Next: CLR TI.

MOV SBUF, A (not bit addressable. byte addressable. SFR)

(Start Transmit)

Here: JNB TI Here.

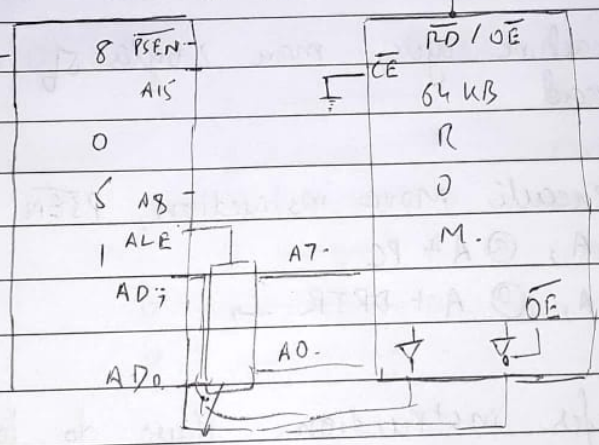
SJMP Next

Transmitting starts automatically as soon as loaded in SBUFF.

* MOV TMOD, #20H.
MOV TH1, #-3
SETB TR1
CLR TI.
MOV SCON, #40H.

SETB PSW.5
MOV SP, #07H.
MOV IE, #81H. (EA, INT0)

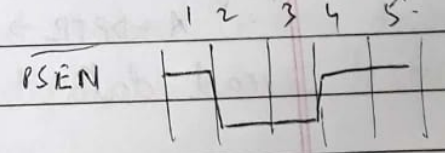
* External Memory Interface : (ROM)



$\overline{CE}$ decides whether chip accessable or not. if only 1, can directly to ground.

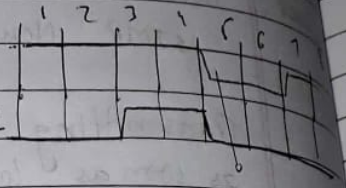
High Impedence, Open ckt, when $\overline{OE} \to 1$, enable output.

$\overline{RD}$ / $\overline{OE}$ → same func. generated itself.



$\overline{PSEN}$ is generated in the start of every instruction. each instruction is read from memory. PSEN goes low in 5th & 6th keeps low for 2 machine cycles.

Then in 5th & 6th, PSEN low, then read. now if instruction is of 24 cycles, after this the whole process starts again.



When PSEN low, whatever loaded in the data output is read.

it auto increment the address,
make ALE high, send address,
make PSEN low, read, execute,
repeat.

1 byte of instruction takes atleast 6 cycles.

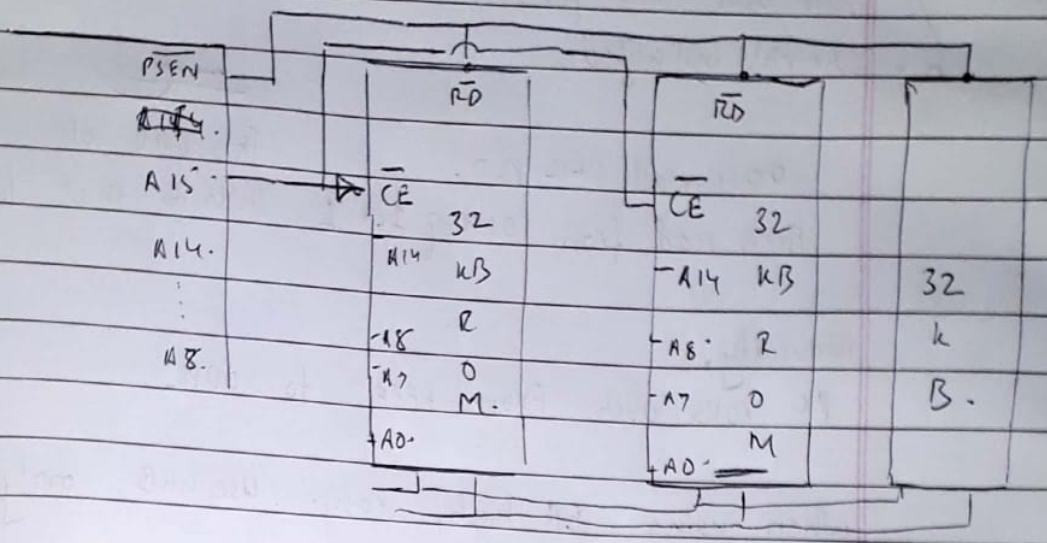
$\overline{PSEN}$ → read from memory. no matter from where. it is also connected to $\overline{RD}$ of internal memory.

in 1 machine cycle, max 2 bytes of instructions can be read

⇒ When I execute MOVC instruction, $\overline{PSEN}$ generated.
MOVC A, @A+PC.
MOVC A, @A+DPTR →
↓
Data transfer instruction. Have to load data from progam ROM (code memory).
A+DPTR → calculate address again PSEN low, read data.

Can connect $\overline{PSEN}$ to $\overline{CE}$. so whenever reading, $\overline{PSEN}$ 1, ∴ $\overline{CE}$ also one. ∴ only enabling chip when required...
Also, @ ALE ~~low~~ high, add

when No need for chip to be enabled when adding the address. Now when $\overline{PSEN}$ low, take the address, & load the data in the data bus



Can connect PSEN to any no. of chips. Whatever be the chip, will anyways use this to read from anywhere.

P3.0	A15	— —	AD.
0	0	0 ——→	The different address ranges for P3.0 & A15 ∴ can use P3.0 to increase the range from 64KB → 128KB
		1 ——→	1.
0	1	0 ——→	0
		1 ——→	1.
1	0	——→	0
	0	1 ——→	1.

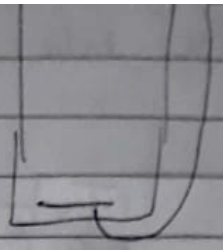
If EA high, & use internal memory to first set P3.0 & clear P3.0. Just before it rolls down. at the end of chip 2, will have to set P3.0.

```
P3.  Org 0H.
     LJMP 100
100H:
     CLR  P3.0.
```

here. set P3.0.

So A0 → A15 initialized with 00H. but ports are not initialized.

6. ?



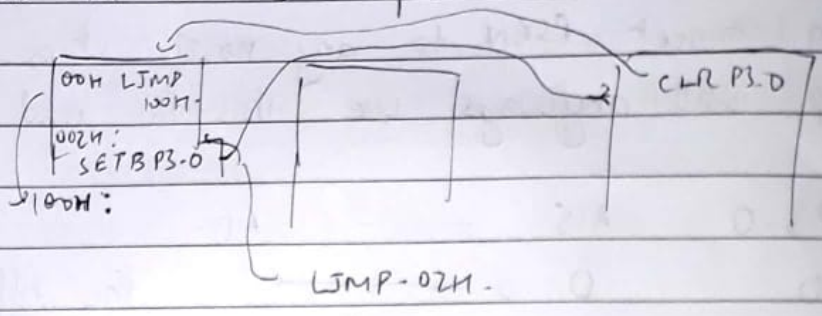
00H read CLR P3.0. This will roll
Then read from 01H of 1st L. over to that location.

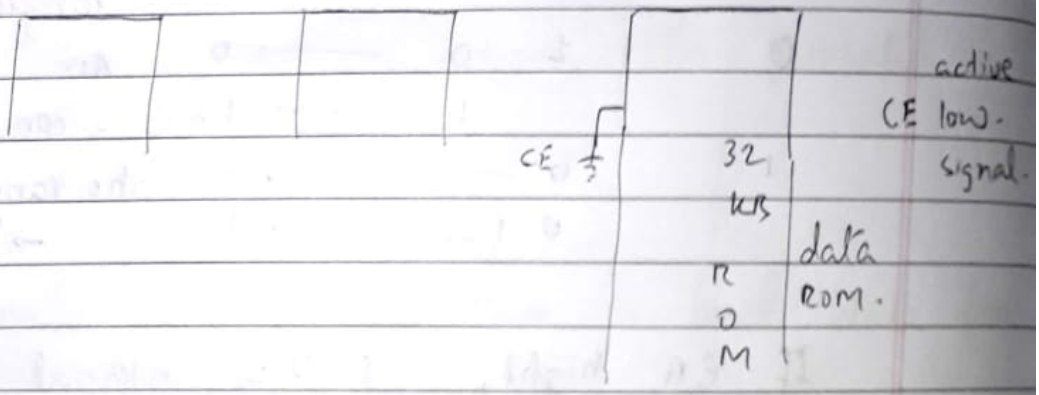
Naturally,
PC rolls over From FFFF to 00H.

when using EA high, easy. Use 4KB. only check for pointers in 2nd 3rd chip.

Check This. ?

Check.





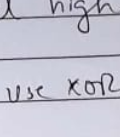
Interfacing Data is easier, as can set address & everything easily. for code, always start with 000H, so complication increase.

Disable REN in SCON. as P3.0 used, can't receive serially. though can still use Transmission.

P3 already have $\overline{INT0}$, $\overline{RD}$. ∴ Can use other port pins also to expand. but won't be able to use it for its implicit function.

Transmission is logic level high.

Check for transition. detect. use XOR. count.



as soon as load in SBUF, add start bit, start transmitting. when finished, last stop bit, make TI → 1.

Illy when receiving, after stop bit, set RI.

Both have logically same space, same address. but diff. physically. Based on Reading or writing, use different sbuf. byte addressable like SFR and 256 bit indirect memory.

Have to keep REN → 1 to be ready to receive.

MOV TMOD #20

It will generate a signal after every overflow, at 9600, this signal taken by UART to receive or transmit.

```
MOV TH1, #-3
MOV SCON
SETB TR1.
CLR TI
MOV SBUF , #('.')
JNB TI, Here.
CLR TI.
SJmp send_next
```

is not set.

Jump if bit ~~not zero~~.