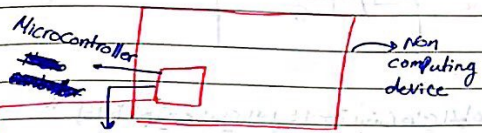
Embedded system is:



Computer system embedded in non computing device → Control non computing device

Now it's called smart device

device ... one or few dedicated functions.

real time constraints → (Action) ... deadline

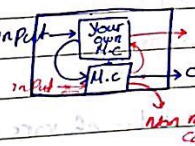
↳ There are deadlines the Embedded system should stick to them.

real time constraints ... Embedded system ...

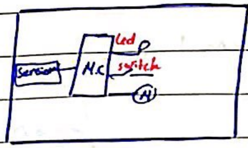
How to start modifying an Embedded system ?

1) You need to understand inputs and outputs.
2) User interaction (how the user interact with the Embedded system)
3) link to other systems
4) Hardware (Embedded system what contains)
5) Software (code)

↳ Do you have access to the code ? → Yes → take backup & enjoy
→ No

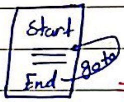


Software driven



" إذا انا اجيت ببدي اعمل ثلاجة ببعض هاي المواصفات بركبت الدائرة
هل رح تشتغل ؟

Code



لا، ما رح تشتغل إلا إذا نزلت عليها الكود

⇒ Always running

Reliable : هنا الجهاز بعطي نتائج صح

Almost gives right results (100% times)

if it's failed there is backup

Example Slide 8 :

* If the Actual temperature larger than required temprature the compressor will turn on (بعمل على تبريد الثلاجة)

* if the Actual smaller than the required temp the compressor will turn off

Main Components of Computer (M.C)

- → CPU : control & excute code
- → I/os : to deal with external world
- → Buses : to connect different components
- → memory : Storage
  - → data memory → hold value of vars (volatile: lost on Power down)
  - → Program memory → holds code intr (Permenent)

Scanned with CamScanner

Code

```
void name int ( ) {
int x, y;
cout ...
--
cin
--
--
x = ;
y = ;
}
```

→ Compiler (translate the high level language to machine language)

if the syntax correct
→ machine code
↓
M.C
↓
Program memory
↓
CPU will run the code
↓
At run time the CPU stores var values in the data memory

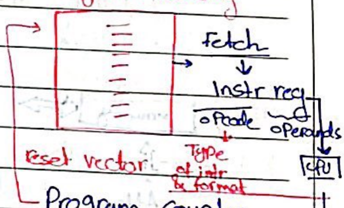
data memory volatite?

* عشان بتخزن قيم الـ variables وكدا وكدا بتتغير او بتتمسح

Program memory permenant?

* لأني طبعًا بحاجة للكود عشان ما احتاج ادرج او اعيد البرنامج

Program memory



ليش ما في permenant؟

To write the data memory it takes less cost (Power & time) عكس الـ program memory

← بس الـ Program memory بتنكتب مرة وحدة عكس الـ data memory

Program counter (holds the address the next inst to be excuted)

e.g if PC: 7 → will excute intr address 7

on power up or reset

PC ← default value called reset vector

⇓

Auto increment

data memory

I/O

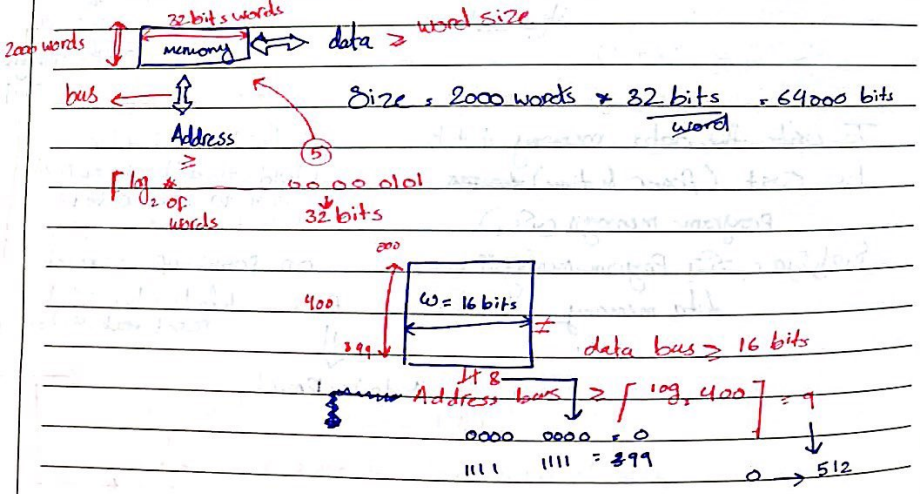
* While executing the current inst the CPU is reaching the next one. (pipline)

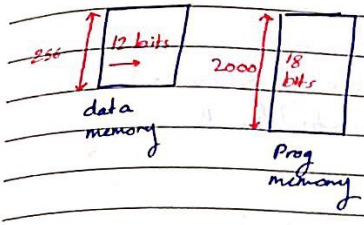
The Von neuman Architecture:
Only 2 busses
(Pauls Jobs Bill)
Tuffling

The Harvard Architecture
2 busses for data memory
" " " I/Os
" " " Program memory

Note
bus
data highway
is a communication system that transfer data between components inside a computer or between computers

* Von neuman Architecture
  ↳ less busses less complex
* Harvard Architecture
  ↳ more complex but faster
  Pipelining more efficient





		data memory	Program memory
Human Arch:	data bus	12	18
	Address bus	$\lceil \log_2 256 \rceil = 8$	$\lceil \log_2 2000 \rceil = 11$
VNA	data bus		
	Address bus	Max {12, 18} = 18	
		Max {8, 11} = 11	

* الاسبوع الثاني *

There are to type of instruction set

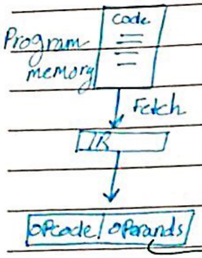
Cisc

* Every operation there is instruction will excute this operation ↓ to many instruction Including complex
* ~~shorter code~~ ~~longer~~ shorter code
* Faster compilation
* many formats for each instruction & variable size instruction. many Addressing mode

Risc

* only simple instruction will build the program And few format
* There is no complex instruction And you need to built it
* ~~shorter~~ code → longer
* Faster execution due mor due to more efficient piplining (same fetching time & execution time (we will use this to write instr in this course)

* Addressing mode ?

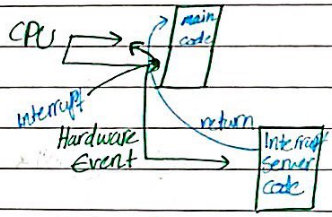


Addressing mode : How to define the var.

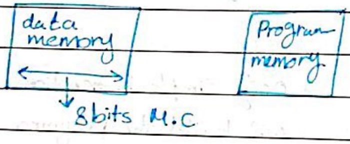
How to define this?

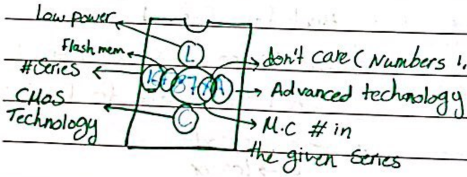
↓ Addressing mode

Interrupt :-



Single working register (Accumulator)
holds the result of the last operation.





بالسوق ما بكون
87 X
ال x بتكون رقم (1,2,3...)

Serial Port ⟹ 16F87xA

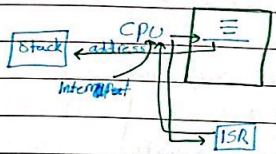
Stack :- Volatile memory

↳ Automatically written & read.

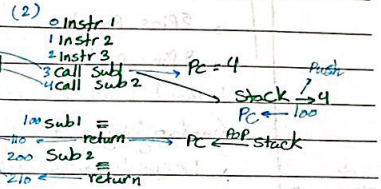
it hold the return Address (interrupt handling) e.g

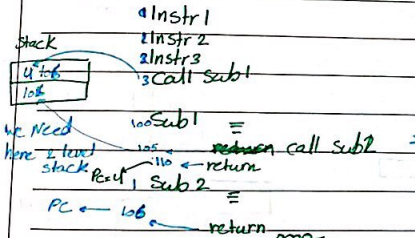
E.g :-

(1)



(2)





⇒ Nested call

if we have 8 stack levels

⇓

8 Nested Call

Stack 8 first in last out

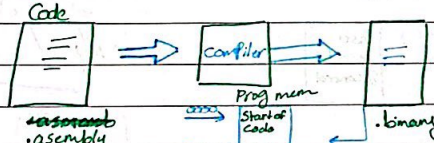
Interrupt vector :- (reset vector) → default value of pc on reset

↓ 0004 ↳ default value of pc on interrupt

Single interrupt vector :

Address ...

eg 5 interrupt types but Single interrupt vector

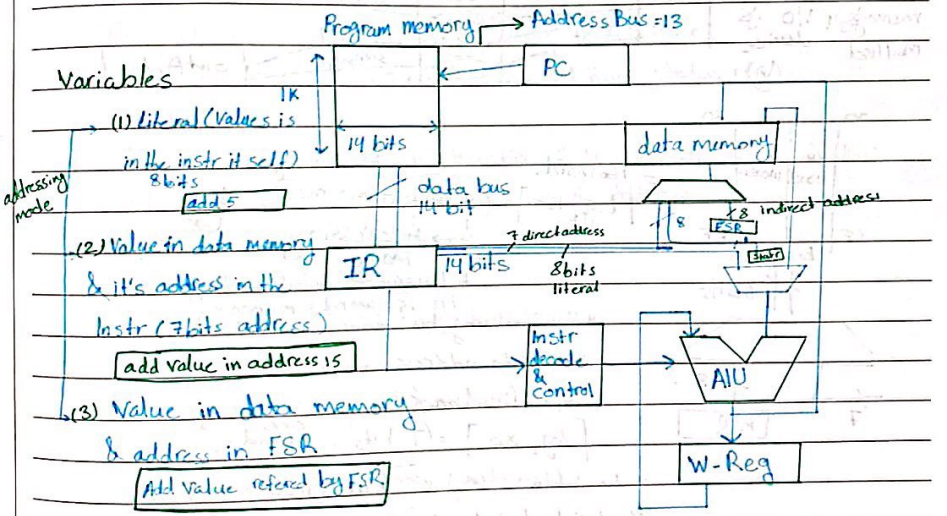


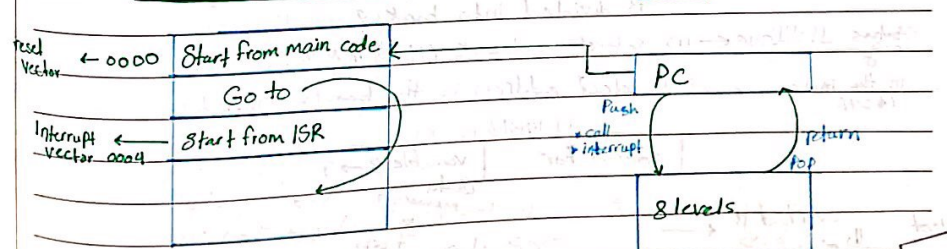
Should write code to check the reason behind interrupt

Chapter 2:

Pic 16F84A:

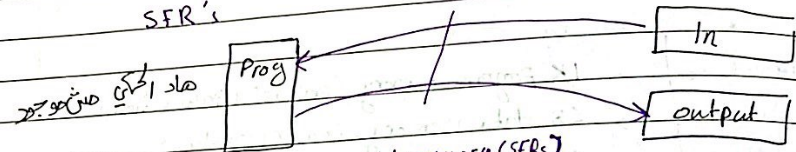
- 3 Peripheials
  - 8 bits timer ø → delay
  - 5 Pins port A → I/Os
  - 8 Pin port B → I/Os
  - 13 Pins for I/Os
- 3 memory types
  - for cak (Permenat) ← 1K Program memory (max 1024 instruction)
  - for var (Volatite) → 68 data memory (max 64 variables)
  - for var values (Permenat) → 64 EEPROM



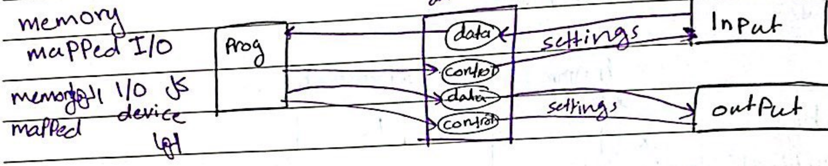


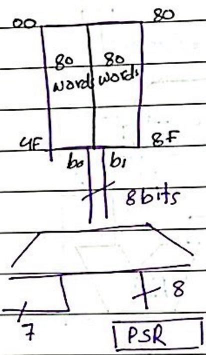
Week 3:

SFR's



memory mapped I/O





= 160 words

$\lceil \log_2 160 \rceil = 8$ bits → ال memory كلها

↳ Addressing mode

if I want to Address Any word in 160 words memory

* How many bits Address we want to address a word in specific bank?

$\lceil \log_2 80 \rceil = 7$ bits → one band

* to address any word in a memory that is divided into banks?

Status reg مش خالفة ال ← (1) select the bank (1bit) if not selected

in the instr itself ← (2) select address in the bank (7bits)

14bits

address for

variable 8 → 7 (data memory) ↳ بالتقسيم صرت احتاج بس ل 7 bits Address

select bank ال overhead ← ارجع يقسم ال

وصار عندي اوفر 1bit وفعليا بعرف مين

M.A.M

Address

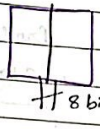
bank 1

85 1000 0101

97 1001 0111

2F 0010 1111

4A 0100 1010



bank 0 ← 7 bits

Address ← 7 bit

(1) select bank 1 (Status(5))

RD 05

RD 17

select b0 (Status(5))

RD 2F

RD 4A

The goal from division the memory to two banks to save 1 bit → (7 bit Address + 1 from Status reg)

Exersise 8

Assume you have an instruction code RD AD, which reads the address AD use it to read the Addresses

75, 98, 8C, 3F, 9A, 2B

75 · 0111 1111

↳ Bank (0) → 7F in bank(0)

98 1001 1011

↳ Bank (1) → 1B in bank(1)

8C 1000 1100

↳ bank (1) → 0C in bank(1)

3F 0011 1111

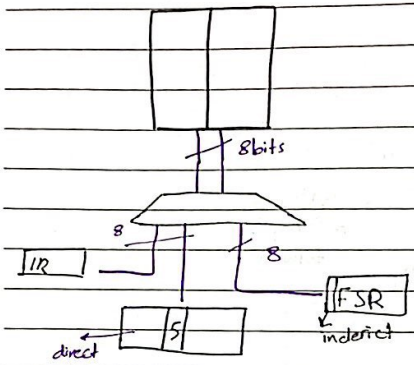
↳ bank (0) → 3F in bank (0)

9A 1001 1010

↳ bank (1) → 1A in bank(1)

2B 0010 1011

↳ bank (0) → 2B in Bank(0)



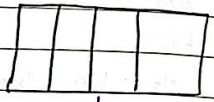
Banks selection:

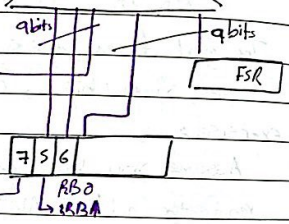
* 2 Banks

direct
status file(5)

indirect
FSR(7)

4 Banks



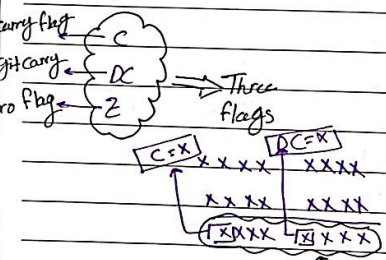


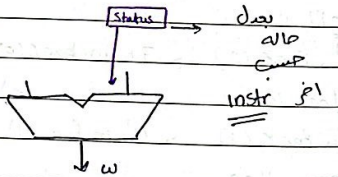
Bank selection:

* 4 Banks

direct
- Status(6)
- Status(5)
- RB1
- RB0

indirect
- Status(7)
- FSR





Data memory → var

Program memory → code & instr

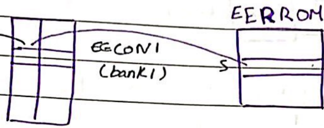
EEPROM → Data → ما بتمسح

↳ (ذاكرة الثانوية)

To Read:

* Address → EEADR (bank 0) → Select bank (1)
* Set RD bit (Start reading)

* اذا بدي اقرأ مرة تانية لازم اعيد كل الخطوات *

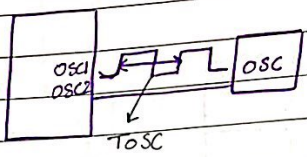


** RD Bit: Set by Software & cleared by hardware

لـ نفس خطوة (2) لا اخليها على طول بتصير (0)

To write:

* address → EEADD bank(0) EEADD(15)(16) 5 → 15
* value → EEDATA → bank 1 EEADATA(15)(6)
* Enable WREN in EECON1 ×
* write (55) EECON2 ✓
* write (AA) EECON2 ✓
* set WR bit (Start writing) ✓ → set by S.W cleared by H.W
* when writing is over EEIF = 1
* if error in writing WRER = 1



$$FOSC = \frac{1}{TOSC}$$

* Tosc : is not enough to fetch or execute one instr
* Tinstr = the time enough to fetch or execute one instr
= x * Tosc
in pic Tinst = 4 * Tosc

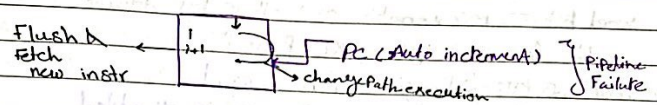
Example: M.C1 Fosc = 10 MH2, Tinst = 2 Tosc
M.C2 Fosc = 20 MH2, Tinst = 8 tosc

* $Tinst(1) = 2 * \frac{1}{10MH2} = 0.2\ M.s$
* $Tinst(2) = 8 * \frac{1}{20} = 0.4\ Ms$

To fetch & execute one inst =
* on M.C1 $\Rightarrow$ 0.2 Ms + 0.2 Ms = 0.4
* on M.C2 $\Rightarrow$ 0.4 Ms + 0.4 = 0.8
M.C2 is slower

Example Fosc = 4 MH2
$\Rightarrow Tosc = \frac{1}{4MH2} = 0.25\ Ms$
$\Rightarrow$ Tinst = 4 * Tosc = 1 Ms
To fetch & excute one inst
1 Tinst (fetch) + 1 Tinst. (exec)
1 Ms + 1 Ms = 2 Ms
$\Rightarrow$ To fully excute a Prog of 10 inst
$10\ inst * \frac{2\ Ms}{inst} = 20\ Ms$
↳ without Piplining
With Pipelining time goes approximetely half

* Some instruction will make Pipeline to fail.



All instruction need 1 Tinstr to be Fetched & execute with Pipline except the inst that cause Pipeline failure

Flush & fetch جديدة

S-R latch

S	R	Q	$\bar{Q}$	
	0	0	Q	$\bar{Q}$ (No change)
exit reset	0	1	0	1 (reset)
reset	1	0	1	0 (set)
	1	1	undefined	

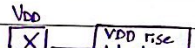
* when does reset happend?

$\bar{S}=1$ & $\bar{R}=0$ when $\bar{S}=1$ → Reset

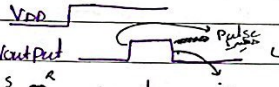
↳ Auto matically the S-R will be in reset mode

3 cases that make $(\bar{S})=1$

* When $\overline{MCLR}=0$
* When WDT is reset on reset mode & when sleep is on not sleep mode → (reset)
* When the Pin of VDD is (1) → Power up on VDD



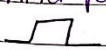
Power لو كانت مش موجودة ع يبقى ال reset كل مرة بيشتغل ع طول

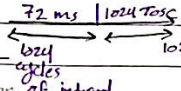


After power up, when does the mc exists reset?

R=1

After power up, how long the PIC stay in reset?





enable PWRT from Configration
word bit 72 ms * enable OST : It is by default
enable for OSC types XT, LP or HS
لـ بتكون enable لواحزت اشي تاني بتكون disabled
if OSC types = RC ⟹ enable OST is disabled

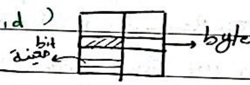
Internal is the best way
And I choose External And Internal when the VDD rise slow

Week 4 B-
chapter 4

Categrics of instruction

(1) byte oriented (file reg) → data momry reg
f(Address) → 7bits, destination (1bit) (op f, d)
0 → W / 1 → data momery

(2) bit oriented file reg



f → (7bits), b → bit رقم ال (3bits), (op f, d)

(3) literal operation → القيمة الي اشتغل عليها
Var (value) K → (8bits) (op K)   add 7 ← موجودة بال instr

(4) Control → بتغير ال Path ال instr
K → (11 bits) op (K)   goto ≠ return / cal

⇓

هيكل ال instr عبارة عن 14bit

* Arithmetic   العوامل التي الحسابية على الأرقام

ADDLW K   // [W] ← K + [W]   literal Always save in [W]

ADDWF f, d   [W] ←(d=0)— [F] + [W]
Address [F] ←(d=1)—

SUBLW K   SUBWF f, d
INCF   f, d
DECF   f, d
COMF   f, d

Example • What are the final values of the file registers W, 21, 22 & 23 after executing the following code? Assuming initial values:

[W] = 3 [21] = 5 [22] = 7 [23] = 9

ADDLW 3 → 3 + 3 = 6 [W] = 6

ADDWF 22,1 → data memory → 7 + 6 = 13 (D) → [22] = D (13)

ADDWF 21,0 → working → 5 + 6 = 11 → [W] = B

ADDLW 23 → 11 + 23 = 34 → [W] = 22

↳ This is value not address because it's literal

Example

W = 2 = F8 = 5

INCF 22,1			
		22	6 8
DECF 23,1	Comp 7	23	56
	111	24	7
	000 x		
COMF 24,0	0000 0111		
↳ W	1111 1000 = F8		

DECF 22,0

SUB LW K [W] ← K - [W]

SUBWF f,d ← [f] - [W]

* We Always Subtract W-Reg

A - B = A + 2's complement of B

A = 5 = 0000 0101 → 2's Complement → 1111 1011

B = 7 = 0000 0111 → 2's Complement → 1111 1001

A - B: 0000 0101 + 1111 1001 = 1111 1110

B - A: 0000 0111 + 1111 1011 = 0000 0010

Subtract

C = 0 → Negative C = 1 → Positive

I need here 2 Var → 2 version

ANDLW K

ANDWF f,d

IORLW K

IORWF f,d

XORLW K

XORWF f,d

* I will use this operation for bits masking

Word

0 And — بدي اشتغل على جزء منها مش كلها

1 Ior — جزء من الـ word ← بدي اشتغل كلية والباقي اخليه زي ما هو

Complement ⊕

* الشغل الي بدي اشتغله كليه يا اخليه (1) ، (0) ، (complement) واخلي الباقي زي ما هو

Rules

$X \cdot \emptyset = \emptyset$ $\quad X \cdot 1 = X$

$X + \emptyset = X$ $\quad x + 1 = 1$

$x \oplus \emptyset = x$ $\quad x \oplus 1 = \bar{x}$

Example write one instr to Clear the least sig 4 bits of working reg (اول 4 bits)

ل (اصفر) ← And $\qquad$ ل ← bits masking

ANDLW K → احتاج اعمل لها Address علشان

: ANDWF f,d

ANDLW F0

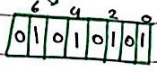
بغير اول 4 ← ← خلي الباقي زي ما هو

Example: Most Sig 3 bits in W-reg set
↳ (one's)

IORLW E0 1110 0000

Example: Complement the even positions bits in W-Reg

XORLW 55

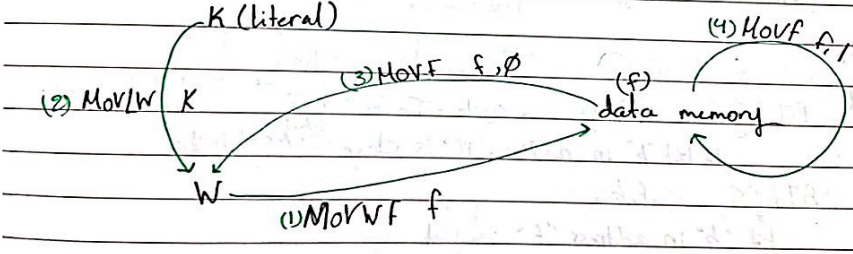


Example: Set last Sig 4 bits in address 15

Working reg. ——— of → MOVLW of

IORWF 15,1

Data movement instructions



(1) To Set a value in Working reg (MOVLW K)

(2) To write value(K) in data memory at address f
same 12 in 13
↳ Need 2 steps (1) MOVLW 12 (2) MOVWF 13

* (3) To copy from address f_1 to f_2
25 → 26
↳ MOVF 25,0
MOVWF 26

(4) MOVF 27,1 [27] ← [27] affects in Z flag

To check whether a value in address X = 0 or not

MOVF X,1 [code to check Z-flag]

SWAPF f,d

بدل ال 4 ↷ 4 ببعض

SWAPF 21,1

SWAPF 22,Ø

=

↓

[W] = B7

21	5A ~~A5~~	
22	7B	

Call K → Stack ←(Push)— PC ; PC ← K

ايش الفرق بينهم؟ (1) الاثنين بقلبوا الـ PC بالـ K

(2) اما الفرق ال Call بتعمل Push لـ PC على الـ Stack

GO TO K PC ← K (You can't return)

اما ال GoTo بتعملش

يعني بال Call بتقدر ترجع لوين انت كنت موصل

* return

RETLW → subroutine

RETFIE → Interrupt

PC ←(Pop)— Stack

This for:

* conditional
* branch
* if
* else
* loops

BTFSC f,b → cycle = Tinstr 2 1/2 (1: condition false, 2: condition True "because of flush")

if bit "b" in address "f" is clear → (skip) next instr

BTFSS f,b

bit "b" in address "f" is set

INCFSZ f,d

if "b" in address 'f' when I do increment will be clear →

~~INC~~ DECFSZ f,d

if "b" in address f become clear when I do decrement →

* "S" stand for Skip: don't excute next instr if cond = true

CLRF

f ← addr مسح كل القيمة اللي بال

destination bit ما الهدف

CLRW

Working مسح كل القيمة اللي بال

NOP

↳ Do Nothing (No operation) → when I do delay

↳ بس لإضاعة وقت

BCF

clear the bit "b" that in address "f"

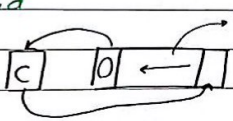
BSF

Set the bit "b" that in address "f"

→ I use this for bank selection

bank 1
BSF Status (5)
chooses b1

RLF f,d

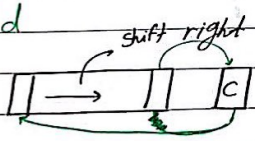


* 2

multiply by 2 if C = 0

* 2 + 1 if C = 1

RRF f,d



Integer div by 2 if C = 0

F [21] = (6) 0000 0110

C = 0 C = X0

RLF 21, 1

0000 1100 = 12

0000 1101 = 13

Example: write code to multiply by 4 the value in Address 18

```
BCF   Status, C      <- if C=0
RLF   18,1
BCF   Status, C
RLF   18,1
```

Example:

```
MOVF  22,1          Z=1  [22]=0
BTFSS Status, Z
INCF  22,1          *This instruction will be executed only if the condition ≠ true
INCF  22,1                                          ↓
                                             zero flag=0
                                                    ↓
                                               [22] ≠ 0
```

■ High level language ⇓

```
if ([22] != 0)
[22]++;
[22]++;
```

```
MOVLW  9
MOVWF  22      [22] <- 9
  └-> Counter           ↓
                         8
                         7
                         ⋮
Loop { do something

DECFSZ 22,1
GOTO   Loop
```

■ high level Language ⇓

```
for (i=9; i>0; i--)
do something
```

→ Note: Every loop need Counter

■ How To represent the instr into binary.

1. op f,d

opcode	d	f
6bits	1bit	7bits

→ Byte oriented

2. op f,b

opCode	b	F
4bits	3bits	7bits

→ Bit orianted

3. op K

opCode	K
6bits	8bits

→ literal

4. op K

opcode	K
3bits	11 bits

→ GoTo

* كيف لبده يعرف ال cpu هاد الكود
عنده ال opcode ← 3,4,6 bits ؟
00 1010 1011 1101
* اذا كانت وحدة من ال op بتبلش برقم
والها رقم معين مستحيل يكون في op ثاني
الو نفس الرقم
* زي اذا كان عندي رقم ال opcode
لل 3bits ← 101 مستحيل تاع
ال 6bits يكون 101xxx
* ال cpu كيف بشتغل ← اول اشي
بطلع على اول (3bits) اذا هم Valid
→ opcode
وبكمل لغ ال instr اذا مش
Valid opcode ← بطلع على ال 4bit ....

→ Note: the "Assembler" Convert from Assembly to machine code "binary"

→ NOTE: opcode should be unique

■ 35 instr = $\lceil \log_2 35 \rceil$ = 6bits → opcode

→ Note: the default value of Assembly is Hex

← اذا كانت Hex value بلبش جوي لازم احط (H)/(0X)

→ Label
↓
* at the begining of the line

الـ label لازم يكون في اول السطر

* Case sensitive "a" ≠ "A"

← بعكس الـ Assembly

* once written, it can be used as operand it's value = the address of first inst after it

* label can be in a stand alone lines

→ label ≡ GoTo label ⇒ GoTo "K" Call

- الـ compiler بجد الـ Address تاع اول instr بعد الـ label

* ليش اول instr؟

* لانه الـ label ممكن يكون بـ line لحال وما في بعده instr على نفس السطر

→ Comment

Start with ";"

→ None executive Statment.

← ما بنفذ الـ compl بطنش اي عملية جاي بعد الـ ;

→ The goal of writing comments: for readability of code

[1] #Include

→ opens file & use anything defined on it

→ #include <iostream> → in C++

[2] ORG

```
ORG 005
instr1 → save in address "5"
Instr 2 → // // // "6"
Instr 3 → // // // "7"
```

most common usage:

ORG 0000 → rest vector

ORG 0004 → interrupt

[3] end

→ end compilation

```
end
=  → will not compiled
```

[4] equ

→ It defined costant

→ The value of this constant doesn't change

```
PJ    equ  3
Status equ 03  x
RP0   equ  5   x
BSF   Status, RP0
        ↓3  ↓5 → ?
```

← وبعدين الـ instr اللي جوه

```
BSF RP0, RP0
BSF 3, 3
```

← متشابهة بس مش متطابقة

[5] cblock

```
21
Var1 → 22      =⇒   Var1 EQU 21
Var2 → 23           Var2 EQU 22
endc
```

→ Note: # include

لما استخدم ال reg باسمائهم
لازم اعمل هيك

Status EQU 03
C EQU Ø

ع اضطر افتح ال data sheet وادور على قيمهم

BSF Status, C

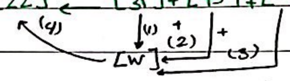
With # include

#include PIC16F84A.INC
BSF Status, C

* we don't need the to go back to data sheet

Example: slide 24

[22] ← [31] + [45] + [47]



MOVF 31,0
ADDWF 45,0
ADDW 47,0
MOVWF 22

⇓

الكود مش كامل

⇓

Slide 25

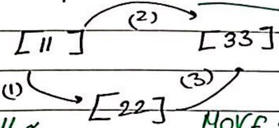
* لازم بالبداية اتأكد انه كلهم بنفس ال Bank
واذا بستخدم ملي ← BCF
BCF RPØ

* ما في داعي لل GOTO بس هنا
اذا كان في loop او ما شابه
← لازم يضل البرنامج Always running
← اذا بدي اعرف ال Status, RPo
لازم احط #include
← مش حاجة اعمل define ← Interrupt vector
الا اذا بدي اعمل disabled

* Slide 25 → some questions:
← قديه بحجز بال Prog memory ?
* بعد ال Instr
← org c equ ما بتخزنوا
+ بعطيني حجم ال bit = 8 X 14
Status → 14 bits RPØ → 3 bits
go to → Address (10)
Done = 10

Example: Slide 26

Ineed temp var ← SWAP لما اعمل



MOVF 11,0
MOVWF 22

MOVF 11,∅
MOVWF 22

MOVF 22,∅
MOVWF 33

10 instr → 140 bits

DONE = 12

label DONE → Disabered

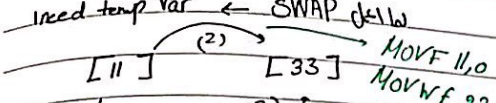
bank switching بحتاج 10* ← بس بختار ال bank بالبداية

org4 ، org 0 ، Interupt ، reset veetore

#include ، GOTO ، End

Example: Slide 26

need temp var ← SWAP data



MOVF 11,0
MOVWF 22
MOVF 11,0
MOVWF 22
MOVF 22,0
MOVWF 33

bank 11 bits ow bank 2 bits to switching

10 instr → 140 bits
DONE = 12
label DONE → Disabeled

org 4 C org 0 C Interrupt C reset vector
#include C GoTo C End

Week 5
chapter 5

Example: write code to do the following:

if ([22] > 7)
    code block1
else
    code block2

MOVLW 7
SUBWF 22,0 ← [22] > 7 C=1 ; [22] < 7 C=0 ; W = [22] - 7
BTFSC Status,C
GOTO block1
GOTO block2

block1 =
block2 = → GoTo Next
Next

Another Solution :

```
    MOVLW 7
    SUBWF 22,0
    BTFSC Status,C
    GOTO block1
block2
    GOTO Next
block1
Next
```

Example :

```
for(int i = 30, i > 0, i--)
    block1
    MovLW D'30'
    Movwf 22
Loop
    block1
    DecfSZ 22,1  ──→  DECFSZ 22,0 = infinite loop
    GOTO loop
```

Example

```
for(int i = 5, i < 20 , i++)
                        ↳ block2
    MovLW 5
    MovWf 24
Loop
    Block2
    INCF 24,1
    MovLW 0'30'
    SUBWf 24,0
    BTFSS Status,Z
    GOTO loop
```

Z=1
if. SuB. 0
E24] . 20

MAM

Example:

If ([22] == 0)

{ Block1

Block2

```
MOVF 22,1
BTFSC Status,Z
GOTO Block1
Block2
  ≡
Block1
  ≡
```

Slide (yes) ⟹

```
MOVF 22,1
BTFSC Status,Z
GOTO Block1
Block2
  ≡
  GOTO Next
Block1
  ≡
  GOTO Block2
Next
```

Code Slide 88

[11] + [22]

If C=0

[33] ←

else

[44] ←

Now

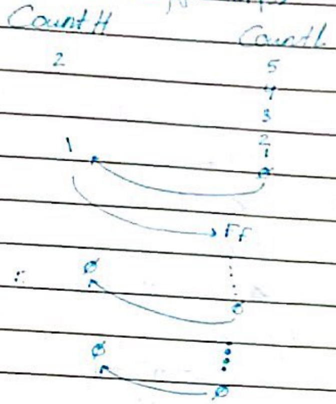
```
MOVF 11,0
ADDWF 22,0
BTFSS Status,C
GOTO S33
GOTO S44
```

```
S33
  MOVWF 33
  GOTO Next
S44 MOVWF 44
```

Example Slide 10:

- حجم أكبر Var → 8bits → 255
- أكبر loop بتاك → 256 ⟺

بدنا نستخدم متغيرين



⟹

```
Loop    MOVF  CountL,1
        BTFSS Status,Z
        GOTO  DEClow
        MOVF  CountH,1
        BTFSC Status,Z
        GOTO  DONE
        DECF  CountH,1
DEClow  DECF  CountL,1
        GOTO  Loop
DONE    GOTO  DONE
```

Example (1)

[25] ← [22] - [24]

Example (2)

```
if ([27]} > [28])
    [27] = 9
else
    [28] * = 4 ;
```

Example (3)

```
for (i= 7 , i<28, i+ =2)
    [28] ++ ;
```

Example (4)

```
if ([24]>7)
add 3 to [24]
if ([24]>9)
  decrement [24]
else
  INC [24]
```

Example (1) Solution:

```
MOVF 24,0
SUBWF 22,0
MOVWF 25
```

Example (2) Solution:

```
MOVF 28,0
SUBWF 27,0
BTFSS Status,C  → GOTO Halt
MOVLW 9
MOVWF 27
GOTO Next
Halt
BCF Status,C
RLF 28,1
BCF Status,C
RLF 28,1
Next GOTO Next
```

Example (3) solution:

```
MOVLW 7
MOVWF 24
Loop
INCF 28,1
INCF 24,1
INCF 24,1
MOVLW D'28'
SUBWF 24,0
BTFSS Status,Z
GOTO Loop
```

Example (4)

```
MOVLW 7
SUBWF 24,0
BTFSS Status,C
GOTO Next
MOVLW 3
ADDWF 24,1
MOVLW 9
SUBWF 24,0
BTFSS Status,C
GOTO Else
DECF 24,1
GOTO Next
Else INCF 24,1
Next GOTO Next
```

Subroutines

- The goal of writing subroutines?
  - When you need to call a code that you'll need in a program in different places.

- Subroutines?
  - Code starts with label & ends with Return or Retlw k (2 Tinst; Movlw k, Return)
  - And executed when called by call label → 2 Tinst

- How to Do this in Assembly?

```
int Sum(int i, int j)
{
  return i+j
}
```

```
call sum(3,5)
call sub(i,j)
```

⇩

```
MOVLW 3
MOVWF 21 ----> i
MOVLW 5
MOVWF 22 ----> j
call Sum
```

Prog Subroutine

```
Sum
    MOVF 21,0
    ADDWF 22,0
    return
```

Example Slide 16

Multiply

```
MOVF 31,1
BTFSC Status,Z
retlw 0
MOVF 31,0
MOVWF Temp
CLRW
Loop ADDWf 30,0
DECFSZ 31,1
GOTO loop
MOVF Temp,0
MOVWF 31        → Error here
Return
```

Delays

- Every Embedded device should contains some types of delays Timer

The time delays can be done by:

- Software → Nop
- Hardware → Timer (Alarm) → chapter 6

```
Loop turn on LED
     500 Nop instr → 0.5 second      Note FOSC = 4KHz
     turn off LED                    Tinst = 4/4KHz = 1 ms
     500 Nop → 0.5 second
     GOTO Loop
```

‖ Another Way (better)

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

max 256 iteration

Another way (more better)

```
MovLW X
MovWF CounterL
MovLW Y
MovWF CounterH
Loop
    Nop
    Nop
    DECFSZ countL,1
    GOTO Loop
    DECFSZ CountH,1
    GOTO Loop
```

256 × 256 iteration

⇒ equal

```
MOVLW X
MOVWF CountL
MovLW y
Movwf CountH
loop call DelSub
    DECFSZ CountH,1
    GOTO loop
DELSUB MOVLW X
MOVWF CountL
Loop2 Nop
    Nop
DECFSZ CountL,1
GOTO Loop2
return
```

← Internal loop
Inside external loop

Delay = $\frac{4}{FOSC}$ * # of instruction

↓ Trace the code

```
Initialization
Loop
decfsz
GOTO Loop
```

→ # of Tinst.
Initialization + (X-1) iterations + last iteration

Code Slide 20 :

The loop iterate → 200

Delay = 4/Fosc * # of Tinstr

$= \frac{4}{800 \times 10^3} * \#$ of Tinstr

# of Tinstr = (1+1) initialization + 199 * (1+1+1+2) + (1+1+2)

(nop, decfsz, nop, goto; nop, nop, decfsz; final iteration)

= 2 + 199 * 5 + 4 = 1001

Delay = 5,005 ms

* What if this code to be used as a Subroutine?

Call Sub → 2 Tinst

Return → 2 Tinst

Delay = 5.005 + 4 * 5 μs = 5.025 ms

↳ (2×5) → call + (2×5) → return

* Modify the code in Slide 20 to give exactly 4ms delay:

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

$4 \times 10^{-3} = \frac{4}{800 \times 10^3} * \#$ of Tinst

↳ = 800

800 = 2 + (X-1)*5 + 4

800 = 2 + 5X - 5 + 4 800 = 5X + 1 $\frac{799}{5} = X$

X = 159.8 → X = 159

↳ 2 + 158 * 5 + 4 = 796

Example: Write Subroutine that takes exactly 10 ms Including call inst, Assume Fosc = 1MHz

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

$$10 \times 10^{-3} = \frac{4}{1 \times 10^{6}} * \#Tinst$$

$\#$ of Tinst = 2,500

table ←

```
Delay 10 ms   MOVLW X
              MOVWF 21
       Loop   Nop
              Nop

              DECFSZ 21,1
              GOTO loop
              return
```

Using (7 Nop):

$2500 = 4 + (x-1)10 + 11 = 4 + 10x - 10 + 11$:

$2500 = 10x + 5 \quad 2495 = 10x$

$\boxed{x = 249.5}$

$x = L \rightarrow 249$

$2500 = \overset{call}{2} + \overset{initial}{2} + (X-1) * 4 + 5$

$2500 = 4x + 5$

$2495 = 4x$

$X = 623.75$

ما بقدر استخدم هاي القيمة لانه

أكبر قيمة ل X (value) ← 255

فالحل انه نزيد عدد ال Nop

$\rightarrow 2500 = 2 + 2 + 6(x-1) + 7$

$2500 = 5 + 6x$

$2495 = 6x \rightarrow x = \frac{2495}{6}$

$x = 415.3$

Also we can't use this

بدنا نزيد عدد ال Nop

Nested loops

Initialization +

First External { All internal Except last + last internal iteration }

+ all external iterations except first & last

+ last external iteration

Slide 21:

10 ms Fosc = 4MHz

$\Rightarrow$ Delay $= \frac{4}{Fosc} \times$ # of Tinst

$\Rightarrow$ # Tinst = 10,000

Solve 8-

call → 2 + 5 → init + (249 * 3 + (2+1+2)) → goto First External + decfsz + decfsz

decfsz, goto, decfsz CountL = 256 / 0 CountH = 13 / 12 (first external)

(11 * [255*3 + 5]) + (255*3 + (2+2+2)) = 10,000

decfsz | return

All External iterations except first and last → last iteration

Working with data

There are three types of Addressing modes:

1- Literal: to deal with values directly, no need for data memory.

2- Direct Address: to deal with the data memory → The address of the value is in instruction itself → choose the bank from the Status

3- Indirect Address: To deal with the data memory → The address of the value is in FSR → The last bit choose the bank

Example: Write code to clear the data memory from address 0x10 → 0x4F (64 location)

CLRF F (This instruction clear the address in the data memory)

64 instruction

CLRF 0x10	Second solution: MOVLW 0x10 / MOVWF Counter
CLRF 0x11	Loop CLRF INCF f,1 → I'm stuck here
CLRF 0x12	DECFSZ Counter,1
CLRF 0x13	GOTO Loop
⋮	
CLRF 0x4F	

We need the Indirect Address mode when I want to manipulate the address (Increment, decrement, ++, --)

Using indirect:

(1) write the address in FSR

(2) replace the operand F by either (00) or INDF

↳ refrence or pointer for Address

Example: write code to increment the value in address 15 by 1 using Indirect address

```
(1) ← MovLW 15
      MovWF FSR
(3) ← INCF 00,1 → INCF 15,1
```

⇓ حل السؤال السابق بطريقة أبسط

```
MovLW D'64'
MovWF Counter
MovLW 0X10
MovWF FSR
Loop CRF INDF
     INCF FSR,1 → يعني بتشيل 1
     DECFSZ counter,1
     GOTO loop
```

Banks

Direct Address	Indirect address
4 banks → 2bits	4 banks
→ Status(6), Status(5)	→ Status(7), FSR(7)
→ RP1 , RP0	→ 2banks → FSR(7)
2banks → RP0	

Example: write code to read address 0x97 using both Direct & indirect

Direct	Indirect
BSF Status, 5	BSF FSR, 7 → 1001 0111 FSR
MOVF 0x17, 0	MovLW 0x17 → 0001 0111
	MovWF FSR
	MovF INDF, 0

Indirect address

```
MOVLW 0X97
MOVWF FSR
MOVF INDF,0
```

look up table

in C++ & java:

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

Count << squares[2]; // 4

look up table: is a way to define an array and access any element in it. but it will be stored in the program memory معناه حجمها أكبر

```
Squares
        ADDWF PCL,1        → بترجع الـ pcl القيمة الي بالـ working
        retlw 0
        retlw 1              int [6] squares=
        retlw 4              ⇒ [0,1,4,9,16,25]
        retlw 9
        retlw D'16'
        retlw D'25'
                             PCl = 2  →  working = 4

        MOVLW 2
call    Squares      =      Squares[2]
        MOVLW 5
call    Squares      →      W = 25
```

13 bits

Pc: [7 ... 0] → PCL

No address (Not readable or writing directly)

بنتعامل في الـ pcl بدال الـ PC

```
Squares
    AddWF PCL,1          MOVWF Temp
    retlW D'9'           MOVLW 3
    retlW D'16'          SUBWF Temp,0
    retlW D'25'
    retlW D'36'          هيك بزبط الكود
    retlW D'49'
    retLW D'64'
```

Squares 3 → 8
(9,16,25,36,49,64)

MOVLW 5
call squares
W = 64

Week (6)

Chapter (6)

Software delay ← hardware (timer) ← Software

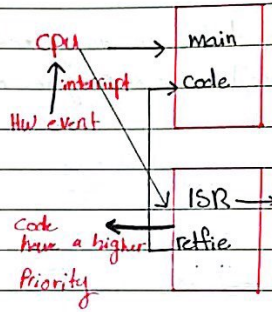
cpu busy

delay 1 sec ← Timer settings

cpu Interrupt 1 sec

→ Timer use interrupts.

Interrupt: Code must start at address 4 & ends with return from interrupt invoked by Hardware event. (interrupt the cpu)



Polling

Interrupt

check

* Must check if the delay has been reached or not

Types of interrupts:

1) External interrupts (from something outside the pic)

2) Internal interrupts (from something inside the pic)

→ Maskable interrupts → I can disabled

→ Non-Maskable interrupts → can't by disabled

All the interrupts in the pic is maskable

The Interrupt occurs when:

* GIE(1) & Interrupt x enable (in one maskable interrupts) & hardware Event (set) → flag

Scanned with CamScanner

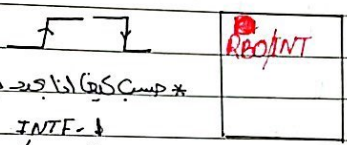
Conditions for interrupts:

1. GIE = 1
2. local Enable = 1
3. Interrupt flag = 1 (Hardware Event)

- 4 types of interrupt: each has enable bit & flag bit
- All enables are Set / cleared by software.
- Flag bits are Set by hardware event.

* Sources of interrupts

1- External interrupt



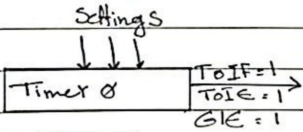
* حسب كيفي انا بدي يشتغل ع ال positive او ال negative Edge

INTF = 1 → flag

- Interrupt بيصير بشرط
- INTE = 1 → local enable
- GIE = 1

Software ← (INTE, GIE)

2- Timer zero over flow interrupts



3- Port B change interrupt

change
0 → 1
1 → 0
لو تغير احد ال4 بوابات الأخيرة

→ RB4, 5, 6, 7

Port B have 8 Pins

RBIF = 1
RBIE = 1 GIE = 1

4- EEPROM write complete interrupt

بس نكتب ال WR ونخليه WR = 1

↳ This means start writting on EEPROM from EEData → when finishing EEIF = 1. This is a flag for interrupt when need to EEIE be one EEIE = 1 and GIE = 1

4 types of interrupt:

We need 10 bits to deal with interrupt

1) (1) GIE bit (if disabled no interrupt Event can happend)

2) (4) interrupt ~~enable~~ enable s (each for one interrupt type)

3) (4) interrupt flags → hardware Event

4) 1 bit ↑ ↓

* Single interrupt vector (To know which interrupt happens we check the flag bit)

* When we use option register we need to switch to bank 1

Interrupt operation slide 11

(1) Complete the current instruction

(2) Save PC for the next instruction on the stack

(3) Clear GIE To stop the other interrupt that will happen when I handle with this interrupt

(4) Put in PC value 0004

(5) Continue the program execution until there is return statement

(6) Set GIE to 1

(7) Pop the PC from the stack

This steps is done by the cpu

How To use interrupts (what programmer should do)

(1) Start at 0004

(2) ends with Retfie

(3) Enable GIE

(4) Enable local enable

(5) Before return from the interrupt you have to clear the flag

Programmer ← clear ال flag يعمل ← hardware — set ال flag يعمل ←

* All interrupt flags are set by hardware but cleared by software

(6) (Not Always only for PortB change) Clear the flag bit at the beginning of the code

on reset flag bits = ∅

clear لازم [illegible] في البداية

لان لو reset [illegible] interrupt [illegible]

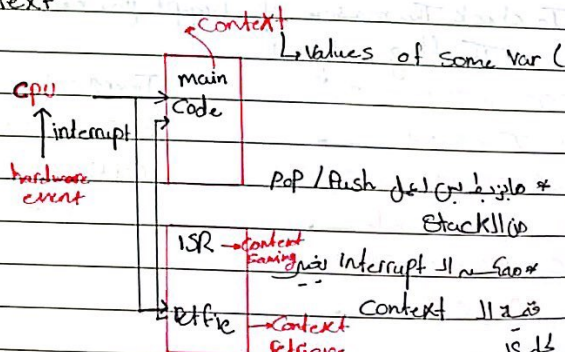
Example slide 13 :

```
        ORG   0000
        GOTO  Start
        ORG   0004
        GOTO  ISR
Start
        BSF   INTCON, GIE
        BSF   INTCON, INTE
        BSF   option_Reg, 6
        BCF   STATUS, RB0
        CLRW
Loop    ADDWF 0X0A, 0
        GOTO  Loop
```

```
ISR
        MOVWF 0X10
        CLRW
        BCF   INTCON, INTF
        RETFIE
        end
```

Context Saving :

"out of context"



* How to do Context Saving?

(1)

For Any register except 2 special cases

```
ISR
MOVF (x),0(any reg)
MOVWF Temp
MOVF Temp,0
MOVWF x=05
retfie
```

(2)

```
ISR
MOVWF Temp

MOVF Temp,0
retfie
```

This for W-R

(3)

```
ISR
SWAP Status,0
MOVWF Temp

SWAPF Temp,0
MOVWF Status
retfie
```

This for Status

Multiple Interrupts:

```
BSF INTCON, GIE
BSF INTCON, T0IE
BSF INTCON, INTE
BSF INTCON, RBIE
```

→ if I only write this I don't need to check the reason of interrupt

→ There is three interrupts

To check The reason of interrupt you need to check The flag ... how?!

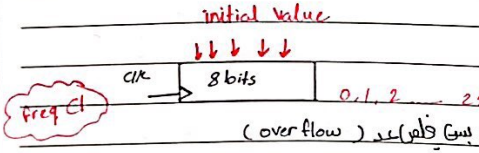
```
ISR BTFSC INTCON,T0IF
    GOTO Timer0 code
    BTFSC INTCON, INTF
    GOTO external code
    BTFSC INTCON, RBIF
    GOTO PROB Code
    retfie
```

```
Timer0 Code
BCF INTCON, T0IF
retfie
```

بدي اكون اكيد سو

→ Timer & counter

Timer is a counter



* To use Counter as Timer:

(1) Know the driving clock freq:

→ if freq of clock = 1KHz, When the TOIF become 1? (Interrupt)

cycle time = $\frac{1}{1KHz}$ = 1ms → 1×256 = 256 ms → Interrupt

(2) Initialize it to a value:

→ if freq = 1KHz, & I want overflow after 100 ms

Initialize it to 256 - 100 = 156

100 ms = $\frac{1}{1}$ * (256 - x) → 100 · 256 - x → 156

→ Timer 0 :

Clock

- Internal : Fosc/4 (T0CS = 0)
- External : RA4 (T0CS = 1)
  - ↑ Positive Edge T0SE = 0
  - ↓ Negative Edge T0SE = 1

→ Prescaled if PSA = 0, Prescale = $2^{(PS2, PS1, PS0)+1}$

→ Not Prescaled if PSA = 1

→ e.g. PSA = 0 & PS2, PS1, PS0 = 100

Ans: 2^{4+1} = 32

→ The goal of the Prescale is to increase the delay

"32 ll)s freq ll pmis)"

Delay of TMR0 = (T) * (256 * Initial)

$\frac{4}{Fosc}$ ($\frac{4}{Fosc}$ * Prescale) * (256 - IN)

Fosc/4 — [Prescale] → (T) → Initial Value

↳ The cycle that is Fit to the Counter ↳ # of cycle needed by Timer 0 to finish Counting

Max delay of TMR0 $\frac{4}{Fosc}$ * 256 * 256

↳ Max Prescale

Example Slide 22:

$$delay = \frac{4}{Fosc} * Prescale * \underbrace{(256 - IN)}_{X}$$

$$5X10^{-3} = \frac{4}{800X10^{3}} * Prescale * x$$

$$Prescale * x = 1000$$

↳ $Prescale = 8 \rightarrow x = 125$

$IN = 256 - 125 = 131$

4 → x = 250

IN = 6

The code

```
MOLW D'131'
MOWF TMRØ
Select b1 → option Reg
MOVLW B'XXØXXØ10'   (internal)
MOVWF option_Reg
Select bØ
```

78) (Prescale)

$2^3 \rightarrow 2^{2+1}$ 2 = 010

Example: Write code that increment a counter stored in address 23 Every second. use timerØ & interupt, assume Fosc = 4MHz

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

$1 = \frac{4}{4X10^6} * Pre * x$ $\quad Prescaler * x = 10^6$

$Prescaler = 256 \rightarrow x = 10^6/256 > 256$ بايظ

Max delay of TMRØ $= (4/4X10^6) * 256 * 256 = 65.53\ ms$

→ Dlay = Val * delay

1 Second = val * 10 ms $\quad Val = 100$

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

Prescale $* x = 10{,}000$

Prescale $= 32 \xrightarrow{x} = 312.5 \longrightarrow$ [illegible]

$= 128 \longrightarrow x = 78 \quad IN = 256 - 78 = 178$

Code:

```
        ORG 0000
        GOTO Start
        ORG 0004
        GOTO ISR
Start
        BSF INTCON, GIE
        BSF INTCON, T0IE
        MOVLW D'178'
        MOVWF TMR0
        MOVLW D'100'        →(Val)
        MOVWF 30
Bank1 ←
        MOVLW B'xxx0x011'
        MOVWF option_Reg    This will give 10ms delay
Loop    GOTO loop

ISR
        BCF INTCON, T0IF
        MOVLW D'178'
        MOVWF TMR0
        DECFSZ 30,1
        retfie
        INCF 23
        MOVLW D'100'
        MOVWF 30
        retfie
        End
```

Watchdog timer

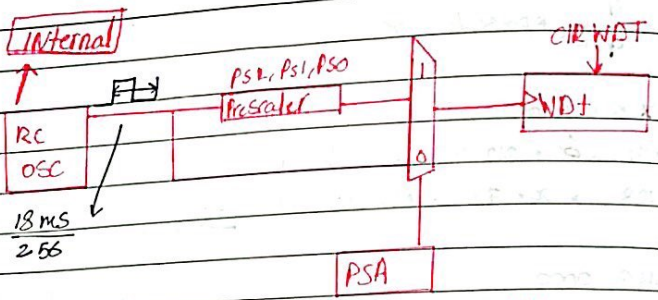
WDT is a counter (8 bit)

*What happens when Watchdog timer finish the counting? (255 → 0)

(1) Wakup if in Sleep mode (2) Reset if not in sleep mode

CLRWDT: Start counting from zero

prescale $= 2^{PS2, PS1, PS0}$



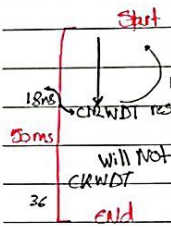
Every cycle $\frac{18\text{ ms}}{256}$ ISR

Delay of WDT = 256 * Prescale * $\frac{18}{256}$

= 18 ms * Prescale

Max Delay = 18 ms * 128 = 2.3 second

Watchdog timer have enable bit in configuration Word



if WDTE is Enable

PSA = 0 → WDT is Not Prescale

Work Even in Sleep mode because it has its own RC OSC

Code

.....

SLEEP → make CPU enter the Sleep mode

If GIE = 0, GIE = 1, PC = 0004 → Code to wakeup ✗

How to wake up?

(1) Interrupt flag = 1 while it's local Enable = 1 regardless of GIE

↓ except timer0 → because OSC is off

(2) Watchdog timer overflow if Enable → WDTE = 1 2.3 s.

(3) $\overline{MCLR}$ = 0 → MCLR 1 2.3 s.

PC = 0 → 0.xyz s. instr

if WDTE = 1

longest Sleep mode 18 ms * 128 = 2.3 seconds

Week 7, Chapter 7

* Serial : Single bits * Parallel : Multi bits (Set e.g. 8 bits)

* PIC 16F84 → (Parallel Ports)

Port B (8 Pins) Port A (5 Pins)

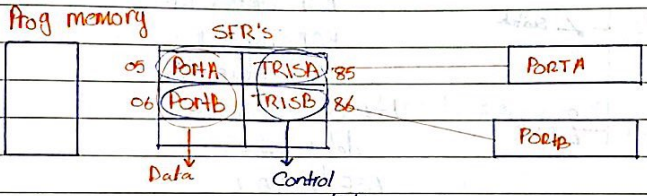
* Some Pins have Symbol "/" That's means → multiple func

→ TRISA : control pin (input or output) 1 = input, 0 = output

→ PORTA : Data

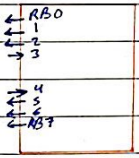
→ TRISB : Input = 1 or output = 0

→ PortB : Data register



*Example Slide 7:

PIC



BSF Status, RP0

MOVLW B'0001 1000'

MOVWF TRISB

* I can't let the pin to be input/output at the same time.

* All pins are independent

Note about Slide 8a

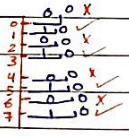
bsF Status, RP0

clrf TrisB → Port B output

MOVLW 0XAA

bcF Status, RP0 → AA → portB

MOVWF PortB

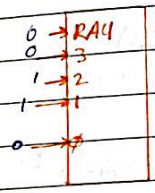


The Effect

4 Leds will be on & 4 leds will be off

#Note about Slide 8:

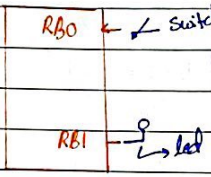
```
bsf Status, RP0
MOVLW 0xFF
MOVWF TRISA
bcf Status,RP0
Movf PORTA,W
MOVWF 0x0D
```



0x0D :
B'0000 0111'

→ RB0 should be (input) in external interrupt not output

Example Slide 9:



Flashing the led but All the time

```
BSF Status, RP0
BCF TRISB,1
BCF Status, RP0
Loop BSF PortB,1
     delay 1sec
BSF PORTB,1
delay 1sec
GOTO loop
```

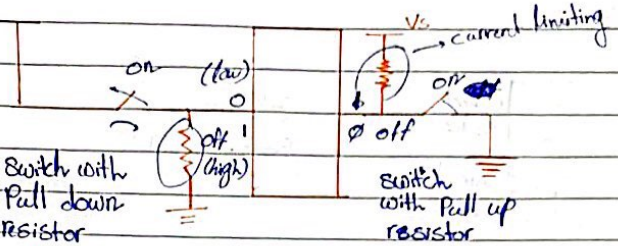
The Code in Slide 9:

flash = [ ] [0 → not flashing, 1 → flashing]

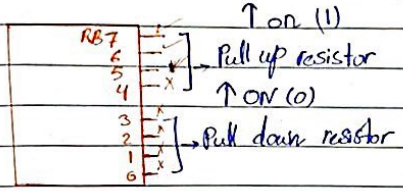
```
ORG 0000
GOTO Start
ORG 0004
GOTO ISR
Start MOVLW B' XXXX XX01'   (B1 → input, output)
  MOVWF TRISB
  BSF Option_Reg,6
  BSF INTCON,GIE
  BSF INTCON, INTE
  CLRF flash        → not flashing
  CLRF PORTB        → led is off
Loop BTFSS flash,0
  GOTO loop
  MOVLW B'0000 0001'
  XORWF PORTB,1
  delay 1 Sec
  GOTO loop

ISR
BCF INTCON,INTF
MOVLW B'0000 0001'
XORWF flash,1
retfie
```

Switches



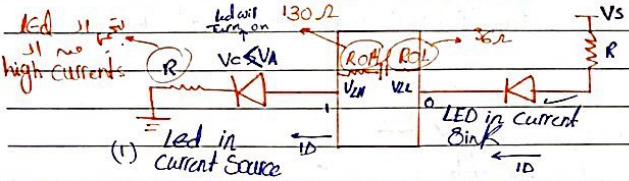
→ if we don't have resistor the input will be floating but I want to force it to be low or high.



Write code to configure PortB & read it into register 0X19

→ B1
MOVLW 0XFF
MOVWF TRISB
→ B0
MOVF PORTB, 0
MOVWF 0X19 → what is the final Value in 0X19?

0X19 = 0000 1000



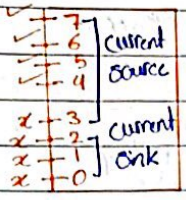
(1) $V_{LH} = V_D + RI_D$ $\quad R = \frac{V_{LH} - V_D}{I_D}$ $\quad V_{LH} = I_D \times R_{OH} + V_D + R_{OL} I_D$

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

(2) $V_S = RI_D + V_D + V_{LL}$

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

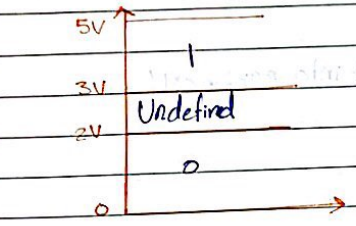
→ Write code to configure a PIC that has 8 LEDs connected to PORTB as follows

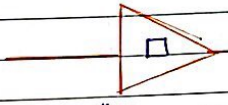


Sink on(0)
Source on(1)

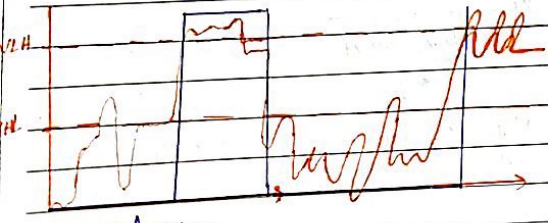
→B1
MOVLW 0X00
MOVWF TRISB

→B0
MOVLW B'1111 0111'
MOVWF PORTB

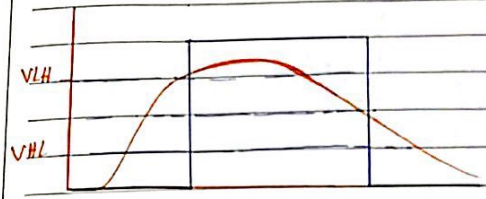




VLH & VHL if initially the output is low, to convert into high the input voltage must go above VLH
& if initially the output is high, to convert into low the input voltage must go below VHL



* cancel Noise



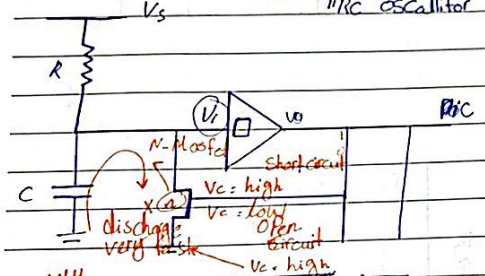
→ Slow Switching

* After Schmitt the Switching will be fast

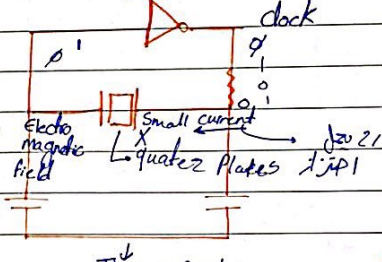
→ oscillator is a device make generation to clock wave Signal (clock)

* first 2 bits determine which OSC I will use

"RC oscillator"



C و R بعتمد على قيمة ال



بالبداية

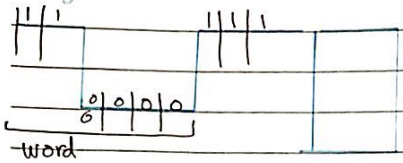
These capacitor to cancel Noise & clock stablise faster

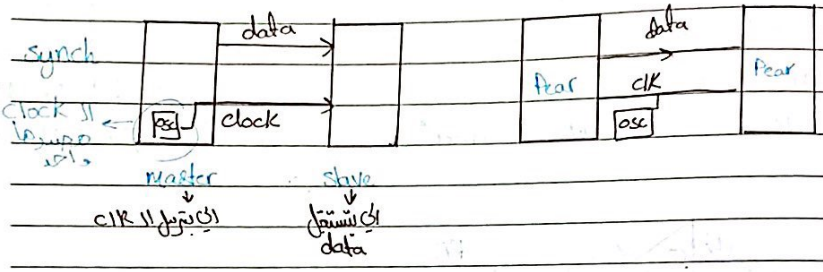
بتكون بالبداية عشان تشحن الـ Capacitor

Week 8 :

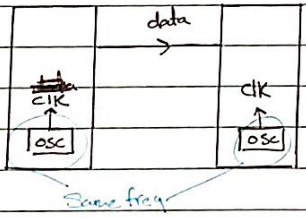
Chapter 10 :

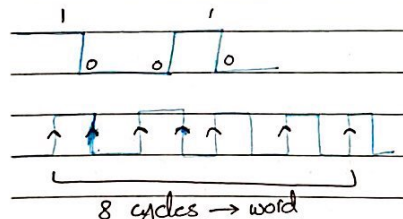
→ Note the two PICs in the same family that's means they have the same Instruction Set & same format.



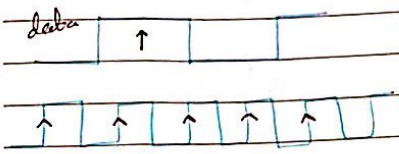


Asynch.



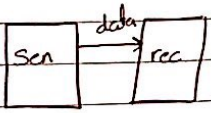


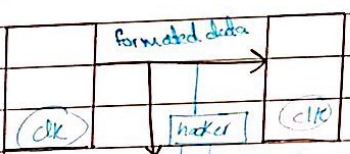
skew, small but not affect single word usually



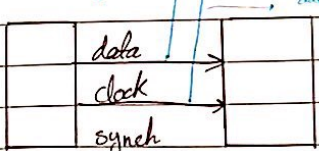
To solve clock skew problem s

→ let the sender & receive resynchronize each other after each word by going to a default states (stop & start bits)



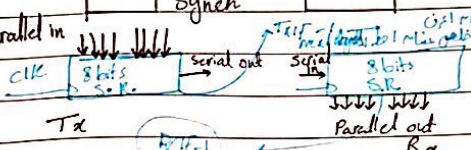


according to a protocol s stop bit, start bit, parity, data rate, encoding



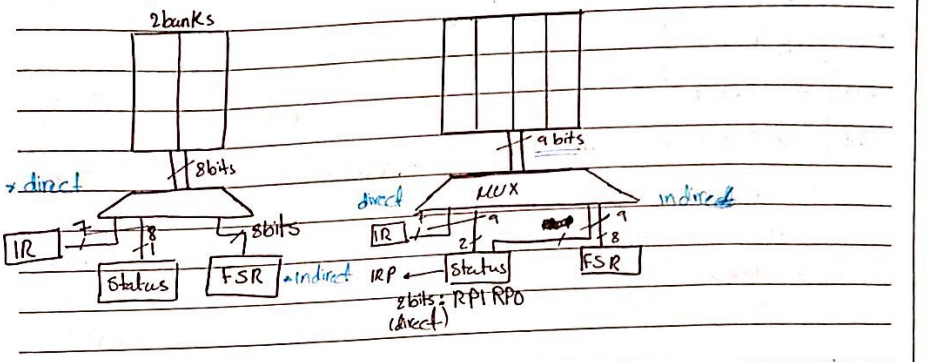
data ll pass directly

Parallel in



byte data

Slide 18



* write code to read the data memory from address 0x185 (9 bits: 1 bank bit + 8 address bits)

Direct:

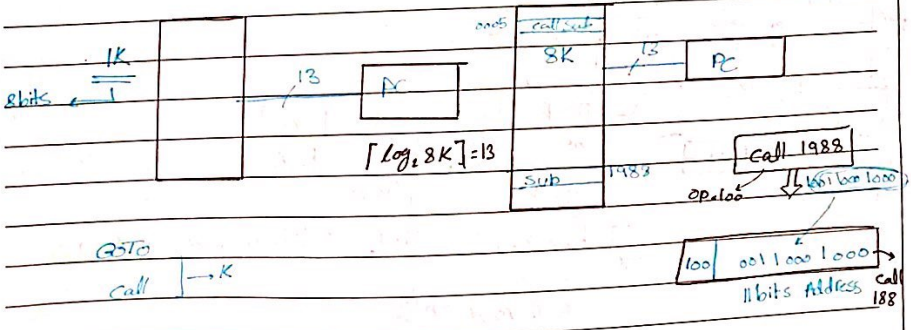
```
BSF Status,6
BSF Status,5
MOVF 0X05,0
```

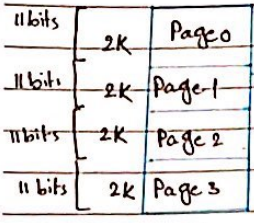
Indirect:

```
BSF Status,7
MOVLW 0X85
MOVWF FSR
MOVF INDF,0
```

bit in FSR → Status(7) to bank 1/IRP

→ 8 bits → FSR



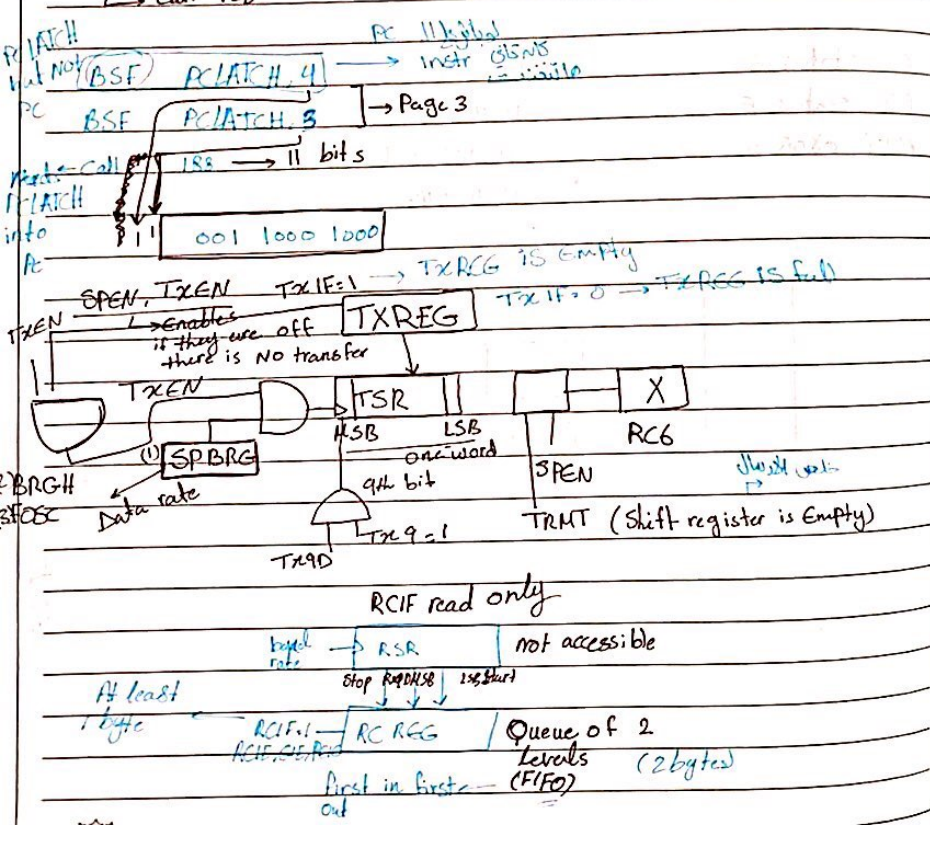


→ To Select the Page = 2bits

→ you Need to choose Page in call & goto only, if the address is in Another Page.

Call 1988

↳ call 188



RCIF read only

At least 1 byte

→ If stop bit of 3rd byte gets received before reading first 2 bytes
↑ → Over run Error

OERR = 1
(1) reception stops → (بوقف استقبال)
(2) 3rd byte is lost

* To avoid OERR, read the bytes as soon as they are received
How ?!
اذا قرأت بيرجع فضي ← MOVF RCREG, ∅ (first in first out)
او ما قدرت = اقلب المشكلة
↳ (OERR = 1) → No reception

ماتقدر اتعامل معها ← BCF [ ], OERR
Read only

* To solve OERR:
(1) clear CREN bit → OERR = 0
(2) read the 2 bytes
(3) Set CREN = 1

SPEN	TX9	TXEN	Sync	BRGH	TRMT	TX9D
1	0	1	0	?	x	x

→ Slide 33:
* Example
(TX9 = 0)

$$\text{baud rate} = \frac{FOSC}{\frac{16}{64} \times (1 + SPBRG)}$$

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

$$9.6 \times 10^3 \times 16 \times (1 + SPBRG) = 20 \times 10^6$$

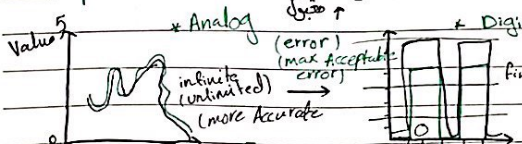
$$SPBRG = \frac{20 \times 10^6}{9.6 \times 10^3 \times K} - 1$$

[redacted]

اذا K بيطلع 256 القيمة اكبر من 256 مش مناسبة (BRGH)

Week 9:
Chapter 11:

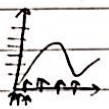
* Analog * Digital



خطأ مقبول وخفيف
↑ مقبول
بقدر ارجعها
مش قادر يتغير عند ال Accurate
انا استقبلت ال signal ما بقدر ارجعها لأصلها
لأني ما بعرف شو اصلها بقدر ارجعها على قيمه لما تتغير

ADC

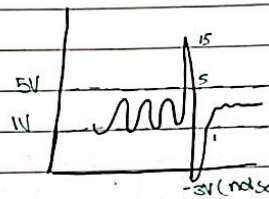
- Sampling: read analog input every times unit (ts)
  - fs = 1/ts ↑ more accurate (number of samples per sec)
  - ⊖ Power ⊖ Processing ⊖ memory ⊖ Bandwidth
- Quantization: round the read value to the closest acceptable Digital Value (number of bits) (n)
  - n ↑ 8 ⊕ more Accurate
  - ⊖ Power ⊖ Processing ⊖ memory ⊖ Bandwidth ⊖ Slower



fs ≥ 2 finput

overwise → aliasing

reference voltages: range of Acceptable Analog inputs



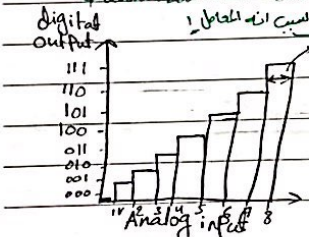
VREF⁺ = 5

VREF⁻ = 1

(more Accuracy)

Quantization:

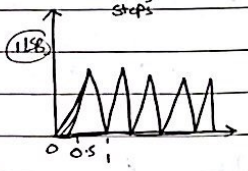
Quantization characterestics graph:



minimum change in input which change the output

(LSB) = (VREF⁺ − VREF⁻) / 2^n = (8 − 0) / 2^3 = 1V

n: steps

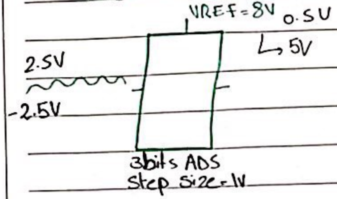


error = Vinput − Voutput

step size (LSB) = Vr / 2^n

max error = 1/2 Vr/2^n → Vr/2^(n+1)

Slide '12':



0 <-> 1V = 000

* Sample & Hold = Fix input
  - Take sample
  - Convert
  - release input

Slide 14:

$V_C = V_S * (1 - e^{-t/\tau})$, $\tau = R*C$

We should wait t until open the Sw

We should wait t until $V_C \simeq V_S$ — * هاد الخطأ مش مقبول *

≃ 10% error $V_C = 0.9 V_S$

$t = 2.3\tau$

Error = $\frac{1}{2}$ LSB

$V_C = V_S - \frac{V_S}{2^{n+1}}$ $t = -\ln \frac{1}{2^{n+1}} * \tau$ → بستنى هاد الوقت وبفتح ال switch

n → # of bits

Note:

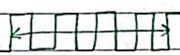
for 8 bits → $t = -\ln \frac{1}{2^{8+1}} * \tau = 6.2\tau$

for 10 bits → $t = 7.9\tau$

كل ما زاد عدد ال bits ال Accuracy بتزيد والعكس صحيح
يعني في tradeoff بين ال Accuracy وال Speed وعمومًا بتقل ال speed

→ TAD ≥ 1.6 μs for correct conversion

for the 10 bits ADC, We Need 12 TAD



→ we open switch & the M.C starts conversion.

Note: there are two Additional cycle for the opening switch

10 bits take → 12 cycle

10 cycle for bits And 2 for opening switch

*Example: Assume FOSC = 1MH what is the min Quantization time

TOSC = 1MS TAD = 2TOSC = 2MS

10 bits ADC → 12×2 = 24MS

*Example 8- FOSC = 10MH2

→ TOSC = 0.1MS → TAD = 16 TOSC = 1.6MS

→ TC = 12.1.6MS = 19.2MS

*Example: FOSC = 100KH2

TOSC = 10MS → TAD = 2TOSC = 20MS

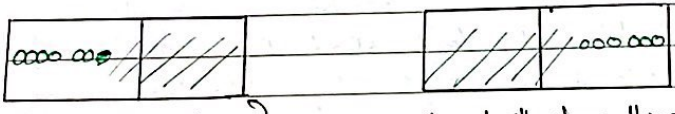
→ TAD from Internal RC = 2MS

Note: freq < 500KHz You should Always choose the Internal RC.

24MS

right justified

left justified



Always choose this

When I will choose this?

When I need the 8 bit result

→ 10 0000 0001 = 513

8 bits: 228 8 bits: 1

228 × 4 = 512 → 4×1 = 4

*Configure ADC (1-4)

Wait for sampling time - $-\ln\frac{1}{2^{n+1}} * \tau$ (you should write code)

Set GO/DONE bit (5)

Wait 12 TAD (No need for code) because we know (finish conversion from) read result

GO/Done = 0, ADIF = 1

(G/D)

Week 10

Chapter 11.8

* for 10 bits & $\frac{1}{2}$ Step accuracy

$(-\ln \frac{1}{2^{n+1}} * \tau) = R*C$ → $2ms + (R_S + R_{SS} + R_{IC}) * C_{hold} + Temp\ Coef$ (Op Amp Setting time)

→ Sampling delay = $2\mu s + 7.6 * (R_{IC} + R_S + R_{SS}) * C_{hold} + Temp\ coef$

| = 0.05 ms for each degree > 25°

* Example: $C_{hold} = 120pF$ / $R_S = \emptyset$ / $R_{SS} = 7K\Omega$ / $R_{IC} = 1K\Omega$ / op_Amp = 2μs

Temp = 35°C

→ Answer:

Aq time = $7.6 * (1*10^3 + 0 + 7\times10^3) * 120\times10^{-12} + 120\times10^{-12} + 2\times10^{-6}$

$+ 0.05\times10^{-6} * 10 = 9.8\mu s$

Sampling time = 9.8 μs

Conversion time = $12 * T_{AD} = 19.2\mu s$ (we Do Not write code(delay))

One Sample needs $9.8 + 19.2 = 29\mu s$

ONE Sample assuming repetitive Sampling = $29\mu s + 32\mu s = 32.2\mu s$

Enter Sample time

| 9.8 μs Sample | 19.2 Convert | 32 μs |

↳ Enter Sample time = $2 T_{AD}$

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

$f_{max} = 15.5\ KHz$ ← $\frac{32.2}{2}$

Slide 33:

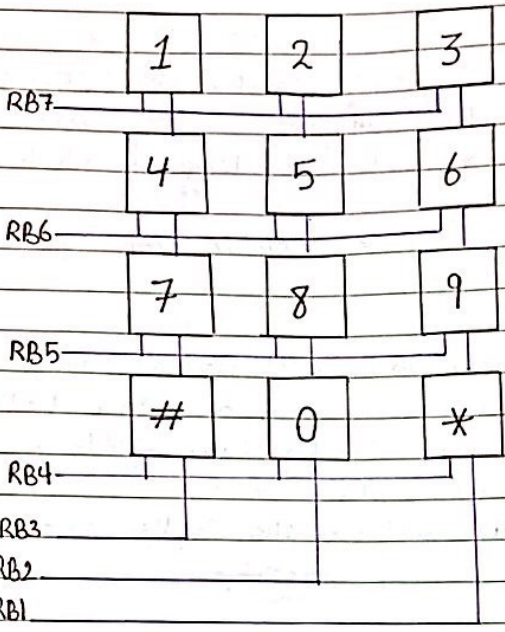
Sample time 10μs / ADCON1 = 10XX 1110 / ADCON0 = 01000X1

$T_{AD} \geq 1.6\ \mu s$ $F_{osc} = 20MHz$ → $T_{osc} = \frac{1}{20MHz} = 0.05\mu s$

$T_{AD} = 8 T_{osc}$ X → $32 T_{osc}$ ✓

$0.4\ \mu s < 1.6\ \mu s$ ↳ $32 * 0.05\ \mu s \rightarrow 1.6\ \mu s$

chapter 8s



6

row Index = 1dll xxxx

Column Index = xxxx 1102

common Cathode

Display number (5)

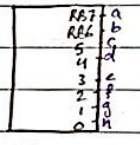
Select bl

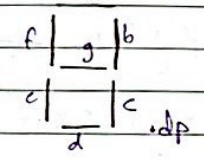
clrf TRISB

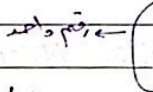
Select bo

MOVLW B'1011 0110'

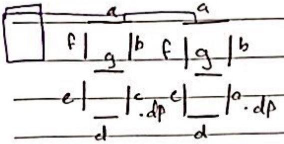
MOVWF PORTB







look up table	ADDWF PCL,1		Display 3
Look up table	retlw	B' 1111 1100'	MOVLW 3
	retlw	B'0110 0000'	call Lookup
(9-0) no plus	retlw	B '1101 1010'	MOVWF PORTB
	retlw		
	retlw		
	retlw		
	retlw		
	retlw		
	retlw		
	retlw		



(2) 7 segment

→ common cathode

بصير اشبك الـ com لـ a بنفس الـ pin ، جوا بصير بس 8 →

Display Number 25

↓
8 pins instead of 16
↓
10 pins

MOVLW B'5'
MOVWF PORTB

هاد الكود بطلع رقم 55

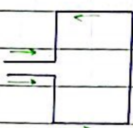
```
loop BSF PORTA,0        → MOVLW D'50'
     BCF PORTA,1          MOVWF Counter
     MOVLW B'5'
     MOVWF PORTB
     delay 10 ms
     BSF PORTA,1
     BCF PORTA,0
     MOVLW B'2'
     MOVWF PORTB
     delay 10ms          → DECFSZ Counter,1
     GOTO loop
```

الكود رح يضل شغال
Unlimited

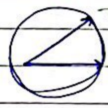
بعد ما يشتغل بصير يضوي
1 sec

Week 11
Chapter 8:
Slide 33

* اتجاه الـ current
* سرعة الدوران بتعتمد على
قوة الـ current



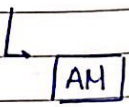
Stepper motor:



To make it smoother
* more steps
↳ you need
* more inductors

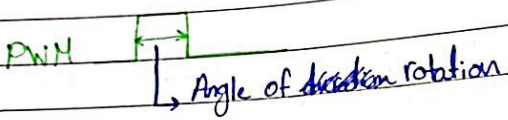
Servo motor:

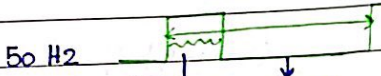
PWM (Pulse with modulation)



AM
5V → 1
2V → 0

FM
freq
1K H2 → 0
10 KH2 → 1

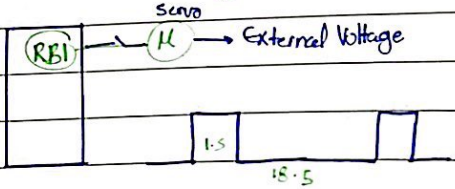




1.75 ms = 90°

1.75 ms = 180°

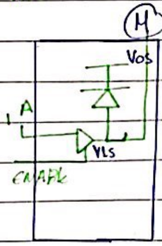
$35^\circ = 1.25 + \frac{1.5 - 1.25}{90} \times 35$



90°

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

Slide 43:



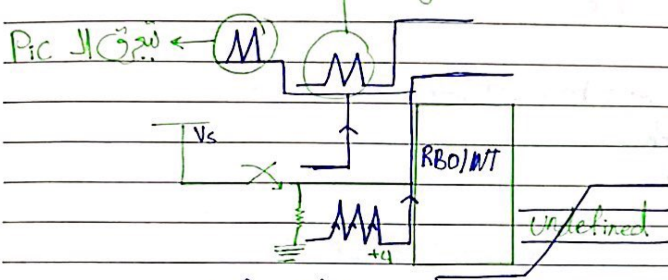
VLS: logic high voltage ≡ VLS

Vos: chip Enable

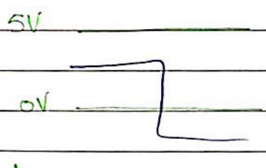
VLS < Vos

Vs = 0 →

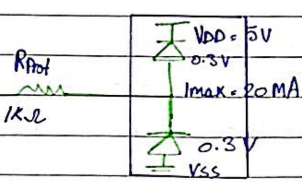
No Voltage will rotate




```
ISR  INCF Counter, 1
     BCF INTCON, INIF
     retfie
```



Slide 48:

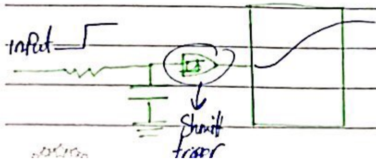


$Max = 1\times10^{3}\times20\times10^{-3} + 0.3 + 5V = 25.3V$

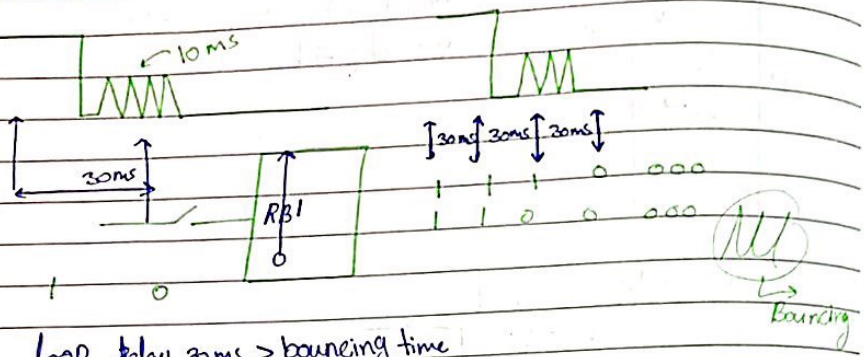
$min = -0.3 - 1\times10^{3}\times20\times10^{-3} = -20.3V$

← حاول مشكلة ال noise بلزم

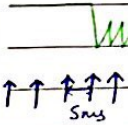
استخدم filter



Slide 52:



loop delay 30ms > bouncing time
read switch
GoTo loop



1 1 0 1 0 0 → Bouncing

chapter 9
Slide 7

→ 16 bit counter → TMR1H_TMR1L
* ON/off bit
→ The clock can be synchronized or Not
→ The prescale values (1,2,4,8)
→ clock sources: Internal , external RC0 for counting
external RC1 for External OSC up to 200KHz
* if clock is External → TMR1 Keep counting in sleep mode & may wakeup M.C
→ Delay of TMR1:

$\frac{4}{Fosc} \times prescale \times (2^{16} - N)$

Max delay = $\frac{4}{Fosc} \times 8 \times 2^{16}$
8 x TMR0 Max delay

* What are the setting to make TMR1 give delay of 1sec if $F_{osc} = 400$ KHz

→

$$delay = \frac{4}{F_{osc}} \times pre \times (2^{16} - IN)$$

$$1 = \frac{4}{400 \times 10^3} \times pre \times x \qquad Pre \times x = 10^5$$

Prescale = 4 → $x = 25000$ $\qquad IN = 2^{16} - 25000 = 40,536$

→ 40,536 → Binary → 1100 1100, 1010 0000 → القيمة أكثر في جوها

(1100 1100 → TMR1H, 1010 0000 → TMR1ø)

TMR1CON = [ ] → setting for Internal clock (TMR1 → ON) (Prescale = 4)

Slide 10

* 8 bits counter
* Counts from ø to PR2 value
* When TMR2 = PR2 it's start counting again from zero
* ON/off bit
* only Internal clock from $\frac{F_{osc}}{4}$
* doesn't count in sleep mode → can't wakeup PIC
* has postcode (1-16), when TMR2 = PR2, # of times = Prescale → TMR2IF = 1

علشان يصير فيه Interrupt
TMR2IE, GIE, PEIE

$$Delay\ of\ Timer\ 2 = \frac{4}{F_{osc}} \times prescale \times Prescale \times (PR2 + 1)$$

$$Max\ delay = \frac{4}{F_{osc}} \times 16 \times 16 \times 256$$ → Max delay of TMRø

* Example: what are the setting to give delay using TMR2 of 10ms, $F_{osc} = 4$ MHz

$$10 \times 10^{-3} = \frac{4}{4 \times 10^6} \times Pre \times Post \times (PR2 + 1)$$

$Pre \times Post \times (PR2 + 1) = 10,000$

16 × 16 , PR2 → 255 لازم تكون

او اقل عشان يكون أكثر مقبول