with 4 processor

2^{16} → 64KB

No. of address lines 2^{10} → 1KB

$2 \cdot 2^{10}$ → 2KB = $2^{(11)}$



In the 62K memory, we will give 16 lines (6
but it will be used only for 62K

LDA ⇒ Processor is reading from memory
$\overline{MRD}$ = 0

multiple locations having same address ...

HCF 2k 62k = 2k

$2k \rightarrow 2^{11}$

Size of decoder = 16 − 11

eg here 16 − 11 = 5

5 × 32 decoder

decoder divides the memory into smaller chunk

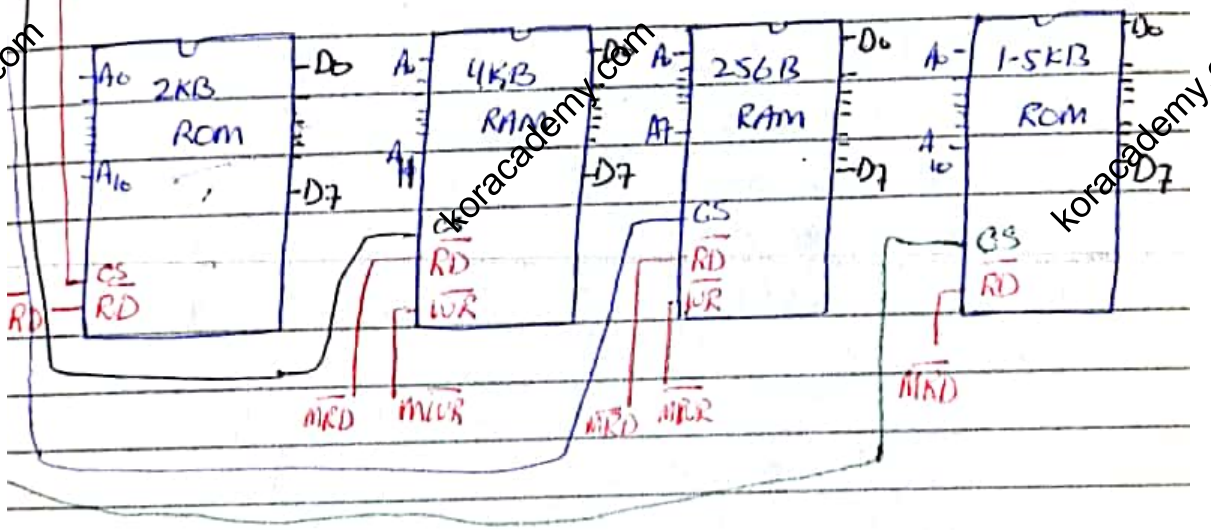
000 1st → 2^0
001 2nd → 2^1
010 3rd → 2^2
011
⋮
⋮

0 → 7FF ⇒ 2KB memory

0000	0000	0000	0000
0000	0000	0000	0001
↓			
0000	0111	1111	1111
0000	1000	0000	0000

Example paper Question

2KB , 4KB , 256B , 1.5KB
ROM RAM RAM ROM



HCF of 2, 4, 0.25 and 1.5 = ~~0.5~~ 0.25
0.25 KB = 2^8
Size of decoder = 16 - 8 = 8 x 256

Instruction Set

Copy operations

MOV R_d R_s (Register addressing)
destination source
1 byte instruction

B → 20H MOV A, B
A = B = 20H

Immediate Addressing

2 Byte instruction MVI R, data 8 bit

MVI A, 30H

A = 30H

LXI Rp 16 Bit data

LXI B, 2022H

B = 20H C = 22H

Direct Addressing

IN 8 Bit Address OUT 8 bit address

eg IN 20H read from 20H location Send data from accumulator to address of port eg OUT 30H

LDA 16 bit address

eg LDA 2200H → load the accumulator from address 2200H.

STA 16 bit address store in the accumulator from the address.

Indirect Addressing

STAX Rp] → BC
LDAX Rp] → DE
HL X

STAX B

B	C
22	44

LDAX D

D	E
33	54

Go to D, fetch E and load accumulator with

H L (only)

8 bit Indexing

M → states that address is in H, L.

eg MOV B, M

Lecture 5

Arithematic And Logic operations

ADD R/M
A = A + R/M

ADD B
A = A + B

ADD M
A = A + M

SUB R/M
A = A - R/M

SUB B
A = A - B

SUB M
A = A - M.

ADI 8 bit data

ADI 20H
A = A + 20H

SUI 8 bit data
A = A - 8 bit
A = A - 10H.

INR R/M
DCR R/M

INR B
B = B + 1

INR M
M = M + 1

DCR B
B = B - 1

INX Rp → BC, DE, HL
DCX Rp

INX B

B	C
20	00
20	01

DCX B

1000

INR B

B	C
21	00

DCR B

2000.

INR and DCR will effect the flag which means there will be arithematic logic operations in ALU.

Flags

We have five flags.

Zero flag After any arithematic and logic ope the zero flag will be 1. If result is 0.

Sign flag When we are dealing with signed numbers arithematic and your result will otherwise it is zero.

Parady flags- (parity → no. of ones).
When parady is even it will be zero and if it is odd the flag will be one.

Assignments
1. Combining decoders to make an 8x256 decoder.
2. Square root function in any simulation software.

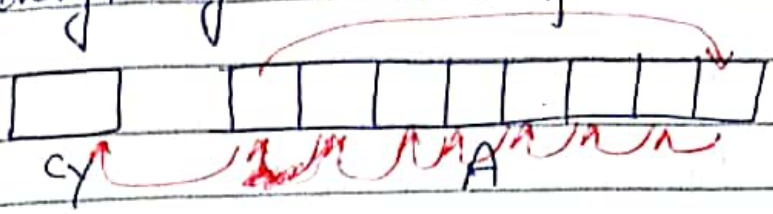
Logical operations-

ANA R/M
ORA R/M → A = A || R/M
XRA R/M → A = A ⊕ R/M
CMA

Rotate Instructions (Also logical operations)

RLC

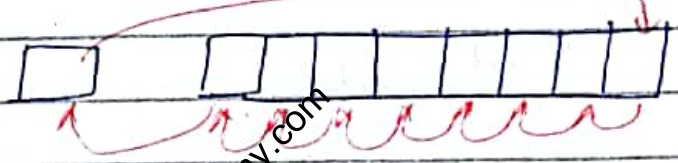
Shifts everything to the left.



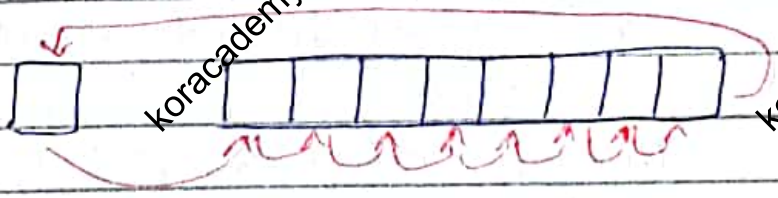
RRC

Shifts everything to the right.

RAL



RAR



let a number 1 10011011

RLC 1 00110111

RRC 1 11001101

RAL 1 00110111

RAR 1 11001101

Compare operations (logical)

CMP R/M

CPI 8 bit data

if (A == R/M)
  Z=1, S=0
else if (A < R/M)
  Z=0, S=1
else
  Z=0, S=0.

CPI 20H

if (A == 20H)
  Z=1, S=0.
else if (A < 20H)
  S=1, Z=0
else
  S=0, Z=0.

Branching Instructions

Jump 16 bit address
Unconditional jump. — you cannot come back SA
you can jump to another address eg jump 2000

Similarly conditional jump.

JZ → jump on zero — JNZ → jump on no zero
JP → sign flag is zero — JM → jump if sign flag is
JC → jump on carry — JNC → jump on no carry.
JPO → jump of parity odd — JPE → parity even.

Call 16 bit address.
Ret
Similarly conditional call and returns.

CC, CNC, CZ, CNZ, CPO, CPE, CM, CP.
RC, RNC, RZ, RNZ, RPO, RPE, RM, RP.

4 Special Instructions.

① PUSH

eg PUSH B → the value of B and C gets stored in stack center and stack pointer increments by 1. 4

(2)

PUSH B SP 4400

B C
22 46

4400H	22
4401H	46
	45
	67

Ex- if we have another register D.

Push D

D	E
25	60

PSW → Accumulator + Flag register.

POP

Retrieves the values; first the low then the upper and discrements the stack pointer.
(LIFO)

```
MVI C, 00H
PUSH B
POP PSW .
```

NOP An empty instruction → means not to do anything.
4 T states useless.

HLT

Halt disconnects the ~~connect~~ clock pulse from the system.
3 to fetch and 1 to execute → total 4 T states.

Stops the program.

Block Data Transfer

Q. Transfer 200 bytes of data from a location string from 2200H to a location string from 3300H.

(source pointer, destination pointer, counting pointer)

```
LDA  2200H
STA  3300H
LDA  2201
                    (200 → in hexadecimal
                     11001000 → C8)
Start:
     LXI   H, 2200H    (source pointer)
     LXI   D, 3300H    (Destination pointer)
     MVI   C, ϕC8H     (counter)
Rpt  MOV   A, M        (transfer from memory to accumulator)
     STAX  D           (store M destination (D,E))
     INX   H           (Increment)
     INX   D           (Increment)
     DCR   C           (Decrement counter)
     JNZ   Rpt
     HLT
```

Q. What if we have to transfer the same data in reverse order (first of the source to last of destination).

```
LXI   H, 2200H
LXI   D, 33C7H
MVI   C, ϕC8H
```

```
Rpt   MOV A,M
      STAX D
      INX H
      DCX D
      DCR C
      JNZ Rpt
      HLT.
```

Lecture 6

Timing And Delays:

F = 1 MHz

1 T state = 1 μsec

let delay = 10 msec.

MVI C,NH → 1 time → 7t

Rpt; DCR C → N time → 4T

JNZ Rpt → N-1 time, 1 → 7T/1

$D = (7T \times 1) + (4T \times N) + (10T \times N-1) + 7T$

$10^{-2} = 4T + 14TN$ $(D = 10^{-2})$

$10^{-2} = 10^{-6}(4 + 14N)$ $(T = 10^{-6})$

$N = \frac{9996}{14} = 714 > 255$

So we cannot assign this to a single register and hence it cannot execute this much delay.

let use this program to have a delay of 1 ms.

71 (decimal) → 47 (H)

F = 1MHz 1T = 1μs Delay = 1ms

```
MVI C, 47H
Rpt: DCR C
     JNZ Rpt
```

What if we want to have a delay of 10ms?
Repeat the program 10 times.

```
     MVI B, 0AH
Rpt2: MVI C, 47H
Rpt: DCR C
     JNZ Rpt
     DCR B
     Rpt2
```

If we want a delay of 1sec?
Repeat the program for 100 times

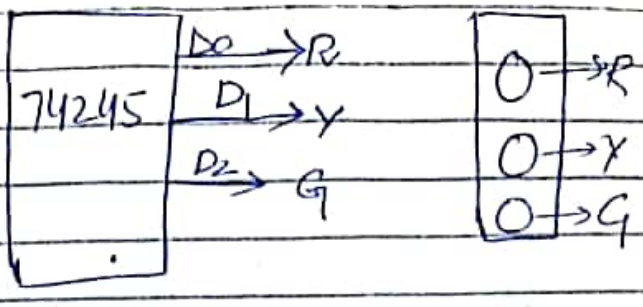
```
     MVI D, 64H
Next: MVI B, 0AH
Rpt2 MVI C, 47H
Rpt: DCR C
     JNZ Rpt
     DCR B
     JNZ Rpt2
     DCR D
     JNZ Next
```

2 X 10ms

$2 \times 10 \times 10^{-3}$ s

Program for traffic signal

Red 10 sec
yellow 5 sec
Green 15 sec.




```
Start : MVI  A, 01H   } Red ON
        OUT  20H      }
        Call delay    }
        call delay    } 10 sec.
MVI  A, 02H           }
  OUT  20, H          } Y ON
  call delay
MVI  A, 04H           }
  OUT  20H            } G ON
  call delay          }
  call delay          } 15 sec.
  call delay          }
  jump  start
  HLT.
```

delay program for 5 sec

```
delay : PUSH  PSW
        PUSH  D
        PUSH  B
        MVI  D, 64H
Next    MVI  B, 32H
Rpt2 :  MVI  C, 47H
Rpt     DCR  C
        JNZ  Rpt
DCR  B
```

JNZ Rpt

DCR D

JNZ Next

(5,6,7,8) dmp exercises.

Process of automation;

(1) IN

(2) OUT

(3) DELAY

What if we use 1 register pair for generating delay?

Advantage: bigger number.

LXI B, NN → 10T × Repeated 1 time

Rpt: DCX B → 10T → N OR B and C Sw it cannot be don directly.

MOV A, B → 4T → N

ORA C → 4T → N For ORing the val to either of them has to be N at

JNZ Rpt → 10T/7T → N-1

$D = 10T + 6TN + 4TN + 10T(N-1) + 7T$

let $D = 100ms$ $T = 1\mu s$

$10^{-1} = 10^{-6}(7 + 24N)$

$24N = 99993$

$\Rightarrow N = 4166.375 = 1046$ (H)

Chapter 8

Q7. Clear initial flags. → load FFH in Acumulator → INR → Mark all flags except Carry → Display Carry at Port 0 →

```
        MVI  C,00H
        PUSH B
        POP  PSW
PUSH PSW    MVI  A,0FFH
POP B       INR  A
MOV A,C     ANI  80H
            OUT  Port-0
            HLT
```

→ ADI 01H → for the repetition part.

@ PUSH PSW

Assuming Carry is number 1.

1 0000000 → 80H

Q8. 20ms subroutine using BC. CLR Z Flag without effecting —

Assume Zero flag is at D4

x	x	1	Z	x	1	1	1
1	1	1	0	1	1	1	1
E				F			

```
MVI C,0EFH
PUSH B
POP PSW.
```

X Because this effects the others

The prgrm will be as;


```
PUSH  PSW
POP   B
MOV   A,C
ANI   0EFH
MOV   C,A
      PUSH B
POP   PSW
      HLT
```

Lecture 7

XOR any register with itself → zero

B → count A → addition.

Multiplication program:

```
      XRA  A
Count: ADD  C
       DCR  B
       JNZ  Count
```

eg multiplying 3 and 4

C = 3 B = 4

A = 0

A = 0 + 3 + 3 + 3 + 3

B = 4, 3, 2, 1, 0

But what if we have to multiply 70 and 100 because 500 > 255

We use two registers.

50		
50		C
50	100 150	0
50	200	0
50	250	0
50	44	1
50	94	0
50	144	0
50	194	0
50	244	0

D = 1 A = FF

IFF → 500.

102C → 44.
(H)

```
MVI  D, 00H      higher byte of product
XRA  A           lower byte of product.
Int  ADD C
     DCR B
     JNZ Cont
```

200 x 5 ? 1000

D	A			Cy
0		200		0
1	144	200	400	1
2	88→88	200	344	1
3	32→32	200	288	1
3			232	0

```
MVI  D, 00H
XRA  A,
Cont  ADD C
      JNZ Next
      INR D
Nxt   DCR B    JNZ Cont    Ret.
```

Multiplication program

```
MUL
        PUSH  PSW
        PUSH  D

        MVI   D, 00H
        XR    A  A
Cont    ADD   C
        JNC   NEXT
        INR   D
NXT     DCR   B
        JNZ   Cont
        MOV   C, A
        MOV   B, D
        POP   D
        POP   PSW
        Ret
```

6000	44
6001	01
6002	55
6003	01
6004	22
6005	01
	33
	01

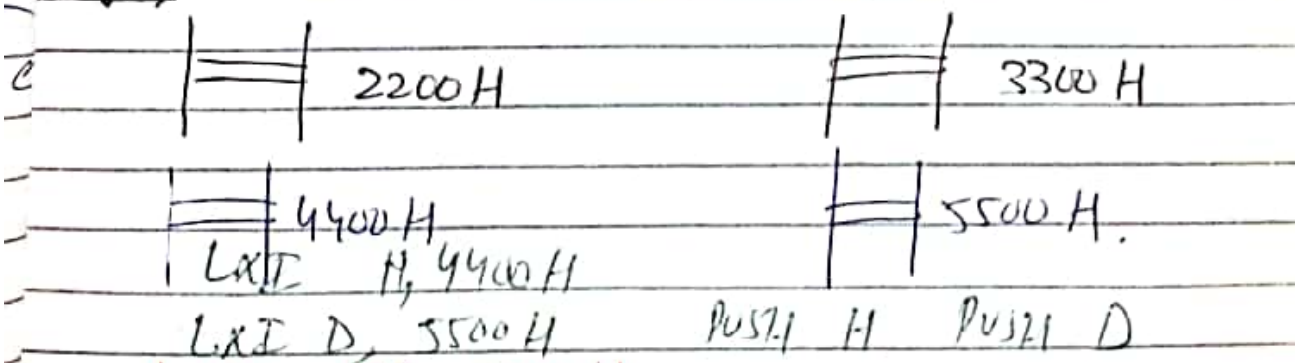
INC

1. Take input from 20H and 30H, give the product as output at 40H and 50H.

```
Start   IN  20H
        MOV B, A
        IN  30H
        MOV C, A
        CALL MUL
        MOV A, C
        OUT 40H
        MOV A, B
        OUT 50H
        JMP start
        HLT
```

(If you want the program to run continuously)

Q(X).




```
        LXI H, 4400H
        LXI D, 5500H
        PUSH H
        PUSH D
        LXI SP, 6000H
        LXI H, 2200H
        LXI D, 3300H
Rpt     MVI C, 0AH
        MOV C, M
        PUSH B
        LDAX D
        MOV B, A
        CALL MUL
```

```
MOV A, B
STA 4400 H
MOV A, C
STA 5500 H
```

```
INX D
INX H
PUSH D
PUSH H
POP H
POP D
POP B
POP H
MOV A, B
STAX D
MOV A, C
STAX H
INX D
INX H
PUSH H
PUSH D
INX SP
" "
" "
" "
POP H
POP D
POP B
DCR C
JNZ Rpt
```

What if registers are falling short?

you pop and manually reach the destination. or by the decrementing the stack pointer.

Multiple times popping does not effect values in the stack memory.

(Past paper).

2. Write a program to count up and down continously a number from 22H to 44H and display it at port 02H after every 2 seconds. Microprocessor speed, f = 2MHz

```
        MVI A, 22H
Next    OUT 02H
        Call Delay (28H)
        INR A
        CPI 45H
        JNZ Next
Rpt     DCR A
        OUT 02H
        call delay (28H)
        CPI 22H
        JNZ Rpt
        Jump Nxt.
```

Q. 15 bytes data read from 2200H. Swap the nibbles (72H to 27H) Store updated result at 3300H.

(To swap numbers, rotate it 4 times)

eg. 0010 0111
2 7

0
0
0
0
0111, 0010,
7 2

Program

```
        LXI   H, 2200H
        LXI   D, 3300H
        MVI   C, 04H
Rpt :   MOV   A, M
        RRC
        RRC
        RRC
        RRC
        STAX  D
        INX   H
        INX   D
        DCR   C
        JNZ   Rpt
        HLT
```

Q. 5 readings stored at XX50H (furnace A) → 5 readings stored at XX60H (furnace B). A is expected to be higher than B.

Write a program to check if A > B → port → D_0 → 1

else if A < B Stop → FF → port 1

else A == B → D_7 → 1 → port 1

continue.

```
furnace A   LXI   D, XX50H   (Source)
furnace B   LXI   H, XX60H   (Destination)
            MVI   C, 05H     (Count)
Next :      MOV   B, M
            LDAX  D
            CMP   B
```

```
JM END                          END MVI A, 0FFH
JZ EQ                               OUT Port 1
EQ: MVI A, 80H                      HLT
SKIP: OUT Port 1
      MVI A, 01H
      JUMP SKIP
      INX H
      INX D
      DCR C
      JNZ Next
      HLT.
```

Q. We have 10 numbers and hae to find the max.
↳ Stored at 55 H.
↳ Display at 20H.

```
      LXI H, 8055H    (Sarce)
      MVI C, 0AH
      MVI A, 00H
Rpt   CMP M
      JP Next         (jump if prime.
      MOV A, M
Next: INX H
      DCR C
      JNZ Rpt
      HLT.
```

FINAL

Lecture 1

1981 → introduced µcontroller → by Intel →

Fundamental component → microprocessor.
µcontroller cannot work without µprocessor

8051

40 pin IC. with internal 4KB ROM, 128 B RAM, two timers T_0 and T_1.

Assembler converts assembly language to machine.
compiler converts one form of language to another

Directives are for compilers
Instructions are for µcontrollers → ORG
→ END

eg HLT is an instruction, END is a directive

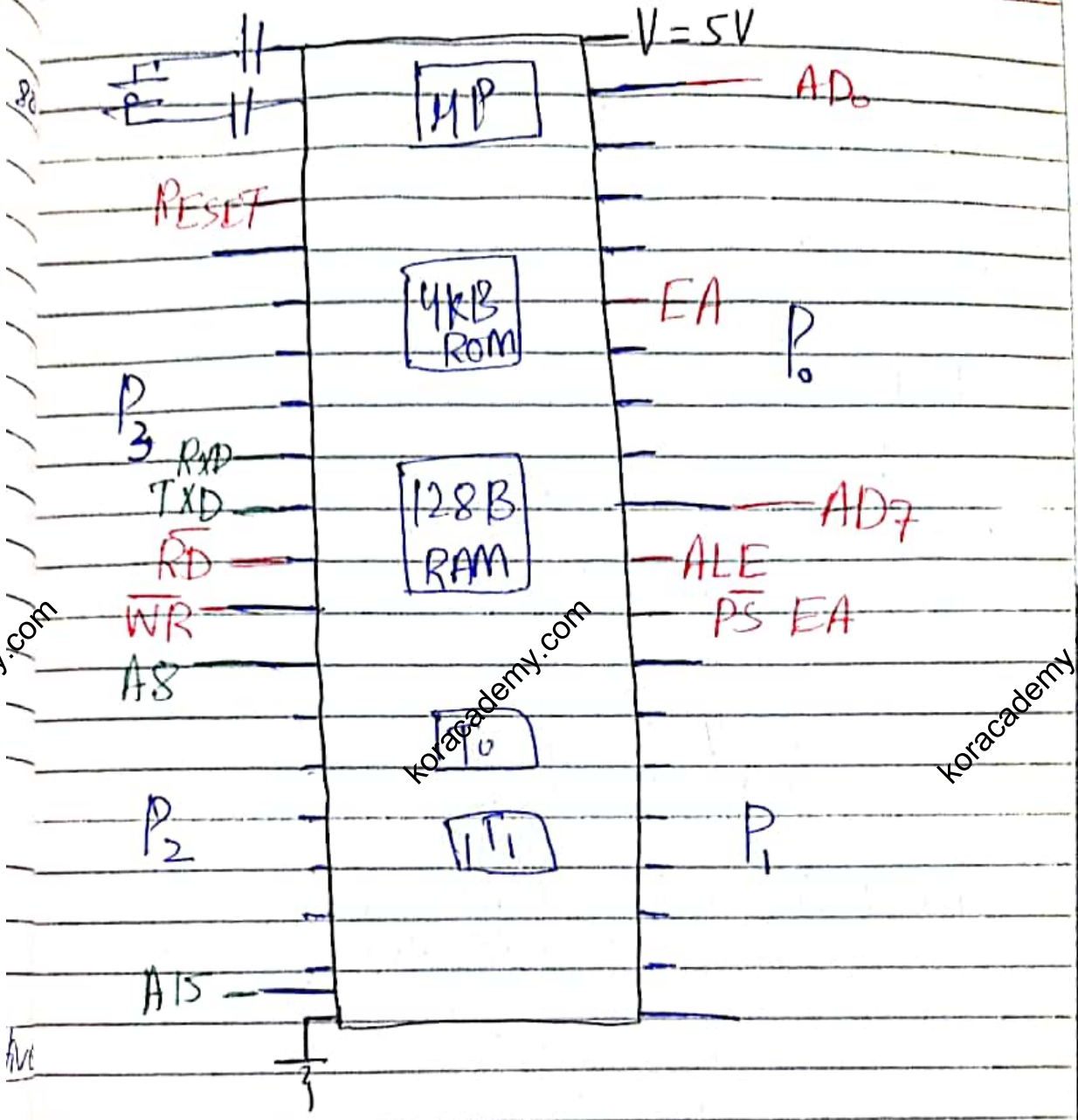
8051 also available in 20 pin and 64 pin chips.

89C51 → by Atmel

Difference b/w 8051 and 89C51?

8051 is one time programmable → cannot reprogram it

89C51 has flash memory → reprogrammable



μ controller.

It has an 8 bit μ processor inside it.

⇒ 8 bit ALU. → (not 8085)

↳ this processor is also capable of multiplication and division.

μ controller is a specific purpose device whereas μ processor is general purpose.

⇒ based on the application.

High level language eg C are portable
↳they are platform independent.

~~Keil transforms the assembly language~~

It can access each and every location

4 I/O Ports P_0, P_1, P_2, P_3

Can be configured either as input or output ports.

Output → send 0 to it.

Input → send FF to it

```
MOV P1, #0FFH
MOV P2, #00H
```

(No MVI instruction).

If we want to read data from it;

```
Rpt: MOV A, P1
     MOV P2, A
     SJMP Rpt
```

If we want to continue reading this data

ORG → tells the compiler where in the chip to store the code

ROM → program memory
RAM → data memory

Every programmable device has fundamentally two modes.
(i) Program mode
(ii) Run mode

yellow line → 12 V

How to transfer the code to µcontroller?

We need address and data lines.
Write, read, ALE.
P_0 works as AD_0 to AD_7
P_2 works as A_8 to A_{15}
Two pins from P_3 work as $\overline{RD}$ and $\overline{WR}$.

If we have a code size of more than 4KB, then we need external memory.

EA (external access)
tells the µcontroller to whether the program is inside the controller or in the external memory.
external memory → ground.
internal memory → high voltage.

Disadvantage?
Only P_1 is left for usage.
The internal 4KB ROM can also not be used.

In Assembly language, we know size of each instruction and can add it up to know the total memory of code.

Whereas, in C we do not know, what is the size of the code.

Each different µcontroller has its own assembly language.

We can configure individual pins as input or output.

→ 32 pins.

input → 1 output 0.

pin → can't → another pin

we can't transfer data directly b/w pins.

Set b P1.0 first pin of port 1 set as input.

CLR P1.1 second pin of port 1 set as output.

1 bit of data can only be accessed at each pin.

Externally we can connect 64kB of program memory → MOVC do access the external program.

We can also connect 64kB of data memory.

Disadvantage?

We lose three ports.

Data word size is on the basis of ALU.

Maximum limit of crystal oscillator → 48 MHz

Capacitors — 10 pico farads

P0 is open drain.

Open drain } pull up resistor if you want
Open collector } to collect loads etc.
BJT

No Ready pin in µ controller.

TTL based communication → 0V, 5V

Lecture 2

Some basic registers; PC (16 bits)

A B

DPTR has addresses of HL used to have.

SP → 8 bits → access internal RAM.

DPTR
DPH | DPL

MOV X
MOV C

CLR A

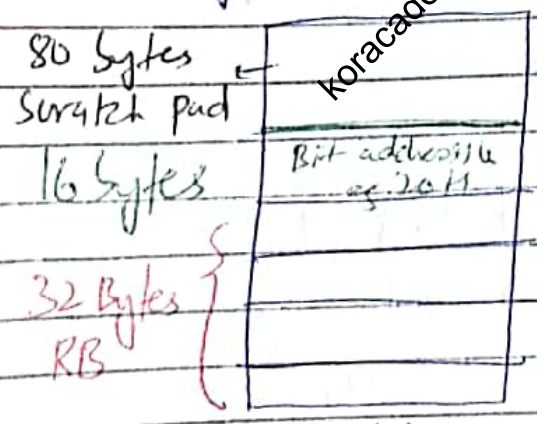
eg MOV A, 20H

There is a defined range can be label called RB (register banks)

4 register banks,

RB_0 00H – 07H
RB_1 08H – 0FH
RB_2 10 – 17H
RB_3 18H – 1FH

128 bit location to store each bit.
128 byte location to store each byte.



128 bytes of RAM

What is register bank?
32 bytes of memory. → (virtual registers)
We can

By default first 8 locations are used as R_0 to R_7.

eg MOV A, 00H
or MOV A, R0
→ the same thing.

We can change

We have two flags RS1 and RS0 which shows which registers are being used.

	RS1	RS0
RB_0 ←	0	0
RB_1 ←	0	1
RB_2 ←	1	0
RB_3 ←	1	1

CLR RS0
Setb RS1
MOV A, R0H
or MOV A, 00H MOV A, 10H

- PSW in microcontroller refers to the flag register.
In microprocessor it is combination of flag register and accumulator.

CY	AC	RS1	RS0	F_0	P	OV

No zero flag.

Overflow → when out of range. -128 to 127.
or eg when divided by zero.

Flag register is 8 bit.

Timers

There are two 16 bit timers. T0 and T1

TMOD register →

G	T/C	M_1	M_0	G	T/C	M_1	M_0
T1				T0			

M_1 and M_0 controls the mode of operation

M_1	M_0	
0	0	→ mode 0 - 13 bit timer/counter
0	1	→ mode 1 → act as 16 bit timer/counter
1	0	→ mode 2 → act as 8 bit auto reload
1	1	→ mode 3 → 8 bit split mode → 2 reg

can be used independently in 8 bits.

T0 [TH0 | TL0] T1 [TH1 | TL1]

T/C If 0 ⇒ timer If 1 ⇒ Counter

When it is configured as a timer, it finds how to generate delay.

$\frac{CLK}{12}$ eg $\frac{12MHz}{12} = 1MHz$

which means next count is after 1 µs.

Counter?

It responds to interruption of T0 or T1

It counts the number of interruptions.

Lcall → long call → range 64KB → 3 Bytes
Acall → range 0 to 2K
5 bit opcode, 11 bits address.

Gate (G)
If 0 → software control.
If 1 → hardware control.

TR0 starts timer T0. } when software
TR1 starts timer T1 } controlled.

INT0 and INT1.

INT0 → If you provide signal to it, it will start the timer.

Program
let using T0

TMOD : 0000 0101
G C/T M1 M0 G C/T M1 M0
0 SH

ORG 00H → MOV P1, #00H
MOV TMOD, #05
MOV TL0, #00H
MOV TH0, #00H
CLR TR0
CLR TF0
Setb TR0
RPT: MOV P1, TL0
SJMP RPT
END

2 byte instruction
SJMP → short jump
-128 backwards ~ 127 forwards

3 byte ← LJMP → long jump
can jump anywhere in the 64 bytes

How to use it as a timer for generating delay?
Use mode 1. 0000 0001
TMOD 01 H

Max Init 65535.
We can generate 0.5s delay (not 1s).

50ms sec delay so cnt = 50000

We have a cnt up here and come to know when we reach the maximum

Maximum — how much to cnt

here 65535 − 50000 = 15535

+1

3CB0 ← Max ← 15536

```
        ORG 00H
Delay   MOV  TMOD, #01H
        MOV  TL0, #0B0H
        MOV  TH0, #3CH
        CLR  TR0
        CLR  TF0
        SETB TR0
Here:   JNB  TF0, Here
        CLR  TR0
        CLR  TF0
        END  (Return)
```

Usually this delay is in the middle of a function → so ↑

The program with delay 50 msec.

Lecture 3

Serial Communication:

Motherboard → Internally → mostly parallel

Microcontroller → two lines
one for receiving, one for transmission.
Configured first which to be used for which purpose. Speed is entered before

TTL → 0V → 0
→ 5V → 1

RS 232
-3 → -5 → 1
+3 → 25 → 0

Max 232 (IC) converts RS 232 to TTL and TTL to RS232. (Inter converts)
(external capacitor to be installed).

Max 233 → same as 232 → capacitors already installed internally

A register called SBUF. → 8 bit → any one
Anything for outside first comes to SBUF.
Anything to go outside first is copied to SBUF

It can either receive or transmit at a single time.

3 types of serial communication
- Simplex
- Half duplex
- Full duplex

Simplex
If system is capable of only transmitting or receiving (TX are simply there)

Half duplex
Transmit + Receive ⇒ but not both at the same time

Full duplex.
Transmit + Receive → and capable doing both at the same time

Microcontroller is a half duplex.

There is a bit called REN in Microcontroller
Set → configure device for receiving
Clear → configure for transmitting

Speed?
Timer 1 is controlling speed of serial communication

TMOD ←

20 → 0010 0000

Two bits TI and RI in Scon (flags).
(Interrupts).

Serial → TMOD → DO.

SCON → register → always put SOH →
→ to transmit & receive 8 bit data

The more negative → the faster it is.

Program for transmission of data

```
ORG 00H
MOV TMOD, #20H
MOV SCON, #50H
CLR REN
MOV TH1, -3
MOV A, #'A'
MOV SBUF, A
```

Copy A to accumulator
65H is the ASCII
so 65H goes to accumulator.

Transmission starts when we start the timer

```
Setb TR1        time start.
Here: JNB TI, Here   Stay here until it is transmitted.
CLR TI
CLR TR1
END.
```

When TI is set → it shows that the data has been transmitted

```
Transmit: MOV TMOD, #20H
          MOV SCON, #50H
Subroutine CLR REN
          MOV TH1, -3
          MOV SBUF, A
          Setb TR1
```

ORG 200H
"This is a cat"
DB

Here: JNB TI, Here
CLR TI
CLR TRI
Ret

Transmitting a Name?

We have to store it in ROM.
DB is a directive.
Dptr is a register pair → used for storing data → and then use as an address.

DB: ' ' → stores first character at first location and so on.

Only Method of storing data in ROM is DB and ORG specifies the location

eg ORG 40H
DB SAL Stores S at 40H
A at 41H and so on.

eg Store This is a cat at memory Location starting from 200H.

ORG 200H
DB: "This is a cat".

MOV A, @A+DPTR adds accumulator and DPTR

Program

```
        ORG 40H
DB:     "Salman" 0
        ORG 00H
        MOV DPTR, #40H
Next:   CLR A
        MOVC A, @A + DPTR
        ACall Transmit
        INCR DPTR.
        JNZ Next.
```

Program for Recieving

```
Rec:    MOV TMOD, #20H
        MOV SCON, #50H
        Setb REN
        MOV TH1, -3
        Setb TR1
Here:   JNB RI, Here
        MOV A, SBUF
        CLR RI
        CLR TR1
        Ret
```

RI → recieve interupt.

LCD

LCD's → work on ASCII

LCD → usually has a RAM (eg text message upto)

LCD → read
→ write

V_{SS} → ground V_{EE} → variable voltage (to control contrast)

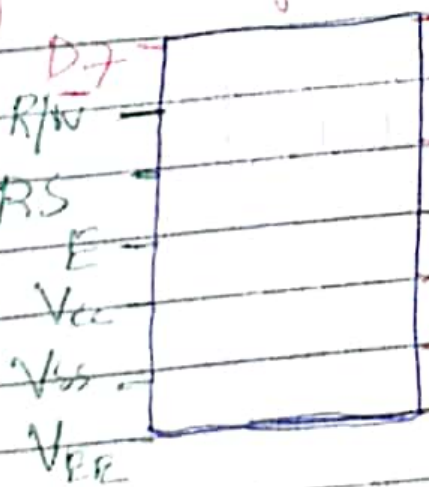
LCD always in the form of matrices (rows and columns)

Command mode → where to display what

RS → command and data select

RS → 0 → command
RS → 1 → data

Enable → always edge triggered → high to low → negative edge triggered



To ready the LCD for use and bring cursor on first location; following 4 commands

38, 0E, 01, 06, 84

Assignment 2

Connect D_0 to D_7 to P_1
R/$\bar{W}$ to P2.0
RS to P2.1
E to P2.2

Subroutine for generating high to low signal

```
Command:  MOV  P1, A
          CLR  P2.0
          CLR  P2.1
          SetB P2.2
          ACall 1ms delay
          CLR  P2.2
          Ret
```

To Send data to LCD (?)

```
Data:  MOV  P1, A
       CLR  P2.0
       setb P2.1
       setb P2.2
       Acall small delay
       CLR  P2.2
       Ret.
```

Program

configuration of LCD

configuration of ports.

```
MOV  P1, #00H
CLR  P2.0
CLR  P2.1
CLR  P2.2
```

```
MOV A, #38H
ACall Command
MOV A, #0EH
ACall Comand
MOV A, #01H
ACall Comand
MOV A, #06H
ACall Comand
```

```
MOV A, 84H        ; to display A.
ACall Comand
MOV A, #'A'
ACall Data
```

To display "This is great".

```
ORG 200H
DB: "This is great" 0
    MOV A, #\84H
    ACall Command
Next: CLR A
    MOV A, @A+DPTR
    JZ Exit
    ACall Data
    INC DPTR
    SJMP Next
Exit SJMP Exit      ; looping at the end
```

Lec

KeyPa

They

Matrix

Everyth

column.

Scanned with CamScanner

Lecture 4.

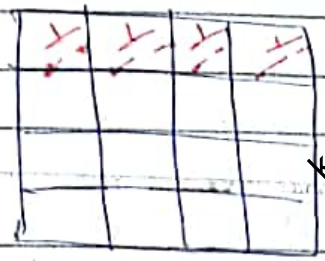
Keypad Interfacing

They are in the form of matrices (rows & columns).

Matrix → because it reduces the hardware needed.

Everytime a key is pressed, a row is shorted with a column.

Before starting, by default, all the keys should be unpressed. open state is considered 1 // connected → grounded.



These vertical 0s and horizontal 1s indicate that all the keys are unpressed.

Checking through controller, connect these to two ports.

```
ORG 00H
MOV P1, #00H        → configured as output
MOV P2, #0FFH       → configured as input
Again. MOV P1, #00H  → MOV DDR, #00H
       MOV A, P2
       ANL A, #0EH   → 00001111
       CJNE A, #0EH, Again.
```

Now the computer needs to know which key is to be pressed. Check for other than 1 in the horizontal 1s.

After the first step, generate a little delay as there are transients. (time delay) and then the second step.

```
A Call s-delay (1ms)
Rpt: MOV P1, #00H
     MOV A, P2
     ANL A, #0FH
     CJNE A, #0FH, Key-p
     SJMP Rpt
Key-p: A Call s-delay
```

The first three parts here to check the any key is pressed and the last one to know which key is pressed.

The third part is to know which key is pressed. So we check row by row.

	3	2	1	0
1110	3	2	1	0
1101	7	6	5	4
1011	11	10	9	8
0111	15	14	13	12

1st row

1110 → 0E
1101 → D
1011 → B
0111 → 07H

```
MOV P1, #0EH
MOV A, P2
ANL A, #0FH
CJNE A, #0FH, R-E
MOV A, R0
ADD A, #04H
MOV R0, A
```

To s... the se... add... ch...

```
MOV P1, #0DH
MOV A, P2
ANL A, #0FH
CJNE A, #0FH, R-E
MOV A, R0
ADD A, #04H
MOV R0, A
```

2nd row

```
MOV P1, #0BH
MOV A, P2
ANL A, #0FH
CJNE A, #0FH, R-E
MOV A, R0
```

3rd row

```
ADD A, #04
MOV A, R0
MOV P1, #07H
MOV A, P2
ANL A, #0FH
CJNE A, #0FH, R_F
SJMP Rpt
```

4th row

R0 is used as a counter check columns.

```
R_F : MOV R1, #04H
Next : RRC
JNC K_F
INR R0
DJNZ R1, Next
SJMP Rpt
K_F : MOV A, R0
MOV A, @A+DPTR
```

Perform right through carry.

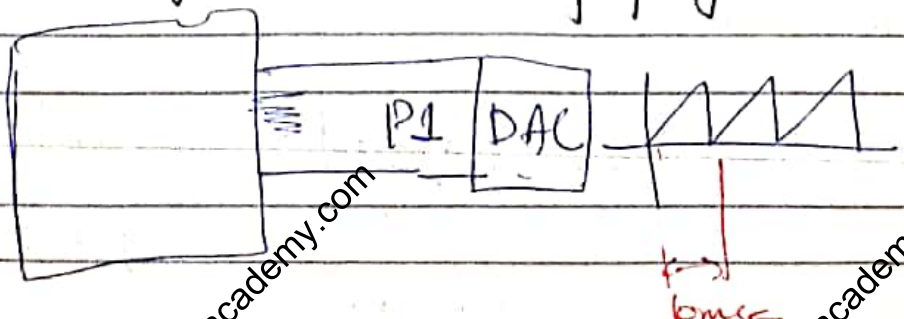
CRG 200H

DB : " " to read the data of the key.

Interfacing DAC

Digital to Analog Converter

Q. To generate a sawtooth waveform using microcontroller through DAC with frequency = 100Hz.



delay?

0 - 255 cnt to be cupleted in 10msec

$$\text{1 cnt} = \frac{10\ \text{msec}}{256} = \frac{\quad}{2} \rightarrow \text{time}$$

D to A converters used to generate different waveforms

```
MOV P1, #00H
MOV A, #00H          (increase)
Rpt: MOV P1, A
     Acall delay
     INC A
     SJMP RPT        CJNE A, #0FFH, Rpt
```

To generate triangular wafom

```
Rpt2: MOV P1, A
      Acall delay
      DJNZ A, Rpt1
      SJMP Rpt2
```

(decrease)

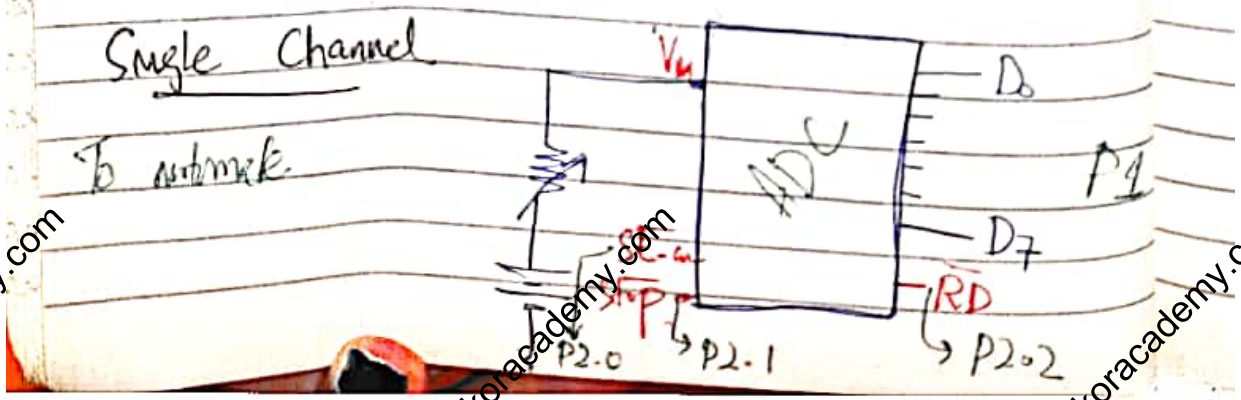
For generating SM wave, put the values in a lookup table first.

Interfacing ADC

Analog to Digital Converters

Single Channel

To interface

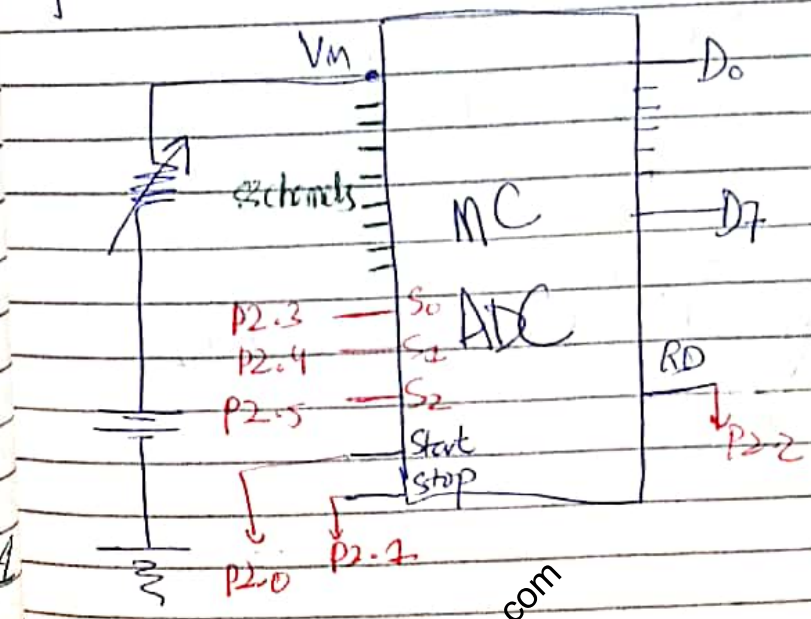


D0 to D7 is used s/c we need 8 bit digital value in the form of 0s and 1s.

```
MOV P3, #00H   → o/p
MOV P1, #0FFH  → i/p
CLR P2.0
Set b P2.1
CLR P2.2
Next ACall ADC
     MOV P3, A
     SJMP Next
ADC : CLR P2.0
Here : JB P2.1, Here
       CLR P2.2
       MOV A, P1
       Ret.
```

Multiple Channel ADC

If 8 channels → 3 select lines.



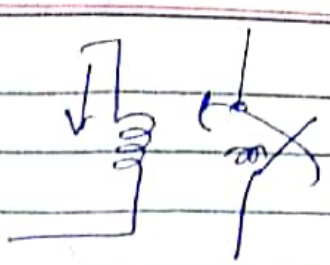
```
CLR  P2.3
CLR  P2.4
CLR  P2.5
MOV  P3, #00H
MOV  P1, #0FFH
CLR  P2.0
Setb P2.1
CLR  P2.2
CLR  P2.3
CLR  P2.4
CLR  P2.5
Next: Acall ADC
MOV  P3, A
SJMP Next
Setb P2.3
CLR  P2.4
CLR  P2.5
Acall ADC
CLR  P2.3
Setb P2.4
CLR  P2.5
Acall ADC
```

Lecture 5

Interfacing AC/DC MOTORS AND OTHER Appliances

electronic controlled.

electromagnetic switch



solid state ⇒ PN junctions

The fundamental thing is replacement of mechanical switches with electronic controlled switches.

How?

We connect them (devices) with a pin.

P2-0 WM → 2h

P2-1 FAN → 3h

P2-2 light → 4h

1h → delay

sets P2-0

Recall delay

"

Clear P2-0

sets P2-1

"

↓

Interfacing Stepper Motors

Machineries → either AC or DC motor

stepper → move in steps.

Two fundamental things

① Command ?

② step ?

It is supposed to make a step or a command.

Conveyer belt → rotational to translational motion. $L = 2\pi r$.

printers, scanners, photocopiers, lifts

If we have a motor for motor having 360° as 1 step.

second 360° ⇒ move 6°

minute 360° ⇒ hour = 6/12 = 0.5°

Serial and parallel

↳ pulses

↳ 8 pins

specified command to register

subroutine for step

Uses for numbers

eg 72, A4, E7, 47.

```
Step: MOV P1, #72H
      ACALL s-delay
      MOV P1, #A4H
      ACALL s-delay
      MOV P1, #E7H
      ACALL s-delay
      MOV P1, #47H
      RET
```

Step will be after the complete sequence

C Language

Advantage → portable. → we can use the code of one controller for another.

disadvantage → not sure where the code is to be stored.

we cannot access individual locations in controller

Size of the code.

main function is executed first.

header file. # include <Reg51.h>

```
void main (void)
{ Bit Temp;
  P1^0 = 1;
  P1^1 = 0;
  while (1) { Temp = P1^0;
              P1^1 = Temp; } }
```

for can be used instead

```
setb P1.0
CLR P1.1
Rpt: mov C, P1.0
mov P1.1 C
SJMP Rpt
```

```
Simdelay;
MOV P1, #00H
cnt: MOV A, #00H
Next: MOV P1, A
A call delay
INR A
CJNE A, #0AH, Next
SJMP start.
delay: MOV R0, #255
Again: MOV R1, #255
```

character → 8 bit datatype
integer → 16 " "

```
Here: DJNZ  R1, Here
      DJNZ  R0, Again.
      Ret.
```

```
# include <Reg51.h>
Void delay (void)
{   int b;
    for (b=0; b< 250000 : bit);
        Return ();  }

void main (void).
{   char a;
        P1 = 0;
while (1) { for (a=0; a<10, a++)
    { P1 = a;
        delay (): } } }
```

If delay is generated using timers. ie.

```
delay : MOV TMOD, #01H
        MOV TH0, #0FH
        MOV TL0, #0CEH
        Setb TR0.

Here : JNB TF0, Here
       CLR TF0.
       CLR TR0.
       Ret:
```

```
void delay (void)
{
    TMOD = 1;          → for hexadecimal in C
    TH0 = 0X0F;
    TL0 = 0XCE;
    TR0 = 1;
    while (TF0);
    TR0 = 0;
    TF0 = 0;
    Return ();
}
```

Interrupts

timers → software interrupt.

Interrupts Vs Polling.

↳ continuously checking.

Interrupt Service Routine:

IE → register for interrupts.