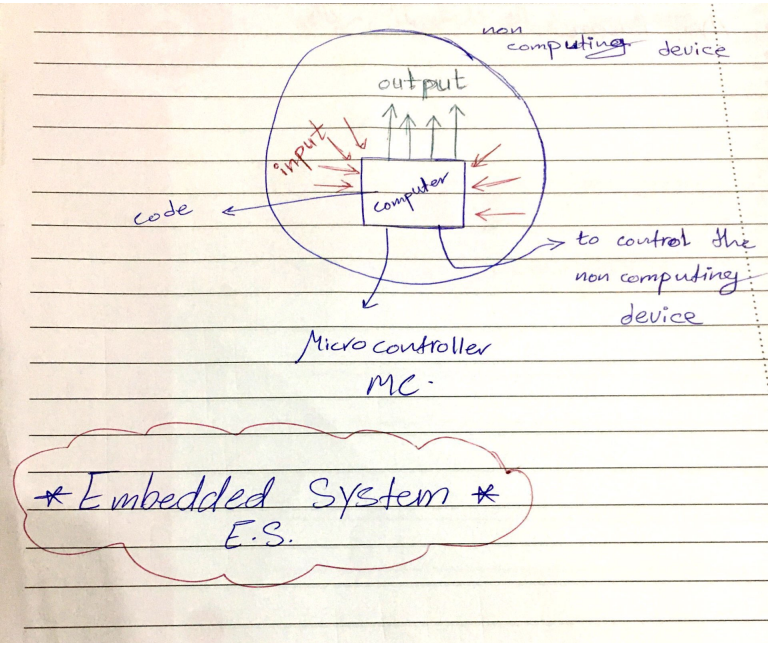
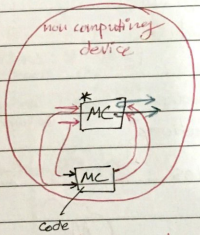


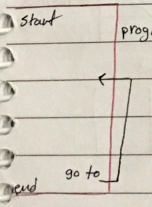
* Embedded System *
E.S.

MC micro computers: do one or few dedicated functions usually with (real time constraints) → timers & delays
↳ there are deadline that must be satisfied, if the action done after the deadline it's useless.

* How to understand & modify an Embedded system?
1- inputs
2- outputs
3- user interaction
4- link to other systems.
5- HW. used (model of MC.)
6- code, is it accessable?
* Yes? take backup & enjoy.
* No? - contact the company.



* software driven: everything in the E.S. is controlled by S/W.



* reliable: almost 100% work correctly.

Ex:

T_{actual} , $T_{desired}$

if $T_a > T_d$ turn on the compressor.

else

Turn off compressor.

* MC: computer on chip

all main components of the computer are existing on MC

* Main parts of computer (MC)

- CPU: control and excute the program
- I/O: to deal with external world
- Memory: storage
  - Data Memo: values of variables
    - ① volatile: lost of power down
    - ② faster & less power to write
  - Program Memo: code "instructions"
    - permenent → kept on power down
- Buses: connect the previous modules

C++:

```
void main ()
{ int x, y ;
  x=3 ;
  y=x+2 ;
  cout
  cin
  x=x+y^2;
}
```

Compiler

* check for syntax error
* convert to machine code

.bin binary

MC

x 3
y 5

data memo

program

① comp.

② exc.

compilation time

power of execution time

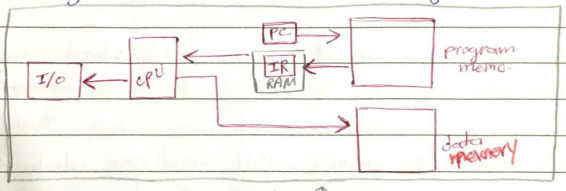
NOTEBOOK

* Program counter (PC): holds the address of the instruction to be excuted
* on power up, PC takes a default value called ((Reset vector)).

* If [PC] = 5
1st inst in your code compilation time should be stored on Reset vector ((0000))

* On power up, PC = reset vector = 0000

* The CPU fetches from program memory into a register called instruction Register (IR)



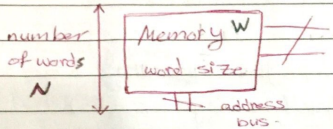
MC.

* PC is auto increment ⟶ power

IR
opcode | operands
* type of inst.
* format of inst

Data
* Volatile → lost on power down
* permenent → kept " " "
prog.

* VNA : 2 buses shuted among all components
  less complicated (less buses)
  fixed size of buses arrive to each component
* Harvard Architecture : 2 buses for each component.
  "faster due to more efficient pipelining
  variable bus sizes.
* pipelining : while the CPU is excuting an inst.,
  it fetches the next one.
  "PC auto increment"



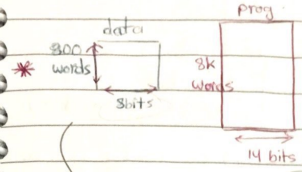
* The size that is reserved when you store any value on the memo.
* when you read the memo., you read one whole word.
* size of memo = $N \times W = \frac{N \times W}{8}$ Byte
* address bus $\geq \lceil \log_2 N \rceil$

ex: $N = 1000$ address bus $= \lceil \log_2 1000 \rceil \rightarrow 10$

$0 = 00\ 0000\ 0000$
$1023 = 11\ 1111\ 1111$ 10 bits

(9 bits 0-511)

* program memo > data memo



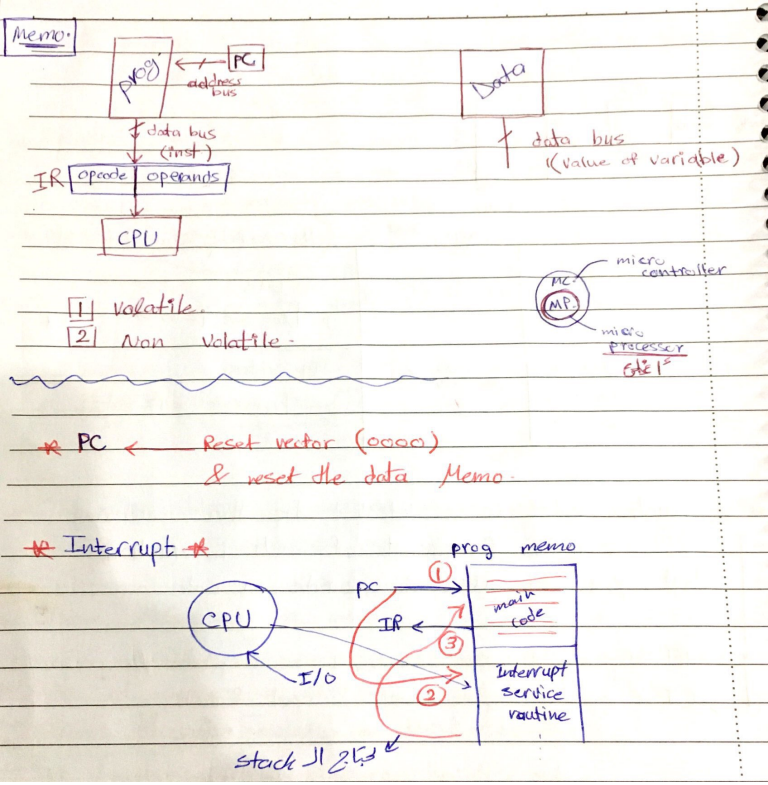
	VNA	Harvard
data memo // bus address bus	max $\lceil 8,14 \rceil = 14$ 13	8 bits $\geq \lceil \log_2 800 \rceil = 10$ bits
program Mem data bus	max $\lceil 8,14 \rceil = 14$ 13	14 bits $\geq \lceil \log_2 8k \rceil = 13$ bits
address //		

* Instruction set : is a detailed doc that contains the description & format of each instruction that you can use to write a code.

→ RISC → only few simple inst / slower compilation / long code / faster execution

→ CISC → many complex inst / several formats / variable size for inst / many addressing modes.

more efficient pipelining (fetch T = execution T) — same size for all inst



* MC's are divided into families:
same family, all MC's share the same core, & same Instruction set & inst. format

* Memo. sizes & peripherals are changing

* MC — Dual in-line packaging ((DIP))
* in the microcontrollers, more pins → more I/O, more size↑
* inter-pins space.

* PIC microcontrollers:
RISC, accumulator: single working reg. which holds the result of last excuted inst."

• 32 bits computer: the word in the data memo is 32bits, the data bus that is connected to the data memo is 32 bits, each variable 32bits
• largest variable = $2^{32}-1$.

• 8bits MC. are classified into 3 families:
1- base line
2- Mid range.
3- High performance.
* to know the family of MC. we should check the "data sheet".

* we will deal with MC. from 16 series in the Mid range family.

MC: ex. 16 L F C 87 X A

- 16 → series num.
- L → low power
- F → flash memo.
- C → CMOS technology. طريقة تصنيع الترانزيستور
- 87X → member name in the given series
- X → don't care (أي رقم مكان x)
- A → advanced technology

* **Stack:** Memo, Volatile, neither readable or writable by SW., it's automatically written & read.
  - it holds the return address.

code:

```
1) main ()
2)
3)
4) call subroutine
5) .
   .
50) Subroutine
   .
   .
60) return
```

(60 → address)

1) (PC = 5) —push→ stack.
2) PC ← 50
3) PC ←pop— stack (5)

* بيحتاج one level stack بهالحال

بس بال (nested calls) رح نحتاج أكثر من stack

CH 2

Subject Date No.

* Interrupt vector: default address that is loaded in the PC on interrupt.

● Slide 27 interrup = 1

"Slide 5"

* PIC 16F84A

→ mid range family.
→ 8bits mc.
→ harvard arch.
→ RISC → 35 inst.
→ 18 pins.
→ 3 peripherals:
  - 8bits timers
  - 5 pins PORTA
  - 3 pins PORT B
  - for I/O

→ max clk speed is 20 MHZ can connected
→ 3 types of memo
  - data memo (68 bytes) → word = value of variable volatile
  - program (1k) → inst 14 bits permenent
  - EEPROM (64 words) → variable permenent

→ * MCLR : external reset.
if connected to "low" it reset.

→ 18 pins:
1. one pin MCLR
2. 2 pins VDD & Vss
3. " " external clk (osc)
4. 5 " PORT A
5. 8 " PORT B

at most (3 I/O's concurrently)

1k (955) variable

* max I/O → 18

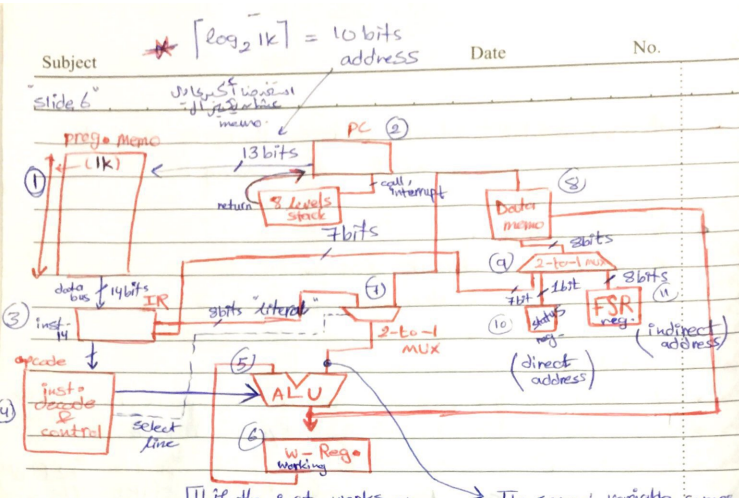
Subject

Date

No.

slide 6

* $\lceil \log_2 1k \rceil$ = 10 bits address



[1] if the inst works on 2 variables, one of them always from (W-Reg.)

[2] The second variable comes either from inst (literal) or from data memo.

"Add the value (5)" from the inst.

* Watch Dog timer: it reset the MC. on crash.

* Status: holds info. (flags) of last excuted inst.

if the address comes from the inst. (direct address), then the address is 7bits, & the 8th bit from Status Reg.

* In PIC, there are 3 addressing modes:

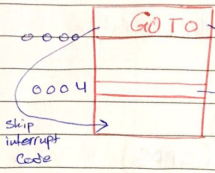
[1] Literal: value in inst. (ex: add 5.

[2] Direct: " " Data memo, address in (inst.) (ex: add the value in address 17.

[3] Indirect: " " " " " " (FSR)

(ex: add the value that it's address is in FSR.

* Reset vector * default address of the pc when powerup or reset.
* Interrupt " * " " " " " on interrupt.



* Configuration word : it's a Word stored in prog. Memo
- it holds configuration info. about the prog.
- You can write or modify it only at prog. download time.
- Once the configuration word was written, to modify it you should redownload the Code.

* What does it contain?

1 FOSC1 & FOSC0 : tybe of "osc" to be connected to the PIC.
("oscilator" bit 1, bit 0)

ex: XT, HS, LP, RC oscilator.
crystal

PIC [external clk.] osc

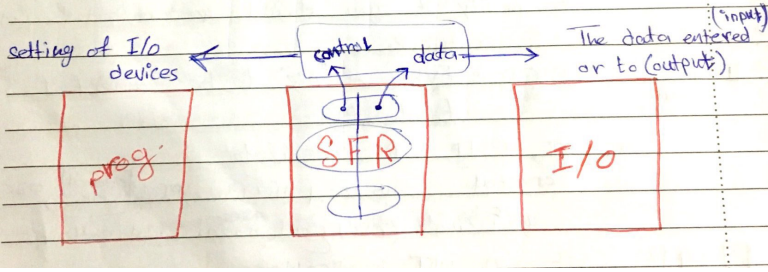
2 WDTE bit → 0 WDT is disabled
((watch dog timer)) enable bit → 1 " " enabled.

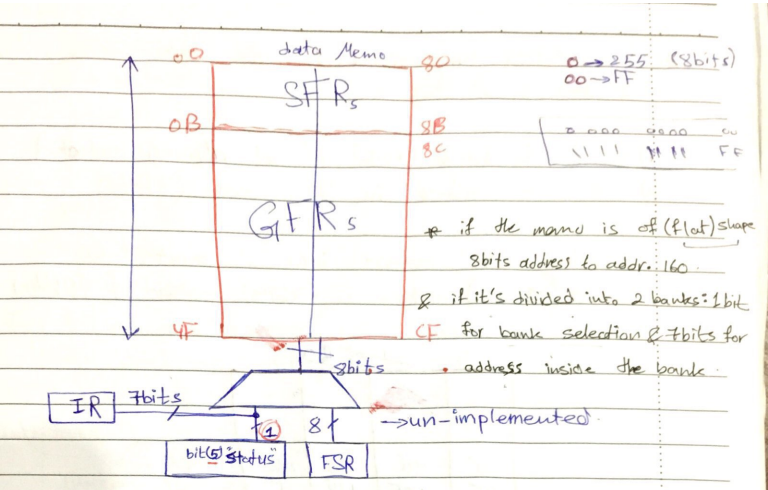
3 PWRTE → 0 enable the MC. will stay in reset mode.
"powerup timer" enable for a while in order for power to Stabilize on all components.

4 CP ((code protection)):
↳ 0 the code is protected, NObody can read it

* Data Memo :- holds values of variable, Volatile
it's dividend in two forms:
↳ ① vertical.
1-special fun. Reg. (SFR)
2-General purpose Reg. (GPRS)
↳ value of developer variables.
↳ ② Horizontal.
2 banks (bank0 & bank1)

* Memo mapped I/O arch.



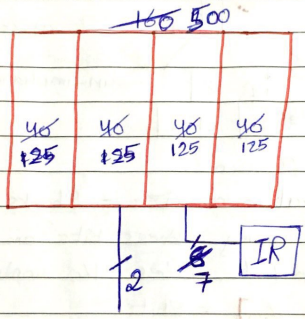


* each bank is 80 words.
  - → total memo size = 2 banks * 80 words / bank = 160 words.
  - → how many address bits are required to address 160 words?
    $\lceil \log_2 160 \rceil = 8$ bits
  - → how many address bits are required to address any word inside a special bank?!
    $\lceil \log_2 \frac{80}{\text{every bank}=80} \rceil = 7$ bits
* However we still to specify the bank?
  1 bit.

* To address any word in the data Memo, it does in 2 steps:

1) select the bank (1bit) if not already selected
2) " " address in the bank (7bits)

→ Dividing the memo. into "2 banks" we saved (1 bit) address (7bits) instead of (8bits).



* To access any word in the data memo.:
1- select the bank if not selected (status reg. (bit 5) RP0)
2- " " address inside the bank.

Ex: 95 H

① 001 له 0101 ← (arrow)

∴ bank 1

عشان اعرف الaddress ع أي bank.
بطلع على (MSB)

Ex: 7A H

⓪ 111 له 1010 ← (arrow)

∴ bank Zero

Ex: Assume you have an inst. called "RD ad" that reads the word at address ad from the data memo, use it to read the following addresss sequentially.

0101   0011   0100   1010   0101

85, 73, 24, 9A, A5.

① 000 له   0111   0010   1001   1010

bank 1   bank0   bank0   bank1   bank1 b1

* RD 85 → compiler 0101 X غلط
(مرفوض bit اضافي مش موجودة، مابتاخذ 7 bits)

RD 05] inst.

① select bank 1   الصح

status(5) = 1

status: [| 1 |] (bit 5)

RD → 05

* RD 73   ① bank Ø   ② status (5) = 0
③ RD 73

* RD 24   ① bank 0   ② status(5) = 0   ③ RD 24 (read)

* RD 9A   ① bank1   ② 〃 = 1   ③ RD 1A

* RD A5   ① bank1   ② 〃 〃 = 1   ③ RD 25

slide 14, skip.

Flags: -

Z , DC , C

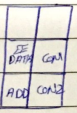
EEPROM Data Memo

* To read the EEPROM *

(EEDATA, EEADDR, EECON1, EECON2) reg.

address

08 09 88 89



different banks

① write the address in EE ADD reg.

② select bank1 (b1)

③ Set RD bit in EECON1 = 1 ← (start reading from EE PROM at address stored in EEADD to EEDATA))

④ select (b0)

⑤ read the value from EEDATA. (1word).

* RD is set by SW. cleared by HW when reading is over.

addr. 16 ... addr. 15 ... (3) ... (4) ... (1) ...

* To write on EEPROM:

ex: to write the value 17 to the EEPROM at address 22.

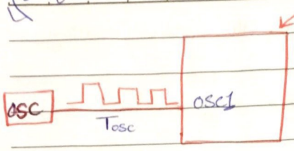
1. write the value "17" in EEDATA.
2. " " " "22" in EEADDR.
3. enable writing to the EEPROM. (WREN = 1) in EECON1 (write enable)
4. Write 55H to EECON2.
5. " AAH " "
6. set WR = 1 bit in EECON1 → start writing 17 → 22 ((set by SW, cleared by HW. when writing is over)).
7. once writing is over EEIF in EECON1, becomes = 1.
8. If an error happened during the writing WRERR = 1 (write error)

B.S: "bank switching"

* EEIF → ((is set by HW., cleared by SW))

Subject

(external osc.)



* $f_{osc} = \frac{1}{T_{osc}}$

* Tinst (factor * Tosc)

in PIC, Tinst = 4 * Tosc

* Tosc is not enough to fetch or excute one instruction.
* New cycle was defined and called "Tinst"
* Tinst is the time that is enough to fetch or excute one instruction.

→ fetch time = excute time due to RISC arch.

Ex: Assume you have MC. [1] asm → x with "freq. = 40 MHz" and "Tinst = 6 Tosc". & [2] MC → y "freq = 30 MHz" "Tinst = 3 Tosc"

* MC x → $T_{osc} = \frac{1}{40\ MHz}$ → "Tinst" $= \frac{6}{40}$

* MC y → $T_{osc} = \frac{1}{30\ MHz}$ → $T_{inst} = \frac{3}{30}$

Ex: Assume Fosc = 4 MHz, what's Tinst = ?

* Tosc = 1/4 MHz = 0.25 MS.
* Tinst = $4 * \frac{1}{4}^{T_{osc}} = 1$ MS. → The time that is enough to fetch or excute one inst. on PIC that is driven by a clock with freq. = 4 MHz.

Ex: Fosc = 1 MHz

* Tosc = $\frac{1}{1\ MHz} = 1$ MS * Tinst = 4 MS.

* one inst. needs to be fully excuted 2MS.

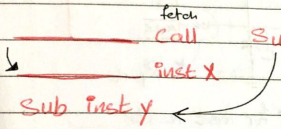
• If you have a prog. composed of 10 inst.

10 inst * $\frac{2\ MS}{inst}$ = 20 MS with no pipelining

= 10 MS " pipelining

• with pipelining time goes approximetly to half

• sometimes the pipelining fails, this happens when the fetched inst. is not the one that will be excuted next.



in this case, the CPU consumes one more Tinst to fetch the correct inst.

* any inst. needs 1 Tinst to be fully excuted except the inst. that (make / causes) pipeline failure, they need 2 Tinst.

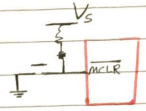
rec 10

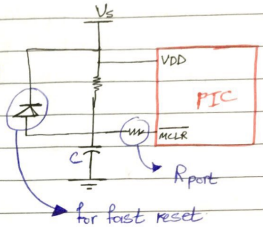
Subject [X] pin Slide 23 Date 15-10 No.

* To keep the MC in Reset mode for a while on powerup :

[1] external

on powerup, $\overline{MCLR}$ (master clear (external reset)) = 0 will keep low until the capacitor gets charged which needs time related to $\tau = R*C$





[2] internal.

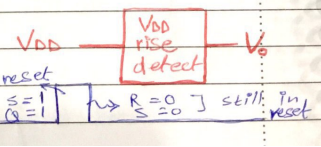
* when chip-Reset = 0
  - reset the PIC.
  - when does reset happen?
    S=1 & R=0
  - when does S=1 ?
    in any of the following cases:

[1]- if $\overline{MCLR}$ = 0

[2]- (WDT) detects crash while the pic is not in sleep, WDT will not reset PIC in sleep.

[3]- on powerup.

S	R	Q	$\overline{Q}$	
0	0	Q	$\overline{Q}$	
0	1	0	1	set
1	0	1	0	reset
1	1	undefined		



* when does the MC. exit from reset mode after powerup ?

when R=1 & S=0

* when does R=1 ?

when all of the following happens:

1. S=0.
2. if enable PWRTE is enabled, we need 1024 cycles of the internal RC osc. which equals (72 ms).
   However, if enable PWRTE is disabled, the time =0 (no need for counter output).
3. if enable OST is enabled, we need 1024 Tosc, However if it's disabled no need to wait for the counter.

* how to enable PWRTE?

→ 10 bit ripple counter → 0, 1, 2, ... 1023

in config word, there is a bit called PWRTE.

* how to enable OST. ?

it is by default enabled if you choose any of OSC. types XT, HS, LP and it's disabled for RC osc.

rec 11 m

Subject: lo

word → 8 bits

CH4

Date

No.

* CU

HLL → compiler → assembly → machine code

* if an inst. need two variables (e.g. Add), it will have 2 versions:

working

W, L literal

W, f data memo.

* result will be stored in:

working reg.

* W - Rag (d=0)

* Data Memo (d=1)

35 inst. in RISC

ALU ← CPU

working reg. (1)

(inst) literal (2)

data Memo (5)

(inst. Illustrate (d) bit)

* destination bit (d)

→ Types of instructions:

[1] Byte oriented file Reg.: (= data memo word)

operands ← f (7 bits data memo. address)

d (1 bit) "destination" (op, f, d)

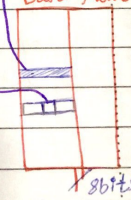
[2] Bit oriented file reg:

f (7bits) (data memo addr.) (op, f, b)

bit location in address (f) ← b (3 bits) → range [0-7],

"No d bit"

(file reg. = Data Memo word)



Subject Date No.

the value in the inst 8 bits.

3 Literal instructions : (op, k) → data from Is.
- k (8 bits) No need to access data memo.
NO f or d
→ the result stored in. W-Reg (working)

4 Control (op, k)
change path of excution :
ex: Call, go to, return, ...
GO TO (k) 11 bits address for the next inst.
call
inst.

k → Value Literal (8)
address Control (11)

opcode	operands

format

* Arithmetic inst. *

assume that initially the data memo & the w-reg. have the following state,
what are the final values in the reg.
21, 22, 23, 8 w-Reg.

1- ADDWF f, d
2- ADDLW literal k
3- SUBWF f, d
4- SUBLF k
5- INCF f, d
6- DECF f, d
7- COMF f, d
complement

W= ~~6~~ ~~27~~ 2A

1- ADDWF 22, 1
2- ADDLW 21
• 3- ADDWF 3, 23, 0
~~4- ADDWF 21, 22~~
5- ADDL 5
5+w-reg
2A + 5 → ~~W~~ w-reg.
2F

1	
22	~~2~~ 8
3	

d: 0 → 1

① [22] + [W] ⇒ [22]

2 + 6 → [22]

→ 8

[21] = 1

[22] = 8

[23] = 3

[W-Reg] = 2A

② 21 + [W] → [W]

21 + 6 → [W]

27 H

③ [23] + [W] → [W]

3 + 27 → [W]

30D

2A H

W = 2 ~~0~~

→ INC

COMF	26, 1	Z = 0
DECF	24, 0	Z = 1
INCF	25, 0	Z = 1

24	1
25	FF
26	~~7A~~ 85

DC = 1 DC = 1

1

0000 0000

0111 1010

1000 0101 complement

8 5

0 → 1

SUB LW → k − [w]

* SUB WF → f − [w]

we always Subtract the working-Reg.

* A−B = A + 2's comp. of B.

		1's	+1	
A = 5	0000 0101	1111 1010	→	1111 1011
B = 7	0000 0111	→	→	1111 1001

[C] is the [comp.] of the sign

* A−B = 5−7

```
 ⊖          0000 0101
  (0) ←     1111 1001  +
           ----------
            1111 1110
```

* B−A = 7−5

```
 ⊕          0000 0111
  (1) ←     1111 1011  +
           ----------
            0000 0010
```

[21] > 7 ?

* write 7 in "(W-Reg)" *

SUBWF 21, 0^W

* عشان ما تتغير القيمة الأصلية
" بس مقارنة ".

check C flag [w] ← [21] − [w]

if "(c=1)" → [21] > 7

[21] − W ⟶ c=1 +ve

7 ⟶ c=0 −ve

عكس

{ SUBLW k
{ SUBWF f,d

* 6 logical inst : we use them for bits masking to modify part of the word ($\rightarrow 0$ AND, $\rightarrow 1$ OR, $\sim$ XOR)

and keep the other bits as they are.

$X \cdot 0 = 0$ $\quad X + 0 = X$ $\quad X \oplus 0 = X$

$X \cdot 1 = X$ $\quad X + 1 = 1$ $\quad X \oplus 1 = \bar{X}$

[ANDLW k
AND WF f,d

[IORLW k
IOR WF f,d

[XORLW k
XOR WF f,d

A	x	A⊕x
0	0	0
0	1	1
1	0	1
1	1	0

AND ← 0 pour les zéros
OR ← 1 " " "
XOR ← comp " " "

Ex: write one inst. to clear the 4 MSBits of the working reg.

ANDLW 0F

ADDWF addr: mettre le résultat

W = XXXX XXXX
0000 1111 → 0F
0000 XXXX

Ex: set the least significant 4 bits of w-reg.

IORLW OF

W = XXXX XXXX
0000 1111
XXXX 1111

Ex:

Complement the bits of even position in w-reg

XORLW 55

7 6 5 4 3 2 1 0
0 1 0 1 0 1 0 1
5 5

Ex: code complement the w-reg.

↳ use easier method on a whole word.

*we can use masking bits

Ex: write code to clear the least sig. 4 bits of address 17.

WF ← write F₀ in w-reg. → w-reg: وعاء

ANDWF 17, 1
address destination

[17] ← [W] • [17]

* The most commonly used inst. are the data movement inst.

MOVF f, 1

MOVWF f

MOVF f, 0

w-reg

MOVLW k

for data memo

* To write a value (15) in the w-reg.

MOVLW 15

* To initialize any data meme. location (21) address with value (30)

a) MOVLW 30

b) MOV WF 21

* اي ما عنا "inst." بتنقل عطول الداتا (Data meme) لاني ما بتوسع

MOV

X ((MOVLF 21)) | 14 bits | inst. | inst. + opcode

8bits + 7bits = 15bits
address

* To copy the value in address 22 to address 23.

a) MOVF 22, ϕ w

b) MOVWF 23

MOVF 22, 1
copy بنسخها لنفسها

• MOVF f, 1 :

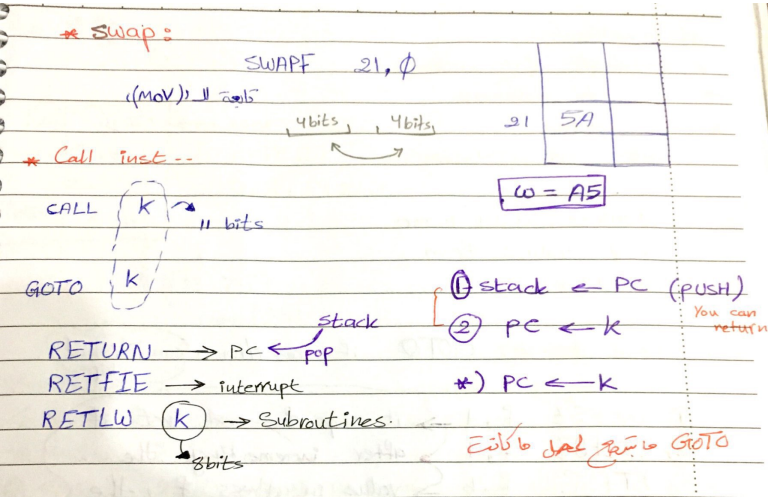
to check if the value in address f is ϕ ?

if (21 == 0) ?

MOVF 21, 1

[21] ← [21]

* check Z flag if Z = 1 → [21] = 0 ✓



* 7 arithmatic
* 6 logic (bit masking)
* 4 data movement
* 9 Control
  - 5 call, GOTO, return

→ to initialize data memo or copy values from data Memo.

* Conditional branch: GOTO if condition, (conditional skips)

file skip if zero

① INCFSZ f,d → it skips the next inst. if
② DECFSZ f,d → after incrementing the
③ BTFSC f,b → value in address f, the
④ BTFSS f,b → result is zero.

③ if clear
④ bit in address f is 1

→ it skips the next inst. if bet ≠ b in address f is 1

bit test file skip if set

these four inst. for conditional statements & loops.

* if ([22] == 0)
  [22] = [22] + 1

bit test file if clear.

[22] ← [22]
if [22] = 0
  Z = 1
else
  Z = 0

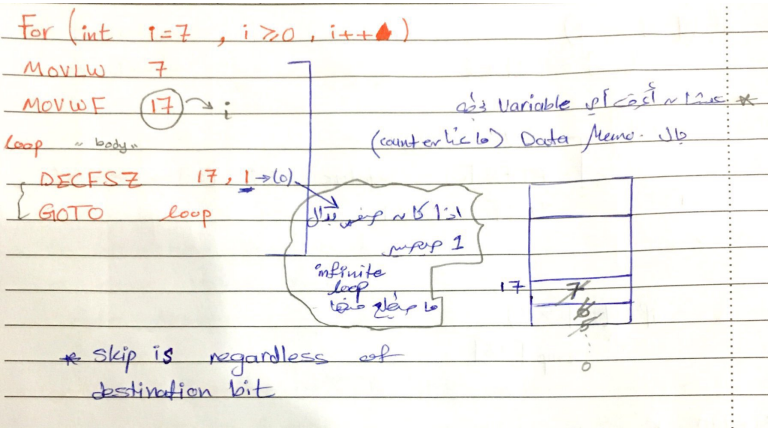
MOVF 22,1
BTFS [C] status, Z; [22] = 0 → Z = 1
INCF 22,1

* if ([23] > 7)
  comp [23]

XSUBLW k
✓SUBWF f,d → address ...

① MOVLW 7
② SUBWF 23,0 → address ... 1 Time

BTFS [C] Status, C C=1 ← [23] > 7
1 Time: COMPF 23,1 without skip
2 Time. with skip [23] - 7



CLRF f (no destination bit) address القيمة في bit

CLRW

NOP do nothing (for delay)

Timer = 0.1 seconds

* Loop inst 1 → (turn on LED)
10 NOPs

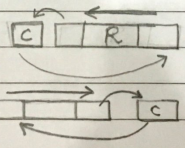
1 sec

inst 2 → (turn of the LED)
10 Nops
GOTO Loop

* BCF f,b → clear bit num. b in address f.
bank selection
* BSF f,b → sets bit number b in address f

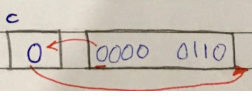
BCF STATUS, 5 → b_0
BSf STATUS, 5 → b_1

* SLEEP sleep.
* CLRWDT watch dog timer.
* RLF f,d rotate left file
* RRF f,d right



*2 if C=0
*2+1 if C=1
• int division if C=0

RLF 15,1



6
12

نزيح في مرتين
2

Ex:

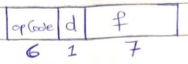
write code to multiply by 4 the value in address 18.

```
BCF   STATUS, C   →   C=0
RLF   18, 1       →   *2
BCF   STATUS, C
RLF   18, 1       →   *2

SUBLW  10          10 - W
```

* Byte oriented file Reg.

op f,d ⟶

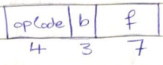


compiler

* code ↓ CPU

* bit oriented file reg.

op f, b ⟶

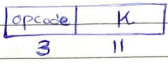


* Literal

op , k

14 bits

* GOTO , Call k



* 6 bits = $\lceil \log_2 35 \rceil$ ⟶ # of bits for Opcode.

inst.

3 bits, if they are valid opcode then this is the inst., otherwise it checks the next bit along with the 3 bits.

10 0001 0001 0001 (14 bit inst)

* Label | instruction mnemonic (35 inst.) | operands | comment

- starts at the beginning of the line.
- case sensitive a ≠ A
- Once defined you can use it as operand and it's value is the address of 1st inst. after it.
- the lable can be in a stand alone line

lable
* Loop * → some address

GOTO Loop
operand

* the compiler who know the address.

* comment
- starts with ;
- none excutable. ... ما بتنفذ
- in a stand alone ~~time~~ line

* how we write values:

MOVF F
MOVL K
MOVF 20, 1
hexa

* Decimal → D'15' , d'15'
* Binary → B'10101' , b'10101'
* Octal → O'32' , o'32'
* ASCII → A'6' , a'6' , '6'
* Hexadecimal

* if the hexa value starts with letters
it must be preceded by either 0X or H

*Hexadecimal : 0XA3 , "H 'A3' , h'A3'.
↳ default

address
MOVF 0x23 , 0
MOVLW 0X23
literal k

* Assembler Directive : it gives info to the assembler at compilation time and after that, they are discarded

① * include it opens a file, and you can use anything inside it.

File
* include PIC16F84A.INC

once included, you can use all registers & bits by their names.

• BSF STATUS , C
bit set file

		Z	DC	C

STATUS

* not included :
① → BSF 03, 0 ✗
OR
② → you define STATUS & C ✓

(2)

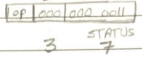
* EQU — equate

↳ it defines constant

STATUS EQU 03 — not variable

C EQU 0

BSF STATUS , C



PI EQU 3 → no floating point for (3.14)

Temp EQU 5

cblock 0x20

var1

Var2

Var 3

Var 4

end C — when the compiler sees it, it stops the execution.

↳ =

Var1 EQU 0X20

Var2 EQU 0X21

Var3 EQU 0X22

Var4 EQU 0X23

end

ORG ox , V — org=origin

↳ it tells the compiler that the next inst. should be stored in the prog. memo.

start ← ORG 0000

Interrupt service routine → start of 1st ISR ← ORG 0004 ← at address V

N O T E B O O K

* assembler details slide 23.

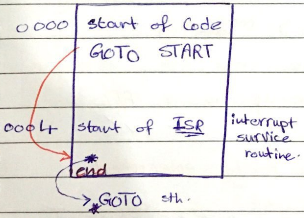
* Var1 EQU 12
  MOVF var1 , Φ → f
  ADDLW var1 → k

Ex: slide 24

[22] ← [31] + [45] + [47]

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

ORG 0004
↳ it's not a must except if you enabled the interrupt because the interrupt are disabled by default.



INVEC goto INVEC — infinite interrupt vector

* DONE GOTO DONE → عشان ما يطلع من الcode يضل فيه وما يرجع فوق يعيد التنفيذ.
  lable

* ORG, EQU مابتهمنا

ex slide 25

↳ 8 instructions (go to, MOVF, - - -)

8 * 14 bits

* MOVF (address = ?) → بطلع لفوق لحين الاقي ORG وبنزل بعد بعدها بزيد

0009 ↙ ORG 0004
↳ 0004
0005

((ORG 0000 مع ما إلها address لأنها directive))

* what is the binary representation of the inst
DONE GOTO DONE ?!

GOTO k → | 101 | KKK kkkk kkkk |

(1)- opcode — 101: 3bits; KKK kkkk kkkk: 11 bits

بافترض من الـ data sheet
" معطى بالسؤال "

((101 000 0000 1010))

ORG 0004
↳ DONE → 101

* Assume that this code is running on a PIC with Fosc = 4KHZ.
- how long does it need to be fully executed.
Assume we removed the last inst. — DONE GOTO DONE } عشان ما يضل infinite

- $T_{inst} = 4/F_{osc} \rightarrow 4/4k = 1\ MS.$
- $7\ T_{inst} \rightarrow 7\ MS.$

- GOTO DONE
  10^6 lines
  code ما بيتعدى بكل الـ
  1 inst عن
DONE

Ex: [33] ⇄ [11] swap

1- temp [22]
2- [11] → [33]
3- [22] → [11]

```
MOVF 33, 0
MOVF 22
MOVF 11, 0
MOVF 33
MOVF 22, 0
MOVF 11
```

slide 27

[w] = [33]

```
ORG 0000
ORG 0004
end
infiniteloop
```

end قبل

لازم أعمل هيك أو لا لأن (bank switching)

* (goto, bcf, mov --- goto) 10 inst. in slide 27

Ex:

write prog. that do the following:

① [17] = [12] - [22] - [23]

② if [22] > [23]

[22] = [22] * 4

③ [17] = [18] * 9.

Sol——.

①

```
MOVF   22, 0
SUBWF  12, 1
MOVF   23, 0
SUBWF  12, 0
MOVWF  17
```

②

```
MOVF   23, 0
SUBWF  22, 0
BTFSC  STATUS, C
GOTO   ENDI
BCF    STATUS, C
RLF    22, 1
BCF    STATUS, C
RLF    22, 1
ENDI
```

③

```
MOVF   18, 0
BCF    STATUS, C
RLF    18, 1
BCF    STATUS, C
RLF    18, 1
BCF    STATUS, C
RLF    18, 1
ADDWF  18, 0
MOVWF  17
```

ubject CH5

Date 31-10

No.

* Conditional branching.

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

* If [22] = 0.

code block1

else

code block2

MOVF 22,1 → 0 إذا كان 22
BTFSC STATUS, Z
GOTO block1
GOTO block2

block1

GOTO NEXT →

block2

GOTO NEXT

block2

NEXT

block1

NEXT

GOTO code1

Code 2

GOTO code1

GOTO code2

GOTO Next Code1

Code1

Next

code2

Next

* if ([22] > [23])

Subtract 7 from [22]

else

Subtract 7 from [23]

```
MOVF    23,0
SUBWF   22,0
BTFS [C] STATUS,C
GOTO    sub22
GOTO    sub23
```

[22]-[23]

if C=0 → [22] > [23]

[22] < [23] ⇒ C=1 BTFSS

```
sub22   MOVLW   7
        SUBWF   22,1
```

[22] = 7 - [22]

```
sub23   MOVLW   7
        SUBWF   23,1
```

[23] = 7 - [23]

Next

Loop

* for (int i = 17 ; i >= 0 ; i--)
add 3 to address 22.

```
Counter   EQU    18
          MOVLW  D'17'      ] i
          MOVWF  counter    ]
loop
          MOVLW  3          ]
          ADDWF  22,1       ] [22] += 3
DECF counter,1
          DECFSZ counter,1  ] i--
          GOTO   loop       ]
```

(i) [18] = 17

17 - 1 = 16

* for (int i = 7 ; i <= 22 ; i++)
add 3 to address 22.

```
Counter   EQU    19
          MOVLW  7          ] i = 7
          MOVWF  counter    ]
loop
          MOVLW  3          ]
          ADDWF  22,1       ] [22] = [22] + 3
          INCFS  =22        → INCF   Counter,1
                              MOVLW  D'22'
                              SUBWF  counter,0
                              BTFSS  STATUS,Z
                              GOTO   loop
```

Zero Flag

C Flag

NOTEBOOK

ORG 0000
Reg size
include: headerfile
end

* Conditional Branching:

Ex-1

```
        MOVF    11,0          [11]
        ADDWF   22,0          [22]+[11]
        BTFS[C] STATUS,C
        GOTO    NOC
        GOTO    WithC

NOC     MOVWF   33
        GOTO    Next

WithC   MOVWF   44
NEXT
```

ADD location 11 + 12
No carry = [33]
with carry [44] = 11+12

" عشان يلف 256 مرة "

FF = 255

```
MOVLW   0 (FF)
MOVWF   counter
loop
        DECFSZ  counter,1
        GOTO    loop
```

((255 ← أكبر loop))
" ← Variable "

اذا بدي ألف الـ loop أكثر
ما بقدر counter → 8bits
أكبرين فمستخدم "2 counters"

Ex-2 ((16 bit counter))

```
        Count H       count L
        3  (2,1,0,FF)  5  (4,3,2,1,0)

START   MOVF    count ,1
        BTFS[S] STATUS, Z
        GOTO    label
        MOVF    countH ,1
        BTFS[C] STATUS, Z
        GOTO    DONE
        DECF    count H , 1
label   DECF    count L , 1
        GOTO    START
        ~~GOTO    DONE~~
DONE    GOTO    DONE
```

NOTEBOOK

* Subroutine ——→ code starts with label & ends with return or return[,w]

function ()
{
:
}

excuted when call by (CALL 'k', label')

main ()
{
call function ();
:
}

Subject **Subroutines.** Date 3/11 No. $2^8 = 256$

int x = sum(3,4).

call sum(3,4).

- parameters & return values are exchanged through the data Memo.

call stack { PUSH → stack — address of call

return — pop → PC

Ex - slide 16.

[W] ← [30] x [31]

```
Multiply    clrw
            MOVF      31,1
            BTFSC     STATUS,Z
            Return
repeat      ADDWF     0X30,0
            decfSZ    0X31,1     → FF
            GOTO      repeat
            RETURN
            END
```

slide 17

* لو برمجتها الـ س ما رح
يرجع أرقامها الـ (data memo)

* لو ضربنا * صفر
[ ] * Zero = 0

بالكود ضربنا * ب 256
فقط ...

كأني بعمل (Shift left)
8 positions

بجمع اللي بـ 30 عدد مرات اللي بـ 31

* Time Delay :

- HW → timers
- SW → non useful code (NOP).

ins 1

if $F_{osc} = 4\,MHZ$

$T_{inst} = \frac{4}{4\,MHz} = 1\,MS.$

```
NOP
 :
 :
NOP
```

max 1024 NOP's

inst 2

```
     MOVLW   X
     MOVWF   counter
Loop NOP
     NOP
     DECFSZ  counter, 1
     GOTO    Loop.
```

256 iterations

[بحط x ← 0 عشان تمشي 256 مرة]

inst 3 Nested loop.

```
     MOVLW   x
     MOVWF   count H
     MOVLW   y
     MOVWF   count L
Loop NOP
     NOP
     DECFSZ  count H, 1
     GOTO    Loop.
```

256X256

(~~[illegible]~~)

delay أكبر

inst. 4

```
        MOVLW   x
        MOVWF   counter
loop    call    subroutine
        DECFSZ  counter , 1
        GOTO    Loop

subroutine  MOVLW   y
            MOVWF   counter2
loop2       NOP
            DECFSZ  counter2 , 1
            GOTO    loop2
            RETURN
```

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

* num. of Tinst = Trance the code.

* Nested loop:

(*) initialization.

(*) First external iteration.

- all internals except last.
- last internal

(*) all external except 1st & last

(*) last external iteration.

* Single loop:

- initialization.
- all iterations except last.
- last iterations.

Slide 20 . Single loop:

* Fosc = 800 kHz .   ب 200 تكرار

|ex| single loop in slide. — تكرار 200 مرة

. Delay = $\frac{4}{F_{osc}}$ * # of Tinst .

$= \frac{4}{800 \times 10^3} *$ [(MOVLW, MOVWF (init.)) 1 + 1 +

199 * (1 + 1 + 1 + 2) +   (NOP NOP GOTO)

200 − 1 ← 199

1 + 1 + 2] (NOP NOP decfsz)   (counter = 1 ← 199 بعد)

"skip Jump لما يصير" (decfsz)

* initialization → لازم أشغل loop .

= 5 Ms * [2 + 199(5) + 4] = 5.005 ms .

* Subroutine مع جزء code إذا كان الـ

Return + Call

2Tinst + 2 Tinst → (+4) → delay = 5 Ms * 1005 = 5.025 ms .   (1001)

Q: write subroutine to generate 10 ms delay including the call inst. Assume: Fosc = 400 kHz .   (S21)

Delay = $\frac{4}{400 \times 10^3}$ (= 4×10^{-5}) * num of inst .

↳ 1000 = 2 + 2 +   (call, init. +1)

(x − 1) * (1 + 1 + 1 + 1 + 2) +

1 + 1 + 1 + 2 + 2   (NOP NOP NOP decfsz return)

1000 = 4 * (x − 1) * 6 + 7

= 4 + 6x − 6 + 7

= 5 + 6x

x = 995 / 6 = 165.8

int العدد الصحيح

4 + (164) * 6 + 7

= 995

```
Subroutine     MOVLW   x        (init)
               MOVWF   Counter
Loop           5 NOPS
               NOP
               NOP
               NOP
               DECFSZ  counter
               GOTO    loop
               return
```

Subject "generating time delays." Date No.

((nested loop))

Slide 21

Fosc = 4 MHz ⟶ Tinst = 1 Ms. ($\frac{4}{4MHz}$)

Delay = Tinst * # of Tinst.

10×10^{-3} = 1 Ms * # of Tinst.

⟶ # of Tinst = 10.000

Count H — count L — * Subro: call / return

13 — 250

1 (الأولى) — 11 — 1 (الأخيرة) — 0 — 1

12

⟶ call + initia. 2 + 5 + (لما أ قبل ال loop)

[249 * 3 + (2 + 1 + 2)] + ⟶ countL 0, countH 12

(decfsz 1, 2 goto; decfsz, decfsz, GOTO)

11 * [255 * 3 + (2+1+2)] + (countL 0 ⟶ 256 مرة تانية)

255 * 3 + 2 + 2 + 2 (decfsz, return, decfsz) — 0-1 = FF = 255

= 10,000

* Indirect addressing:

ex: Write a code to clear the values in address [0X10 – 0X4F] 64 locations.

[CLRF f]

```
CLRF 0X10
CLRF 0X11
  :
CLRF 0X4F
```

(inst)

① literal ⟶ inst — Value ins. direct address / address; data Memo. indirect address. address

② * Direct ⟶ data memo / inst

③ * indirect ⟶ FSR.

* To do any address manipulation like loop on the address "indirect addressing"
  (1) write the address in FSR
  (2) Replace parameter f by either INDF or 0X00

* Write code to complement the value in address 0X22 using direct & indirect

XOR ← bits change
COMP ← address " "

Direct

```
COMF    0X22, 1
```

indirect

```
(1) { MOVLW   22
      MOVWF   FSR
```

address → FSR

→ INDF or 0X00 isn't a physical reg., it's just a signal to tell the CPU to get the address from FSR

0X00 ; 1 → comp

```
        MOVLW   D'64'
        MOVWF   counter
        MOVLW   0X10
        MOVWF   FSR
loop
        CLRF    INDF, 1      (or → 0X00)   10 - 11 - 12 - - -
        INCF    FSR, 1       → 11
        DECFSZ  counter, 1
        GOTO    loop
```

```
MOVLW   0X95
MOVWF   FSR
MOVF    INDF,0
```

8bits: FSR ← address

* bank selections:
  - Literal : NO BS.
  - Direct : status (5).
  - Indirect : FSR

* Indirect:

```
BSF     FSR,7
MOVLW   0X15
MOVWF   FSR
MOVF    INDF,0
```

0001 0101

Indirect:

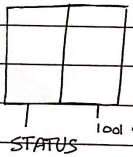
```
MOVF    X10,0
MOVLW   0X15
MOVWF   counter
MOVLW   0X10
MOVWF   FSR
MOVF    0X10,0
```

* Direct:

```
BSF     STATUS,5
MOVF    0X15,0
```

1 0001 0101 (bit 5)

Ex: Write code to read the value in address 0X95 using both direct & indirect. 0X10 to 0X1F result(0X20)



16 address

```
        MOVLW   D'15'
        MOVWF   counter
        MOVLW   0X11
        MOVWF   FSR
loop    ADDWF   INDF,0  => FSR => 10+11
        INC     FSR,1   -> +1
        DECFSZ  counter,1
        GOTO    loop
        MOVWF   0X20
```

NOTEBOOK

* Direct :

```
MOVF    0X10, 0
ADDWF   0X11, 0
ADDWF   0X12, 0      loop
  :
ADDWF   0X1F, 0
MOVWF   0X20
```

(15)

0X20 ← 0X10 + 0X1F

• int squares [6] = {0, 1, 4, 9, 16, 25};

cout << squares [3]; → MOVLW 3 / call squares [3]

* look up table : array in the prog. memo. MOVLW 5 / call squares [5]

PC → prog. Code 35 inst.

inst → squares ADDWF PCL, 1

PC → retlw 0 *

+1 → retlw 1

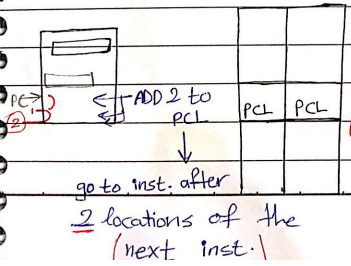
+2 → retlw 4

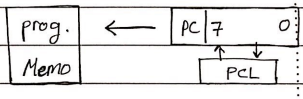
+3 → retlw 9

→ retlw D'16'

→ retlw D'25'

تعديل PCL على قدر working الإدخال





PC isn't writable or readable directly. However I can read & write PCL

NOTEBOOK

* int squares2 [6] = {9, 16, 25, 36, 49, 64};

when I write a value in the W-Reg. & call the lookup table, it returns the square of the value. the lookup table it contain squares of (3-8).

```
        MOVLW   7
        call    squares2
squares2  MOVWF   Temp
          MOVLW   3
          SUBWF   Temp, 0
          ADDWF   PCL, 1
          retlw   D'9'
          retlw   D'16'
          retlw   D'25'
          retlw   D'36'
          retlw   D'49'
          retlw   D'64'
```

Temp=7
W=7-4

PC → retlw

1	
2	6
3	
4	

(1) PCL (3) ADDWF

Subject CH6 Date No.

Interrupt : Code

start 0004

ends retfie return from interrupt

invoked HW event

main code

0004 PC ← retfie

has higher priority than main code.

Subroutine

Lable

return, retlw

call

* polling: we should check for sth. many times.

In PIC → all interrupts are maskable "by default disabled"

* interrupts 1- maskable 2- non maskable.

slide 6.

* For an interrupt to happen:

pc = 0004 & interrupt occurs

1. GIE = 1 global interrupt.
2. Local enable = 1
   - set & cleared by SW.
3. interrupt Flag = 1
   - set by HW. event

In PIC there are 4 types of interrupt. However, we have single interrupt vector = 0004

local enable flag.

CH 6

Subject ______ Date ______ No. ______ الرقم

Interrupt : Code

* 4 types of interrupt :
  - ↳ enable
  - ↳ flag.

→

* start 0004
* ends retfie — return from interrupt
* invoked — [HW event] → main code → 0004 → PC ← retfie

} has higher priority than main code

- on reset the flags are cleared.
- Two flags may be set regardless of enables. However, if the enables are disabled → No interrupt.

* For an interrupt to happen:

by SW. ←
1. GIE = 1 — global interrupt enable.
2. local enable = 1
3. flag = 1 ↳ set by HW. event.

subrotine
* ↳ start lable
* return , retlw
* call

* 4 types of interrupt : "slide 8"

[1] external interrupt.

→ 1 → RBØ /INT

by HW. [INTF = 1 — external interrupt flag

INTE = 1

by SW. GIE = 1

[3] EEPROM write complete interrupt

(WR = 1) start writing.

by HW. ← EEIF = 1 → خلصت كتابة

by SW. ← { EEIE = 1, GIE = 1 }

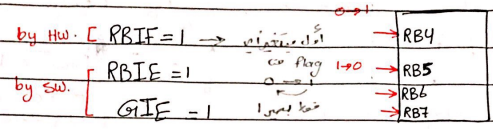
[2] Timer Ø overflow interrupt.

TOIF = 1 ⟹ لما يطفح العداد اللي عدده يصير = ١

→ flag = 1

by SW. { TOIE = 1, GIE = 1 }

4 portB change interrupt.



* To deal with interrupt:
  1) GIE bit (1bit)
  2) 4 local enables (4 bits)
  3) 4 local flags (4 bits)
  4) one bit ↑ ↓ in external (1 bit)

  ] in INTCON except EEIF in EECON1
  ] in OPTION_REG

  interrupt vector

retfie return from interrupt
implicit set GIE

* What the developer should do to deal with interrupt?
  1) Start a 0x0004
  2) ends with retfie.
  3) enable GIE.
  4) enable local enable.

not always. 5) If you deal with portB change interrupt you should clear the flag (RBIF) at the begining of the code.

6) before retfie, don't forget to clear the flag.

* flags set by HW.
  clear by SW.

EX - slide 13.

* OX0A , external interrupt (↑) — +ve edge. — result OX10 ; W in the end zero
OX10 , W - Reg

Sol. ——

```
# include PIC16F84A.INC
        ORG     0000
        GOTO    START
        ORG     0004  → لازم تنكتب
    (1) GOTO    ISR
START
    (3) BSF     INTCON , GIE        → GIE = 1
    (4) BSF     INTCON , INTE       → INTE = 1
        BSF     START , RPØ
        BSF     OPTION - REG , 6    ]  ↑ (rising edge)
b1 →    BCF     STATUS , RPØ        ]
        CLRW
loop    ADDWF   OX0A , Ø   ]
        GOTO    loop       ]
```

* بدي أغير أجع القيمة اللي بـ OX0A لحد ما يصير في interrupt

((خطوة 5 ما إلنا إلها))

```
ISR     CLRW
    (6) BCF     INTCON , INTF
        retfie
```

* بضل بال loop لحد ما يصير interr.

~~for any Reg~~

[Context Saving
slide 15

W - Reg.
```
ISR     MOVWF   Temp.
```

* for any reg. *

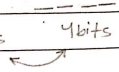
* إذا بدي reg نخزن وقت الـ interrupt

```
10  ISR     MOVF    OX10 , Ø
            MOVWF   Temp
              (10)
            MOVF    Temp , 0
            MOVWF   OX10
        (2) retfie
```

```
        MOVF    Temp , Ø
        retfie
```

MOVF STATUS, ∅
MOVWF Temp

SWAPF Status, ∅
MOVWF Temp



Z = flag

MOVF Temp, ∅
MOVWF STATUS

SWAPF Temp, ∅
MOVWF STATUS

MOVF → flag يتأثر
MOVWF → لا يتأثر

Subject Multiple interrupts. Date No.

Ex: Write code

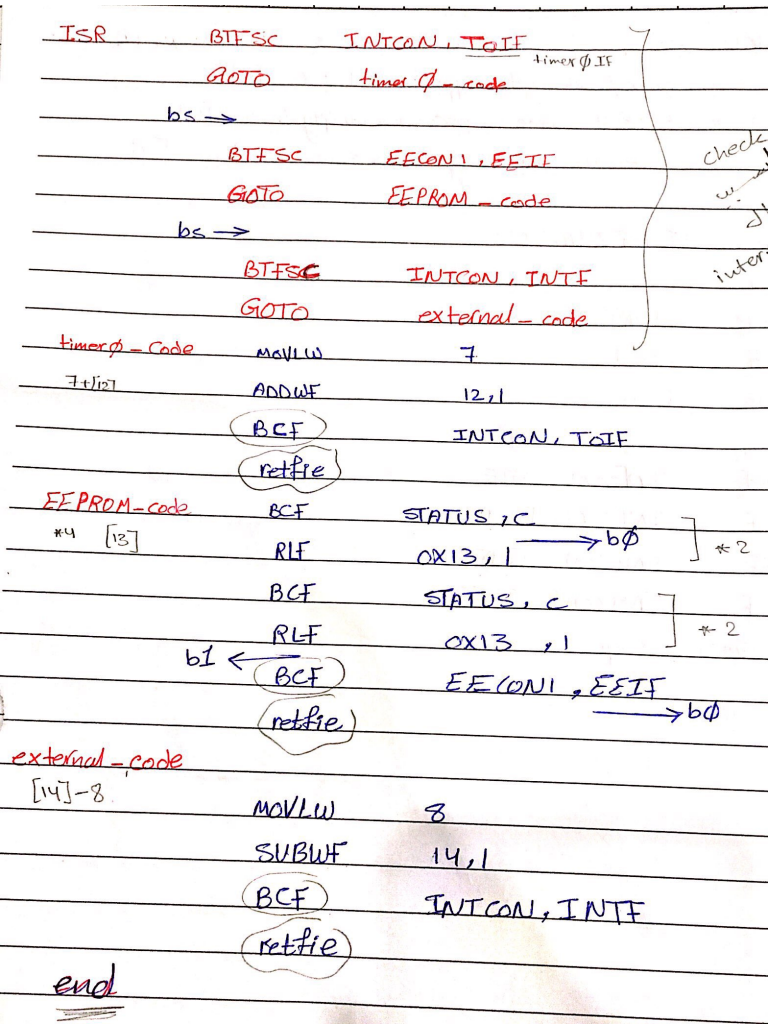
when timer0 interrupt happen → add 7 to [12]

" EEPROM write complete interr → multiply by 4 [13]

" external int happens → Sub 8 from [14]

```
# include  PIC16F84A.INC
    ORG     0000
    GOTO    START
    ORG     0004
    GOTO    ISR
START                          ← interrupt vector
    BSF     INTCON , GIE   → GIE = 1
    BSF     INTCON , INTE
    BSF     INTCON , T0IE  → enable
    BSF     INTCON , EEIE
loop GOTO   loop
```

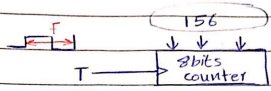
* عشان البرنامج يرجع على الكود إلا اذا
جاء interrupt غير هيك ما يرجع
ويبقى اسير.



Subject Counters & Timers Date No.

* Timer Ø is ((8)) bits counter:

initial value
TMRØ (01)
OPTION-REG (81)



TOIF = 1

* To use counter as timer:

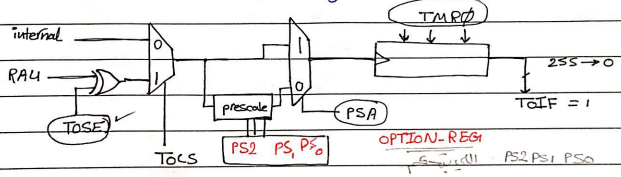
① $T = \frac{1}{f}$

if you know that F = 1MHz.

→ Interrupt happens after $\frac{256}{1MHz} = 256\ \mu s$.

② to make interrupt happen after 100 μs.

start counting from 156.



f	prescale by	f/a
	16	

16 cycles ← 1 cycle

delay

* prescale value = $2^{PS2\,PS1\,PS0+1}$

* PSØ → 0 No prescale
→ 1 Yes.

Clk → prescaled / not

Ex:

If PSA = Ø & PS2 PS1 PSØ = 101

→ prescale = 2^{5+1}

= 64

64 cycles

* Delay of TMR0 = $\left(\frac{4}{Fosc} * prescale\right) * (256 - IN)$ (IN: initial value)

→ max delay of TMR0 = $\frac{4}{Fosc} * 256 * 256$

Ex 8 Write code to generate 10 Ms delay using timer 0 $Fosc = 1\ MHz$.

$10 \times 10^{-3} = \frac{4}{Fosc} * prescale * (256 - IN)$

$10 \times 10^{-3} = \frac{4}{10^6} * pre * x$

→ $(prescale * x = 2500)$

assume prescale = 4 → $x = 625$ ~~256~~ → max 256

" = 32 → $x = 78$

$IN = 256 - 78 = 178$.

Ex: write code to increment the value stored at address 18 every 1 second. use timer 0 & interrupt.
Assume Fosc = 10 MHz.

* max delay = $\frac{4}{Fosc} * 256 \times 256$

$= \frac{4}{10 \times 10^6} * 256 \times 256$

$= 0.026$ sec → 26 MS.

* Delay = 10 MS * 100 times.

$10 \times 10^{-3} = \frac{4}{10 \times 10^6} * pre * x$

$pre * x = 25000$

prescale = 256 → $x = 97$ → IN = 256 − x = 158.

* Delay = $\frac{4}{Fosc} * Pres * (256 - IN)$

$1 = \frac{4}{10 \times 10^6} * Pre * \frac{x}{(165 - IN)}$

$Pre * x = \frac{10^7}{4} = 25 \times 10^5$

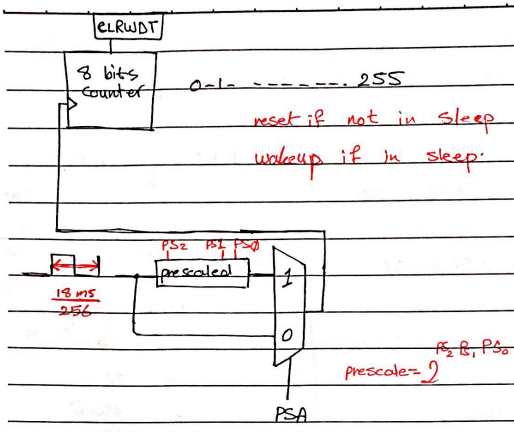
Assume pre = 256

→ $x = \frac{25 \times 10^5}{256} = 97.65$

```
ORG 0000
GOTO START
ORG 0004
GOTO ISR
START BSF INTCON, GIE
      BSF INTCON, TOIE
      MOVLW D'100'
      MOVWF counter
      MOVLW D'158'
      MOVWF TMR0
      MOVLW B'XXXX0111'   ; PSA used for timer0
      MOVWF OPTION_REG    ; prescale selection
LOOP  GOTO LOOP           ; loop
ISR   BCF INTCON, T0IF
      MOVLW D'158'
      MOVWF TMR0
      DECFSZ counter, 1
      RETFIE
      INCF 18, 1
      MOVLW D'100'
      MOVWF counter
      RETFIE
```

1 sec

Subject "Watch Dog Timer". Date No.

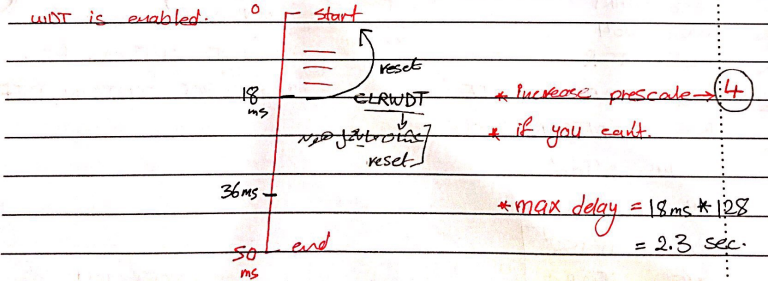


* Delay until reset = $\frac{18\ ms}{256} * 256 * prescale$.

$= 18\ ms * prescale$

- without pre scal → delay = 18 ms.

WDTE from Configuration word.



Subject "Sleep Mode" Date No.

* Sleep: mode the CPU enter it by excuting.

SLEEP instr.

* بما أنها بتطفي osc.
clear WDT
وبوقفها تنفيذ أي
اشي بالمره

* WDT keeps counting in sleep mode because it has it's own osc.

Timer ∅ is off, can't wakeup MC from sleep.

. you cant write code to wake it up.

SLEEP

* To wakeup the CPU:

1- $\overline{MCLR}$ = 0 → reset vector

2- WDT if enabled. → PC

3- Interrupt flag = 1 & its corresponding enable = 1 regardless GIE. → PC if GIE = 0, or if GIE = 1

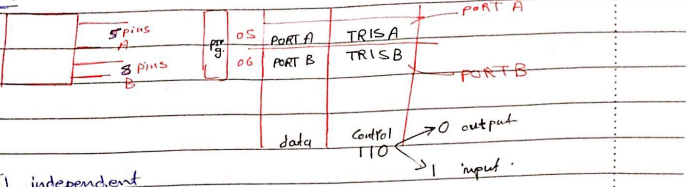
* Write code to make the CPU sleep for 2.3 sec.

2.3 = 18 ms X prescaled
prescale = 128

```
MOVLW B'xxxx1111'     ; PSA, 111 -> 128
MOVWF OPTION_REG      ; <- * bank switching.
SLEEP
MOVLW B'xxxx11110'
```

Subject CH3. Date 14/11 No.

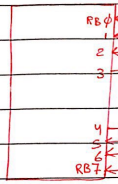
Slide 4



[X] independent

half duplex → input or output but not at the same time.

Slide 7



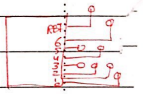

```
BSF     STATUS, RB0
MOVLW   B'1110 0111'   ← control
                out  in
MOVWF   TRISB
BCF
```

Slide 8

ex①

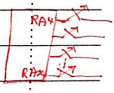
```
b1
MOVLW   00
MOVWF   TRISB      } CLRF TRISB
b0
MOVLW   0XAA
MOVWF   PORTB
```

ex→ LEDs.



ex②

```
b1
MOVLW   B'XXX1 1111'        MOVF    PORTA, 0
MOVWF   TRISA               MOVWF   0X0D
b0                          CLRF    PORTA   writing to Reg.
```

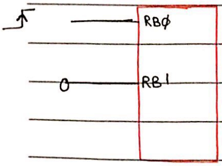


portA & portB will not affect input pins. affects only output pins.

NOTEBOOK

Can write to PORTA using inst. MOVLW

slide 9   +ve edge , external interrupt.
"   10




```
BCF   PORTB , Ø
BSF   PORTB , Ø
```

ما يعمل interrupt.

* بدي بس تجي external interrupt يشتغل الضو وبس تجي
واذا كان يطفي الضو.

Flash [*] → 0 : not flashing
→ 1 : flashing.

```
ISR     MOVLW    0X01
        XORWF    Flash, 1

        BCF      INTCON, INTF
        retfie.

START   BSF      INTCON, GIE
        BSF      INTCON, INTE
        → b1
        BSF      OPTION_REG, Ø
        MOVLW    B'XXXX XX01'
        MOVWF    TRISB
        → bØ
loop    BTFSS    flash, Ø
        GOTO     loop
        MOVLW    B'0000 0010'
        XORWF    PORTB, 1
        delay 1 sec.
        GOTO     loop
```

(INTCON, GIE / INTCON, INTE / OPTION_REG, Ø → rising edge)

I/O (MOVLW B'XXXX XX01' / MOVWF TRISB)

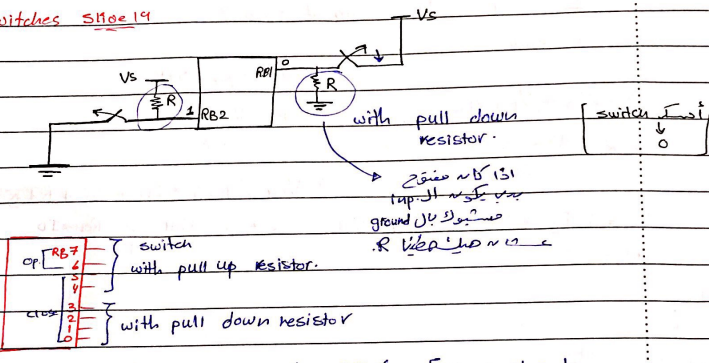
* turn on led & / " off. (XORWF PORTB, 1)

NOTEBOO

* Slide 16-17 → H.W. X

* Slide 18 ((* electrical char. *))

* Switches Slide 19

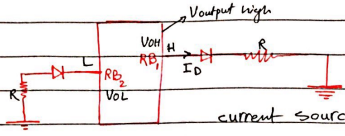


switches connected to RB0 — 5 are closed & remaing are open.

Read switches state into Reg. 0X1A.

```
b1
MOVLW   B'11111111'
MOVWF   TRISB
b0
MOVF    PORTB,0          [0X1A] = 1100 1111
MOVWF   0X1A
```

5-27



* VOH + ~~~ series
VOL + ~~~

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

$V_S = RI_D + V_D + V_{DL}$ $\rightarrow R = \frac{V_{OH} - V_D}{I_D}$

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

$V_{OH} = R_{OH} * I_D + V_D + RI_D$

$V_S = RI_D + V_D + R_{OL} I_D + V_{OL}$ $\rightarrow R = \frac{V_{OH} - V_D - R_{OH} * I_D}{I_D}$

$\rightarrow R =$

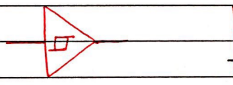
24/11

PORTA } data
PORTB }

TRISA } control
TRISB }

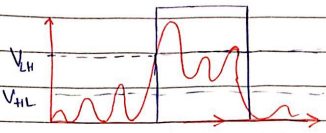
0 → output
1 → input.

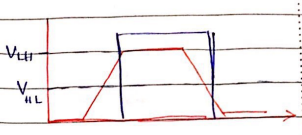
* Schmitt Trigger input.



* V_{LH} : if initially the output is "low", to convert into "High", the input voltage go above (V_{LH-}).

* V_{HL} : " " " " " "high", " " " "low" " " " " below (V_{HL}).



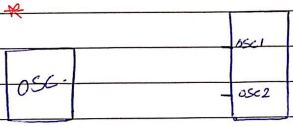


* benifits:

1- cancels noise.
2- reduced undefined region.
3- fast switching.

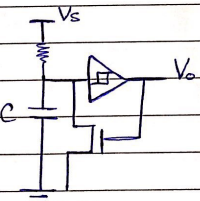
* The Oscillator *

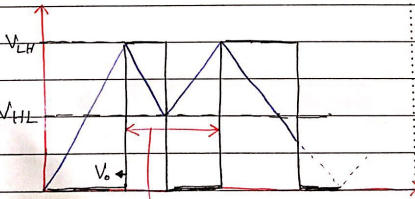
" slide 29".



*code or settings: Configuration word.

V_H: on $V_g = V_H$ → short circuit.





slide 30

① RC osc.

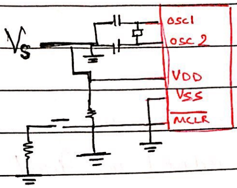
$\tau = R * C$ T E B O O K

② crystal osc.

* The PIC 16F84A OSC.

Fosc1 ; Foscϕ

↳ slide 31 . مهمة
مش حفظ



* The power Supply *

slide 34.

جدول V و I لكل PIC

* OSC2 → إذا مش موصل بإشي بتكون RC.

Subject CH 10 · Date No.

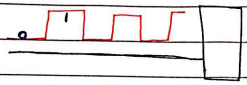
Serial

16F84A
16F877A — same family.
↳ Serial port.

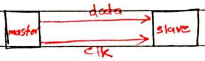
formatted data
stop bit
start bits
parity

Clock ((protocol)) Clock
some rules

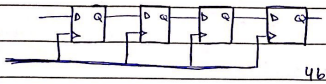
Slide3 parallel, serial.



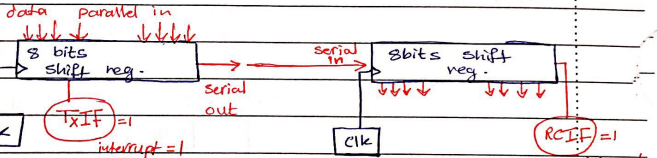
① Asynchronous
② synchronous — up! clk ji jaye



* Serial: shift registers.



4bits.

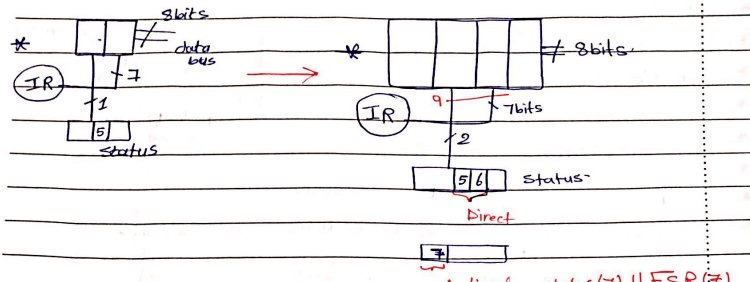


transmetion
Tx

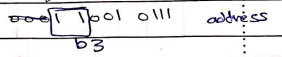
Reception
Rx

Slide 11→16 skip.

Memo:



* Write code to read the address 0x107 using direct & indirect.



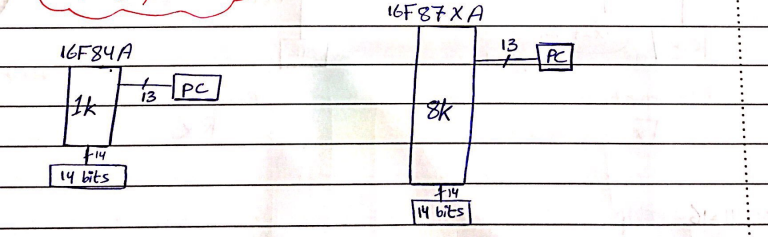
Direct

```
BSF   STATUS, 5
BSF   STATUS, 6
MOVF  0X17, 0
```

Indirect.

```
BSF   STATUS, 7
MOVLW 0X97
MOVWF FSR
MOVF  INDF, 0.
```

* prog. Memo. *



GOTO k — address of inst.
Call k
(k = 11 bits)

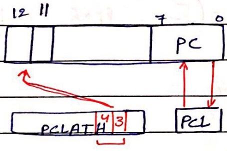
e.g. Assume PC = 0001 & you want to call the Subroutine at address 0X1855

صرنا بدنا بدال الـ 11 bit لل call / Goto
بدنا أكبر من هيك

Call k
Call 0X1855 → compiler → | 3bits | 000 0101 0101 |

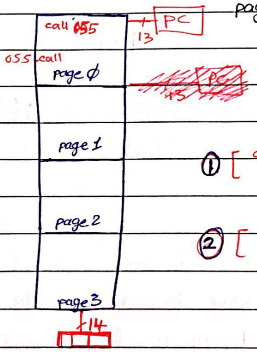
(11)

بنستخدم زي قبل banks بس هون
اسمها pages



Pclatch

BSF PCLATH , 4
BSF PCLATH , 3
Call 0X055

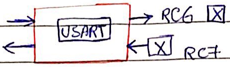


① [data Memo address → bank

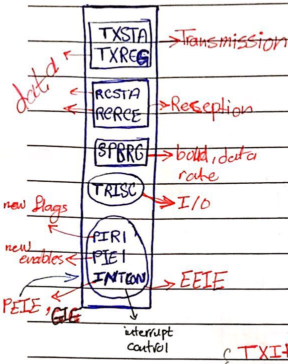
② [Call / GOTO] → page selection

USART

universal sync. Async. reciever transmission.



* Async. → بيرسل وبيستقبل



* if "TSR" is empty, the word writen in "TXREG" moves to "TSR".

* 2 enables:
  - . TXEN = 1 → Tx only.
  - . SPEN = 1 → for Tx & Rx.
    (serial port enable)

* baud Rate = F(Fosc, SPBRG, BRGH bit)

- TXIF = 1 → if TXREG is empty.
- TXIF = 0 " " " full.
- TXIF ⟹ it's Read only
- TRMT = 1 if the shift reg. is empty.

Reciver

recive shift reg. ← RSR not accessable by SW. (no address)

RCREG is accessable

↳ queue of 2 words. ↳ FIFO . 1st in 1st out..

TX9

TX9D bit(9)

Rx9 = 1

RX9D = bit(9)

2 enables:

continuous reception enable

- CREN = 1 → reception only
- SPEN = 1 →

.. 2words بيستقبل

* أول ما أبعت "word" بيروح على (RCREG)

RCIF = 1

NOTEBOOK

*baud Rate = F(Fosc, BRGH, SPBRG)

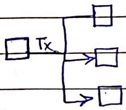
*if the 3rd byte gets recieved before read the previous. « over run error » OERR

2 bytes → OERR = 1.
↳ read only.

RCREG → 2 words only.

→ as long as : OERR = 1
① reception stops. ② 3rd byte is lost.

* How to prevent overrun error ?
read the byte when received.



* OERR can't be cleaned by SW. (read only)

* To clear it :
↳ ① CREN = 0 → OERR = 0
② read the data in RCREG.
MOVF RCREG, 0 read 1st byte
MOVF RCREG, 0 " 2nd byte
③ CREN = 1

stop bit = 0 → framing error.
بس ما بوقف للإستقبال.

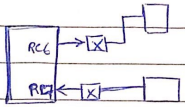
* بنوقف الاستقبال بعدين بس نرجع بنشغله وبنستقبل اللي بعده.

* baud Rate = $\frac{Fosc}{*(1+SPBRG)}$

high speed ← BRGH = 1 ← 16
low " ← BRGH = 0 ← 64
BRGH: baud rate generating high

Subject rec 27 Date 1-12 No.

* same data Rate (same reg, Fosc.)



Ex:
* 40, 41, 42
no parity
9.6 k bps
Fosc = 20 MHz

slide 24

لازم يبدأ TXEN = 1

asynchro. 8 sync. high = 16 = ①

* TXSTA = X0100X0X
* RCSTA = 1XXX XXXX
* SPBRG = 129
* TRISC = X0XX XXXX

• code → slide 34.

$$9.6\times10^{3} = \frac{Fosc}{\frac{16}{64} * (1+SPBRG)}$$

$$9.6\times10^{3} = \frac{20\times10^{6}}{16 * (1+SPBRG)}$$

$$16 * 9.6\times10^{3} * (1+SPBRG) = 20\times10^{6}$$

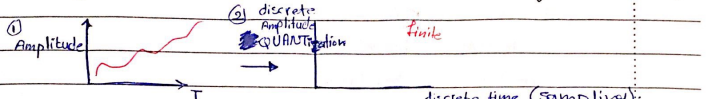
$$\boxed{SPBRG = 129}$$

لازم 16
if x > 255
لازم 64

Subject CH11 Date No.

"Data Acquisition & manipulation"

* Digital : discrete, limited num of values in a range.
* Analog : infinite num of values within a range. , Continuous.



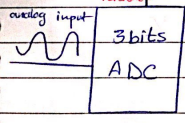
Analog signal Digital

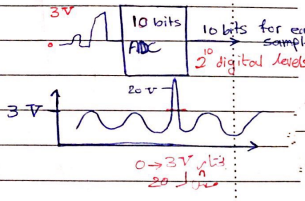
* more accurate → less accurate
* suffer from noise (interference) → error.

* Analog to Digital converter

* Voltage references *

1> range of acceptable input.



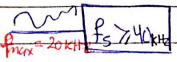


* ADC:

sampling $ts = \frac{1}{fs}$

⊕ * more accurate, more storage and bandwidth.
⊖ time & processing & power.

min $fs \geq 2\,fmax$



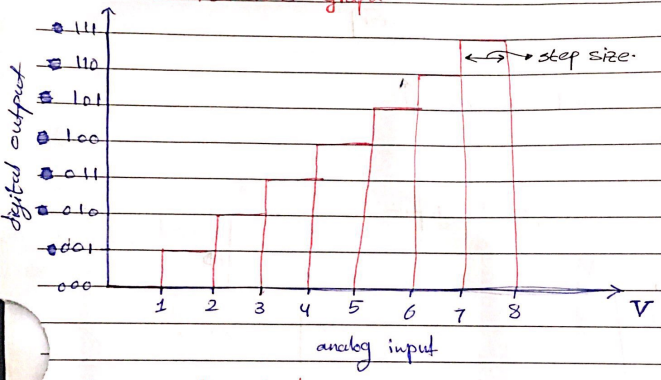
② Quantization:

[n] # of bits.

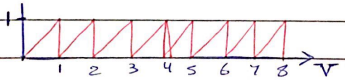
n↑ ⇝ ① more accurate

- storage & band width.
- power, cost, time.

*To do quantization, we usually draw quantization characterestics graph.



*error = input − output.



*max error = 1 V

= 1 LSB

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

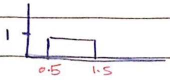
*"resolution" step size = $\frac{Vr}{2^n}$

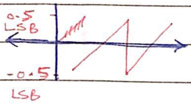
↳ LSB: least significant bit voltage: min change in input that will difinitely change the output digital values.

↳ is a measure of resolution.

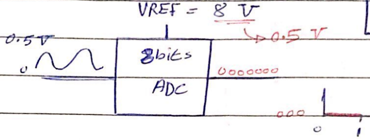
* max error $= \frac{1}{2}$ LSB.

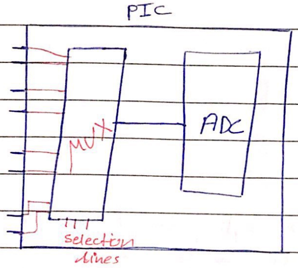
$= \frac{V_r}{2^{n+1}}$



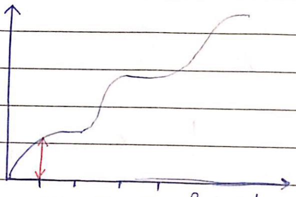


[slide 8
" 11





ما بدي يطلع zero فبتغيّر ال (VREF)



sample & hold.

* taking the sample while the input is changing → error

تفصيلي:

→ fix the input, take the sample, convert it, Release the input

slide 14

- if the switch is closed.

$V_c \approx V_s$.

- to take a sample, we should wait until $V_c \approx V_s$. then open the switch (fix input).

* for how long we should wait until opening the switch

$$V_c = V_s * (1 - e^{\frac{-t}{\tau}})$$

* assume the error = 10%

$\rightarrow V_c = 0.9\,V_s$

$0.9\,V_s = V_s * (1 - e^{-t/\tau})$

$t = 2.3 * \tau$

- max error $= \frac{V_r}{2^{n+1}}$

$V_c - V_s = \frac{V_s}{2^{n+1}}$

$t = -\ln \frac{1}{2^{n+1}} * \tau$ → n ↑ ⇝ sampling time ↑ for 10 bits ADC → $t = 7.6 * \tau$

* for n = 8bits → $t_s = -\ln \frac{1}{2^{8+1}} * \tau$

$= 6.2\,\tau$

* on the pic 16F877A, you can connect 8 possible analog inputs:

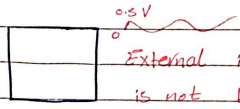
RA0 – 3 RA5
RE0 – 2

→ A/D using RCFG bits in ADCON1

1→ Voltage refrences internal or external

V_{DD}
V_{SS}

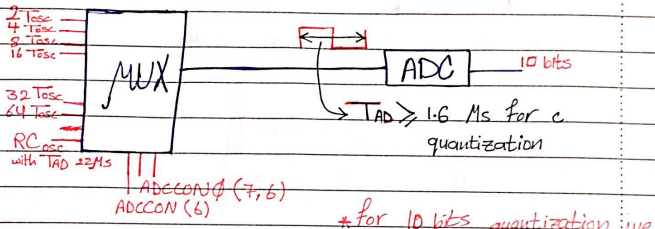
RA3
RA2



External is more accurate if range is not between V_{DD} & V_{SS}.
5V , 0

* Set quantization speed:

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



* for 10 bits quantization we need for 12 * T_{AD}.

* quantization time $= (n+2) * T_{AD}$.

Ex: if Fosc = 1 MHz.

what is the fastest quantization time in PIC16F877A ?

→ $T_{osc} = \frac{1}{1\,MHz} = 1\ \mu s.$

→ $T_{AD} = 2 * T_{osc} \rightsquigarrow 4 T_{osc}.$
$= 2\ \mu s$

→ quantization time $= 12 * T_{AD} = 24\ \mu s.$

Ex: $F_{osc} = 10\ MHz$

$T_{osc} = 0.1\ \mu s$

$T_{AD} = 16\ T_{osc}$

$= 1.6\ \mu s$

$TQ = 12 * 1.6\ \mu s = 19.2\ \mu s$

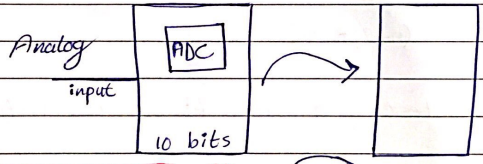
Ex: if $F_{osc} = 100$ * +ts

$T_{osc} = 10\ \mu s$

select RC osc with $T_{AD} = 2\ \mu s$.

GO $\lceil$DONE = 1 open the switch
start quantization.

GO $\lceil$DONE = 0 end the quantization.



10 0000 000 (128)

5B

8bits
left justified

1 * 4 = 4
128 * 4 = 512

- you have to write code for it

configure ADC (1-4) → wait sampling time

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

GIE = 1
step (5) ⇒ wait quantization

12 * TAD
GO DONE = 0
ADIE = 1

NOTE BOOK
GIE = 1
PEIE = 1
ADIE = 1

* configure ADC

(1-4) → wait Sampling time → step (5) → wait quantization time (12 T_{AD})

GO/DONE

you have to write a code for it

(6) read result

- GO/DONE = 0 ← GO/DONE
- ADIF = 1

- for error = $\frac{1}{2}$ LSB

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

sampling time

* Op Amp setting time 2 μs + Tempfuller + $-\ln \frac{1}{2^{n+1}} * (R_S + R_{SC} + R_{SS}) * C$

→ 0.05 μs for each degree above 25°C

* What should be the value of PCFG bits in case you want RE2 to be a external voltage references?

10000

- slide "31"

$R_{SS} = 7k\Omega$, $R_{SC} = 1k\Omega$, $R_S = 0$, CHOLD = 120 pf

Temp = 35°C, $T_{AD} = 1.6\,\mu s$

+ Sampling = 2 μs + (35-25) * 0.05 μs + 7.6 * (+7k + 0) * $\frac{}{120 \times 10^{-12}}$

+ quantization = 12 X T_{AD} → 12 X 1.6 μs = 19.2 μs

→ Time for one sample = 9.8 + 19.2 μs = 24 μs

- $f_S = \frac{1}{32.2\,\mu s} = 31$ kHz

$\frac{f_m}{15.5\ kHz}$ ADC

9.8 μs	19.2 μs	int sample
sampling time	quantization t. (29 μs)	time 2 T_{AD} (32.2 μs)

→ 32.2 μs

E B O O K

* RA0, T_{AD} = 8 T_{osc}.

internal voltage references, Fosc = 20 MHz, VDD = 5V

T = 25°C

right justified.

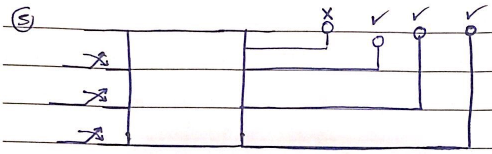
① Tsampling = 10 µs.

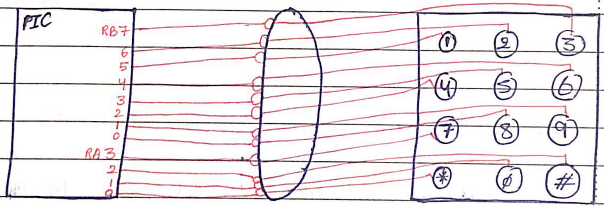
② ADCON0 = 0100 0ØX1

ADCON1 = 10XX 1110

TRISA = XXXX XXX1

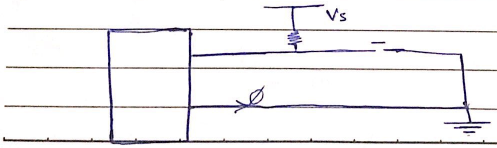
⑤





((12 pins)) save # of pins.

• We saved 5 pins, cost is code completly.



NOTEBOOK

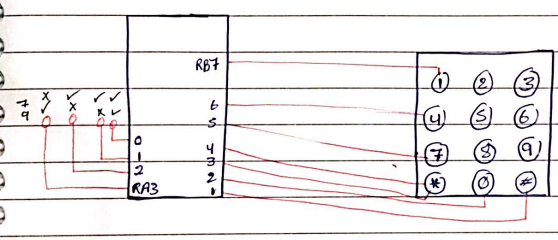
* To read which buttom is pressed:

① read row pins
- a- make row pins input.
- b- " column pins output.
- c- output zero on column pins.

$r_0\ r_1\ r_2\ r_3$: 1101 → row 2

② read column pins.
- a- make column pins input.
- b- make row pins output.
- c- output zero on row pins.

$C_0\ C_1\ C_2$: 110 → Column 2



* ROW_index = 1101 0000 $_2$
* Column_index = 0000 0110 $_0$

3 * 3 + 2 = 11

3 * row_index + Column_index

3 * 2 + 0 = 6

(10)

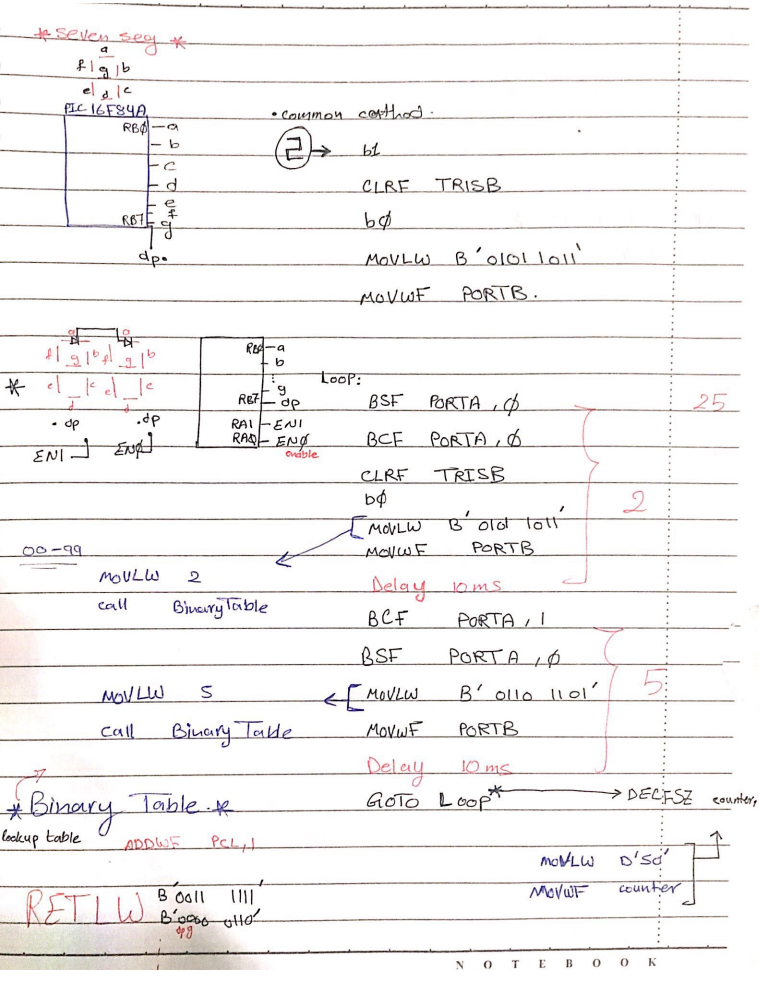
* row_index = 1110 0000 3
* Column_index = 0000 1100 2

3 * 3 + 2 = 11

* portB change interrupt
① the flag should be cleared at the beginning of the code
② To clear the flag:

```
MOVF  portB, 0
BCF   INTCON, RBIF
```

NOTEBOOK



Sensors

- light dependent resistor "LDR"
  - it's a resistor with it's resistance is dependent on light intensity.
  - as light intensity increases, more electron-hole pairs are formed. ⟹ resistance decreases.

* when the light is very luminant.

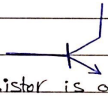
$$V_D = 5 * \frac{R_{LDR}}{R_{LDR} + 90k\Omega}$$

- optical object sensing: . composed of 3 components.
  . works in 2 modes: 1- cut.
  2- reflect.

① IR LED —▷|—

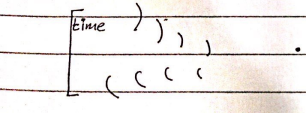
It emits IR when $V_A > V_C + V_{Th}$.

② photo transistor.
when light arrives to the base, the transistor is on.



③ plastic housing: only allows unvisible light to pass.

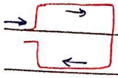
- Ultrasonic object sensor.



. Distance = $\frac{\text{time} * \text{speed vice}}{2}$

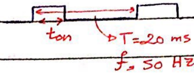
Subject: Motors   Date   No.

DC



- stepper motor
- servo motor

↳ PWM : pulse width modulation.



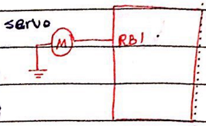
* 40°

→ $1.25 + \frac{1.5 - 1.25}{90} * 40$

* 90°

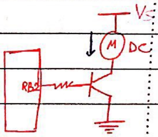
```
loop:  BSF    PORTB,1
       delay  1.5 ms
       BCF    PORTB,1
       delay  18.5 ms
       GOTO   Loop
```

→ احتمال لو عطينا الكود لهاد موتور احتمال ما يشغله.



* To connect Motor to the pic, we usually use the PIC as switch, and the motor is driven by external power suply.

1. BJT "transistor". — Slide (38).
2. MOSFET
3. H-bridge.



- L293D: (4) half bridges: to connect M which rotate in one direction, just to drive M by external power suply.
- (2) Full H-bridges: to make the M rotates in 2-directions.

* RB2 → high

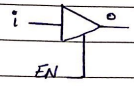
رح يشتغل M باتجاه واحد عند ال I

بنحتاج للاتجاهين بستخدم "H-bridge"

slide 42.

NOTEBOOK

* it has ((16)) pins:
  - (4) pins for ground & heat sink.
  - (4) input pins (1A, 2A, 3A, 4A).
  - (4) output pins (1Y, 2Y, 3Y, 4Y).
  - (2) enable pins.
  - (2) Voltage sources: * V_{LS} : if the input is high. it's converted into V_{LS}.
    * V_{OS}.

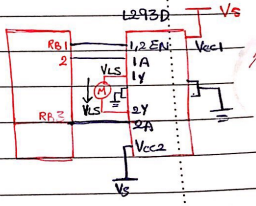


EN	i	o
0	X	Hi-Z
1	0	0
1	1	1

$V_{OS} > V_{LS}$

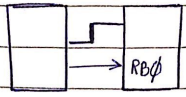
$I_S = 0$ the chip is off

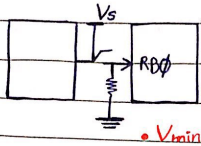
* connect L293D, (M), and PIC 16F84A such that when
  - RB1 = 0, the motor is off
  - RB2 = 1 & RB3 = 0 ⟶ ↓
  - RB2 = 0 & RB3 = 1 ⟶ ↑
  - write code to make the current ↑




```
BSF   PORTB , 1
BSF    "    , 3
BCF    "    , 2
```

• more in digital input s





INCF counter, 1
BCF INTCON, INTF
RETFIE

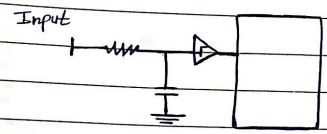
• $V_{min} = -0.3 - 20 * 10^{-3} * 1 * 10^{3} = -20.3\ V.$

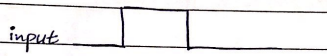
• $V_D = 0.3\ V$
• $I_D = 20\ mA$
• $R_{protection} = 1\ k\Omega$.

* What are the maximum & min. Voltage that can enter?

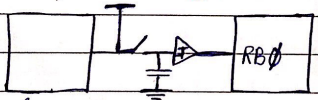
$= (1 * 10^{3})(20 * 10^{-3}) + 0.3 + 5$

$= 25.3\ V.$





* Switch Depouncing *



OPTION_REG(7).

loop

delay 1 ms
BTFSS PORTB, 1
GOTO loop

NOTEBOOK

Subject CH9 Date No.

Timer0 → T0IF = 1
WDT → reset, wakeup.

* Timer1 *

① 16 bits counter ((① TMR1H, ② TMR1L))

② on overflow → T1IF = 1 , T1IE = 1
GIE = 1 , PEIE = 1

③ if TMR1ON = 0
Timer will not count.

④ you have an option to synchronise the clock T1Sync = 0

⑤ prescale value = 1, 2, 4, 8 slide7

⑥ clock may have 3 sources:
[1] internal (Fosc/4)
[2] external
→ RC0 ((T1OSCEN = 0)) → for counting
→ RC1 ((T1OSCEN = 1)) → for external osc. up to 200kHz.

⑦ Timer1 will keep counting in SLEEP mode when the clock source is RC1, and may wakeup the PIC.

* Delay of TMR1 = $\frac{4}{Fosc}$ * prescale * $(2^{16} - IN)$

→ IN = 13300 ✓

↓

1111 0000 , 0000 1010

TMR1H TMR1L

lect 34

slide7

- Delay = $\frac{4}{Fosc}$ * prescale * $(2^{16} - IN)$
- max delay of T1 = $\frac{4}{Fosc}$ * 8 * 2^{16}

8 * max delay of TMR0.

Ex: What are the required settings to generate 100ms delay using Timer1, assume Fosc = 4 MHz.

sol

$100 \times 10^{-3} = 4/4 * 10^{6} *$ prescale * $\underbrace{(2^{16} - IN)}_{x}$

pre * x = 10^5 → pre = 8 → x = 12,500

→ IN = $2^{16} - 12,500 = 53,036$

= 0XCF2C

* Timer2 *

1. 8 bits counter
2. count from 0 to PR2.
3. prescale values 1, 4, or 16
4. only internal clock ($\frac{Fosc}{4}$)
5. There is postscale

if postscale = x → when Timer2 value = PR2 value.
x num. of times → TR2IF = 1 PIE = 1
TR2IE = 1 PEIE = 1

* Delay = $\frac{4}{Fosc}$ * prescale * postscale * (PR2 + 1).

. max delay of TMR2 = $\frac{4}{Fosc}$ * 16 * 16 * 256

= max delay of TM0.

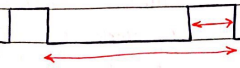
Ex: what are the settings to generate 20 ms using timer2 assume Fosc = 4 MHz.

* Capture / Compare / PWM modules. *

CCP module: it can work in one of 10 modes.

① off

② PWM (Timer2)



③ capture modes when an event happens read the time. (Timer1)

1 — every falling edge on RC2.
2 — " rising " " "
3 — 4th " " " "
4 — 16th " " " "

CCP1IF = 1
CCP1IE = 1
PEIE = 1
GIE = 1

④ compare modes when the time = value → do an event
ex: alarm

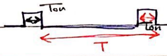
* CCP1CON , CCP1H , CCP1L *

- do nothing.
- trigger special event
- set RC2.
- clear RC2.

} CCP1IF = 1

Ex: what are the required settings to set RC2. after 50ms assume Fosc = 4 MHz, and using CCP1 module

compare
CCP1CON
~~TMR1ON~~
CCP1H
CCP1L

PWM 

- RC2 → output
- Timer ②

* as timer 2 is counting it's compared with 2 values
① CCP1H concatenating with CCP1CON <5:4>
↳ to control "Ton" : increase it.

* when CCP works in PWM there exist an internal 2 bits counter that are connected with TMR2 value.

→ when they are equal → reset RC2 

② PR2 → to control "T"
when they are equal
1- tests timer 2
2- set RC2
~~3- latches CCP1L into CCP1H.~~

there is no ~~postscale~~ in PWM

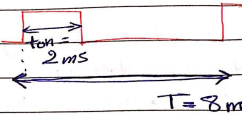
* if PR2 < CCP1H
- Timer2 will count to be equal to PR2.
- NO PWM sig., it will never be equal.

. بنحط ال value بـ CCP1H عشان بـ يخلص عد ويتبعه اوله بيقل غيره

CCP1L → CCP1H عشان نتحكم بال ton.

* $T = (PR2+1) * T_{osc} * 4 * \text{Timer2 prescale}$.
* $t_{on} = (\text{10 bits value}) * T_{osc} * \text{Timer2 presale}$
↳ CCP1L // 2 bits from CCP1CON.

Ex: Write a code that generates the following sig.
assume Fosc = 4 MHz.



using ECPI.