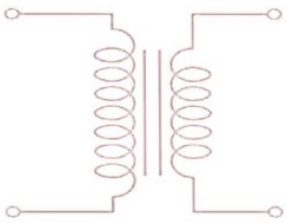
Embedded

Summer017




Dr. Ramzi Seifan

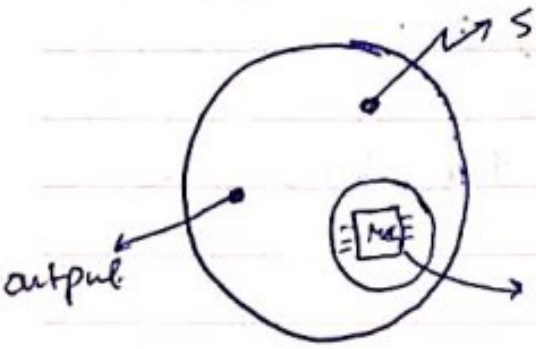
By: Anoud Alhallaq



www Powerunit-ju.com

Embedded

23/7/
SUN



non-compiling device.

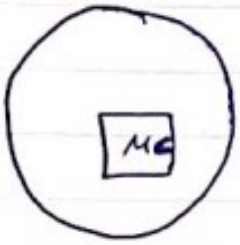
Ch. 1 → introduction.
2 → hardware
download program ~ 4] code.
5 → first. 25%.
6
3 → bouns (programing process (simulater) ← mblab (IDE)
10 → second 25%.
11
8
9 → final. 50%.

lab of embdded. sys. (E.S).

24. 7.
MON

* Embedded system (E.S) :-

non - computing device that has M.C that control it.



- one or few dedicated function
- real time constraints: means there is dead line after which the action is useless

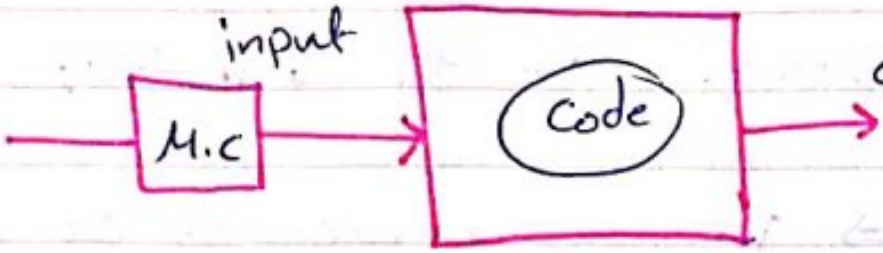
* To modify / understand any E.S :-

1) inputs.
2) outputs.
3) user interaction
4) link to other sys.

* next step :-

1) Hardware (HW) → (M.C, peripherals)

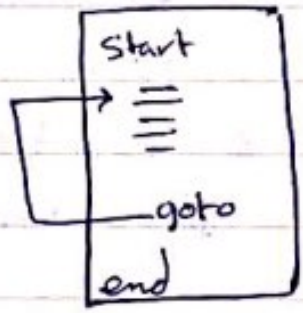
2) Software (SW) → (Program).
   ↳ if you have access to the code backup remanufacture it.

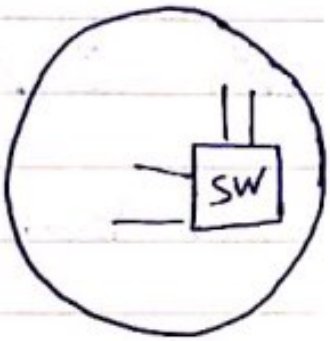


* M.C :-

Software driven.

Program : always running.





* في الأمبيديد مافي نظام لينو
في اشي برجعلك داعماً.

reliable: almost 100% running correct

can I modify it ?!

* slide 8 :-

if $\bar{T}_d < T_a$

"desired" "actual"

switch ON compressor :

else

switch. OFF compressor.

slide 15:-

Main Components of the computer:-

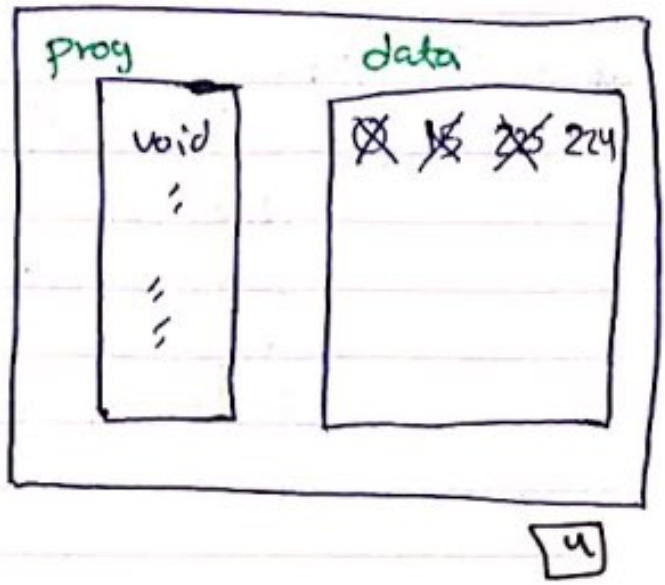
1. CPU ⇒ Processor, excutes code, control.
2. memory & storge.
3. ILOS & to interact with external world.
4. Buses & to transfers data btw the Componets above.

* Memory :-

① prog memory : Prog code.

② data memory : value of variable.

```
void main()
int x;
x=5*3;
x=x^2;
x=x-1;
```

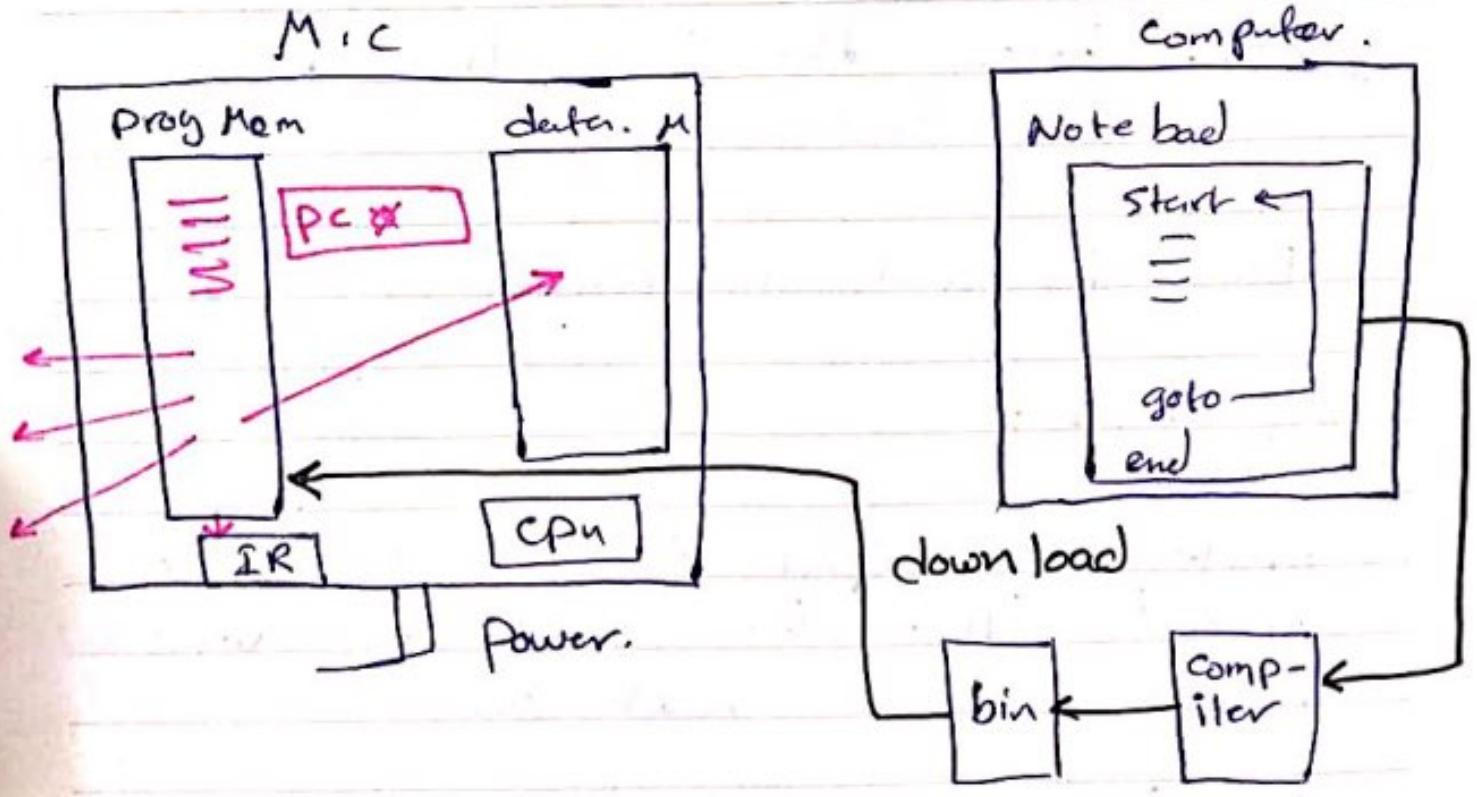


4

prog mem :
permanent : it stay in memory even or power down.

- بكتب الـ code مرة واحدة .

data mem :- it is lost on power down.
- it is fuster and takes less power in writting.



* PC ≡ Program counter &

holds the address of next instruction to be excuted.

* reset vector: the address from where execution start on power up or reset.

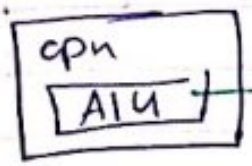
* fetch: graph the instruction from program memory into IR.

* IR: holds the instruction being executed.

PC: is auto increment.

Pipelining :-

while the cpu is execuing the current instruction, the cpu will be fetching the next one.



→ Arethmatic / logic unit

op-code	operands

op code : identify type of instruction.

Slide 16:

Harvard Arch:

- there are 2 busses for each module

→ faster : because pipelining is easier.

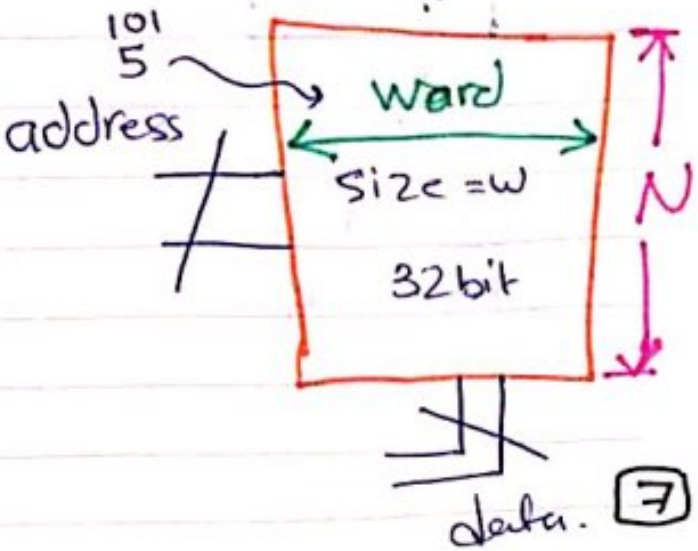
→ the busses have variable sizes.

VNA:

only 2 busses ⊕ less complex.

→ fixed busses size.

word size :
is the size that is reserved when store a variable.



data bus $\geq$ word size (w)

address bus $\geq \lceil \log_2 N \rceil$

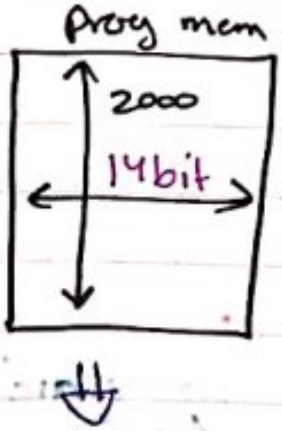
Ex $N = 1000$, data bus = 16.

address bus = $\lceil \log_2 1000 \rceil = 10$

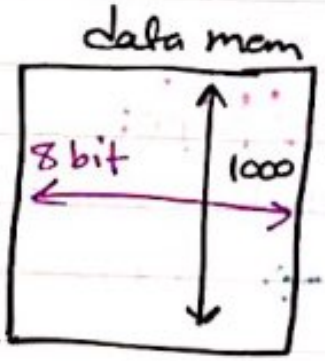
* 9 bit. $\Rightarrow$ address 0 0000 0000 = 0
1 1111 1111 = 511

10 bit $\Rightarrow$ 0 – 1023.

Ex:



$w = 14$
$N = 2000$



$w = 8$
$N = 1000$

	data mem.		prog mem.	
	data bus	address bus	data bus	address bus
Harvard Arch	>8bit	$>\lceil \log_2 1000 \rceil$ $=10$	>14	$>\log\lceil \log_2 2000 \rceil$ >11
VNA	$\max\{14,8\}$ 14	$\max[10,11]$ 11	14	11

25/7/
Tue.

* PIC uses Harvard :-

Instruction set: It is a description for each instruction you can use to write a program.

→ written by a manufacter.

C = A+B

If you didnot stick to the IS ⇒ syntax error detected by the compiler

C = A+B.

logical error ⇒ not detected by the compiler.

* CISC :

→ there are too many types of instruction and addressing modes.

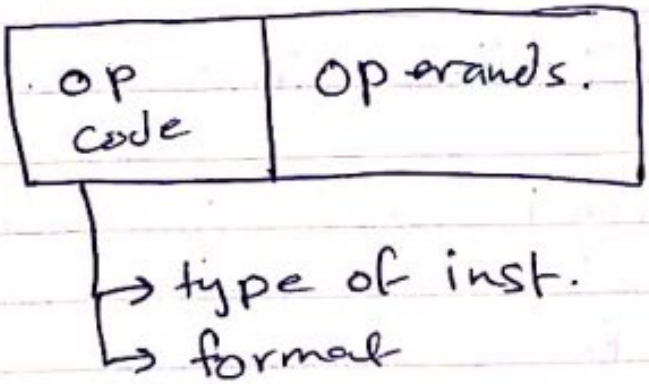
→ many of the inst. are complex (div, multiplication , SQRT).

→ multiple version of the same operation.

* RISC:

→ only few simple instruction and addressing modes.

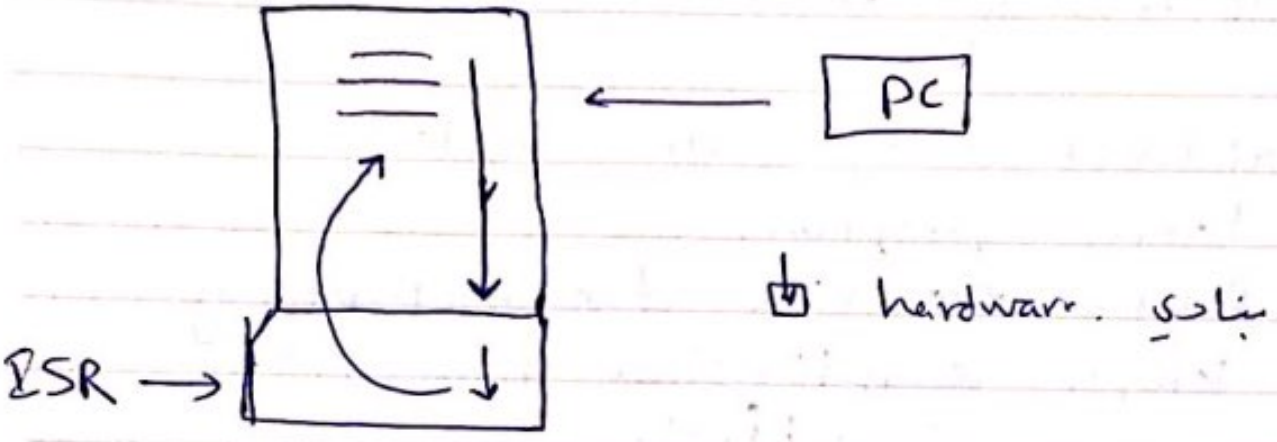
→ all instruction are simple.
→ longer program.
→ faster execution due to pipeling
→ longer ~~compliction~~ Compliction time.
→ PIC is RISC.



* addressing mode: is the way of naming the operand in the inst.

Reset: start execution from reset vector (00)

Interrupt: Specific hardware action that interrupt exculion the main code and start executing the interupt code.



hardware بنادي

* هو code انت بتكتبو لما يجي اي اشي ع HW بروح بنفذو بعدين برجع .

* M.C are divided into families:-

member with in the same family have similliar code and have same inst. set. and inst. format.

when move acode from M.C to another :-

a- within the same family s minimal change.

b- btween differant families & may be totally differant code.

DIP:

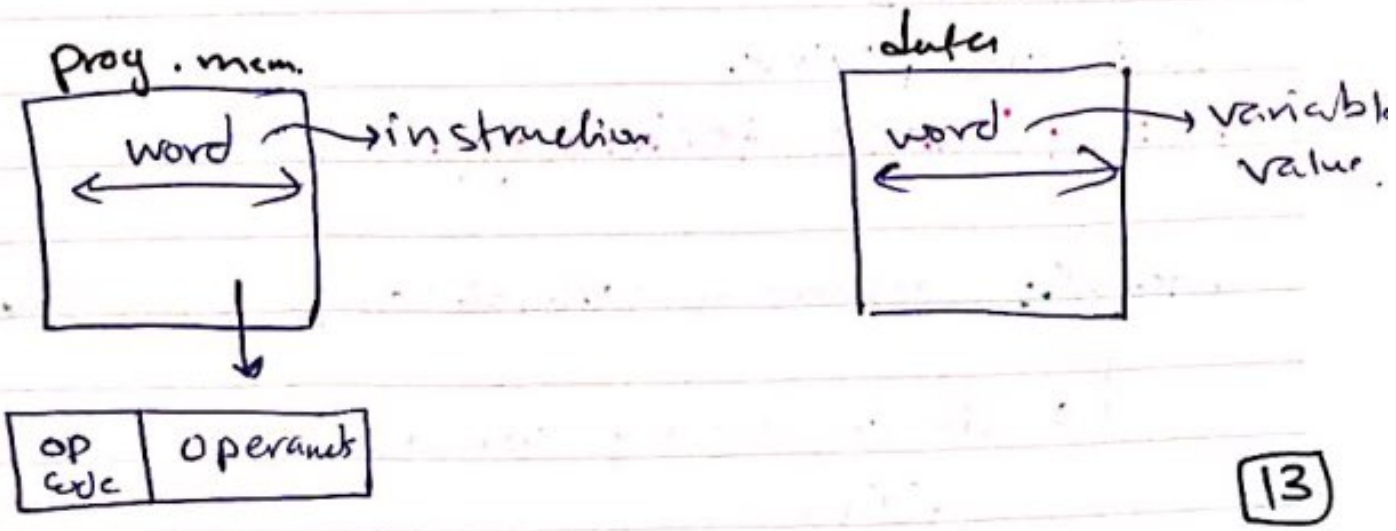
more pins ⇒ more function
size ↑

more inter-pins spacing → size and easier connection

8 bit computer: the word of data. memory is 8 bit.

→ all variables are 8 bit with value 0 - 255

→ data bus on the data memory is 8 bit.



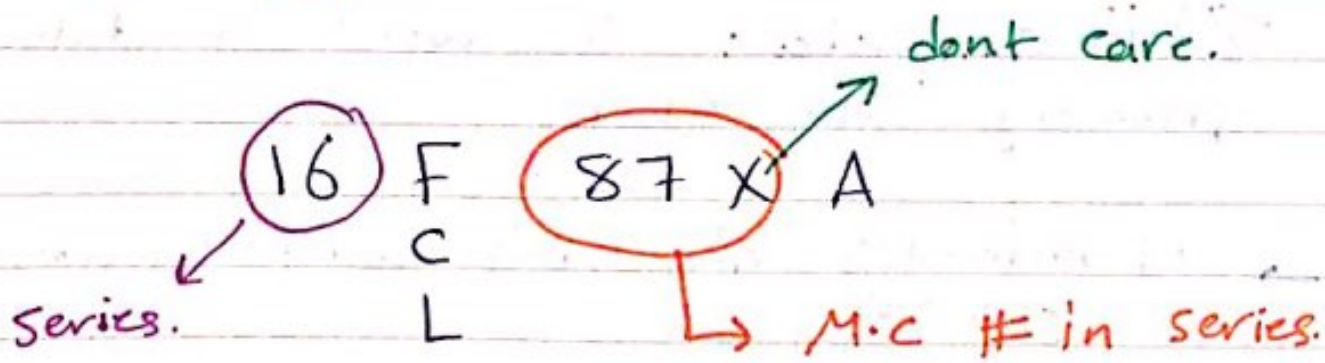
8 bit M.C are divided into 3 families 8

1) baseline.
2) Mid-range.
3) high Performance.

} data sheet.

. family من كل واحد فيهم series *

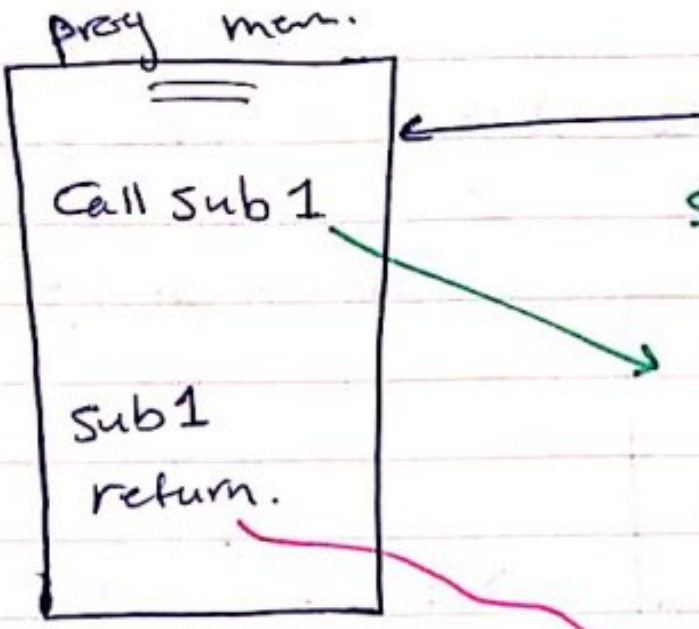
Slide 27



F ≡ Flash memory.
C ≡ C MOS
L ≡ low Power.
A ≡ advance technology.

Stack: memory (Volatile) .. اذا انقطعت بتضيع

↳ automatically written and read FILO.



PC

stack ← PC
push

PC ← address of sub1

PC ← address الاشي إي بدي انفذها.

PC ← stack.
pop

* بخزن بـ (stack. ← PC) عشان عا اعل return

اعرف وين ارجع .

* عا اعل Call بخزن بـ stack .
* عا اعل return بقرأ ال stack .

Ex:

address

a	call sub1 →
b	call sub2.
c	
x	sub1 ≡ return.
y	sub2 return. →

stack [b]
PC ← x

stack [c]
PC ← y

stack
PC ← b

stack
PC ← c

1 level stack

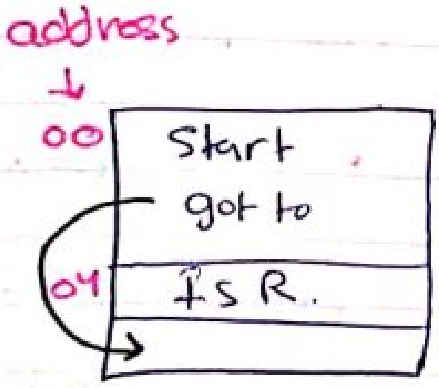
* مير عنك 2 level Stack
ما اعمل Call داخل Sub
قبل ال return.

```
Call sub 1
Call sub 2.

sub 1
    .Call sub 2.   → nested call.
    return.
```

reset vector: the address from where execution start on reset.

interrupt: the address from where execution start on interrupt



on reset

PC ← 00 reset vect.

on interrupt.

PC ← 04 interrupt vect.

*Slide 28:

accumulator (W-Reg) :-

it holds the result of last inst.

*Slide 30:

if the inst. needs 2 variables (+,-,AND) then one of them comes from the W-Reg and the second :-

a- literal (directly from the inst.).
b- data memory : and its address comes :
[1] from the inst. itself (direct address).

[2] from file select Registor. (FSR) (indirect address).

the result will be store either in the W-Reg or in data memory.

MCLR : active low
↳ resent when = 0.

26/7/
Wed

16F84A
↳ mid range family.

* Types of memory in PIC 16F84A:-

① 1K prog. memory ⇒ Code (inst.)
→ permanent.

② 68 bytes data memory → data.
(variable) → volatile.

③ 64 bytes EEPROM → data,
perement, eg setting of device.

peripherals: (I/Os)

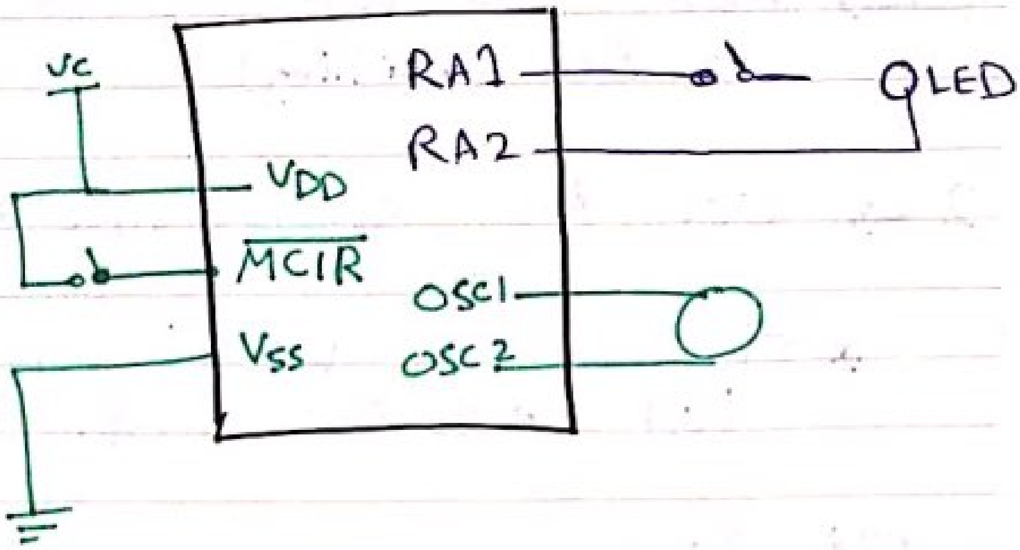
→ 8 pins PORT B
→ 5 pins PORT A
} can be used to connect I/O device.
(5+8=13 I/O)

8 bits: timer Ø
↳ "not I/O"

18 pins :
- → 8 PORT B
- → 5 PORT A
- → 2 V_{DD} , V_{SS}
- → 2 OSC
- → 1 $\overline{MCLR}$

Ex Connect all required pins such that RA1 is connected to SW and RA2 is connected to LED.

Sol:



* RB0 → بقدر اشبك عليه اكثر من function

Ex If the second operand is in the data memory and its address is in the instruction it self, how many bits will be the address that.

comes from instruction ?! :

Sol:

7 bits address, however the address bus of the data memory is 8 bits ⇒ the 8th bit comes from status Reg.

* If I want to add 2 data memory location.

1 → move one ~~from~~ of them to the w-Reg (working)

2 → add the second one to it.

* status reg: hold information about last executed inst. (C?! Z?! N?)

Slide 10:

Configration word:-
14 bit word usually stored in the program memory it hols Configuration Information about the program.

once it is written, it can't be modified except by re downloading the prog again.

* It contains :-

1. OSC ((external)) type that will be connected on OSC1 and OSC2.

RC circuit نوع من ← RC
crystal نوع من ← HS, XT, LP
* تختلف عن بعض بـ speed.

2. WDTE: if = 1 WDT is enabled
if = 0 WDT is off.

WDT: it reset the M.C on crash.

3. $\overline{\text{PWRTE}}$: if = 0 ⇒ ON power up the M.C → stays in reset mode for a while, then it start working.

فترة أول ما يشتغل.

4 CP: Code protection if = 0 the code is protected.

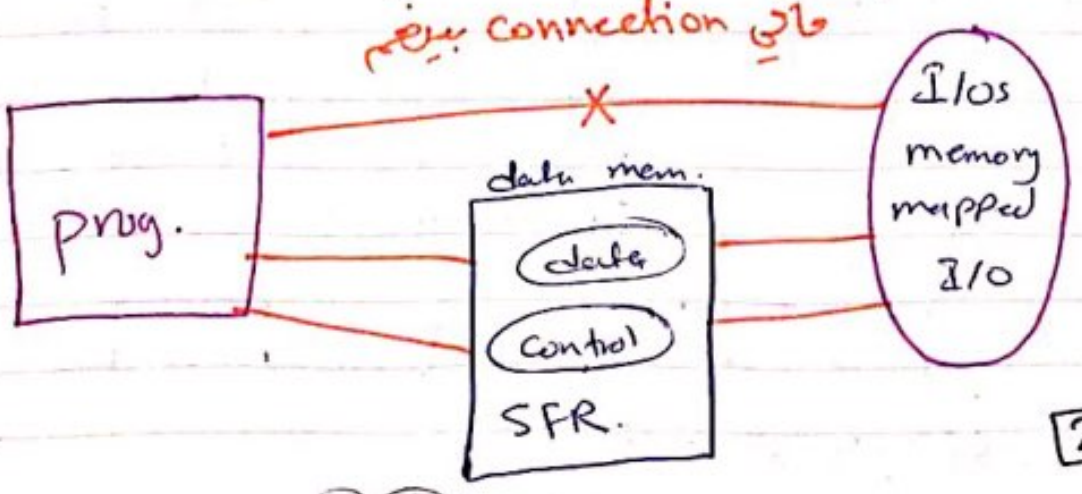
Slide 11:-

* Data memory is divided into 2 ways:-

I Vertically:

→ Special function registor.:
they have additional function inaddition of being data memory location. mainaly to deal with I/Os, Status, FSR.

→ general purpose registor:-
to hold variable vale. I/Os.
memory mapped I/O.



* تعامل مع I/os عن طريق ⇐ SFR

data SFR: contain the control data or the data you want to output.

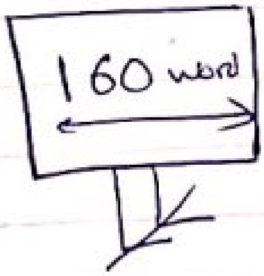
Control SFR: setting of the I/os.

[2] Horizinontaly :-

. is divided into 2 banks :-

- b0 : 00 → 4F → 80 location.
- b1 : 80 → CF → 80 location.

} 160 word.



$\lceil \log_2 160 \rceil$ = 8 bit address.

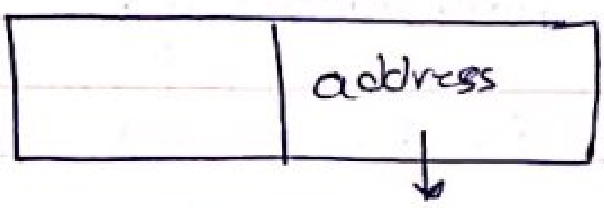
* to address any word inside a specific bank we need $\lceil \log_2 80 \rceil$ = 7 bits.

To address any word in the data memory :-

1bit [1] select the bank if not selected

7bit [2] access the address inside the bank.

1+7=8bit



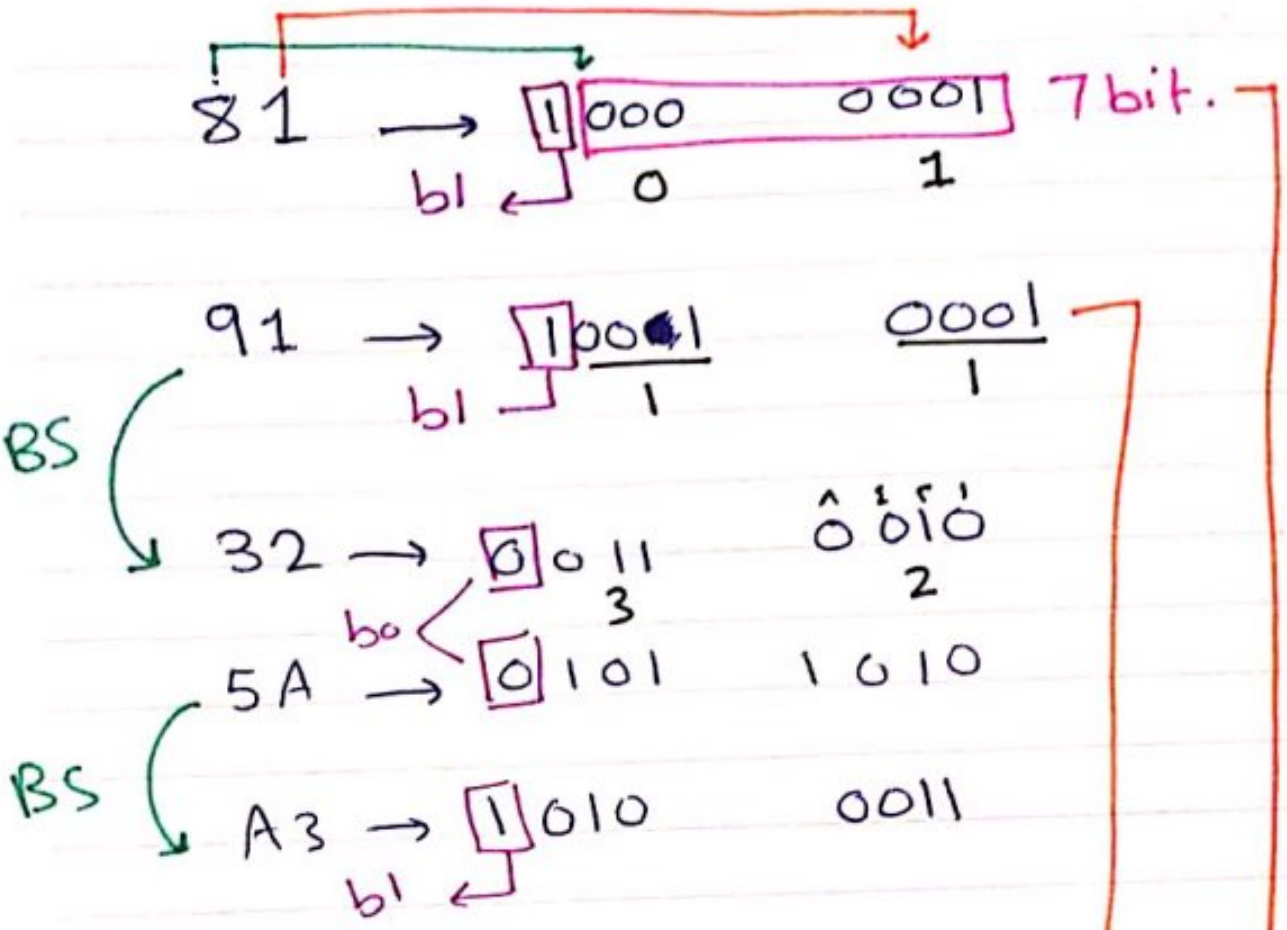
if 2 bank => I saved one bit from the address.

4bank → 2bit for bank selection.

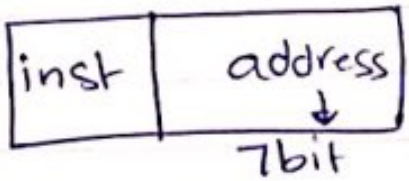
Ex write a pseudo code to read the addresses:
H'81', H'91", H"32", H'5A'
H'A3' ?!

* بحول ال 81 لـ binary ← بطلع على أول رقم وبحدد ال bank

bank
0 → b0
1 → b1




```
select  b1
read    H'01'
read    H'11'
select    bØ
read    H'32'
read    H'5A'
select  b1
read    H'23'
```



27/7/
Thur

* Addressing mode:-

[1] literal:
value is in the instruction

[2] direct address:
value is in data memory. address in the inst.

[3] indirect address:-
value is in data memory address in the FSR.

* Bank selection is dependent on addressing mode 8↓

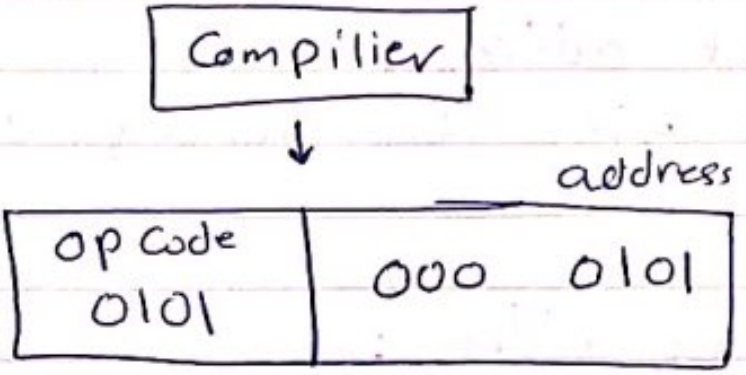
[1] literal: no bank selection.

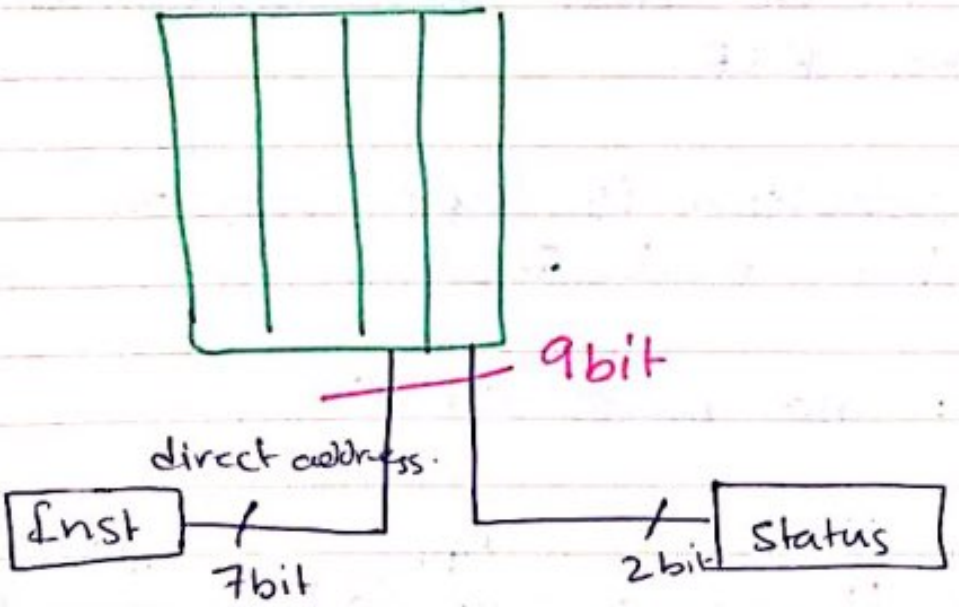
[2] direct address: RP1 RP0
- 4 banks: status(6) and status(5)
- 2 banks: status(5) (RP0)

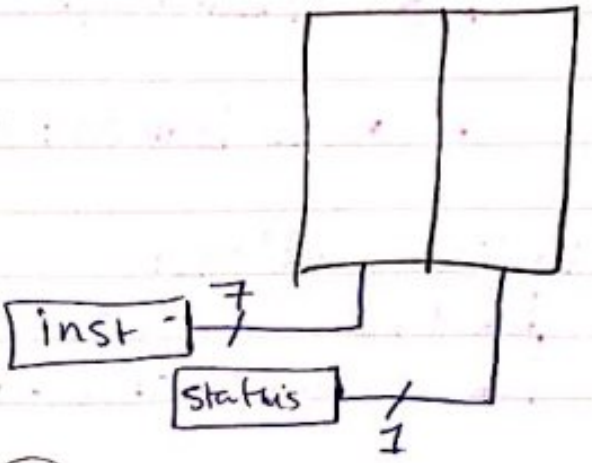
[3] indirect address:
- 4 banks status(7) and FSR(7)
- 2 banks ~~status~~: FSR(7)

H'85' = 1000 0101

Read H'85'







Ex read addresses 0X51 at 0X81, ~~addr~~ direct addressing.

0X51

0 1 01 0001

b0

0X81

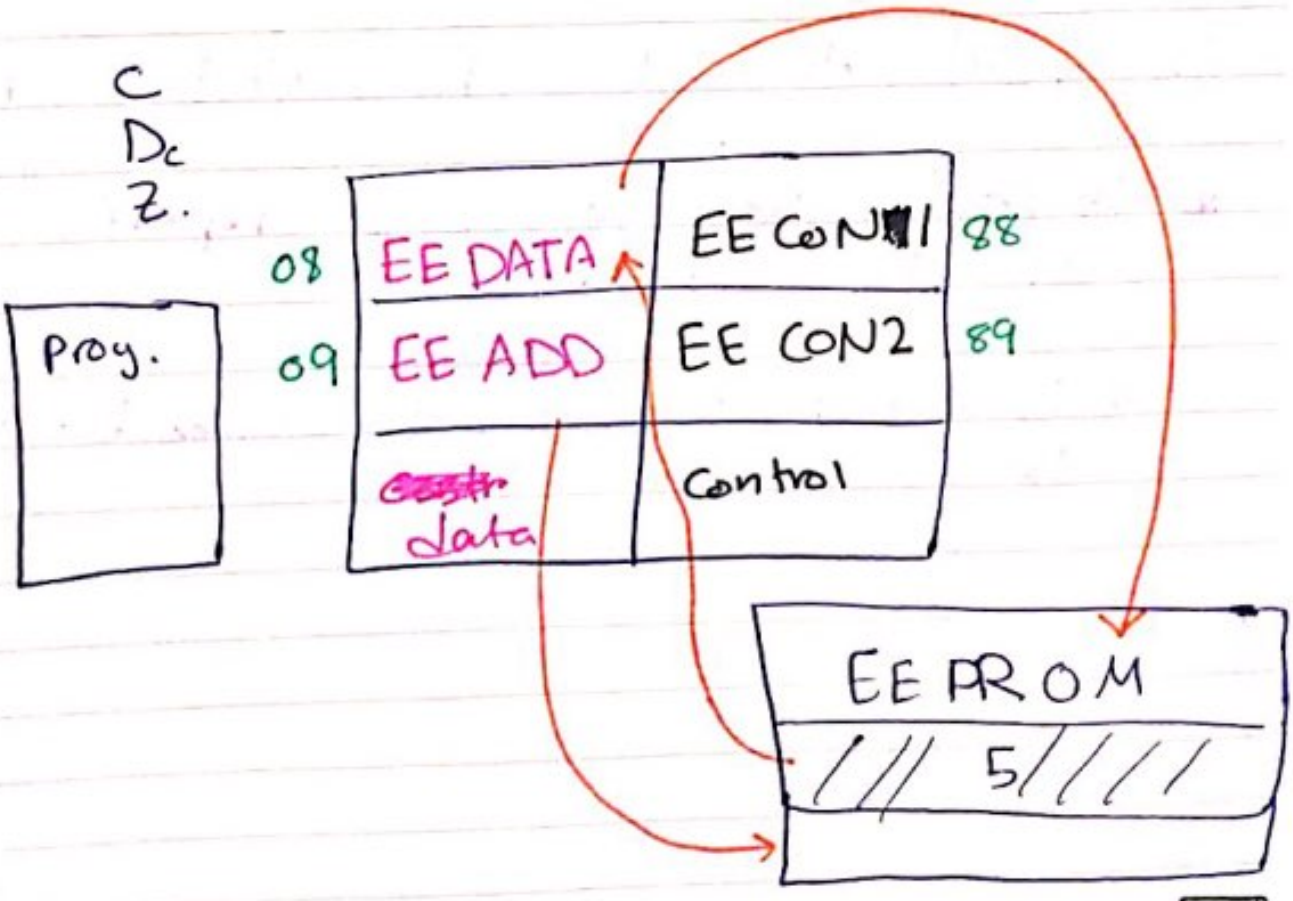
1 000 0001

b1

b0 ← clear RP0
read address H '51'

b1 ← set RP0
read address H '01'

C
Dc
Z.



erse

read EEPROM :-

① store the address to read in EEADD register.

② switch into bank (b1)

③ set RD bit in EECON1 (start reading), start copying the byte at address stored in EEADD EEPROM int EEDATA.

④ read the data from EEDATA Reg.

* to read the next byte repeat all the steps.

RD bit : set by SW cleared by HW to read.

1) value → EE DATA.
2) address to write → EE ADD
3) set : WREN in EECON1 (writting to EEROM is allowed)

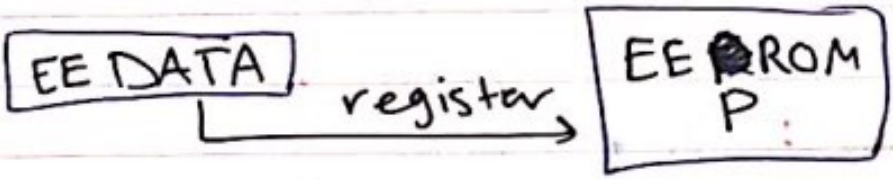
BS

4) write 55H in EECON2
5) write AAH in EECON2

لازم بالترتيب

} to make sure that writting to EEPROM is intensional

6) set WR bit (start writting) in EECON1

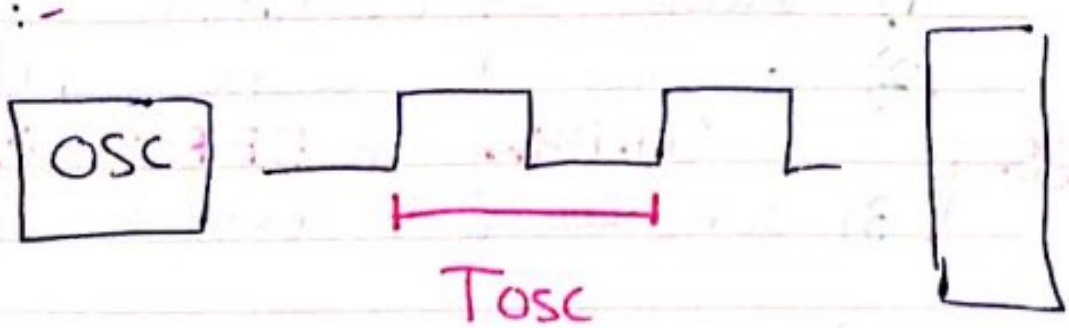


7) When writing is over, EEIR in EECON1 = 1

BS ≡ Bank switching.

* you should repeat all step except 3.

Timing :-



$$F_{osc} = \frac{1}{T_{osc}}$$

Tosc: is not enough to fetch or excute one inst.

$$T_{inst} : X * T_{osc}$$

↳ the time that enough to fetch or excute one inst.

fetch time = excute time = T_{inst}.

* In PIC ⇒

$$T_{inst} = 4 * T_{osc}$$

Ex Fosc = 1 MHz.

sol:

$$Tosc = \frac{1}{1MHz} = 1\mu s$$

$$Tinst = 4 * 1\mu s = 4\mu s.$$

Ex if Fosc = 4 MHz.

$$Tosc = \frac{1}{4} = 0.25\mu s.$$

$$Tinst = 4 * 0.25 = 1\mu s.$$

* each inst requires fetch then execution
  fetch requires = 1 μs
  executed " = 1 μs.

* if we have a prog of size 1 inst ⇒ to execute the whole prog.

1inst x 2 μs = 20 μs if No Piplining

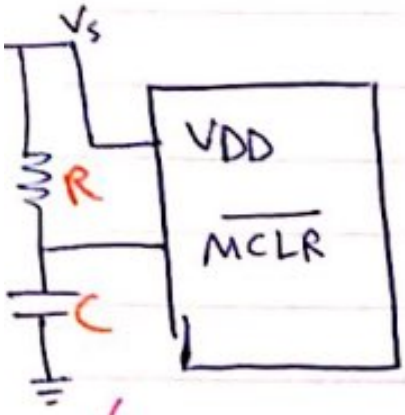
With piplining the time goes to half approxmetaly.

* Some times the Piplining fails because the already fetch inst, write not be excuted. ((eg call, go to, return))

  In this case the cpu need extra Tinst to fetch the new inst.

* all inst need (Tinst to be with piplining) the times goes to half fully excuted except the inst. approxemetly that make pipelining to fail.
  (eg, goto, return, call).

30/7/
SUN



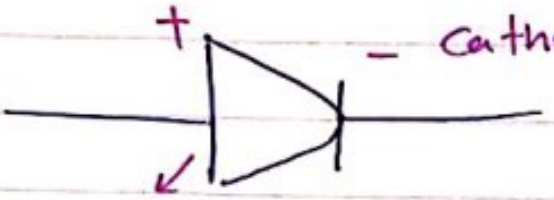
on power up, I don't want the M.C to start immediately it is better to stay in reset mode. for awhile, then start.

- external
- internal

on power up, the M.C will stay in rest mode until the capacitor gets charged which takes time. related to $\tau = R * C$

$R_S \equiv$ is a current limiting resistor to protect the PIC.

Diode:

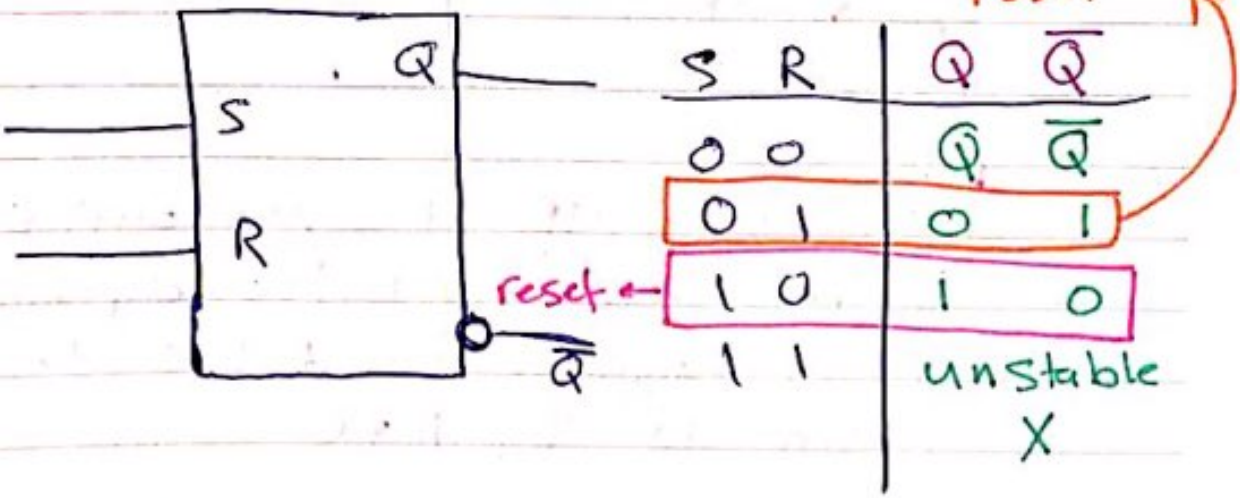


$V_a > V_c$

free wheeling diode will work in forward mode when V_D is disconnected and the discharge will be very fast.

slide 23:

SR latch :-

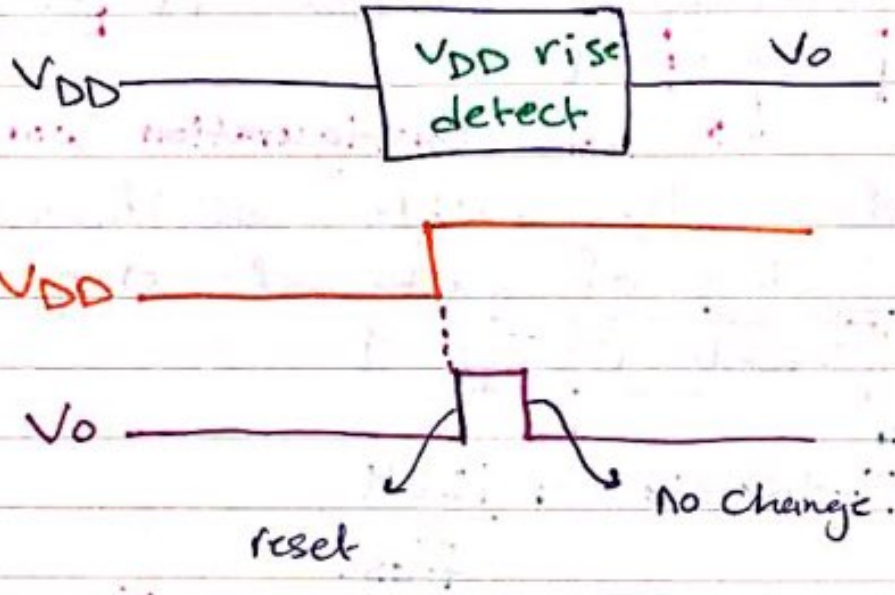


* Internal reset circuit :-
When reset happens. when
S = 1 ((3 cases S=1) =>

a) $\overline{MCLR} = 0$

b) WDT while not in sleep mode ⇒ WDT will not reset PIC in sleep mode. (WDT and $\overline{sleep} = 1$)

c) $V_{DD} = 1$ (on power up).

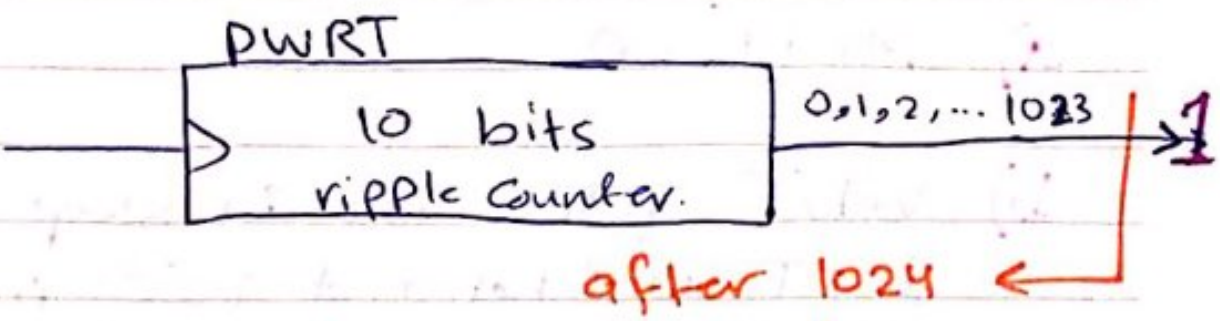


after power up; when does the M.C exit from reset mode ?!

ans: when R = 1

when the following happens concurrently ?!

1) S = 0



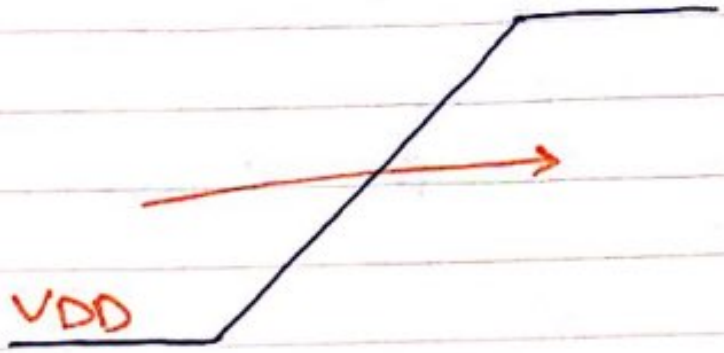
= 72ms

2) after 1024 cycle! of the internal RC OSC if Enable PWRT = 1
   ↳ from configuration word.

3) then, after additional 1024 cycle, of external OSC if enable OST = 1

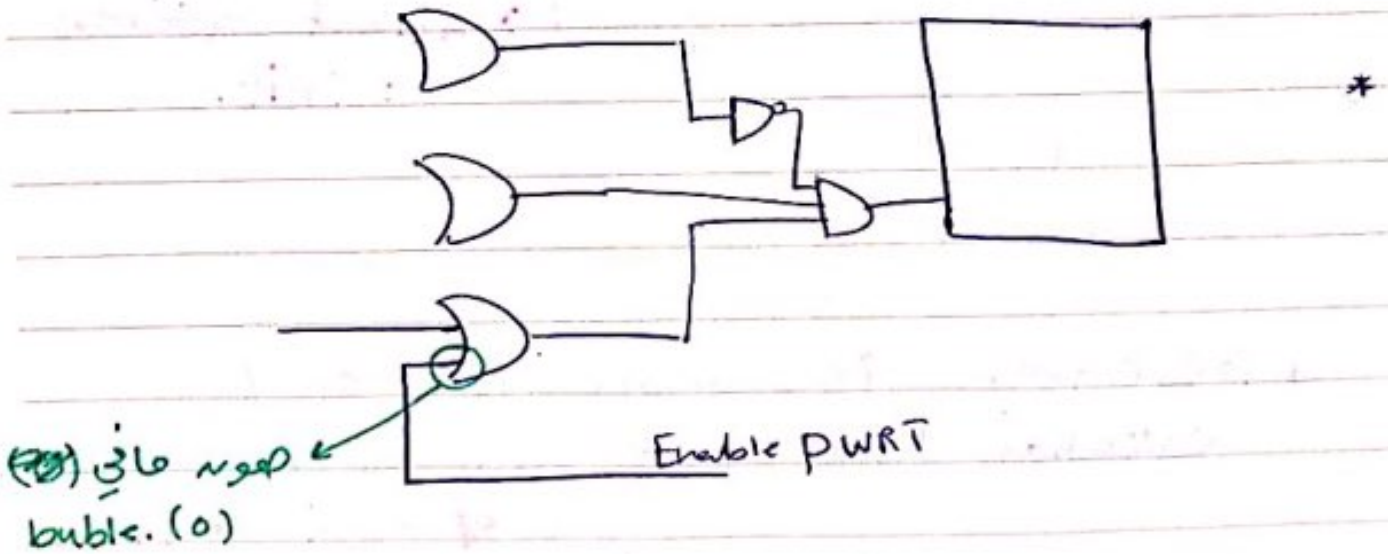
1024 cycle = 1024 / Fosc

is by defult enable for OSC types (XT, LP, HS)



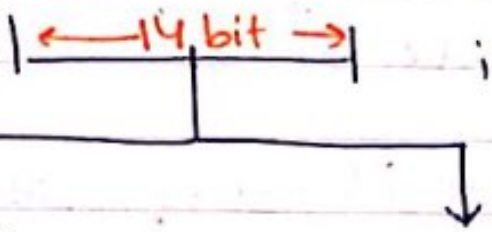
for longer delays you can use both interned and external.

* $\overline{PWRTE}$ ⇒ if = 0 ⇒ Stay in reset mode on power up.



Chapter 4:

00 start. pc=0



op code

operands.
- literal : 8bit
- direct address : 7bit.

end.

* assembly is more efficient execution time and memory usage.

not efficient.

.C → compiler → assembly → machine code

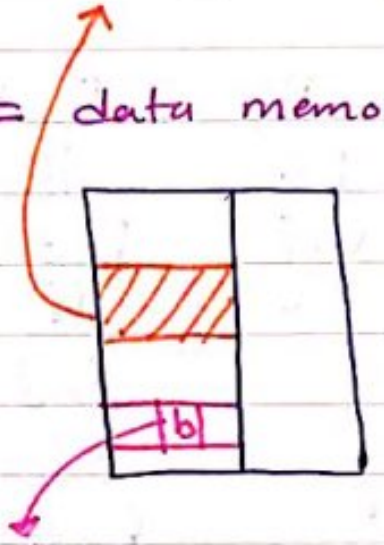
.asm → assembler → machine code

14 bit.

* result will be stored either in W- Reg (if d=0) or in data memory if (d=1).

Instruction Categories

1. Byte-oriented file register.



* file register = data memory word.

f ⇒ 7bit.
d ⇒ 1bit.

2. Bit oriented file register.
   ↳ f ⇒ 7bit , b → 3bit.
3. literal operations, no access to data memory, K ⇒ 8bit , distination is always in W-Reg
4. Control operations.

Call
goto
:

the modification will be carried on the bit itself (no desk)

call k
goto. k

11 bits if it is an address of instruction.

op f, d
op f, d
op K

* ADDWF f, d
ADDLW K
SUBWF f, d
SUBLW K
INCF f, d
DECF f, d
COMF f, d

1 variable

Ex ADDWF 22, ∅

d=0 → w
d=1 → f

[w] ← [22] + [w]

ADDLW 2

[w] ← 2 + [w]

4

22	2
23	3

W 4 ~~6~~ 8

31/7/
MON

35 Instruction.

Arithematic

ADDWF f, d
ADDLW k

SUBWF f, d
SUBLW K

INCF f, d
DECF f, d
COMF f, d

Ex
① ADDLW 20
② ADDWF 21, 1
③ ADDWF 22, 0
④ ADDWF ~~22~~ 22, 1

20	1	
21	2 ~~24~~	
22	~~3~~ 28	

① [w] ← 20 + [w]
② [21] ← [w] + [21]
③ [w] ← [w] + [22]

w | 2 ~~22~~ 25

[f] - [w]

K - [w]

SUBLW 5 ⇒ [w] ← 5 - [w]

* باخذ القيم الموجودة بـ [w] وبطرح منها 5 وبخزنها بـ [w]

SUBWF 22,1 ⇒ [22] ← [22] - [w]

* A-B

A + 2's comp of B

5 - 7

5: 0000 0101 → 2's comp

1111 1010 + 1

= 111 1011

7: 0000 0111 → 2's comp.

1111 1000+1

= 1111 1001

5 - 7 =

5 + 2's comp of 7

C=0 ←

```
  0000   0101
  1111   1001  +
-------------
  1111   1111
```

7 - 5

C=1 ←

```
  0000   0111  ~> 7
  1111   1011  ~> 2's comp 5
-------------
  0000   0010
```

* C is the complement of the Sign

```
COMF  21, Ø
INCF  21, Ø      ← بزيد واحد على القيم المعكوسة بـ 21 وبخزنها بـ w
SUBLW 9
SUBWF 21, 1
```

5 ≡ 0000 0101 → Comp → 1111 1010

F A

21	~~5~~ 2
22	3

w | ~~AØ~~ ~~FA~~ ~~Ø~~ 3

Tinst

ANDWF f,d
AND LW k

IOR WF f,d
IOR LW k

XOR WF f,d
XOR. LW K

* for bit masking :-

$A \cdot \emptyset = 0$
$A \cdot 1 = A$

$A + \emptyset = A$
$A + 1 = 1$

$A \oplus \emptyset = A$
$A \oplus 1 = \bar{A}$

Ex write an instruction to complement the least sig 4 bits of the w-Reg?

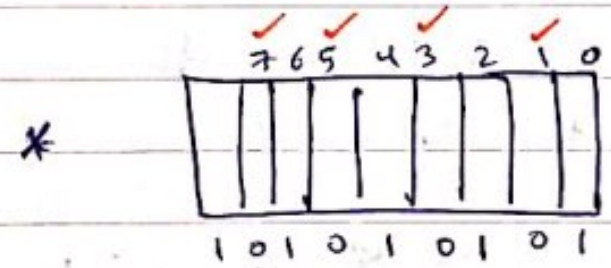
Sol: XORLW 0F

Ex 2: write an instruction to set the least sig 4 bit of the W-Reg?

Sol: IORLW 0F.

Ex 3: write an instruction to clear the odd bit of W-Reg?!

Sol: ANDLW 55



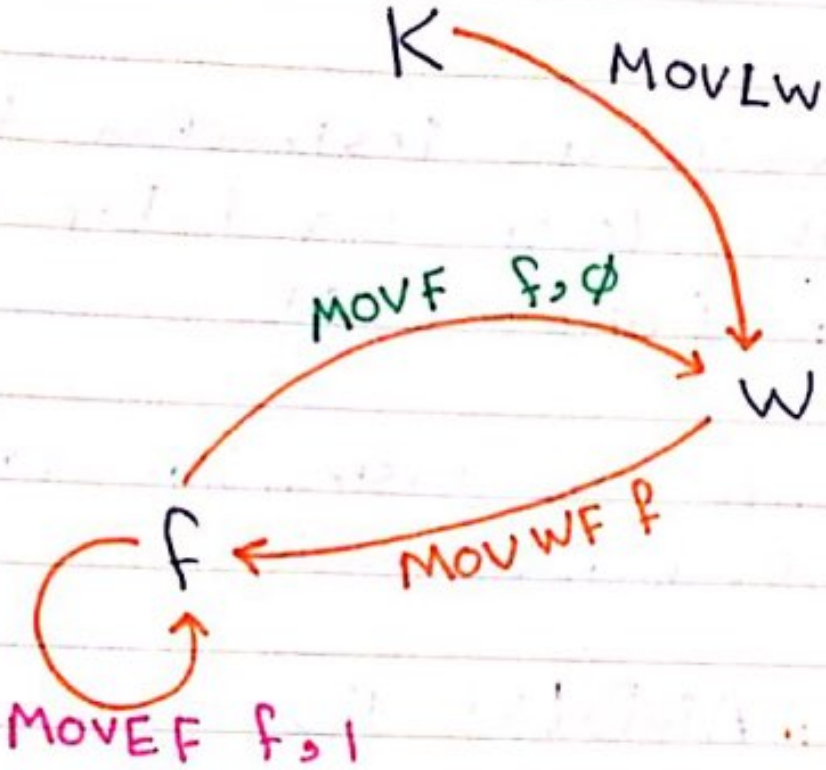
if w < 5

SUBLW 5
check C flag 5 - w.
if C = 0 → 5 < w
if C = 1 → 5 ≥ w

if w = 5

XORLW 5
check Z flag

0000 0101
(+) 0000 0101
0000 0000



Ex write code to initialize the address 22 with value 5

MOVLF 5, 22 ✗ ما عندي إياها → فنعملها بخطوتين.

Sol:
```
MOVLW  5
MOVWF  22
```

* to initialize any data memory location
  1. write the value in W-Reg
  2. move the W-Reg to the data memory location.

* to copy any data memory location into another one.

eg. copy address 22 into 25

copy ⇐
```
MOVF   22,0
MOVWF  25
```

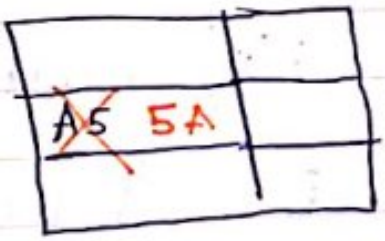
* To check any file reg = 0 ?!

```
MOVF   f,1
check  Z flag
```

* SWAPE 11,1

". 4bit بتبدل 4bit "

11



* Call K ⇒ push stack ← PC, PC ← K } can use return.

* go to K ⇒ PC ← K without return.

* Return ⇒ PC ← pop stack

BTFS C f,b

BTFS S f,b

INCF SZ f,d

DECF SZ f,d

→ Conditional Instruction.

→ Conditional branch.

↓

skip.

↓ Conditional Skip

Skip of the next inst

Ex write code to do the following

① if [22] = ∅

Increment 22

Sol: ⇒ MOV F 22,1

BTFS C STATUS,Z

INCF 22,1

[22] = ∅

Z = 1 → يعني skip.

② if [23] < 5
COMplemat 23.

sol ⇒ MOVLW 5
SUBWF 23, ∅ ⇒ [23] - 5
BTFS [5] STATUS, C [23] > 5
COMF 23, 1 ↳ c = 1
else [23] < 5
↳ c = 0

for (int i = 22, i > 0, i = ...

MOVLW 22
MOVWF 20] → Counter.
loop DECFSZ 20, 1

GOTO loop.

* 2 T instruction if skip
* 1 T instruction if not skip.

BCF f,b clears bit # b in address f.

BSF f,b sets bit # b in address f.

Ex1 write instruction to select bank 1

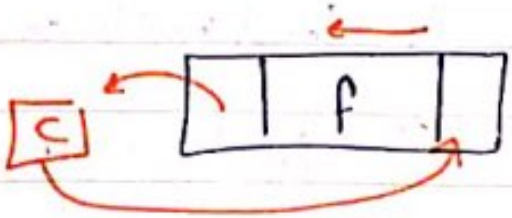
Sol: BSF STATUS , RPØ

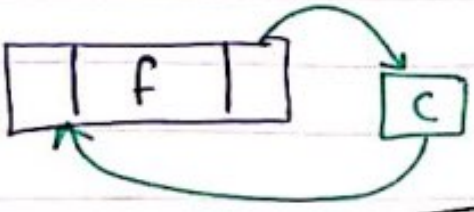
Ex1 write instruction to clear C-flag in Status Reg.

Sol: BCF STATUS , C
select bank Ø
BCF STATUS , RPØ

RLF f,d

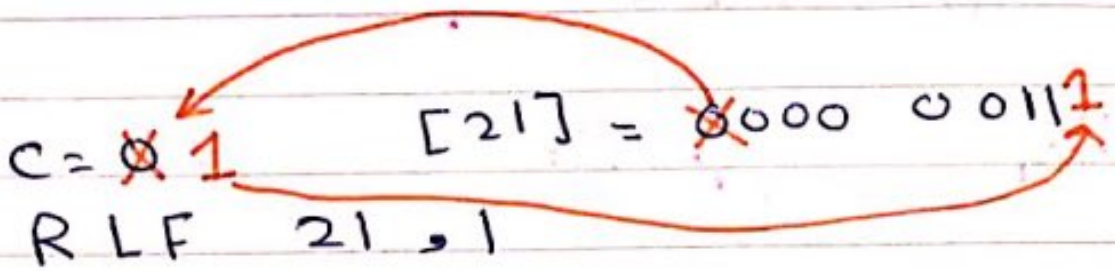
RRF f,d





2 if c = 0.

2+1 if c = 1



* integer division by 2 :-

Ex write code to $\frac{7}{2} = 3$

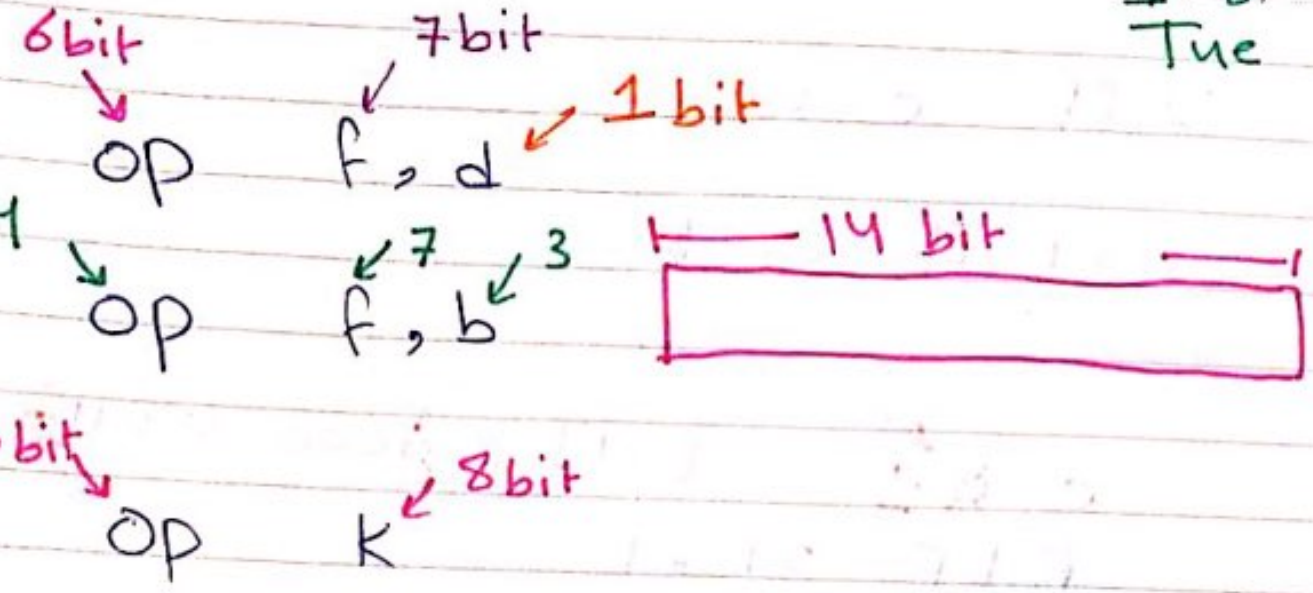
multiply location 15 by 4.

Sol:
```
BCF   STATUS, C
RLF   15, 1
BCF   STATUS, C
RLF   15, 1
```

* ADD WF 03, 1

[03] ← [W] . [03]

1/8/ Tue



3bit ↓
call k ← 11 bits
goto

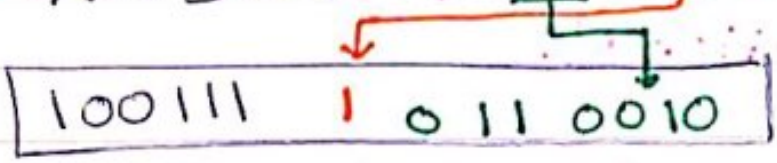
* If the op-code is of fixed size $\lceil \log_2 35 \rceil = 6$ bits

Q: Is there a posibility that one instruction to have op-code = 101 and another inst to have op-code = 101011 ?!

sol: impossible → will never be executed.

Ex| convert the following instruction into its binary interpretation assume the op-code of the instruction ADDWF = 100 111

sol: ADDWF 32, 1



* labels

→ start at the very beginning of the line.

→ start with letter or underscore, but not number.

assemble is case insensitive.
"a" = "A"

→ labels are case sensitive
Once it is defined, it can be used as operand
GOTO lable → is the address of first inst comes after it.

label

Inst 1

Inst 2

GO To label

Ø bit.

11 bit

→ Can be in a stand alone line
→ optional.

* menemonic → 35 instruction.

* Comment
  - → start with ';'
  - → can be in stand alone line
  - → non executable.

* operand

Decimal ⇒ D'23', d'23'

default ⇐ Hex ⇒ H'23', h'23', 0X23

Octal ⇒ O'23', o'23'

Binary ⇒ B'10111011' , b'10111011'

ASCII ⇒ A'G' , 'G' , a'G'

ADDLW 23

* If the value starts with letter (A - F) then it should be preceded by either 0X or H to distinguish it from label.

* Assembler directives :

they give some information to the assembler at compilation time ; then they are discarded

include it opens a file and you can use anything inside it.

*include < IO stream . h >

```
Void main ( ) {
  cout << "Helb";
}
```

X ← ORG 5
5 ← inst 1
6 ← inst 2

} it tells the assembler that after translate. inst 1 into binary to store the ~~binary~~ binary interpretation into address 5

Ex1

```
    Org  1
    goto  start
    Org  04
    goto  ISR
    Org  06
start
    inst 1
    inst 2
    inst 3
    inst 4
    ORg  10
ISR inst X
    inst Y
```

0	goto 06 (*)
1	—
2	—
3	—
4	goto D'10'
5	~~inst 1~~ —
6	inst 1
7	inst 2
8	inst 3
9	inst 4
10	Inst X
11	inst Y
12	
13	

(*) EQU: its defines a const which can be used anywhere in the code.

```
STATUS    EQU   03
RP0       EQU   5
RP1       EQU   6
C         EQU   0
DC        EQU   1
Z         EQU   2
```

select bank ⇐ BSF STATUS , RP0

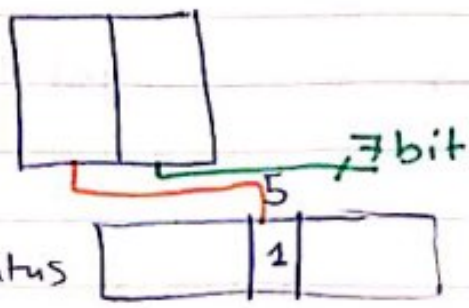
1 ⇐

```
ADDWF   RP1 , C
ADDWP   6 . 0
```

include PIC16F84A.INC

↳ you can use all register and flags by their names.

* اذا كنت كاتب BSF STATUS , C ⇐ و
بيعمه هالكتب الـ include فو الـ EQU بيعاني
syntax error.



```
cblock    20
          var1
          var2
          var3
          var4
          endc
```

define multiple constant:

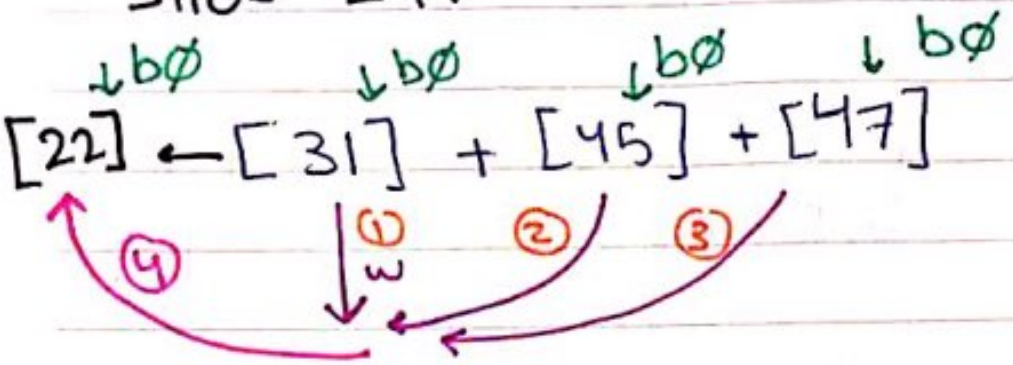
```
s.t       var1 = 20
          var2 = 21
          var3 = 22
          var4 = 23
```

```
var1   EQU  1
  BSF   var1, var1
          ↓       ↓ bit loc
        7bit      3bits.
        address
```

* end: tells the compiler no to compile anything after it.

* while (get line) != "end"

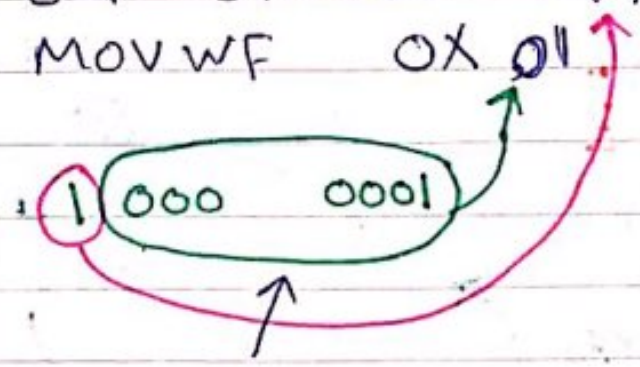
Slide 24:



① MOVF 31 , 0
② ADDWF 45, 0
③ ADDWF 47, 0
④ MOVWF 22

---

* BSF STATUS RP0
MOVWF 0X 01



81

Slide 25:

how many instruction ?!

↳ 8 inst.

8 inst * 14 $\frac{bit}{inst}$ = 112 bit (60)

* go to Start
  - 11 bit
  - 5. تقريبا

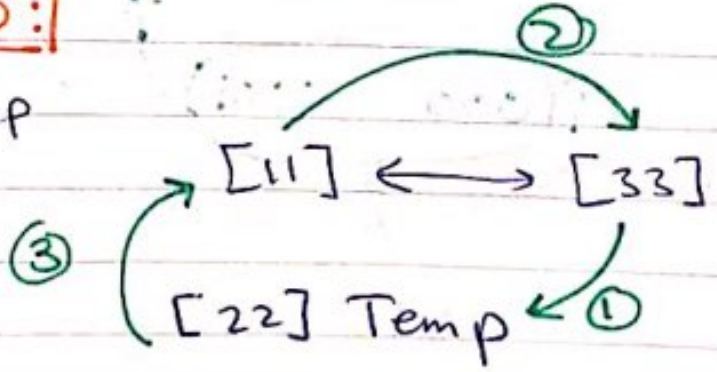
* برنامج حتى ينفذ اذا قمنا بازالة 7 cycle
  Done goto Done.

* Done goto Done → ∞ cycle.

* address ∞ for move 9

Slide 26:

swap



① ≡ MOVF 23,0
MOVWF 22

② ≡ MOVF 11,0
MOVWF 33

③ ≡ MOVF 22,0
MOVWF 11

bank switch تذكر

chapter 5:-

BTFSC	f, b	Conditional skip
BTFSS	f, b	
INCFSZ	f, d	
DECFSZ	f, d	

Ex if [22] x = 0

block 1

else

block 2

sol: code ⇒ sol (1)

if [22] = 0 → z = 1

```
MOVF   22, 1
BTFSC  STATUS, Z
GOTO   block1
GOTO   block1
```

block 1 ≡ goto Next

block 2 ≡

Next.

Sol (2)

```
MOVF    22,1
BTFSC   STATUS,Z
GOTO    block1
block2  ---
        ---
        ---
GOTO Next.
block1  ---
        ---
        ---
Next.
```

Ex H.W

if ([23] < 5)
multiply addres 30 by 8
else.
complement address 31

Ex2 H.W

if ~~address~~ [21] is odd
add to it 1
else
add to it 2.

Ex for (i = 30 ; i ≥ 0 ; i--)
block 1

* "inc" في loop بدي Counter

```
counter EQU 21
        MOVLW D'30'
        MOVWF counter
loop    block 1
        DECFSZ counter,1
        go to loop
```

بتربط بس بهاد المثال اذا كان بينفذ مرة وحدة.

اذا كان 0 بضل loop. ∞

Ex for (i = 15 ; i < 30 ; i++)
block 1

Sol:

```
center EQU 20
MOVLW D'15'
MOVWF counter
loop  block 1
      INCF counter
      MOVLW D'30'
      SUBWF counter,0
      BTFSS STATUS,Z
      go to loop
```

Ex * write code to multiply the values in address D'20 - 50' by 2

* multiply by 2 the value in data memory location which has odd address?

slide 8:

Ex

[11] + [22].
if c=0
[33]
else
[44]

Sol:
```
MOVF   11, 0
ADDWF  22, Ø
BTFSC  STATUS
GOTO   block 1
MOVWF  33
block 1  MOVWF  44
Next  ——
END
```

Slide 10:

Ex

count H	count L
3	2

* بعادل من countL كم مايصير 0 بعدين بعادل من countH
نسى لازم اشيك انه اذا countL و countH مش صفر.

* اذا كان (countL = ∅) وعملت decrement ⇐ FF

count H	count L
2 ~~3~~	5 ↓ 0

كم مرة صيطبقها ؟!

5 → 0 مرات 5

~~3 * 256~~

5 + 3 * 256.

* Subroutines:- code usually start with label and ends with RETURN or retlw K and invoked by call.

call sub 15

sub ⟶ ≡

2 Tinst { Return
retlw K → { MOVLW K
Return.

* Call → stack ← Push pc

pc ← address of first inst in sub

* Return → pc ← POP stack

* to pass parameters to the subroutines, we store them in data memory, and the subroutine work on these data memory.

Slide 167 handling لازم 0 بدي اعمل عليه اذا كان

multiply CIRW.

~~Multiply~~ MOVF 0X31, 1
BTFX STATUS, Z
Return.

Repeat add wf 0X30, 0
decfsz 0X31, D
goto repeat
return.

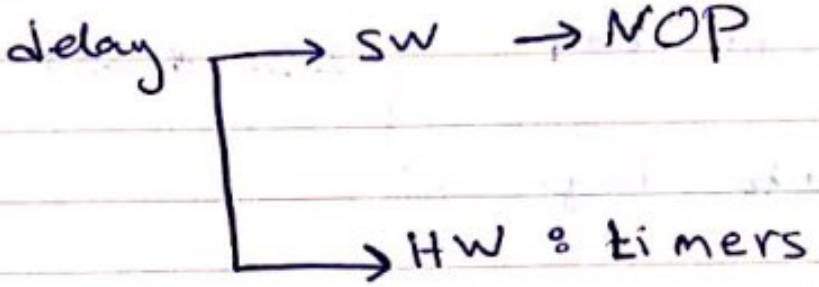
[30] = 3

[31] = 0

call multiply

. ناتج 256 ← 3 ×

* Delays -



* Ex suppose Fosc = 4MHz.

$$\Rightarrow T_{inst} = \frac{4}{4\ MHz} = 1\mu s.$$

inst 1

1Ms ← NOP } 1 Tinst.

1Ms ← NOP

1Ms ← NOP

NOP } 100 NOP

⋮

inst 2

```
Loop
        NOP        } 256 مرة
        goto loop
```

```
loop    Nop            }
        go to. loop    } 256 * 256
        go to loop.    }
```

* Nested loop: using subroutine.

```
loop
        call sub1
        go to loop
sub 1
loop 2    NOP
          go to loop 2
          return.
```

$$Delay = Tins * \#of\ Tinst. * \frac{4}{Fosc}$$

↳ trace the code
تتبع الـ code

3/8/
Thur

$$delay = \frac{4}{Fosc} * \#\ of\ cycle.$$

↳ trace the code.

delay of loop:

- ↳ initialization
- ↳ all itterations except the last.
- ↳ last iteration.

Slide 20:-

$$delay = \frac{4}{Fosc} * \#\ of\ cycle.$$

$$= \frac{4}{800\,KHz} * [(1+1) + 199*(\underbrace{1}_{nop}+\underbrace{1}_{nop}+1+2)]$$

decfsz, goto

$$+ (\underbrace{1}_{nop}+\underbrace{1}_{nop}+\underbrace{2}_{decfsz})]$$

$$= 5\,Ms * [2 + 199*5 + 4]$$

$$= 5.005\ ms$$

If this Code Subroutine, you should add 2 Tinst for call and 2 Tinst for Return ⇒ totaly delay

$= 5.005\ ms * 5\ \mu s * 4$

$= 5.025\ ms$

Ex write a subroutine that gives 5ms delay that Fosc = 1MHz ?!

Sol:- $delay = \frac{4}{Fosc} * \#\ of\ cycle.$

$5 \times 10^{-3} = \frac{4}{10^6} * \#\ of\ cycle$

⇒ **# of cycle = 1250.**

```
sub   MOVLW   X D'207'
      MOVWF   Counter
      nop
      nop
      nop
loop  nop
      nop
      nop
```

```
nop
decfsz      counter , 1
goto        loop
```

Return.

MOV.LW

call

$1250 = 2 + (1+1) + ((X-1)*6) + 5$

$+ 2$

MOVWF

return.

$1250 = 2+2+6X-6+7$

$1250 = 5 + 6X$

$X = \frac{1245}{6} = 207.5$

$5 + 6 * 207 = 1247$

I need 3 more cycles.

nop . فبضيف 1250 ~~2050~~ بدي أنا

Slide 21:

Fosc = 4MHZ.

=> Tinst = 1MS

delay = Tinst * # of cycle.

$10 \times 10^{-3} = 1 \times 10^{-6}$ * # of cycle.

# of cycle = 10,000

nested loop: Initialization. +
First external itteration.
last internal iteration → all internal except last
all external iteration except first and last.

last external iteration.

ex 6 int

5

100 ↓ 0 } 100

4

0
FF
F5 } 256
↓
0

5

call ↙ decfsz ↘ goto ↙

10,000 = [2 + 5 + 249 * (1+2) +

decfsz (2+1+2) + [11 * (255 * (1+2) + (2+1+2)]

decfsz goto

+ (255 * (1+2) + (2+2+2)]

COUNT H = ~~13~~ 12 → 1 → 11 → 1 (لانو بنزل صفر)

COUNT L = ~~250~~ 0 0

* if count L = 0 → صافي داير أخذ First external

Why do we need indirect addressing ?

Ex ⇒ write code to clear data memory location with addressies

10 ⟶ 4F

64 locations

```
CLRF    f

CLRF    10
INCf    10,1
CLRF    11
CLRF    12
CLRF    13
CLRF    .
        .
        .
CLRF    4F
```

Solution : use direct addressing.

(1) write the address in FSR.

(2) work on INDF (0x00)

Ex to clear data memory loc. write address to using indirect address ?!

sol:- MOVLW D'64' D'32'
MOVF counter

MOVLW 10 } (1)
MOVLWF FSR }

MOVLW 11

loop CLRF INDF → CLRF 10.
→ INCF FSR,1

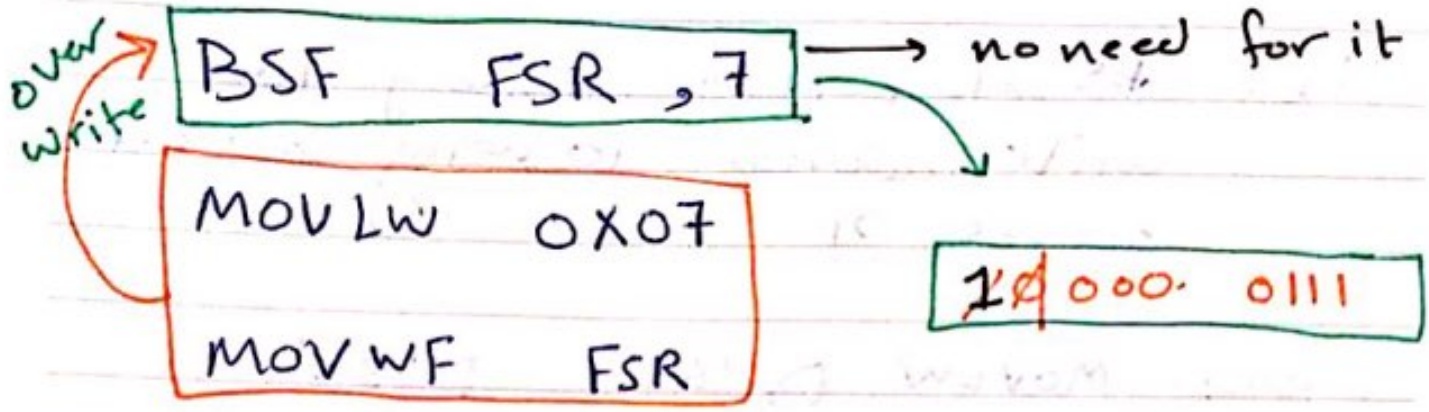
INCF FSR,1
DECFSZ counter,1
go to loop

sol 2. ⇒ حل آخر لازم اغير واخزن

Ex1 write code to read data memory Loc with address 87 using indirect addressing.

(1) 000 0111
b1

STATUS (7) and FSR(7)



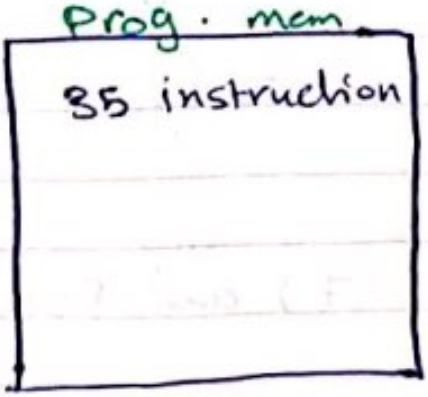

```
MOVF    INDF, 0
```

```
MOVLW   0X87
MOVWF   FSR
MOVF    INDF , 0
```

Slide 26:

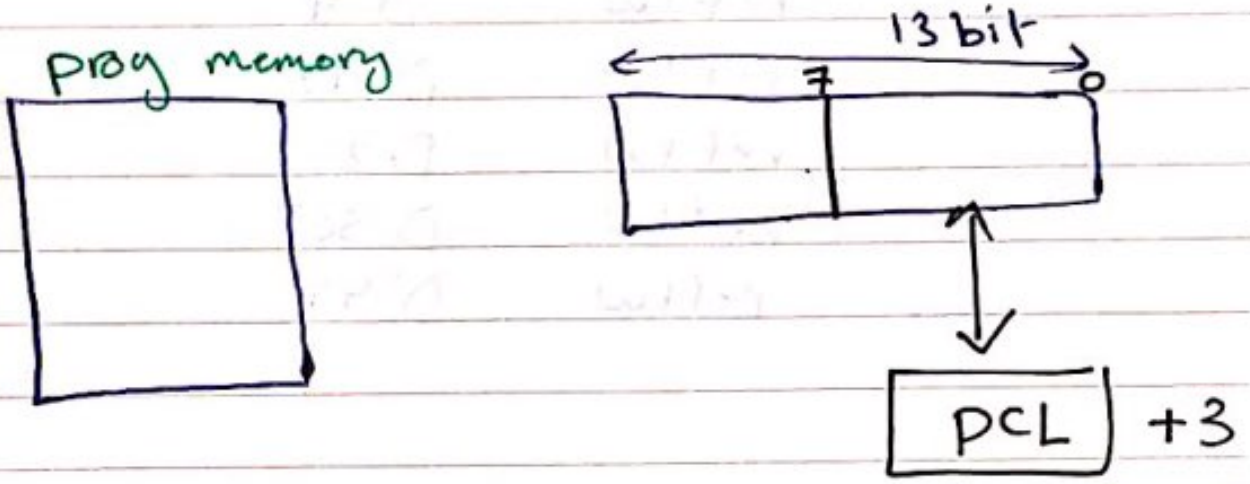
```
int [6] = { 0, 1, 4, 9, 16, 25 };
Cout << X[2]; => output = 4
```

* How to define array in assembly ?!



↳ using lookup table

```
sub   ADDWF   PCL, 1
      retlw   D'0'
      retlw   D'1'
      retlw   D'4'
      retlw   D'9'
      retlw   D'16'
      retlw   D'25'
```



يجمع ٣ على PC.

```
MOVLW   2        } indexing element
Call    sub      } # 2 / sub[2]
MOVLW   5
Call    sub
sub   ADDWF   PCL, 1   → excution time = 2
      retlw D'0'  4
      retlw D'1'  9
      retlw D'4'  16
      retlw D'9'  25
      retlw D'16' 36
      retlw D'25' 49
```

اي اشي بعدل على PC بأخذ 2 instruction.

* بدي ضرب الاعداد من 2 → 7 فبعدل على code.

Sub 2 from w

```
MOVLW   5
Call    sub
sub     MOVWF   Temp
        MOVLW   2
        SUBWF   Temp, Ø
        retlw   D'4'
        retlw   D'9'
        retlw   D'16'
        retlw   D'25'
        retlw   D'36'
        retlw.  D'49'
```

6/8/
SUN

chapter 6:-

* INDF is not a physical registor.

inst 1

loop

NOP

goto loop

1ms

inst2

* inst 2 → inst/no 1ms بتنفذ 2/

HW

↳ timer ∅

[some setting to give an alert after specific delay ~~there~~ and the CPU keeps excuting the code]

inst 1

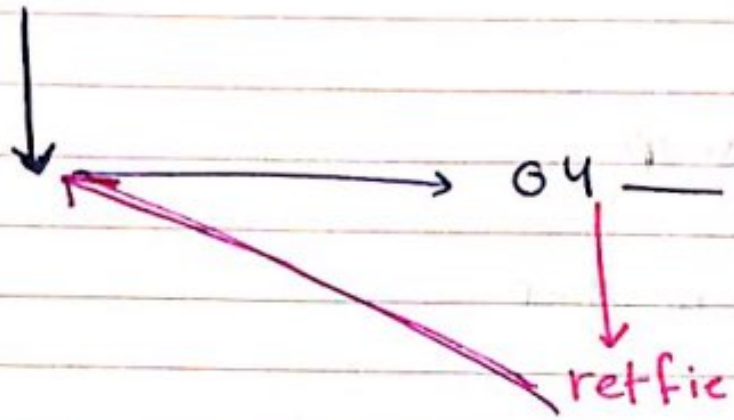
code on timer ∅ to give 1ms delay

inst 2.

↳ interrupt

Continous executing بنفذ ما لم يحدث delay بدون

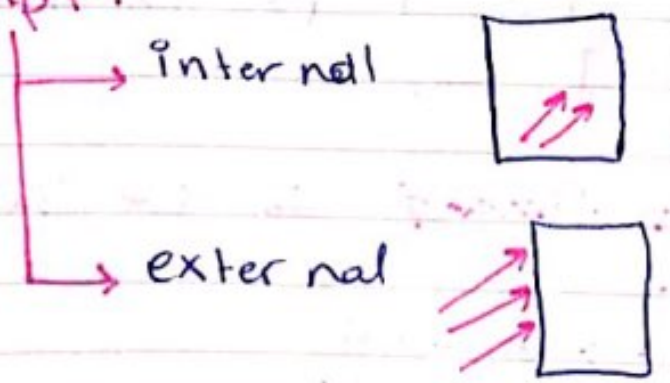
Interrupt: is a subroutine starts at address 04 ends with retfie invoked by HW.



higher priority than the main code.

Interrupt ⟶ polling (Keep Checking)
بضل اشيك اذا خلص ولا
الـ delay

Interrupt:



Interrupt

→ maskable: can be disabled
↳ In PIC, all interrupt are maskable.

→ non-maskable :- can't be disabled.

PC ← 04

* for interrupt to happen:-

1. GIE = 1
2. local enable = 1
↳ there are 4 types of interrupt
• each has its own local enable.
3. local flag = 1
↳ each of the 4 interrupt has its own flag

set by HW

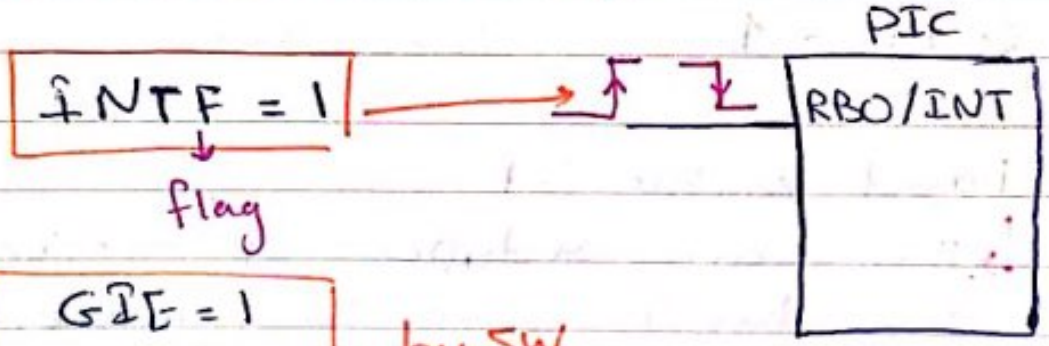
enabled / disabled by SW (code) ⇨ by default they are disabled.

* The flages are set by HW regardless of the enables. However, the interrupt will not happen except if the GIE = 1 and local enable = 1

* The flages will be cleared on reset.

There are 4 tupes of interrupt:

[1] external interrupt



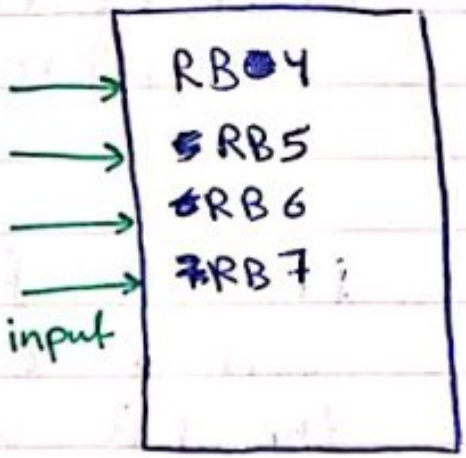
[2] Timer Ø overflow interrupt.

T0IF = 1 → by HW

T0IE = 1
GIE = 1 → by SW

3 Port B interrupt on change.

When any of the pins RB4 - RB7 is input and its value gets changed



RBIF = 1 → by HW

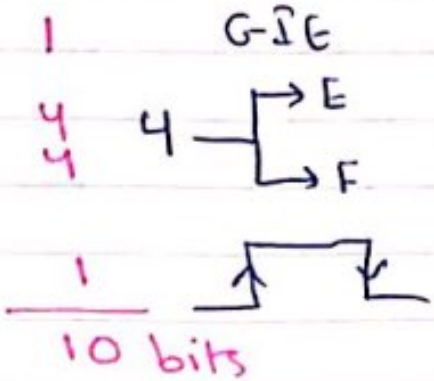
RBIE = 1, GIE = 1 → by SW

4 EEPROM write complete interrupt:-

WR = 1 → start writing

EEIF = 1 → by HW

EEIE = 1, GIE = 1 → by SW



INTCON(8)

EEIF is in EECON1

in OPTION_REG(6)

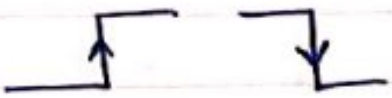
If local flag = 1 while its corresponding local enable = 1 the PIC will wake up from sleep mode regardless GIE value.

8/8/
Tue

GIE

4 types of interrupt:-

each has flag and enable → EEIF in EECON1

 → OPTION-REG

INTCON

for interrupt to happen:-

by SW
1. GTE = 1
2. local enable = 1
3. local flag = 1 (by HW)

⇒ PC = 4

Slide 11 :-

* Clear GIE: to prevent any other interrupt the current one.

Interrupt should end with Retfie

GIE = 1 : set دلوقت

E_1 = 1

E_2 = 1 " Flag = 1 لا interrupt, هو اذ "*

F_1 = 1

GIE=1

ISR

F_2 = 1

retfie

GIE = 1 بعير لا set for لحا اعلى * flag

flag est ⇒ set by HW
cleared by SW

Retfie:- ① pc ← pop stack

② set GIE ..

How to use interrupts—

① start at 04

② finish with retfie

③ enable GIE

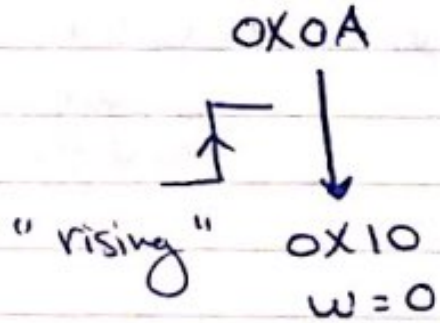
(4) enable local flag.

(5) at the begining of the code, clear the flag if not cleared at reset (RBIF) portB change interrupt flag.

(6) clear the flag before retfic.
flages are set by HW
cleared by SW

slide 13

Ex



Sol:

```
org  0x00
go to  start
org  04
go to  ISR
BSF  INTCON , GIE
BSF  INTCON , INTE
BSF   STATUS , RP0
BSF   OPTION_REG , b
BCF   STATUS , RP0
CLRW
```

```
loop   ADDWF  0XA, 0
       go to  loop
```
→ if counter = 0

```
ISR    MOVWF  0X10
       CLRW
       BCF    INTCON, INTF
       BCF    INTCON, INTF
       INCF   counter, 1
       retfie

       end
```

ممكن اضيفها بس بعاد انو ال ما بكتبها

* بكون بشتغل على W او address 12 وبعدين صار interrupt عندي وصار يخزن فوق اللي انا كنت اشتغل عليه فلازم اعمل context saving ← بعمل انا وين وقفت

[12
  W]

ISR ⇒ save in temp loc

over write W
12

W ISR MOVWF Temp

MOVF Temp, 0
retfie

(12) ISR MOVF ~~12~~, 0 (Status)
MOVWF Temp

modifies Z-flag

MOVF Temp, ∅ → if status=0 ↳ Z=1
MOVWF ~~12~~ * STATUS
retfie

STATUS

ISR SWAPF STATUS, ∅
MOVWF Temp

SWAPF Temp, ∅
MOVWF STATUS
retfie.

Ex write a code to increment the address 0X25 by 1 on external interrupt
Inc by 2 on port B change
INC by 4 on EEPRD ISR complete interrupt

sol ⇒ BSF INTCON, GIE
BSF " , INTF
BSF " , RBIE
BSF " , EEIE

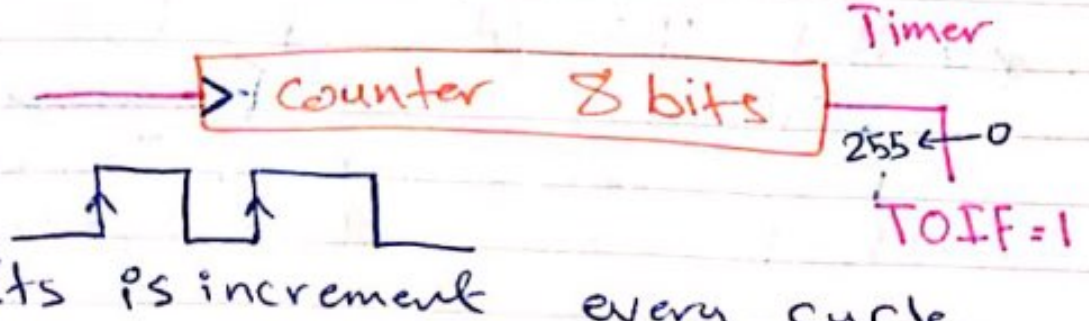
90

ISR BTFSC INTCON, INTF
go to external.
BTFSC INTCON, RBIF
go to port B change.
BTFSC INTCON, EEIF.
go to EEPROM

external INCF 0X25, 1
BCF INTCON, INTF
retfie.

port B change MOVLW 2
clear the flag
[ADDWF 0X25, 1
BCF INTCON, RBIF
MOVF PORTB, 0
retfie.

EEPROM :



Its is increment every cycle.

* When we can use counter as timer ?!

1. Know the driven frequency

eg :-

if F = 1MHZ ⇒ after 256 MS → TOIF = 1

cycle ⇒ $T = \frac{1}{1MHz} = 1MS$.

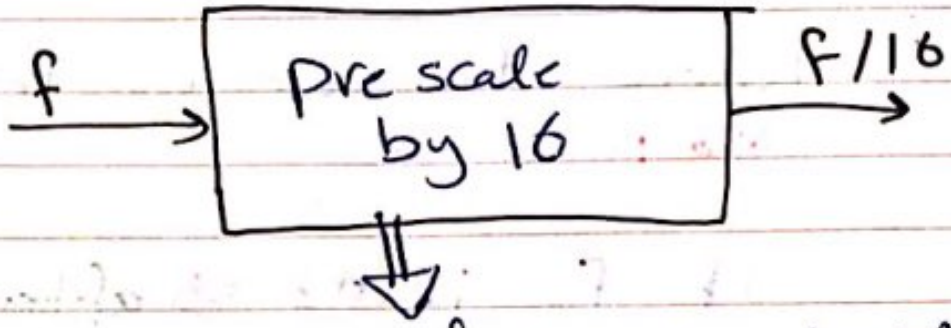
2. ability to initialize.

eg :

If F = 1MHZ and I want TOIF = 1 after 100 MS ⇒ initalize to 256 - 100 = 156

* To program we control the intial value and the clock to give certain delay.

TMRϕ (01) $\Rightarrow$ for initial value



frequency divider
↳ time is longer

If PSA = ϕ $\Rightarrow$ there is a prescale with value = 2.
PS2, PS1, PSϕ + 1

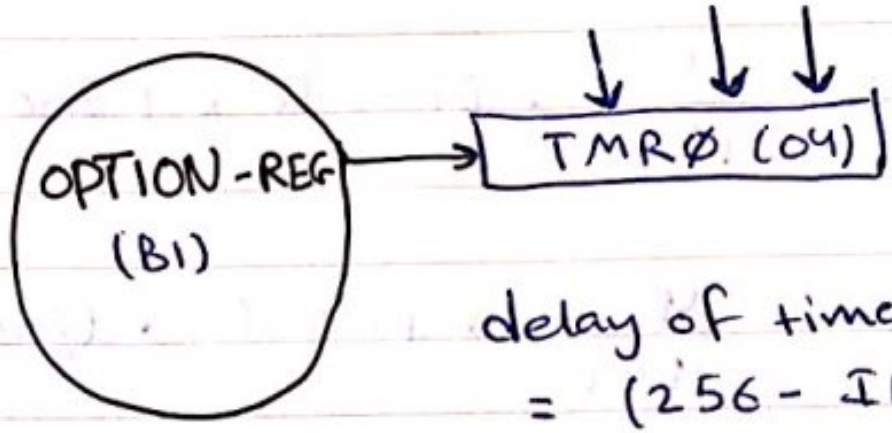
eg :- If PSA = ϕ
and PS2, PS1, PSϕ = 100
$\Rightarrow$ prescale = 2^{4+1}
$2^5 = 32$

clock can be internal with
frequency = $\frac{Fosc}{4}$

or external (RAM).

9/8/
Wed

Slide 19



delay of timer Ø
= (256 - IN) * 4/Fosc
* prescale.

delay of timer Ø = (256 - IN) * [4/Fosc * prescale.]

#of cycle

cycle enter ~~timer~~ TMRØ

max delay of timer Ø = 4/Fosc * 256 * 256

IN = 0

prescale = 256

Ex write code that complement file register 0X21 after 10ms (approximatly) use timer Ø and Fosc = 400 KHz ?!

94

Sol:

$$10ms = \frac{4}{Fosc} * prescale * (256 - IN)$$

$$10 \times 10^{-3} = \frac{4}{4 \times 10^{6}} * prescale * \underbrace{(256 - IN)}_{X}$$

$$pre * X = 1000$$

if $2 \Rightarrow X = 500$ ✗

معادلة بمجهولين
انا بفرض قيم لوصل
منعم.

prescale = 4

$X = 250$

$IN = 256 - 250$

$= 6$

```
* include
  ORG  0x00
  goto  start
  ORG  0x04
  go to  ISR
start   BSF   INTCON, GIE
        BSF   STATUS, TOIE
        MOVLW   6
        MOVWF   TMR0
```

10ms

```
BSF     STATUS , RP0
MOVLW   B' XXOX 0001'
MOVWF   OPTION , REG.
BSF     STATUS, RP0
```

go to loop

```
ISR.    COMF: 0X21, 1
        MOV LW  6
        MOV WF  TMR0
        BSF     INTCON , T0IF
        retfie
        end.
```

Timer 0 = 0

complemate — 10ms بدي اطلع بحيث كل

sol * modify the code such that every 10ms the address 21 is complemat?

2 تمرين * modify the code to complemat 0X21 every 2 second.?!

↳ max delay = $\frac{4}{4 \times 10^5} * 256 * 256$

= 0.65 seconds

اذا كان ال delay اكبر من delaymax بستخدم counter.

delay = counter * $\frac{4}{Fosc}$ * prc * (256 - IN)

2s = counter * 10ms

counter = 200

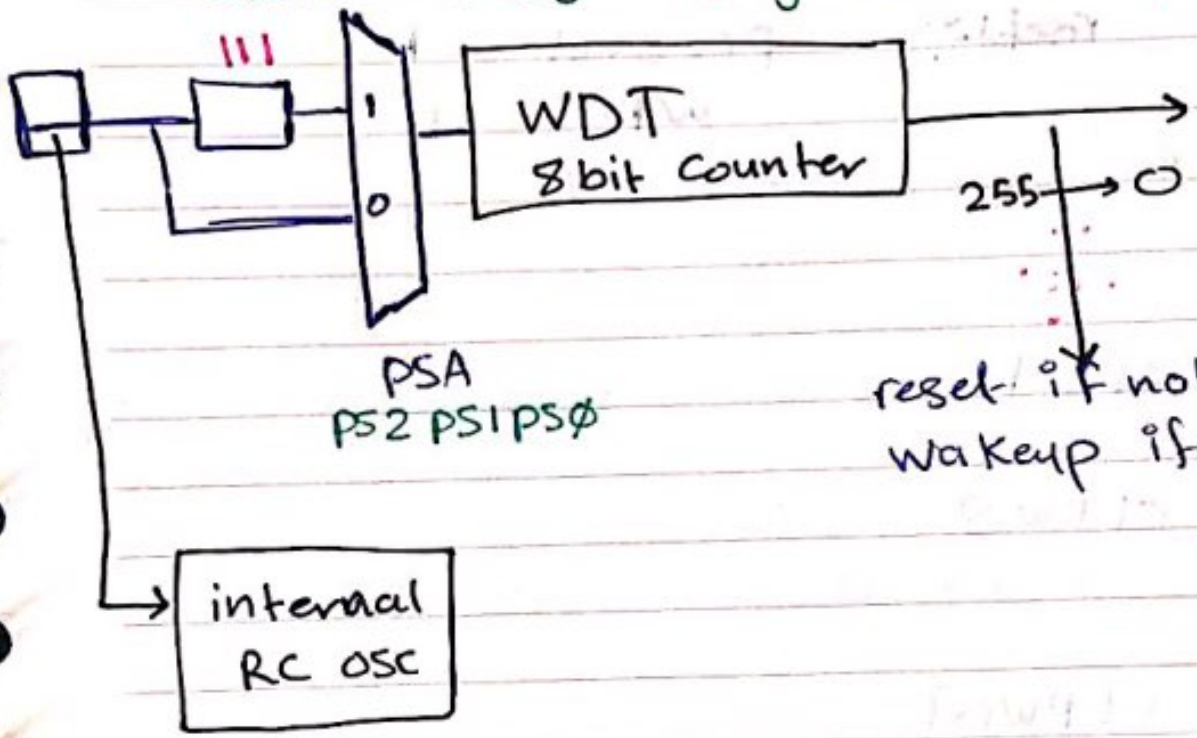
* after goto ISR ♯ code تعديل على ال

```
        MOVLW   D'200'
        MOVWF   Counter
          .
          .
          .
ISR     MOVLW   6
        MOVWF   TMR0
        BCF     INTCON, T0IF
        DECFSZ  counter,1
        goto    ~~comp~~ Next. or retfie
        ~~[illegible]~~
Comp    MOVLW   D'200'
        MOVWF   counter
        COMF    0X21
Next    retfie
        end
```

Slide 24:

* Watch dog timer :-

you can't initialize it, however you can clear it by using CLRWDT instruction



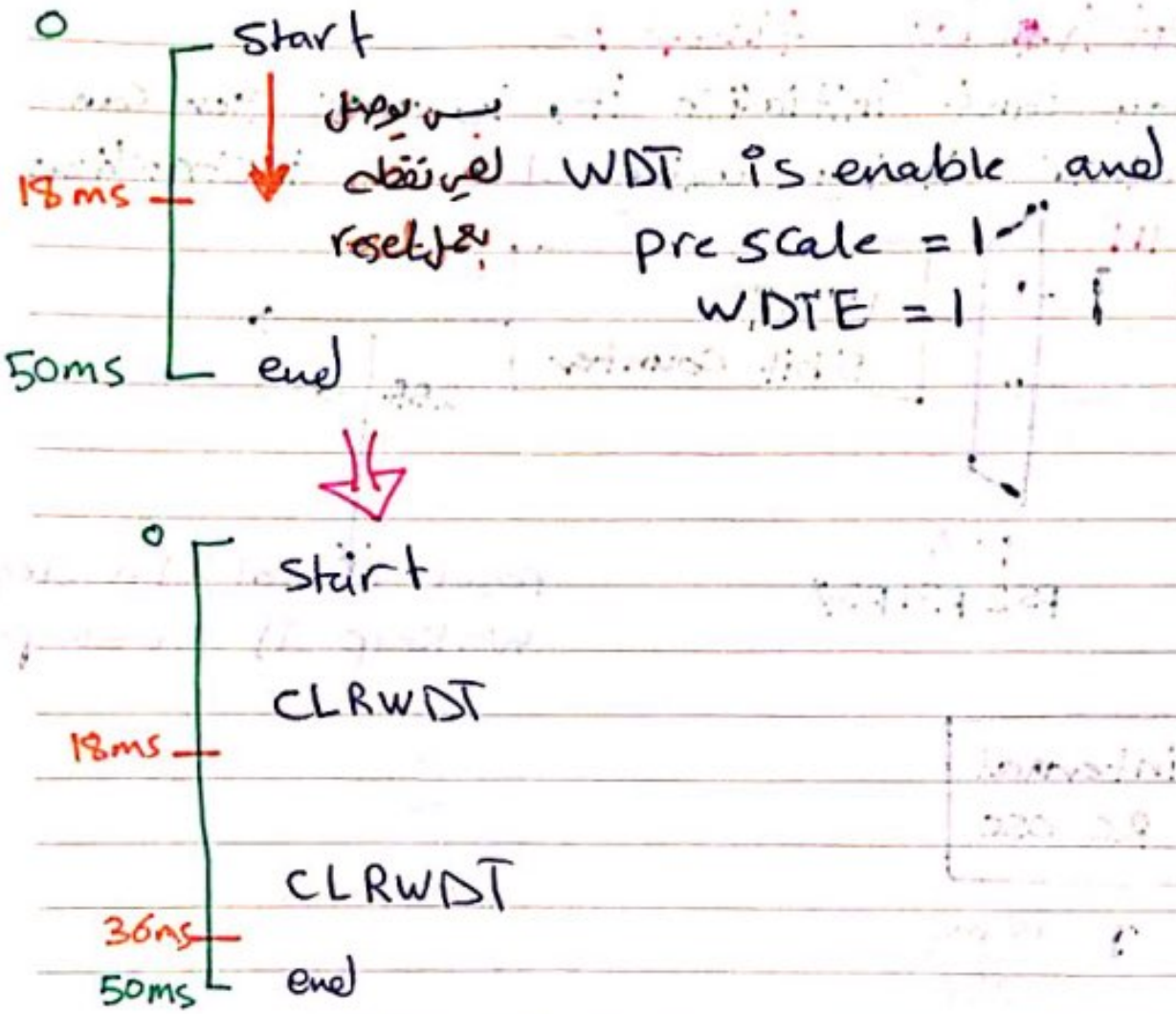
$T = \frac{18\,ms}{256}$

prescale = 2

eg: If option reg = B'0000 1100', after how long the M.C reset.

Sol: delay = 256 * prescale * $\frac{18\,ms}{256}$

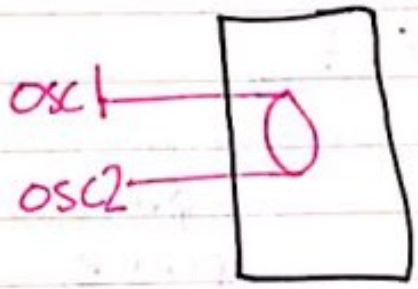
= 16 * 18 ms



max delay of WDT = 128 * 18ms
= 2.3 seconds

10/8/
Thur

Sleep ⇒ enters the M.C in sleep mode to save power.



WDT keeps counting even is sleep mode because it uses it's own RC osc.

sleep
inst 1 ⇒ will be excuted after wakeup

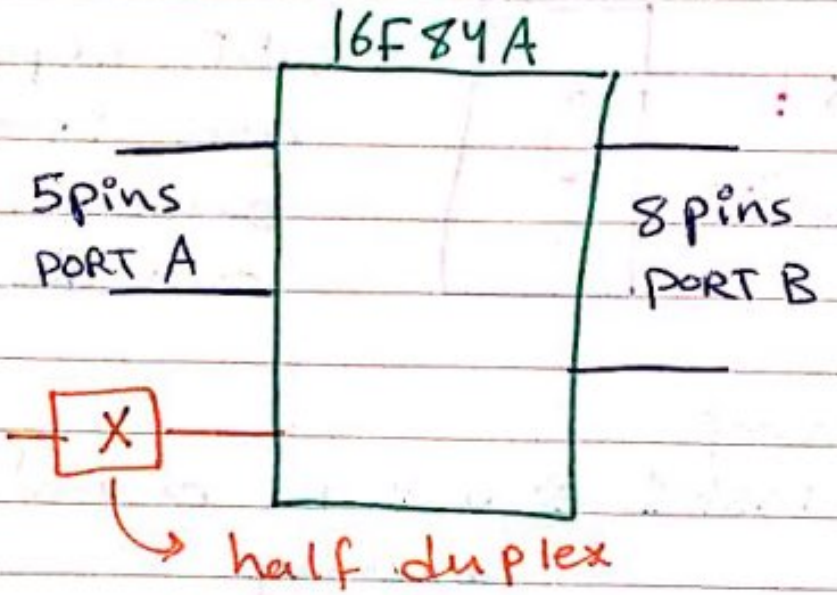
* Can I add some code after sleep inst to wakeup the M.C ?!

* wakeup in 3 cases:-

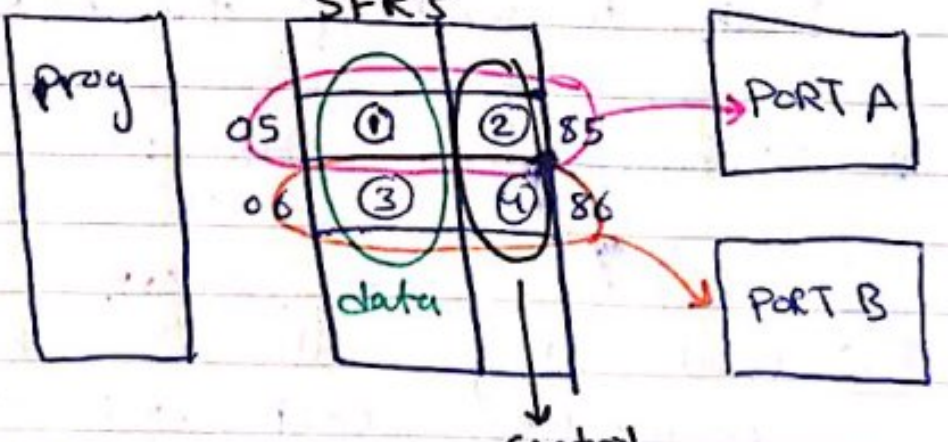
1. WDT → PC ← Zero
2. local flag while its enable = 1 regardless GIE. ⇒ if GIE = 0 → PC
   if GIE = 1 → 04
3. $\overline{MCLR} = 0$
   ← Zero ← 0

* If WDTE = 1, the maximum sleep time ≤ 18ms * WDT prescale. ≤ 2.3 seconds

* Chapter 3 84



Memory meipped I/Os



① ≡ PORT A
② ≡ TRIS A
③ ≡ PORT B
④ ≡ TRIS B

control to identify each pin whether it is input or output.

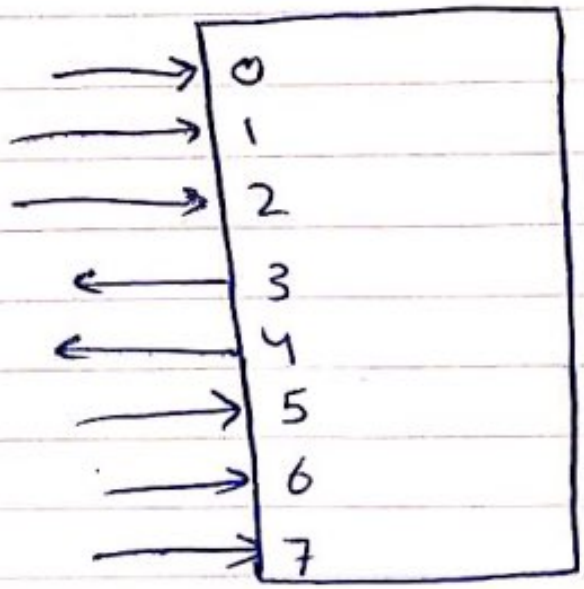
if tris bit = 0 ⇒ the corresponding pins is output

bit = 1 ⇒ input

* all pins are independent of each other.

slide 7:-

Ex:



~~MOVLW 0x00~~

```
BSF    STATUS,RP0
MOVLW  B'11100111'
MOVWF  TRISB
```

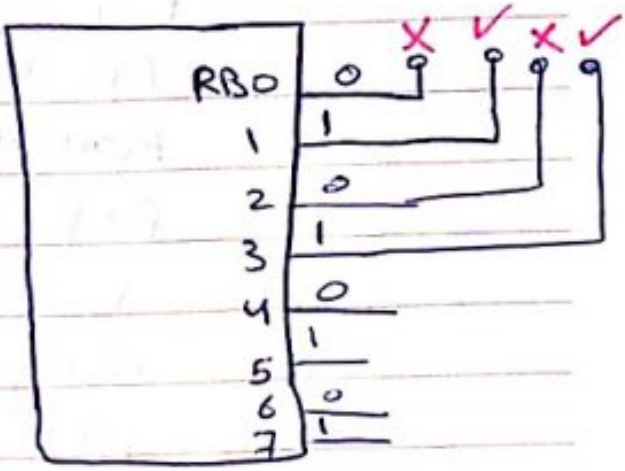
Slide 8:

Ex 1

```
MOVLW 0X00   }
MOVWF TRISB  }  → CLRF TRISB
```

⇒

```
BSF   STATUS, RP0
CLRF  TRISB   → b1
BCF   STATUS, RP0
MOVLW 0XAA
MOVWF PORTB   → b0
```



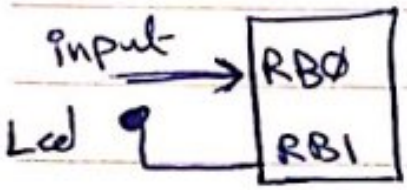
Ex 2

```
BSF   STATUS, RP0
MOVLW 0XFF
MOVWF TRISA
BCF   STATUS, RP0
MOVF  PORTA, 0
MOVWF 0X0D
```

Slide 9:

Ex enable external interrupt.

- GIE = 1
- INTE = 1
- RPØ option-Reg (6) = 1 يلزم يكون input ← RBØ



delay

Fosc = 4MHZ

code لـ led يضوي ويطفي كل 1sec :

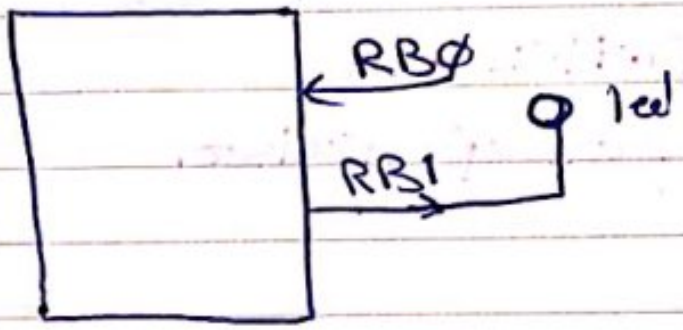
```
BSF     STATUS, RPØ
MOVLW   B'XXXX XX01'
MOVWF   TRISB
BCF     STATUS , RPØ
loop    BSF     PORT B, 1
        delay   1 sec.
        BCF     PORT B , 1
        delay   1 sec
        goto    loop
```

* variable

flash [ ][0] → not flashing

1 → flashing

```
ISR   MOVLW   0x01
      XORWF   flash,1
      BCF     ;INTCON,INTF
      ret fie.
```



CLRF PORTB ⇒ output ال بتاع كل

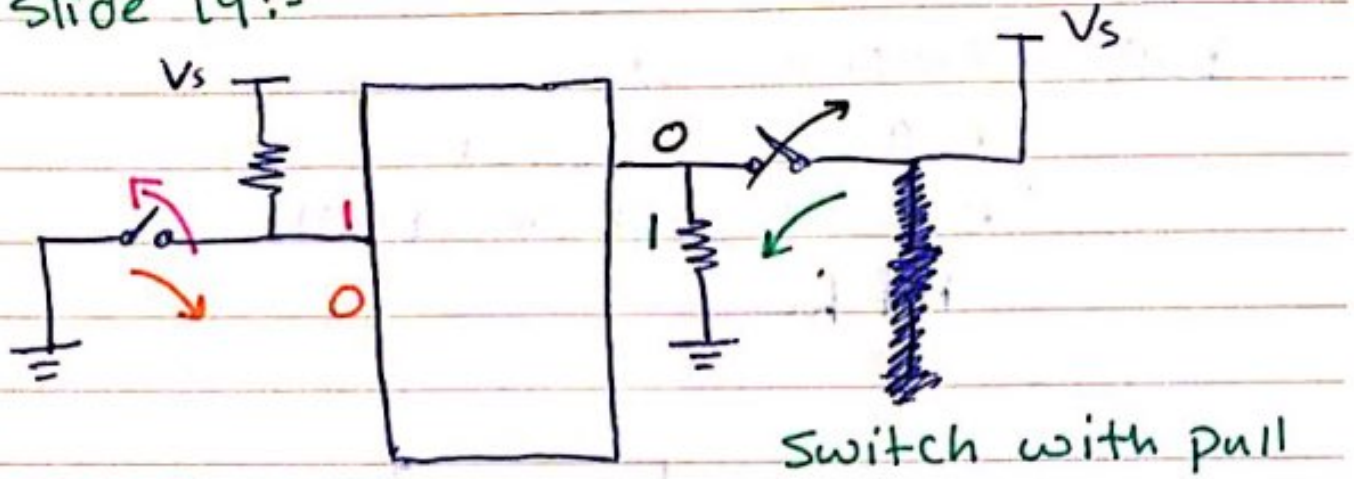
→ When you write PORTB or PORTA register the input pins will not be effected.

Hardware مش عارف

⇐ Hardware ال مطلوب من *

* PORTB Change interrupt flag to clear it
  - → MOVF PORTB,0
  - → BCF INTCON, RBIF

Slide 19:-



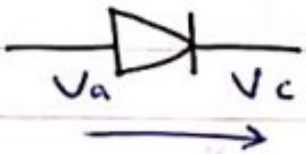
Switch with pull up resistor

Switch with pull down resistor.

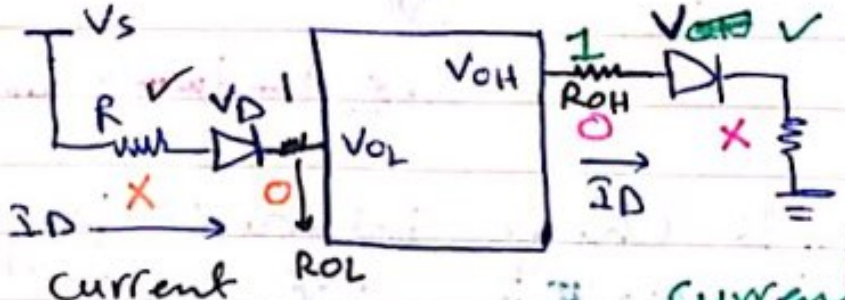
- current limiting
- to prevent floating Input.

* دائماً الـ Input بيكون switch

Slide 20:

LED: 

Led ⇒ output دائماً



current Sink

current Source

$V_S = R * I_D + V_D + V_{OL} + I_D R_{OL}$

$V_{OH} = V_D + R I_D + I_D R_{OH}$

current sink :-

$$R = \frac{V_S - V_D - V_{OL} - I_D R_{OL}}{I_D}$$

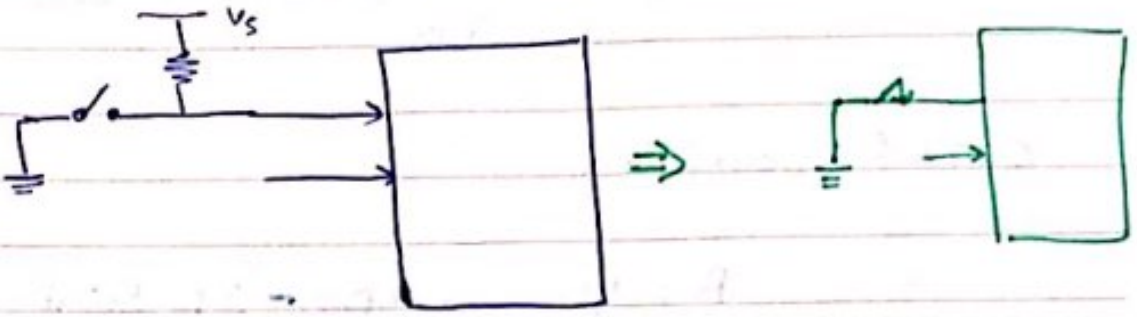
current source :-

$$R = \frac{V_{OH} - V_D - I_D R_{OH}}{I_D}.$$

* $R_{OH} = 130\ \Omega$
* $R_{OL} = 36\ \Omega$

13/8/
SUN

Port B change interrupt.

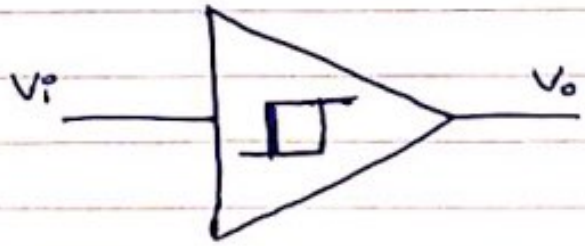


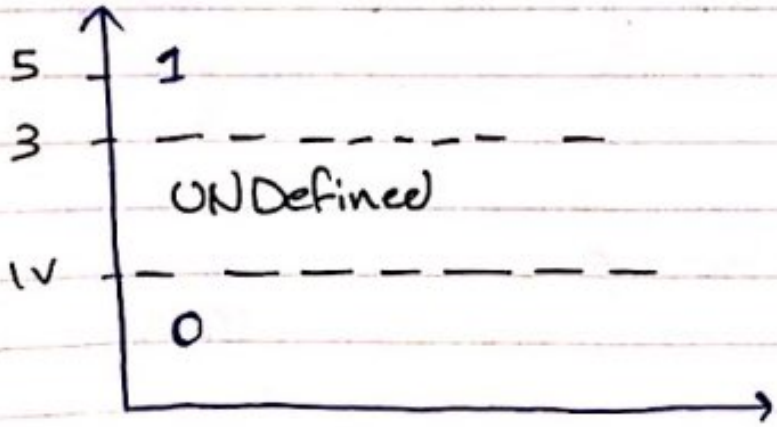
$\overline{RBP\emptyset}$

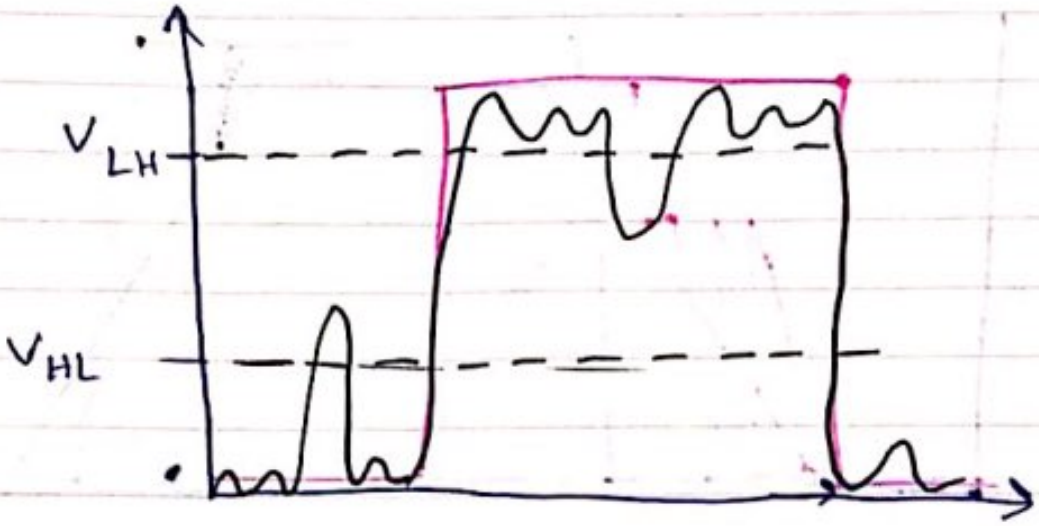
OPTION-REG(7) if =0

⇒ there is an internal pull up resistor.

Schmitt trigger:-





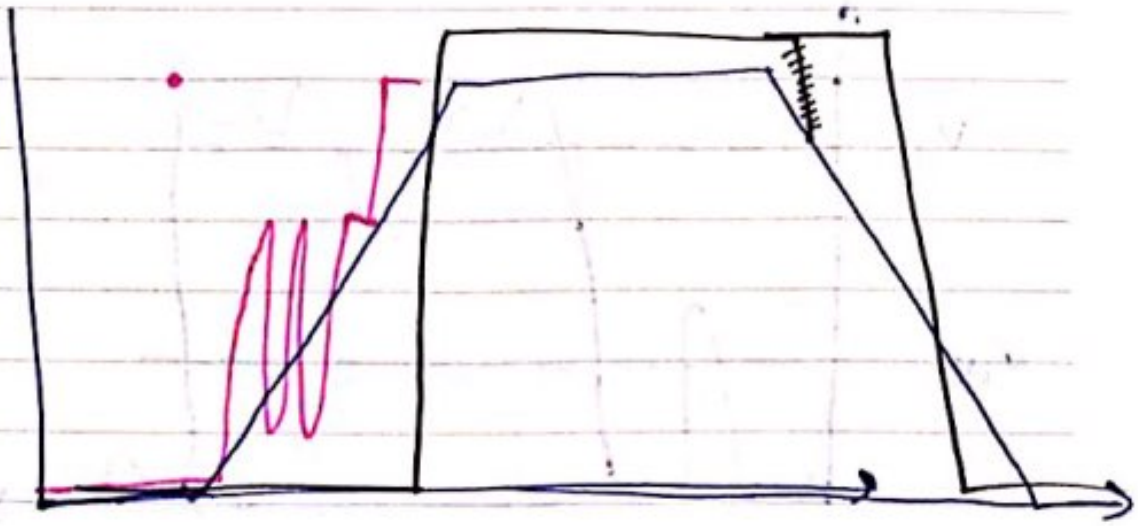


In schmitt trigger, there are 2 threshold values:

V_{LH} (↑): if intially the output is zero in order for the output to change into 1; the input voltage must go behind V_{LH}.

V_{HL}: if currently the output is high in order for the output to change into zero the input must go below V_{HL}

1. it filters the noise.
2. fast switching



*

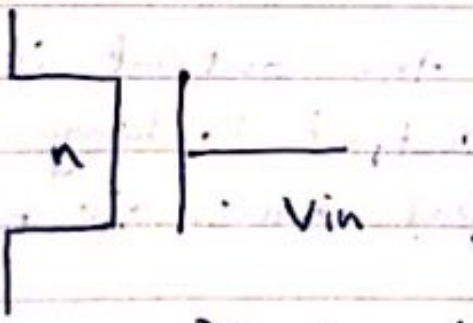
Slide 29:-

RC osc

Crystal osc

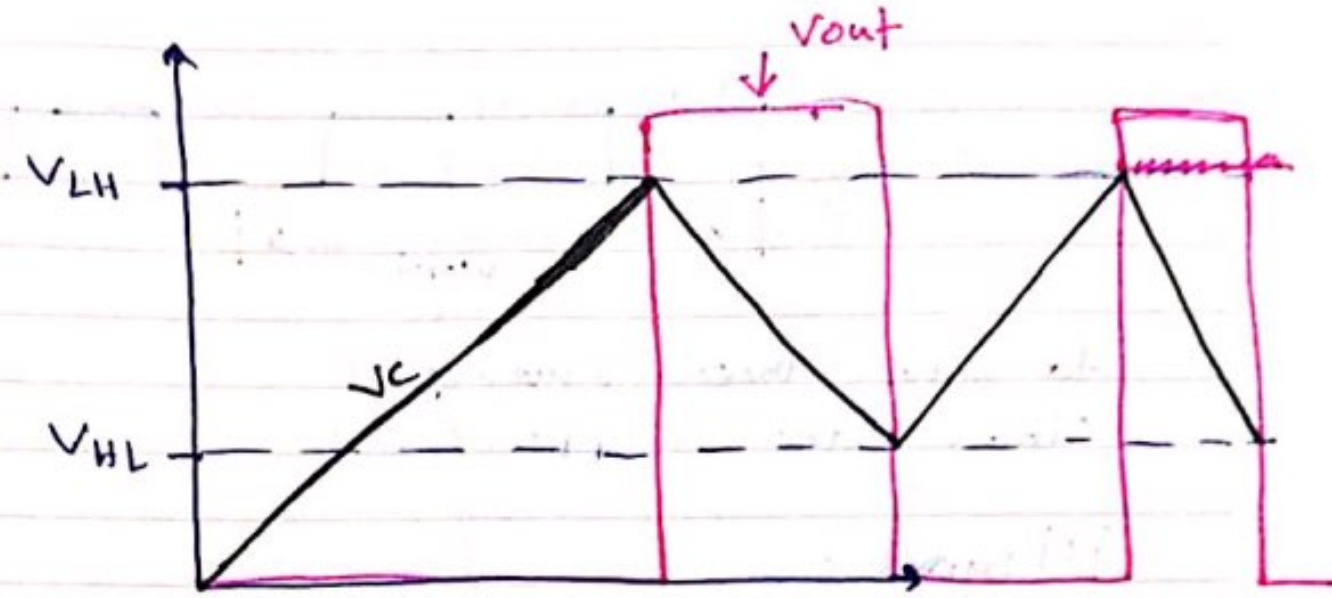
- LP
- HS
- XT

} بفرق عن بعض بـ freq range



if Vin = H ⇒ short ckt

if Vin = L ⇒ open ckt.



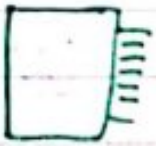
Vc بضل يشحن كد ما يوصل لـ V_{LH} . بعدين بعمل discharge ...

freq is dependant ⇒ R and C.

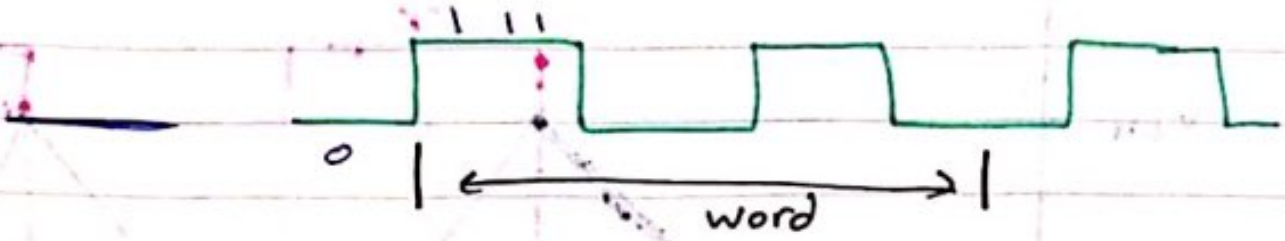
* Chapter 10:

cross line interference. ⇒ you can't send in high data rate and for long distances.

parallel: multiple bit at a time.
- faster.

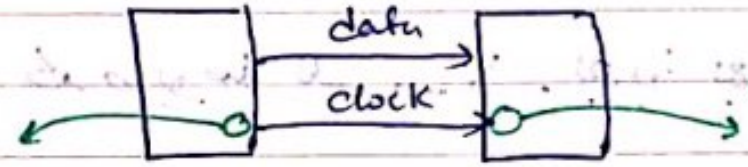


series:- one bit at a time.



to solve these chaleyens:-
There are 2 approaches:-

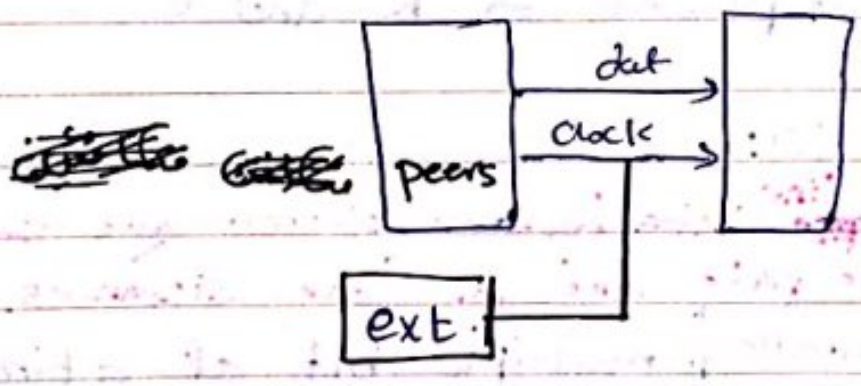
1 Synch :



إ بترسل master...

إ بتقبل slave

بترسل data ، clock.



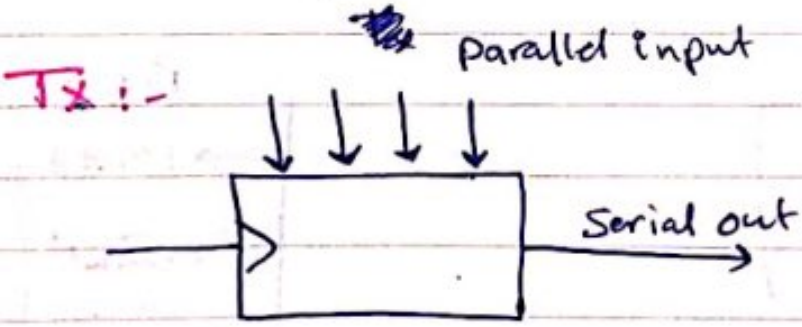
2 Asynch:-

Asynch:- for the receiver to be able to read the received signal, it must know the transmission protect:

- data value
- size of word
- start bit
- Stop bit
- parit
- encoding.

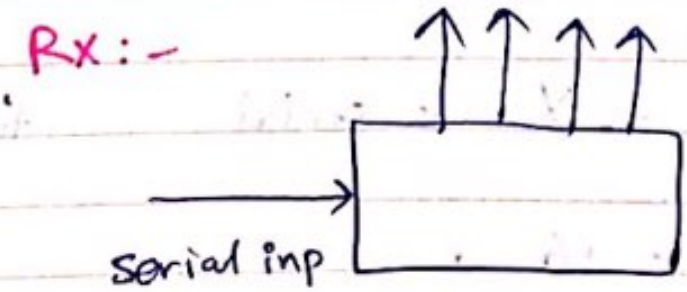
Slide 7:-

Serial port is shif registor:

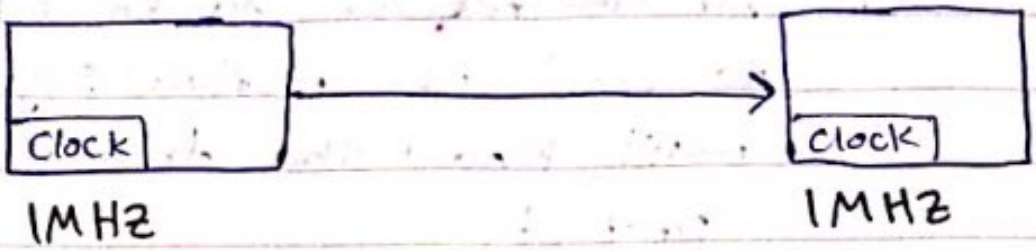


to inform me that transmission is over.

interrupt-

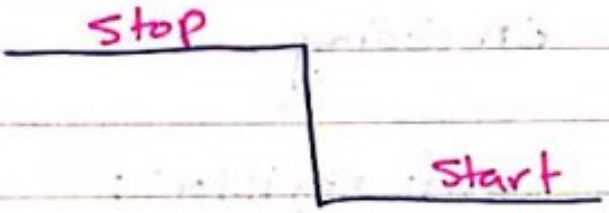


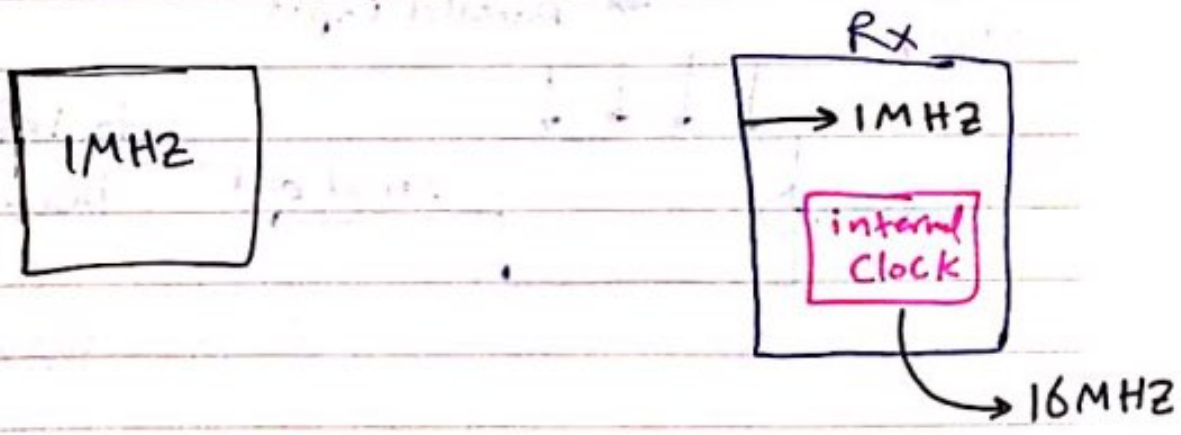
a method to tell me reception of byte.



Clock Skew

⇒ the reciver and sender must resychronous each word. ⇒ Stop bit, and start bit.





* internal Clock:-

clock freq = X * serial freq

in pic := 16 * "

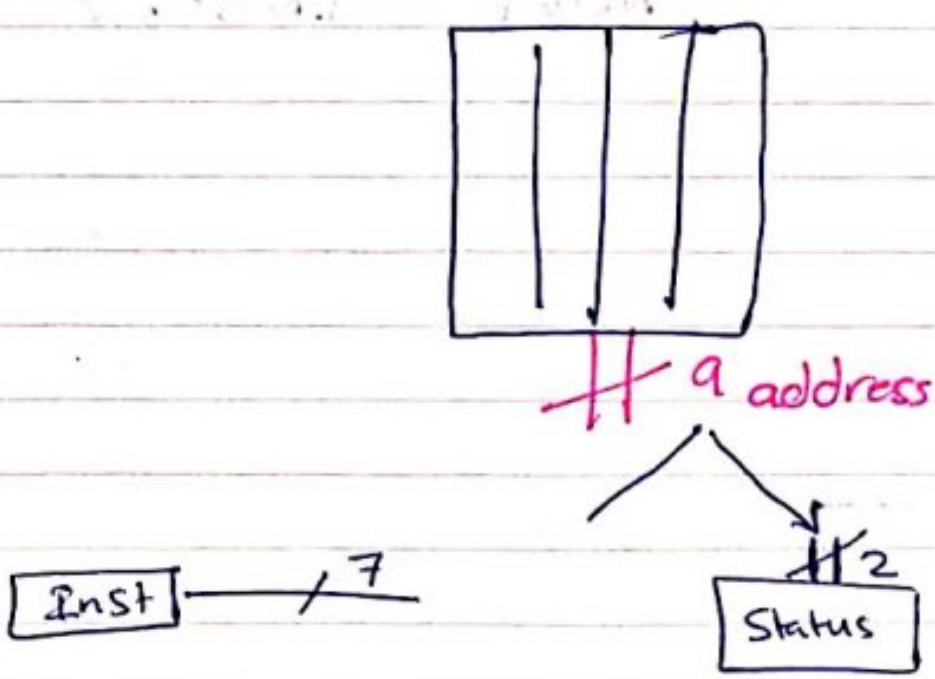
if the majourety of the collected samples = 0 ⇒ it is a start bit

and start receiving the data.

Slide 17:

16F87XA ⇒ have Serial port
↳ mid range.

4 banks ⇒ 2 bits.
↳ direct (RP1 and RP0)
↳ indirect (IRP and FSR(7))



write code to read address 0X185

1 1000 0101

⇒ b3

direct ⇒

```
bsf     Status , RP1
bsf     Status , RP0
MOV     05, 0
```

indirect : ⇒

```
BSF     STATUS, IRP
MOVLW   0X85
MOVWF   FSR
MOVF    INDF, 0
```

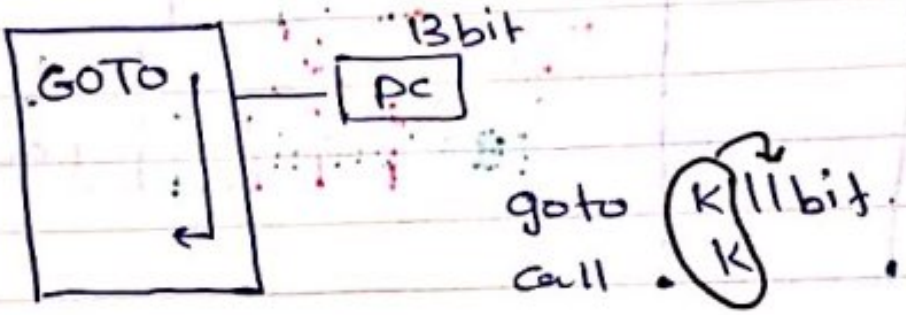
14/8/
MON

16F 87XA

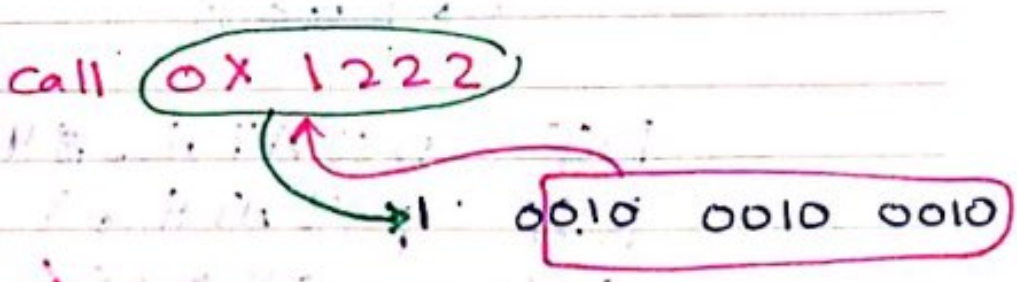
↳ program memory 8K

↳ need 13bit

$\lceil \log_2 8K \rceil$



Ex if currently the pc = 0005 and you want to call subroutine at address 0X 1222



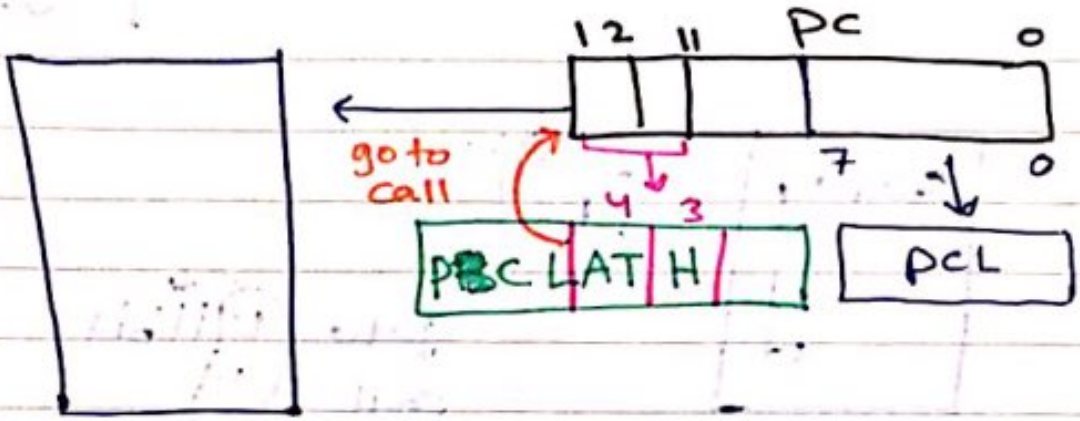
Call 0X 222

8 K

new program memory is divided into 4 pages each of size 2K

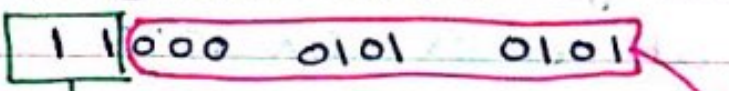
Select the page first then address inside the page

How to choose the page.



Ex: 0005

GO TO : 1855



→ page 3

```
BSF   PCLATH , 4
BSF   PCLATH , 3
goto      0X 0055
```

page selection ⇒ call , go to

bank selection ⇒ data memory

Slide 19

عشان يجي عندي interrupt تبع EEPROM

"الصفحة" فـ صار لازم يكون

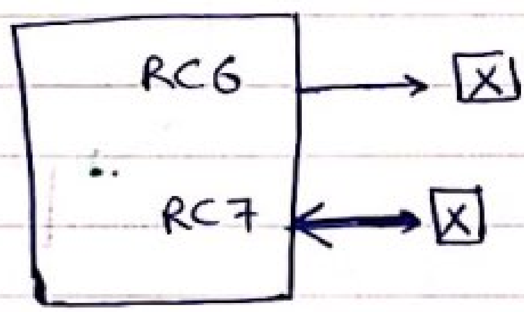
GIE = 1

EEPROM = 1

enable = 1

" = 1

Slide 21: Asynch. انا بدي أشتغل ع



PIE 1 → enable.

flag ← PIR 1

INTCON → GIE
→ PEIE

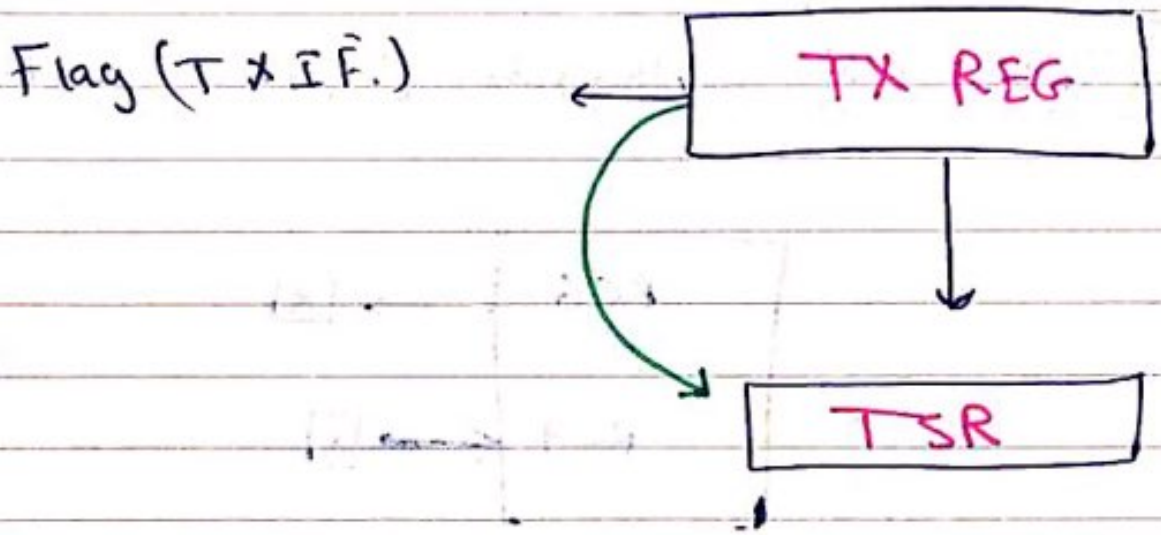
↓ over wrote EEIE

T.SR:-

is not writable or readable because it is not in the data memory.

→ However ~~TSR~~ TXREG has an address.

→ If TSR is empty, TXREG will be moved (not copied) to TSR.



* When TXREG Moves into TSR, TXIF = 1
  GIE = 1
  PEIE = 1
  TXIE = 1

* TXIF = 0 if TXREG is full

[set/cleared by HW ⇐ flag من النوع هاد]

Tx start if :-

[1] TXEN = 1 — اذا واحد منهم صفر ما بصير عندي ارسال. transmitted

[2] SPEN = 1 => for Tx and Rx

TRMT = 1 => last bit was sent

Baud rate = f (SPBRG (reg), BRGH (reg), Fosc)

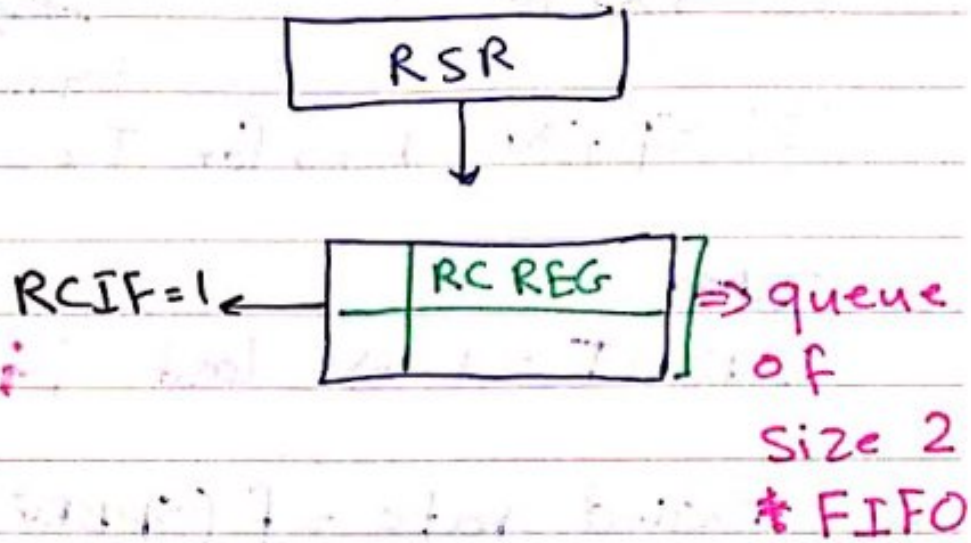
⇒ If TX9 = 1: send with each Byte additional bit that is stored in TX9D

* TXIF = 0
✗ BCF PIR1, TXIF ↓

ما بصفر لحالو ← by hardware

✓ MOVWF TXREG. ← بدي اصفر

slide 27



FIFO => first input first output.

OERR : over run error ⇒ if the stop bit of the third byte gets recived before reading the previous 2 byte.

→ 1) reception Stops

→ 2) 3rd Byte will be loss

* How to prevent it ?!
: interrupt يحدث في

* How to solve it ?!

RCIF = 1 when there is at least one byte in RCREG.

GIE = 1 ; PEIE = 1 , RCIE = 1

* بدي أقرأ أول byte ⇐ MovF RCREG, Ø

RCIF ⇒ = Ø when RCREG is empty

↳ set / cleared by HW

* Reception has 2 enable :-

1. CREN = 1
2. SPEN = 1 for Tx and Rx

OERR = 1 ⇒ reception is Stopped

↳ read only.

↳ to solve it :-

- read the 2 byte ⇒ RCIF = Ø
  Clear CREN ⇒ OERR = Ø

↳ انا هيك وقفت الاستقبال.

set CREN ⇝ receive again

FERR = 1 → when the recived stop bit = 0
→ reception continous.

FERR . and . RX9D are double bufferes.

لذلك لازم اقرئهم قبل الـ RCREG

you should.. read then before the RCREG (word).

16/8/
Wed

slide 32:

Band rate = F(fosc, BRGH, SPBRG)

slide 33

Tx → 3 bytes
40
41
42
8 bits
9.6 Kbps.

Fosc = 20MHz

$$\text{band rate} = \frac{\text{Fosc}}{16 * (1 + \text{SPBRG})}$$

BRGH = 1 ⇐ 16 * (1+ SPBRG)
BRGH = 0 ⇐ 64

$$9.6 \times 10^3 = \frac{20 \times 10^6}{16 * (1 + \text{SPBRG})}$$

$$\frac{20 \times 10^6}{9.6 \times 10^3 * 16} - 1 = \text{SPBRG}$$

SPBRG = 128 ⇒ BRGH = 1

BRGH = 0

SPBRG = 31

TXSTA = X 0 1 0 X 0 X X

TXREG = value.

RXSTA = 1 X X X X X X X

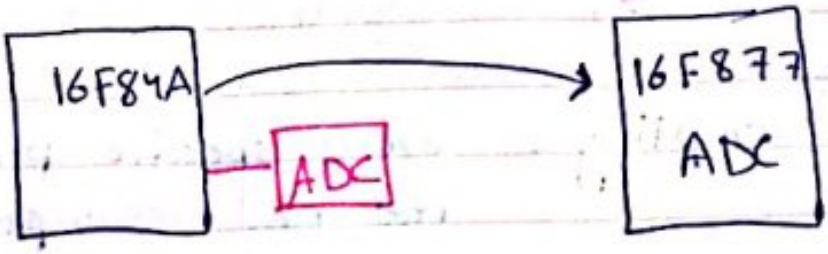
RCREG = X

SPBRG = 128

* Chapter 11 :

* Digital : → takes limited number of values in a specific rang
→ discrete in time in specific time the change may happen.

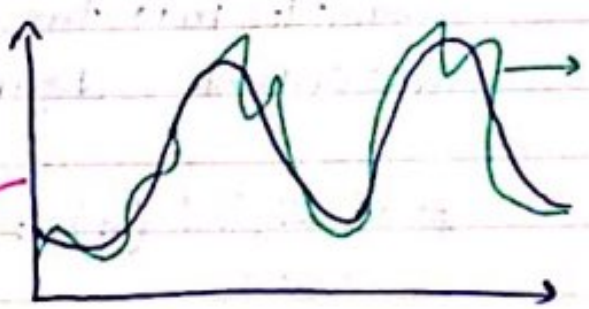
* Analog : → takes unlimited number of values. in aspecific rang.
→ continous in time.

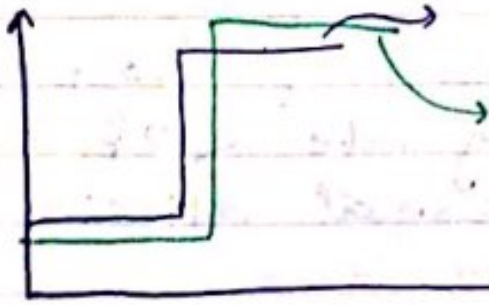


A → loss / error → D

More accurate

Max acceptable error





* بقدر ارجع الـ signal للاصلي

we cant benifit from its accuracy when the data is sent or stored and retrived.

A ⟶ D

→ sampling :- every specific period read a sample of the analog input.

sampling rate = $\frac{1}{\text{period}}$

↑ → ⊕ more accurate.

⊖ time , ⊖ cost , ⊖ storage , ⊖ power

→ Quantization :- convert the read sample into the closest acceptable digital value

# of digital level (n) ⇒ 2^n level.

Sampl⇒ ↓ sampling rate ≥ 2 * fmax

if sampling rate < 2f

⇒ aliasing

discrete in time

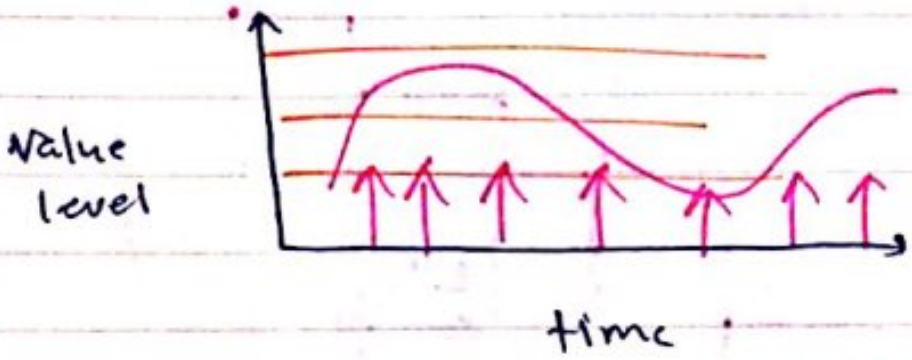
* Quantization ⇒

* 10 bits ADC ⇒ result is represented in 10 bits' value ⇒ 1024 possible digital value.

$n\uparrow \Rightarrow$ (+) accuracy

(−) cost , (−) power , (−) time

$\Rightarrow$ discrete

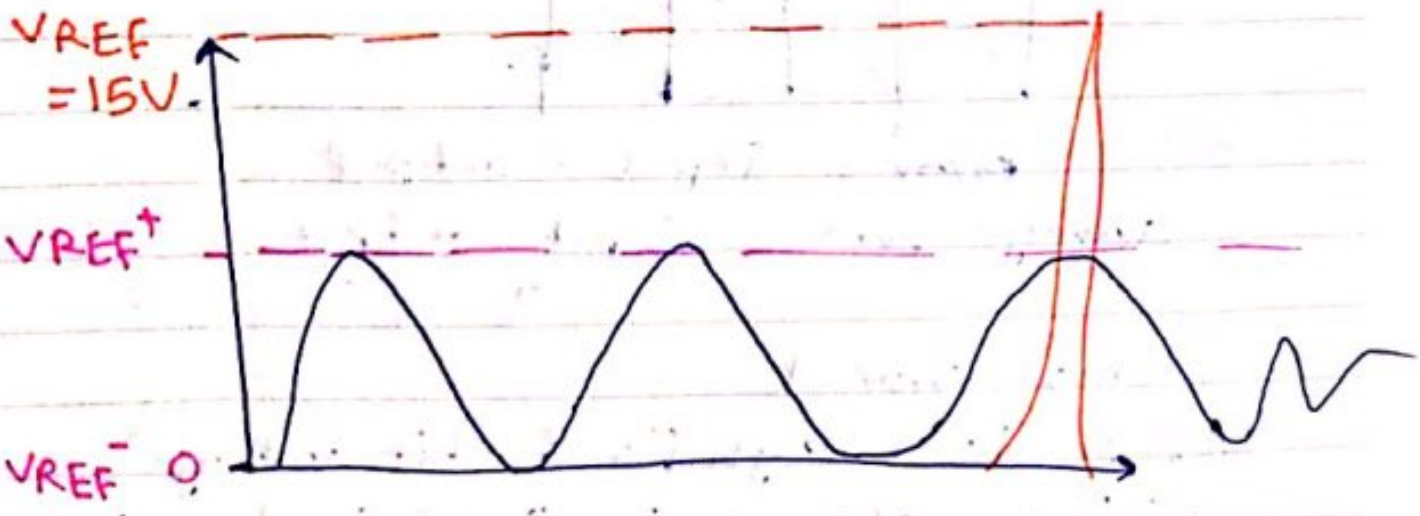


*Voltage reference :-

the range of acceptable analog input

$V_{REF} \Rightarrow V_{REF^+} = V_{REF}$

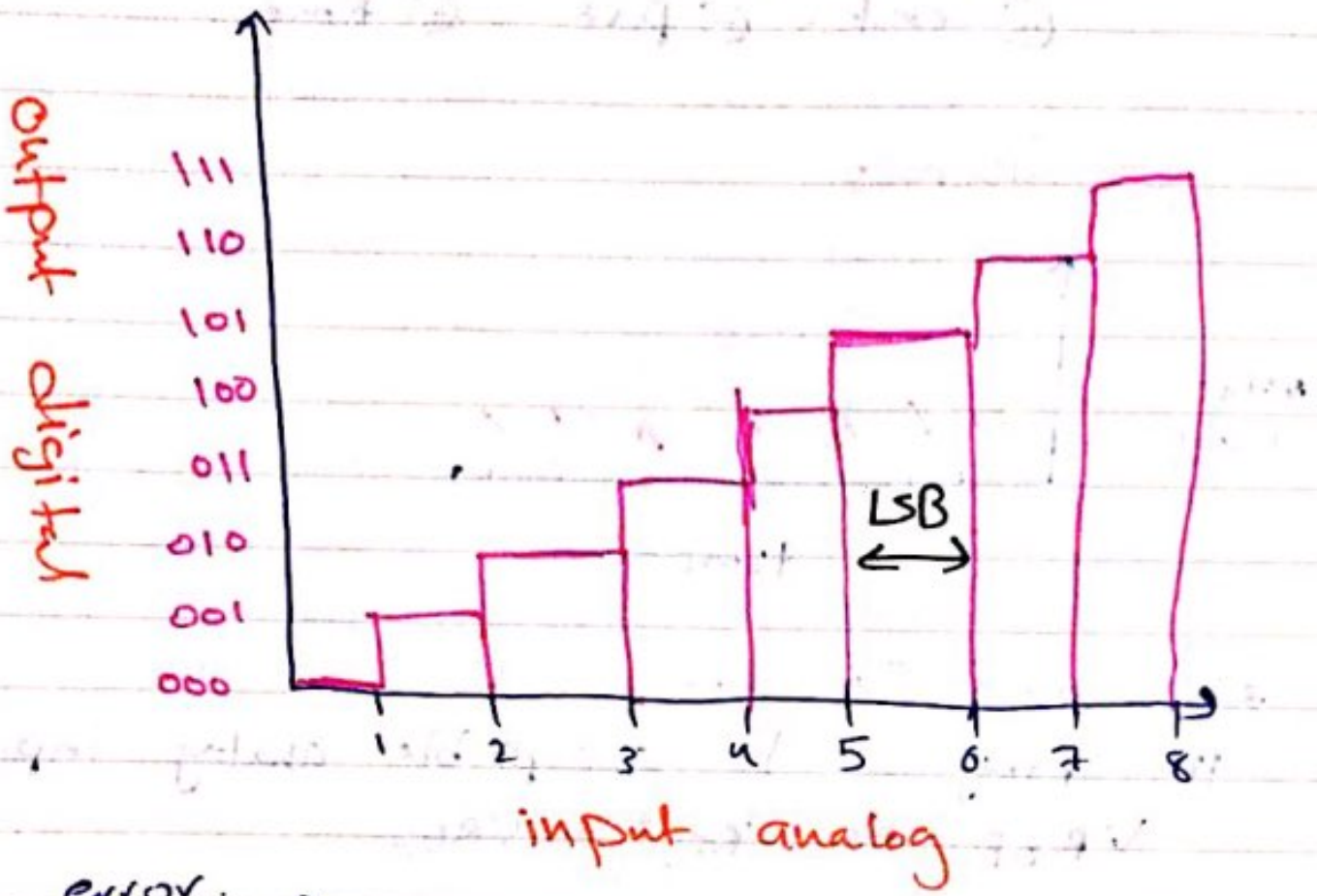
$V_{REF^-} = \emptyset$



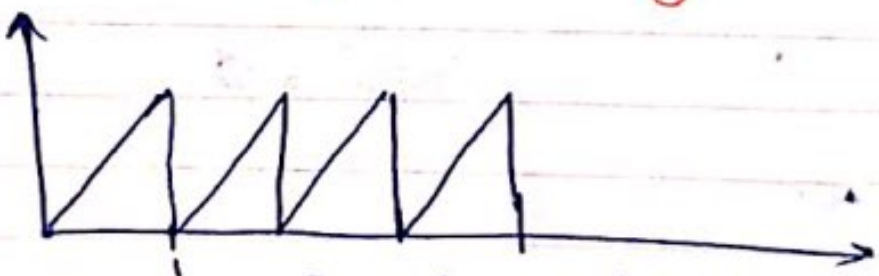
Ex $V_{REF^+} = 8V$, $V_{REF^-} = \emptyset V$

$n = 3$ bit

Quantization characteristic graph :-



error.



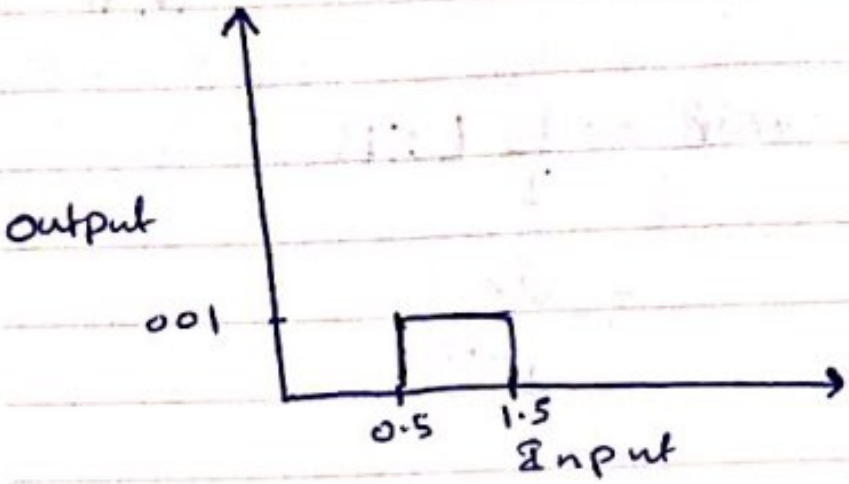
error = input - output.

average error = $\frac{\text{input - output}}{2}$. $= \frac{1-0}{2} = \frac{1}{2}$

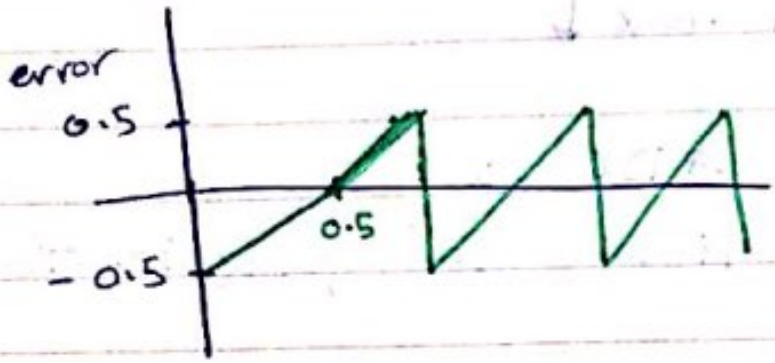
max error = 1

LSB :- "least significant bit voltage"
minimum change in input level voltage that will definitly change the output

output = $\frac{V_{REF}^{+} - V_{REF}^{-}}{2^n}$



0.5 → 1
1.5 → 1
1 → 1



maximum error = $\frac{1}{2}$ LSB

$= \frac{1}{2} * \frac{Vr}{2^n}$

$$\text{max. error} = \frac{Vr}{2^{n+1}}$$ where

$Vr = V_{REF} - V_{REF}$

17/8/
Thur

* max acceptable error = $\frac{1}{2}$ LSB

$= \frac{Vr}{2^{n+1}}$

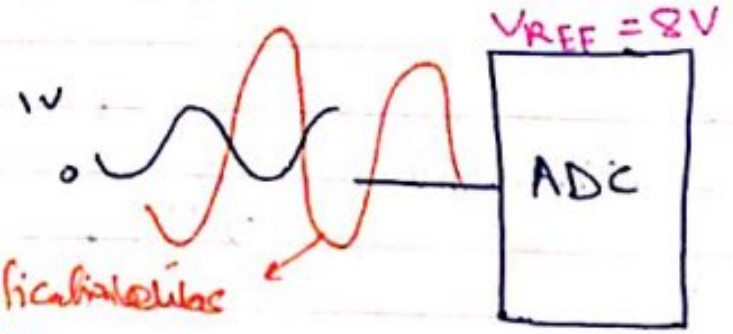
$\uparrow n \rightarrow$ max $\downarrow$

Slide 11

more accurate ADC $\Rightarrow$

Sampling time $\uparrow$ $\Rightarrow$ $\uparrow$ time and quantization time $\uparrow$.

There is a trade off btween accuracy and speed.



* $V_{REF}^{-} = -2.5V$

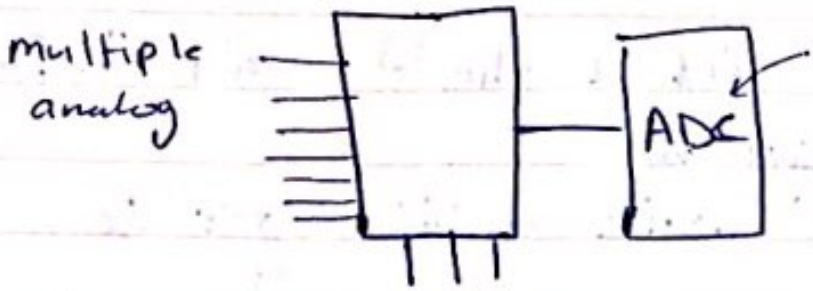
$V_{REF}^{+} = 2.5V$

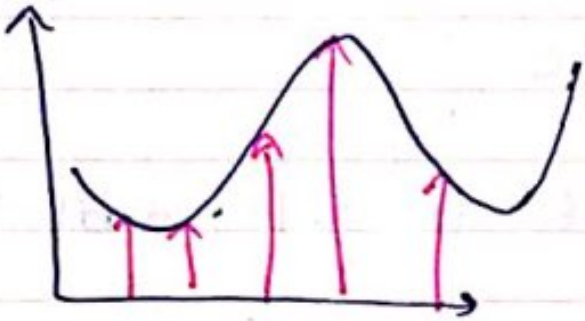
$\Rightarrow$ shifted wala

3 bit ADC

$\frac{Vr}{2^n} = \frac{8}{2^3} = 1V$

132





If the sample was collected while the input is changing ⇒ there will be an error.

Solution ⇒ ① hold the input ② take the sample ③ relase the input.

slide 14:

as a switch is closed, Vc follows Vs

* when the switch is open ⇒ Vc is almost constant.

⇒ close the switch until $V_o = V_s$
⇒ open the SW
⇒ take sample
⇒ convert sample.

for how long I should wait before open the switch.

⇒ I should wait until $V_o \simeq V_s$

$$V_o = V_s * (1 - e^{-t/\tau})$$

$$\tau = R.C$$

Ex If I want to wait until $V_o = 0.9 V_s$

sol: ⇒ $0.9 V_s = V_s (1 - e^{-t/\tau})$

$$t = -\ln 0.1 * \tau$$

$$t = 2.3\tau$$

$$\text{error} = \frac{V_s - 0.9V_s}{V_s} * 100\% = 10\%$$

max error = $\frac{1}{2}$ LSB ⇒ الخطأ المقبول

$V_o = V_{in} - \frac{V_{in}}{2^{n+1}}$ ← V_o التي تعرفني

$\frac{V_{in}}{2^{n+1}}$ LSB

$$V_{in} * (1 - \frac{1}{2^{n+1}}) = V_{in} * (1 - e^{-t/\tau})$$

$$e^{-t/\tau} = \frac{1}{2^{n+1}}$$

$$\frac{-t}{\tau} = \ln \frac{1}{2^{n+1}}$$

$$t = -\ln \frac{1}{2^{n+1}} * \tau$$

↑ wait this time then open the SW

↑ n ⟶ ↑ sampling time.

* سؤال الفيديو " $V_o = 0.7 V_{in}$ ⇐ error = 30% الـ

if n = 8 bit ⇒ time = ?!

$$t = -\ln \frac{1}{2^9} * \tau$$

$$t = 6.2\,\tau$$

if n = 10 bit

$$t = 7.6\,\tau$$

↑ #o bit ⇒ ↓ error

Slide 19 8 possible analog inputs

- 5 → A: RA0, 1, 2, 3, 5
- 3 → E: RE0, 1, 2

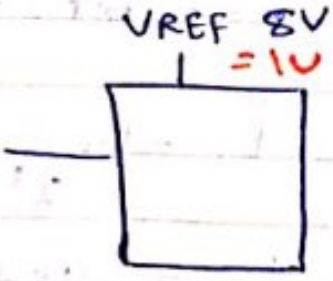
Ex If I want to convert an analog input connected to RE1, what should I put ON CHS ?!

sol ⇒ 110

TRS 1 ←
0 →

→ you should identify each of the pins where In or ~~out~~ out and if Input where A ~~or~~ or D PCFG 3,0

* you can connect voltage refrence externally on RA3 and RA2



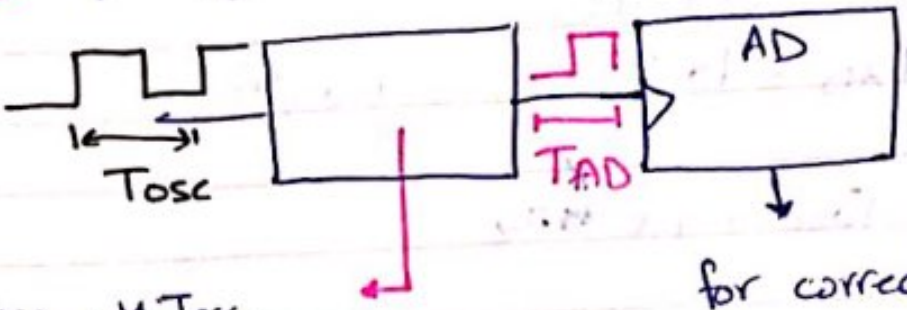
(PCFG)

- → external: RA3 and RA2
- → Internal: V_{DD} and V_{SS}

* have 2 regesitor ⇒ 10 bit

136

Slide 21 : ① switch on
② specify Quantization speed



2*Tosc , 4 Tosc
8 Tosc , 16 Tosc
32 Tosc , 64 Tosc

for correct quantization
TAD should be ≥ 1.6Ms

Rc with TAD = 2Ms

* for 10 bits Quantization
→ ADC needs 12 TAD
→ 8 bits ⇒ 10 TAD

larger n↑ ⇒ ts ↑
tQ ↑
error ↓

Ex If Fosc = 1 MHz , what is the fastest Quantization ?!

Sol: $Tosc = \frac{1}{1MHz} = 1Ms$

$TAD = 2Tosc = 2Ms$

fast Quantization ⇒ $12TAD = 12 * 2Ms$
$= 24Ms$

Ex If $F_{osc} = 10\,MHz$

Sol: $T_{osc} = 0.1\,\mu s$

$T_{AD} = 16\,T_{osc} = 1.6\,\mu s$

Ex If $F_{osc} = 100\,KHz$

Sol: $T_{osc} = 10\,\mu s$

$RC \Rightarrow T_{AD} = 2\,\mu s$

* اذا كان Freq أقل من $500\,KHz$ يتم اختيار RC

slide 24 5) $G/\overline{D} = 1 \Rightarrow$ Start Quantization

before this step you should wait for sampling time.

(open switch)

$G/\overline{D} = 0 \Rightarrow$ when Quantization is over.

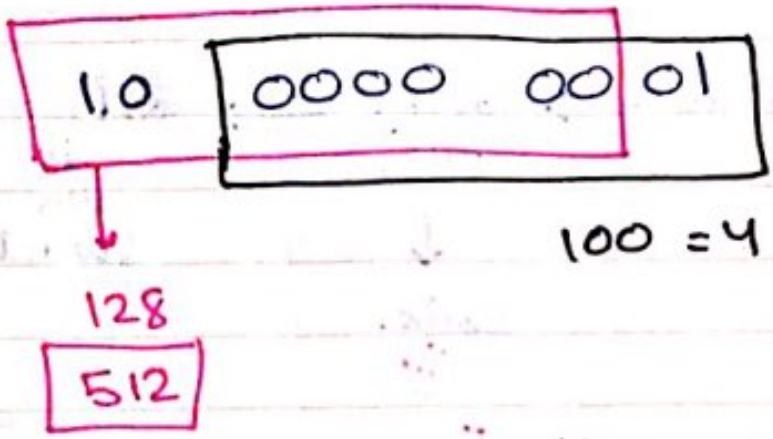
6) read the result

result is → 1) left justified

2) right " "أفضل"

*left justified ⇒ If I want to deal with 8 bit result.

Ex



* $G/\bar{D} \Rightarrow 0 \Rightarrow ADIF = 1$
  - GIE = 1
  - ADIE = 1
  - PEIE = 1

* Configure : ADC ① (1 → 4 steps) →

② then wait sampling time ($-\ln \frac{1}{2^{n+1}} * \tau$)

→ ③ $G/\bar{D} = 1$ → ④ wait 12TAD →

⑤ read result.
  - $G/\bar{D} = 0$, ADIF = 1

* wait sampling ⇒ I have to write code for it.
  - we will find more accurate formula for ts

* wait 12TAD → you don't have to write code for it.

slide 25:-

configration (1-4). → delay 7.6*τ → code. → GIDN = 1 (open switch) → wait 12 T_{AD} (No code) → read result. $\left[-\ln\frac{1}{2^{n+1}} * \tau\right]$

"GIDN = 0
ADIF = 1
↳ GIE = 1
ADIF = 1
PEIE = 1

Ex What is the value of PCFG bits if I want to connect single analog input and external voltage refrance ?!

sol: 1111

C / R
↳ # of external voltage refrance.
↳ # of analog input.

C/R = 1/2 ⇒ 1111

$(-\ln\frac{1}{2^{n+1}} * \tau)$ time ⇒

~~opdap~~

op Amp Settling time (2 µs) + Temp Coef + 7.6 * (R_S + R_{IC} + R_{SS}) * C_{HOLD}.

R_{SS} → dependent on

Temp Coef → $|T-25| * 0.05\ \mu s$

إذا كانت أقل من $T<25$ ⇐ خليه $T=0$.

Q Final 7.6
- 10 bit
- error → 0.5 LSB

Ex

$R_{SS} = 7K\Omega$, $R_{IC} = 1K\Omega$, $R_S = 0$

Temp = 35 °C, $T_{AD} = 1.6\ \mu s$, $t_{op\_amp} = 2\ \mu s$

$C_{Hold} = 120\ pF$

$$T_S = \text{op amp set} + \text{Temp coef} + 7.6 * \tau$$
$$= 2\times10^{-6} + (35-25)\times0.05\times10^{-6} + 7.6\,(7\times10^{3} + 1\times10^{3} + 0) * 120\times10^{-12}$$
$$= 9.8\ \mu s$$

$$T_Q = 12\,T_{AD} = 12 * 1.6$$
$$= 19.2\ \mu s \text{ for one sample}$$

sampling time	Quantization	enter Sample time ⇒ $2\,T_{AD}$
9.8 µs	19.2 µs	3.2 µs

Total: 32.2 µs

for one sample only we need.

$= 19.2 + 9.8 = 29\ \mu s$

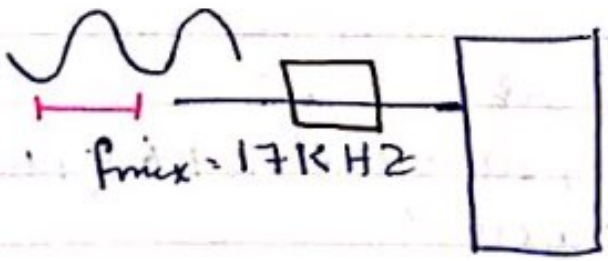
for repetive sampling, one sample required

$= 29 + 2 * 1.6$

$= 32.2\ \mu s$

Sampling rate $= \frac{1}{32.2\ \mu s} = 31\ KHz.$

max input freq $= 15.5\ KHz = \frac{1}{2}$ sampling rate



"10 bit → 12 cycle"
Quantization

10 bits result $\frac{Vr}{2^{n+1}}$

6 bit Quantization
- → 8 cycle with $T_{AD} = 1.6\ \mu s$
- → 4 cycle with T_{AD} minimum

Ex fosc = 10 MHz.

↳ Tosc = 0.1 Ms

→ TAD = 16 Tosc = 1.6 Ms

* 8 cycle with 1.6 Ms = TAD
* 4 cycle with TAD = 0.2 Ms

⇒ Quantization times = 8 * 1.6 + 4 * 0.2

= 13.2 Ms

slide 33 RA∅

FAD = Fosc / 8 , TAD = 8 Tosc

internal voltage refrence , Fosc = 20MHz

VDD = 5V , Temp = 25° , right justifi

Sol: sampling time = 2 + 7.6 * ...

= 10 Ms

ADCON∅ 01 000 ∅X1

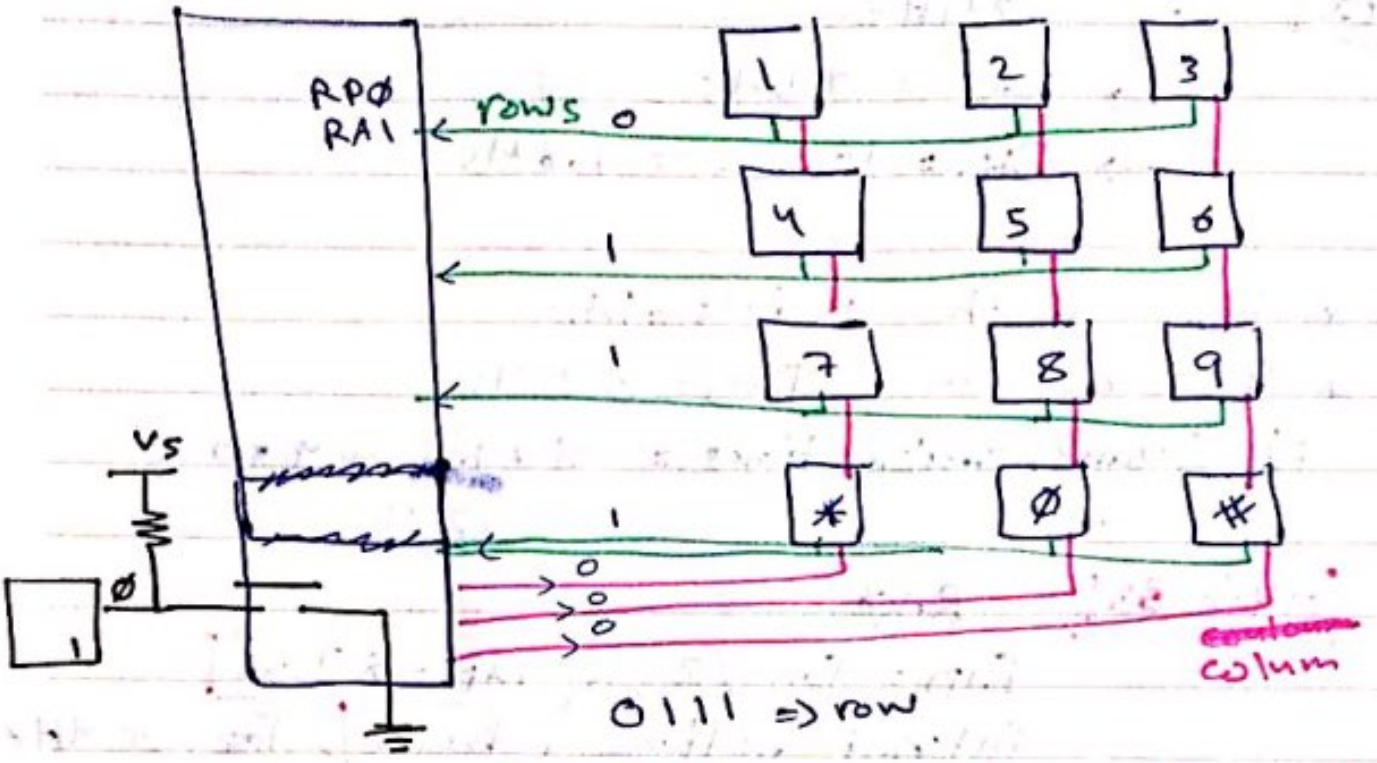
ADCON1 1 0 X X 1 1 1 0

Chapter 8:

Key pad ⇒ مجموعة push bottom مشبكة على شكل row and coulum

* شبك كل كبسة على pin ⇐ 12 pin

slide 7:-



12 → 7 Save pins ⇒ code more complex

30 → 11

To read which push button was pressed :-

[1]. read row pins.

a) make row pins input.

b) " colum pins output.

c) output zero on colum pins.

[2] read colum pins

a) make row pins output

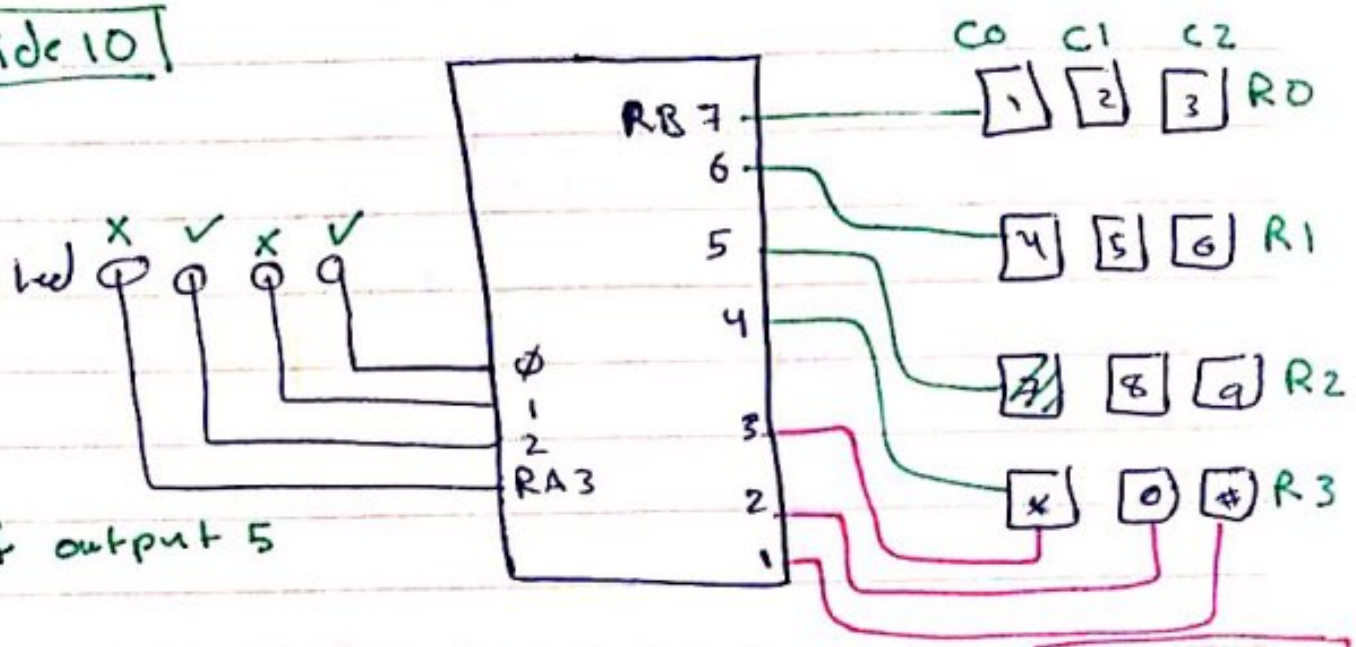
b) " col " input

c) output zero on row pins.

Convert into number ← 0111 → row / 0111 → colum

144

Slide 10



3x3+2=11

Col Index = 0000 0110 = 0

Row Index = 1101 0000 = 2

W = Ø 2 6

3 * row Index + Coulum Index

* اذا كان سين Inter انك ضمت ايدك ⇐ أكبر من ١١.

21/8/
MON

* at pull down → 3 — 0
2 — 0
1 — 0

* كيف إذا كنت بتقول 0 والباقي كلو 1 !

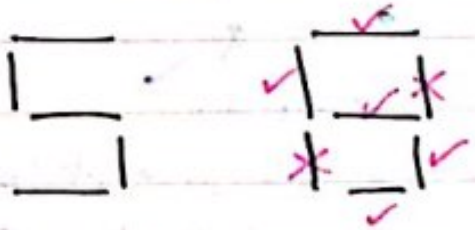
* Port B change ← ما بيغير غير لما row bit ~ إذا كان output

Interrupt.

slide 16

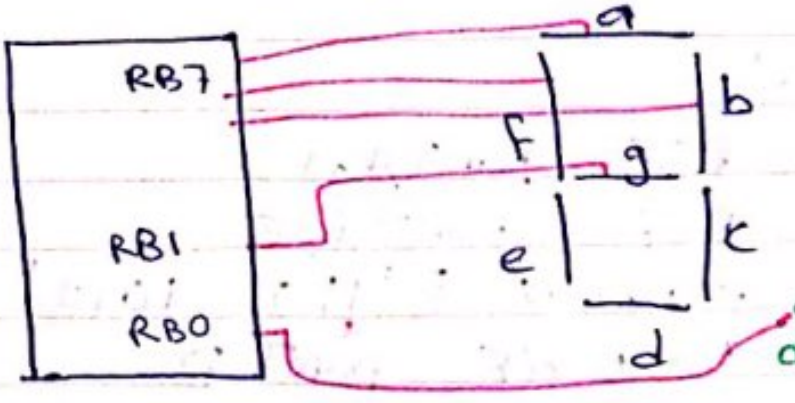
⇒ seven segment.

common anode:-
all anodes of the 7 Segment are connected together to Vcc, to turn on any Segment, we connect it to ground.



→ 14 SD or Star burst.

need 8 pins



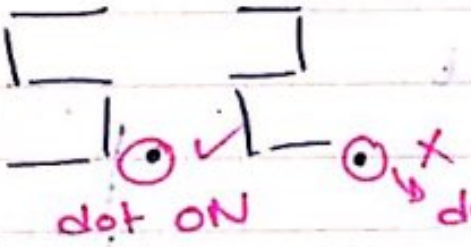
146

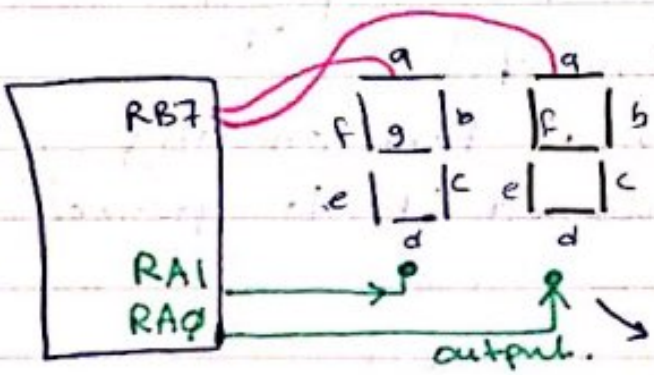
```
CLRF    TRISB
BS
MOVLW   B '10110110'
```

a ... dot

```
MOVWF   PORTB.
```

* "شابك الـ a على RB7 ممكن بغير التشبك".





* لو شبك كتابين
  طبعاً عندي جواب 55
  انا بدي طلع 52 .
  * بضيف control

بدي اطلع متى 25 ⇐ بخلي الاول يضوي وطلع 52 وطفي الثاني وبعدين بدي ثاني وطفي الاول وطلع 5.

→ code ;-

```
loop    BSF   PORTA,1
        BCF   PORTA,0
        MOVLW B'1101 1010'
        MOVWF PORTB
```

a

147

Call delay 5 ms → يعني ما بشوف لما يضوي ويطفي .

* في ثانية وحدة بضوي ١٠٠ مرة فما بحس فيها .

```
BCF PORTA, 1
BSF PORTA, 0
MOVLW B'11011010'
MOVWF PORTB
call delay 5ms

GOTO loop
```

* بدي اعرض فيه 25 عدة ثانية ، وهك ⇐ بضيف counter.

تعديل ع code ←

```
      MOVLW D'100'
      MOVWF counter
loop  BSF PORTA, 1
        ⋮
      call delay 5ms
   →  DECFSZ counter, 1
```

* slide 21

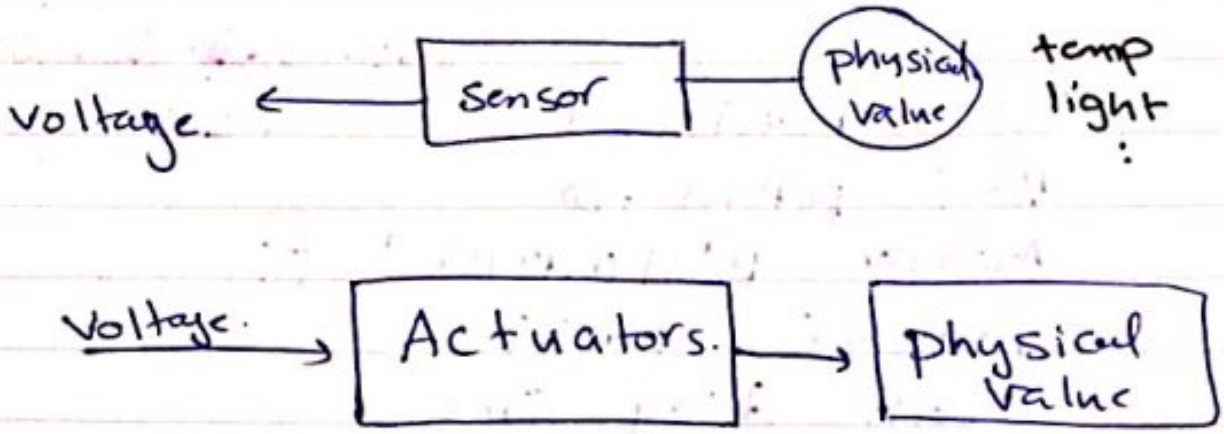
Example

لـ تم أو كانت

00 ↓ 9 → 9 لما يوصل

بعديها بصير 0

Slide 26. Sensors :-



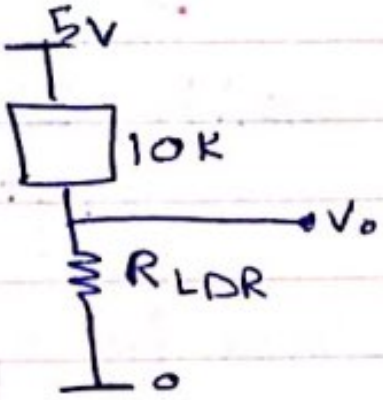
Slide 28 LDR 8

↳ resistor, its resistance value is dependent on light intensity.

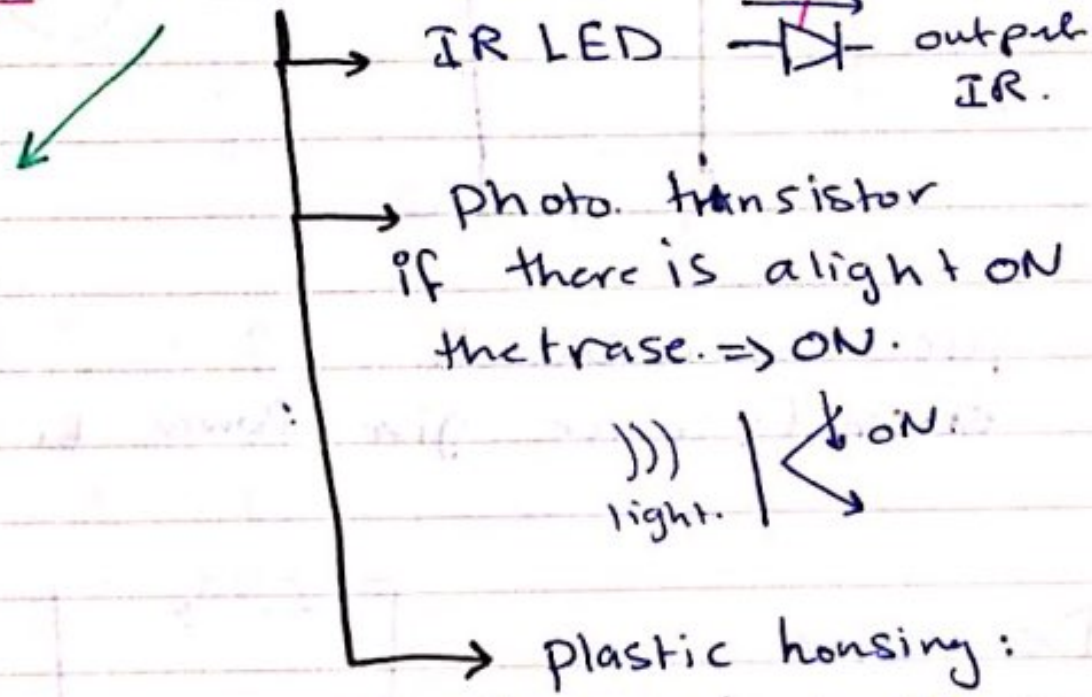
→ as light intensity increase ⇒ more electron-hole pairs are formulated ⇒

conductivity increase.
⇒ resistance decrase.

$$V_o = 5V * \frac{R_{LDR}}{R_{LDR} + 10K}$$



Slide 29: OOS

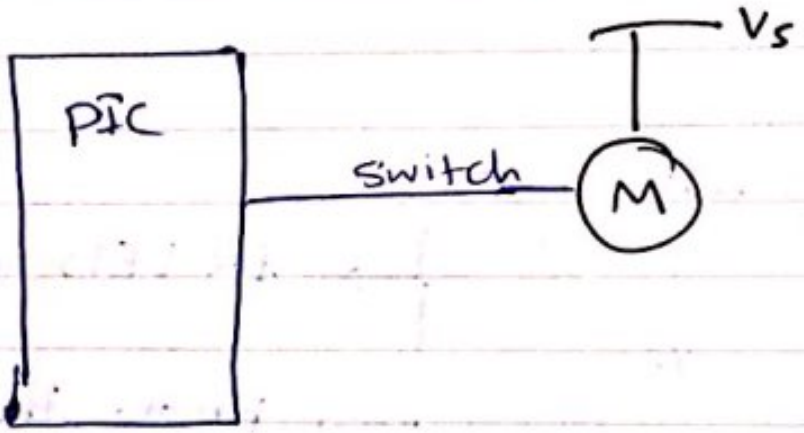


slide 31: Ultrasonic :- "analog" to find the distance to an object.

$$distance = \frac{time * Speed\ of\ voice}{2}$$

slide : 32: ~~Actuators~~

Actuators : " output "

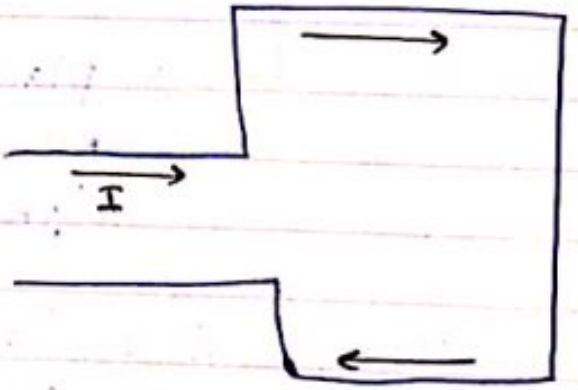


PIC as a switch.
external source give power to motor.

Dc motor:

electric field: تيار بيولد

منبعل بلف الـ motor.



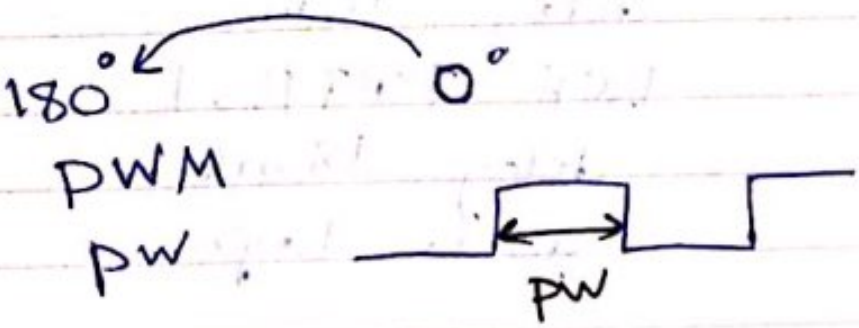
Steps كل ـــ بلف على

ازيد steps بزيد عدد Inductor داخل.

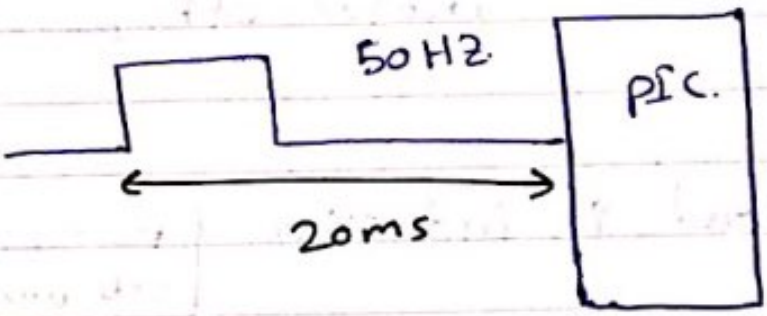
عشان بلف بشكل smooth .

* اتجاه دوران بعتمد على اتجاه تيار .

slide 36: Servo Motors :-



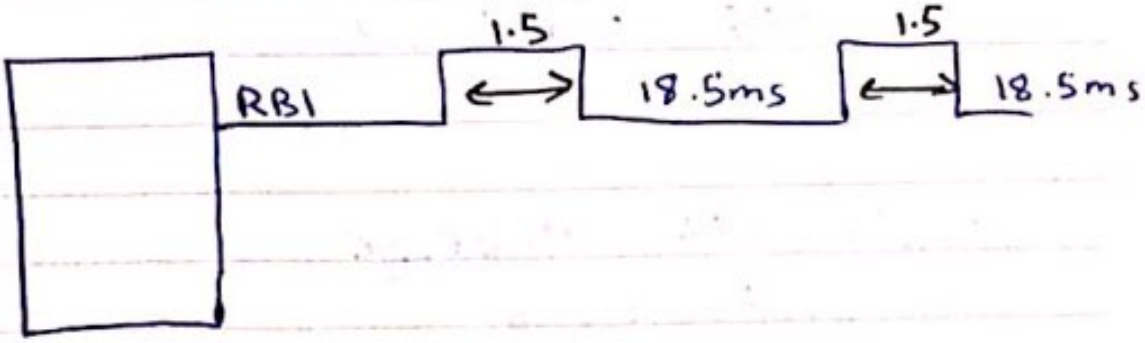
PW ⇒ represent by angle of rotation



at 35° => ?! .35 بدي الف بدرجة

$$\frac{1.75 - 1.25}{180} * 35^{\circ}$$

↳ per degree.



```
loop    BSF PORTB,1
        delay 15ms
        BCF PORTB,1
        delay 18ms
        go to loop
```

slide 38

لما يمر تيار في ال base رح يشتغل ال motor.

لما ما يمر تيار ← motor off

slide 39

load is inductive

[بجرب فيها شوية power
لما افصل بضل يلف
شوي ط بوقف.
اول ما افصل]

slide 41:

mosfet ON → current flow in leds

BSF PORTB,2.

slide 42:

H bridge: ⟶

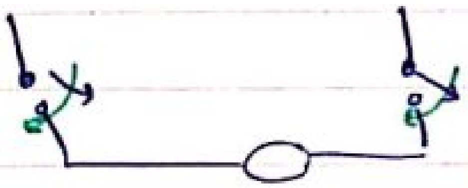
① to make motor rotates in 2 direction.

② to drive the motor with external voltage source.

* اذا كان 2switch مسكرين بنفس الاتجاه ← short ckt

Short ckt ⇐

Closed at same time → No current flow.



Slide 43:

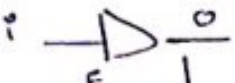
L293D
- → 4 half bridge ⇒ to rotate in one direction
- → 2 full H-bridge ⇒ rotate in 2 directions.

23/8 Wed

* L293D

4 half bridge.

2 full bridge.



E	i	o
0	0	
1	0	0
1	1	1

* 16 pins:
  - 4 of the pins grounded and
  - 2 - 2 voltage source.
    - V_{LS} — logic high ⇒ V_{LS}
    - V_{OS}

$V_{LS} > V_{OS}$

the motor will not rotate any way.

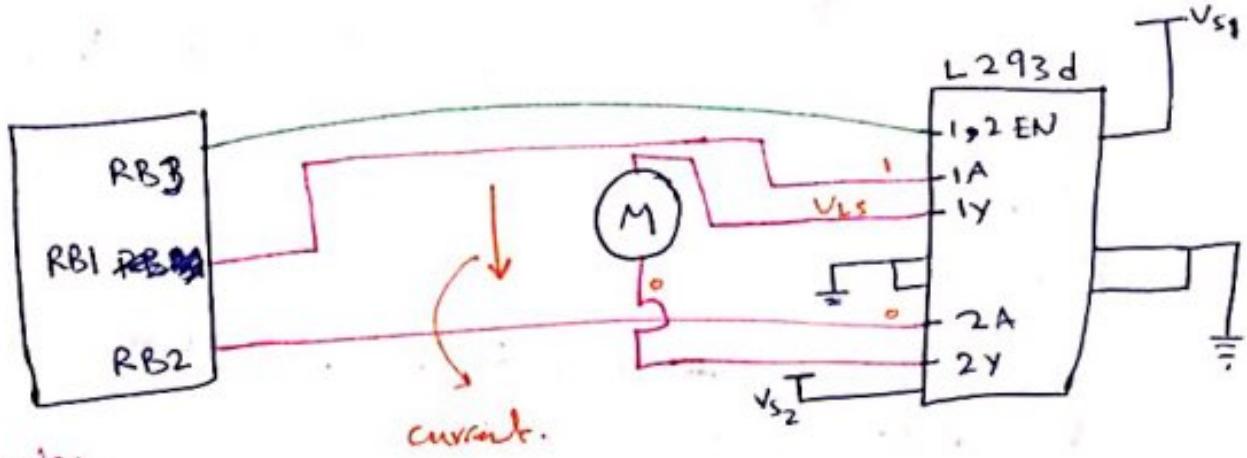
$V_{LS} < V_{OS}$ motor rotate if input =1 and driven by V_{LS}.

  - 4 inputs (A)
  - 4 output (Y)
  - 2 enables
    - enable for its 1st and 2nd half bridge.
    - enable for 3rd and 4th half bridge

Slide 43

Ex make the proper connection, and write the required code to make the current passes

* in the M↓ if RB1 = 1 and RB2 = Ø
* Current ↑ if RB1 = Ø and RB2 = 1
* the motor is totally off if RB3 = Ø

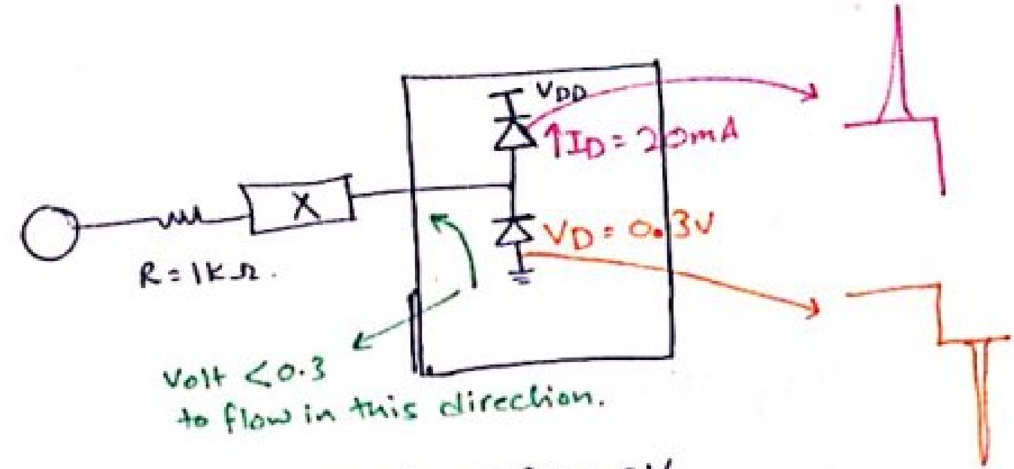


code:

```
CLRF  TRISB
→BS
  BSF  PORTB,3
  BSF  PORTB,1
  BCF  PORTB,2
```

Different families :-

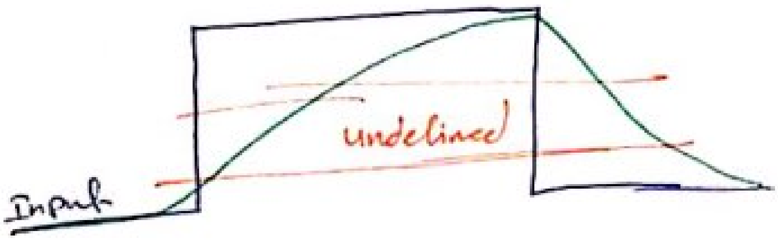
Slide 48



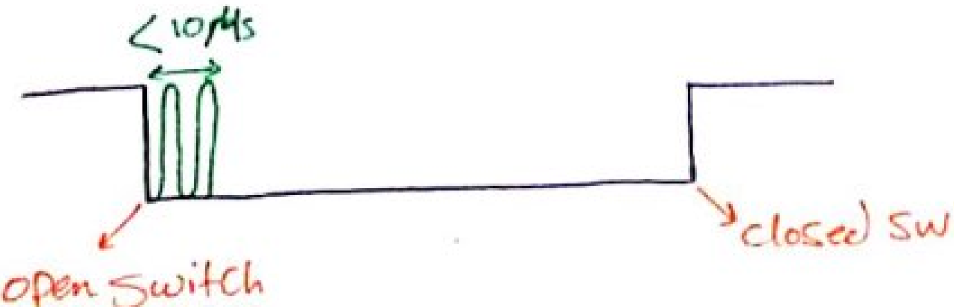
$$\text{max Voltage} = 1K * 20mA + 0.3V + 5V = 25.3V$$

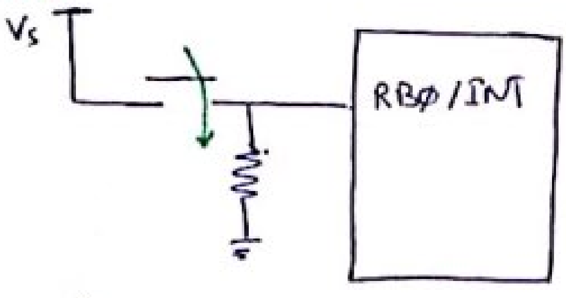
$$\text{min voltage} = 0.3 - 20\times10^{-3} * 1\times10^{3} = -20.3V$$

⇒ use filter to remove noise.



بلغي ال noise
بـــس بدخلني
ب ال undefined
نبـــــكم مو
~~[illegible]~~.
schmetic.





ISR

INCF counter, 1
BCF INTCON, INTF.
retfie.

* 2 solutions to bouncing :-

① by HW

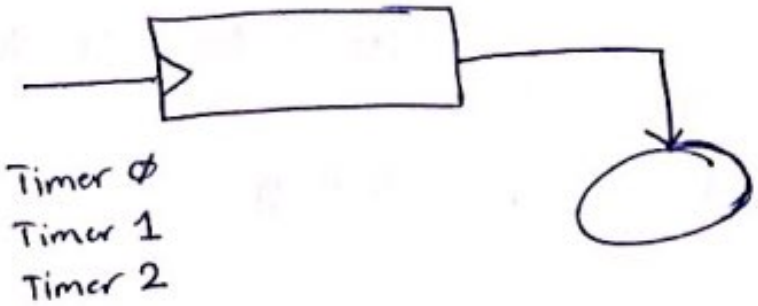
② by SW

* اذا كان مشبوك على RB1 حتى RB0 بدل

↳ Polling

loop delay 15 ms
BTFSS PORTB,1
go to loop.

CHAPTER 9: 😊



Slide 7

Timer 1:- is 16 bit counter.
↳ TIMRL and TIMRH Registor.

* $0 \rightarrow (2^{16}-1 \rightarrow 0)$

TIIF = 1
GIE = 1
PEIE = 1
T1IE = 1

1500
↳ Binary
MS8b ← | → LS8b
least Significat 8 bit

* It has ON/OFF bit
* osc يمكن يكون internal او external بالنسبة لـ Timer
* pre scale 1, 2, 4, 8
* clk can be sync or not.
* clock either internal $\left(\frac{Fosc}{4}\right)$ if TMRICS = Ø
  external if TMRICS = 1
  ↳ RCØ if T1OSCEN = Ø
  ↳ RC1 if T1OSCEN = 1

* RC0 → * you can connect it to external osc up to 200KHZ

* Keep counting in sleep mode.

RC1 → for counting

$$\text{Timer 1 delay} = \frac{4}{Fosc} * \text{prescale} * (2^{16} - IN)$$

TMR1L TMR1H

$$\text{Max delay} = \frac{4}{Fosc} * 8 * 2^{16}$$

$$= 8 * \text{Max delay of TMR0}$$

TMR1CON ⇒ Register.

Slide 10:

TMR2 is 8 bit counter

count from 0 → value in PR2

every time TMR2 = PR2 ⇒ TMR2 reset.

⇒ postscale = X

When TMR2 = PR2 number of times = X

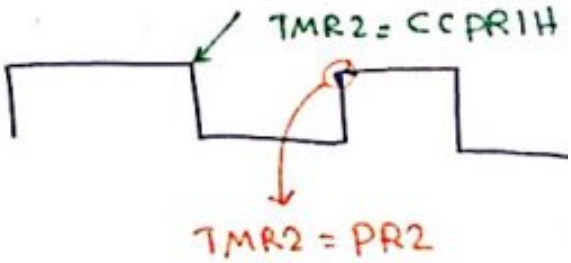
⇒ TMR2IF = 1

* have one clock source.
* No external OSC.

$$\text{TMR2 delay} = \frac{4}{Fosc} * \text{prescale} * \text{post scale} * (PR2+1)$$

Slide 19 :- PWM

when TMR2 = CCPR1H



when TMR2 = PR2 => [1] set. output
[2] reset TMR2
[3] copies CCP1LH

* ton is determined by CCP1H
T " " " PR2

* If PR2 ≤ CCP1H => 1 ہو تو
↳ No PWM signal

* ton => CCP1H || [ ] → least significant bits.
CCP1CON (4:5)

when TMR2 works in PWM mode. there is an internal counter 0 → 3

To modify ton, set the value in CCP1L and this value will take effect on next cycle. (when TMR2 = PR2).

Slide 21: T = (PR2 + 1) * Tosc * 4 * prescale of TMR2
CCP1L || 2 bits of CCP1CON.

ton = (10 bits value) * prescale of TMR2

IN PWM No post scale.

* write code to generate PMW

Fosc = 1MHz.

$10 \times 10^{-3} = (PR2+1) * 10^{-6} * 4 * \text{prescale}.$

$\text{Prescale} * (PR2+1) = 2500$

$\text{prescale} = 16 \rightarrow PR2 + 1 = \frac{2500}{16}$

$$PR2 = 155$$

ton = ?!

$2 \times 10^{-3} = X * 10^{-6} * 16$

$$X = \frac{2000}{16} = 125$$

00 0111 1101

CCP1CON = XX01 11XX

RP2 = D'155'

CCP1L = 0001 1111

T2CON = XXXX 11X

ON

prescale.

Slide 23 مش مطلوب