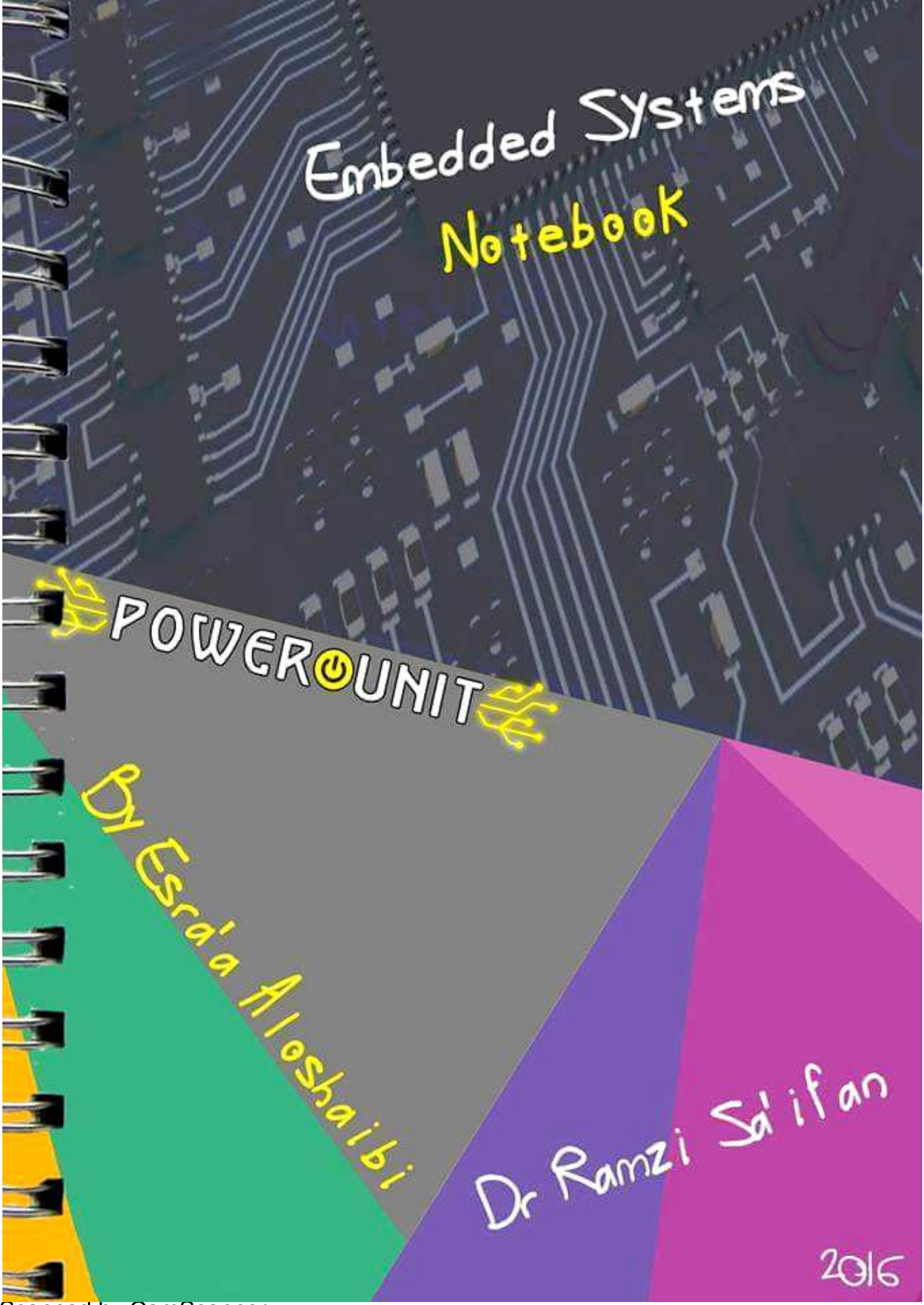


Scanned by CamScanner

* أي non computing device عليه micro controller يعتبر embedded system

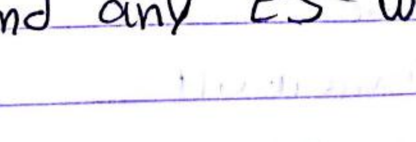
Real time computing: There's a deadline for the result to appear, if results appear after this time, they are useless.

مثلاً لما أكون مبرمجة جهاز معين أنو يحل شغلة بعد ثانيتين ازا عملها بعد ٤ ثواني بكون عالفاضي.

Slide 4 مهمة

* حتى أشتغل على أي Embedded system لازم يكون عندي الكود ومتى ما بشتغل ع الكود لازم نعمله backup.

لو ما قدرنا نحصل ع الكود لنشتغل عال input ونقرأ ال output ونشوف العلاقة.



* To understand any ES we should know:
1. Inputs
2. outputs
3. User interaction
4. Links to other systems.

شرح وصف ال Characteristics

* Software driven : the whole ES controlled by the software (program)
كل تحكمنا بال Es يتم عن طريق الكود

* Reliable : almost 100% work correctly.
دائماً يشتغل

* البرنامج ما لازم يوصل لل End لإنو ازا وصلها مبيفصل النظام
منكون عندي go to بمكان معين جوا البرنامج

* example
desired temperature (Td) درجة الحرارة اللي أنا بدي توصلني
actual Temperature (Ta) الدرجة الحقيقية

if Ta > Td
Turn compressor on وظيفة المبرد
else
Turn it off

* ال microcontroller عبارة عن computer على chip جزء

* slide 15 . مهمة

- The main components of microcontroller or computer:

1) CPU → control و بعمل عمليات حسابية ومنطقية

2) Memory. → Data memory ; it holds the values of variables (Volatile ; lost if power off).

→ Program memory ; it holds the program instructions. (premenant ; kept if power off).

3) I/O

4) Busses.

* البرنامج بكتب مرة وحدة. أما ال variables ممكن أخذ قيمهم اكثر من مرة

Data memory is less writing cost

→ power → time.

طب ما أنا volatile مش لازم!

the values of the variable is dependent on location inside the program

ربما اني بدي اكتب قيم ال variable اكثر من مرة ؛ فمنطقي يكون بمكان سريع الكتابة ويوخذ power قليلة. عشان هيك نحطها ال memory لحالها

Scanned by CamScanner

* int x;

=

x = 1 ;

x = x + 2 ;

⋮

x = x * 2 ;

لو هون انفصلت الـ power ورجعت شغلتو مرة رح يجيب من أول بجملة x = 0 وبكمل عادي . وبمسح وين كان واصل. عشانه هيك ما بخزني ابدا اني

Voltaile

* when power up, the cpu read (fetch) the first instruction from the program.

أول ما أعطي power للـ computer أو للـ ES- بتبدأ الـ cpu تقرأ البرنامج لأنه البرنامج هو اللي بتحكم بالـ system كامل.

بتجيب أول instruction و بتبدأ تنفذها ازا add بتجمع وإذا sub بتطرح وهكذا

و النتيجة ممكن تتخزن بـ variable أو تطلع عالـ output أو تبعت input

وهي بتنفذ أول instruction بتكون بتجيب ثاني instruction وهاد

بنسميه pipelining

* لازم البرنامج يكون مخزن بمكان معين ودائماً أول instruction

الـ address تاعها 0

* While executing the instruction, the CPU fetching the next one. (pipe lining)

PC → Program Counter بأشر عال address تاع ال instruction الي بدها تتنفذ.

وأول ما تدخل ال power بتكون قيمته صفر

→ auto increment → بزيد كاله حتى يجيب ال instruction التالية.

* based on four main components; we have two types of hardware and two types of software.

- Software
  - CISC
  - RISC
- hardware
  - the Von Neumann Architecture
  - the Harvard Architecture

Scanned by CamScanner

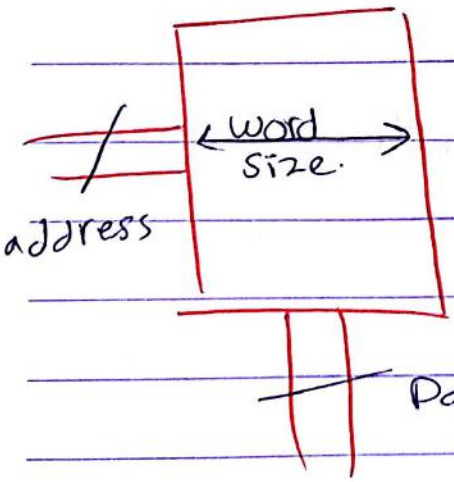
* in Harvard Architecture there are two busses for each of Data memory, Program memory and I/O., in von Architecture there are only two busses connecting all

Harvard → كل واحد الو busses خاصة فيه وهو أسرع من الثاني و حجم الـ busses تختلف.

هون الـ pipelining أحسن

Von → الكل بيعل عليهم access وكا واحد بعل access لباقي بيستنى و حجم الـ bus ثابت وبحدد حسب أكبر bus منهم "fixed size"

* To deal with any memory we need 2 busses Data and address.

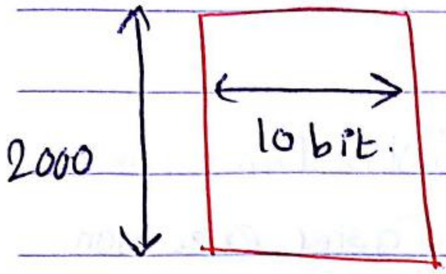


* حجم الداتا بس بحدد الـ word size
data bus

* minimum address bus = $\lceil \log_2 \#\ of\ words \rceil$

Scanned by CamScanner

* ex :

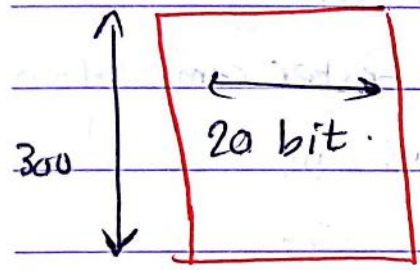


Data memory

Sol : word size = 10 bits.

memory size = $\frac{2000 * 10}{8}$ Bytes

8 لـ للتحويل ال bytes



Prog memory

Sol : Word size = 20 bits

memory size = $\frac{300 * 20}{8}$ Bytes

← تابع للرسمات .

* Harvard → data bus = 10 bits. , databus = 20 bits.

address bus = $\lceil \log_2 2000 \rceil$, address bus = $\lceil \log_2 300 \rceil$

= 11 bits = 9 bits

* VON → data bus = MAX {10, 20} = 20 bits

address bus = MAX {11, 9} = 11 bits.

* بنعمل ال ⌈ ⌉ عشان نغطي ال memory كاملة ونوصل لكل ال locations

ودائما نزيد bits ولا ننقص

هاي اللي استخدمنا

* According to HW : VNA and Harvard

According to SW :

هاي الـ inst مختصرة

(1) RISC → only few simple instructions.

هاي الاشي مختصرين

Longer program , Faster execution

because pipeling is more efficient

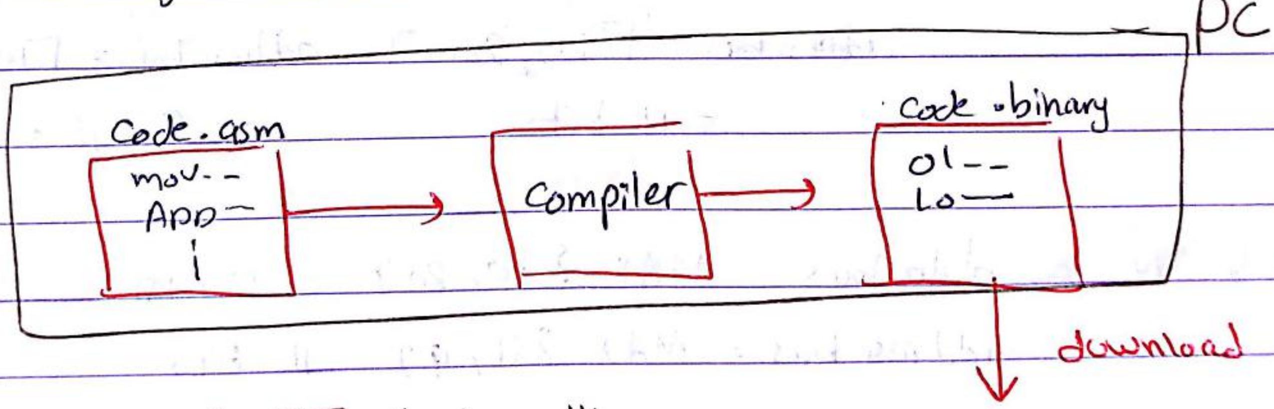
مجموعة كبيرة الـ inst

(2) CISC : → too many instructions including

ex : mul , DIV , SQRT ← complex ones , also many addressing modes. , Faster compilation

↵ عدد الـ inst أقل

* Instructions in RISC take the same execution time same fetching time , and they are of the same length.



اللي بهمني الـ ET يكون أسرع

مش الـ CT لأنو الـ CT مرة

وحدة بدي اياها الـ ET

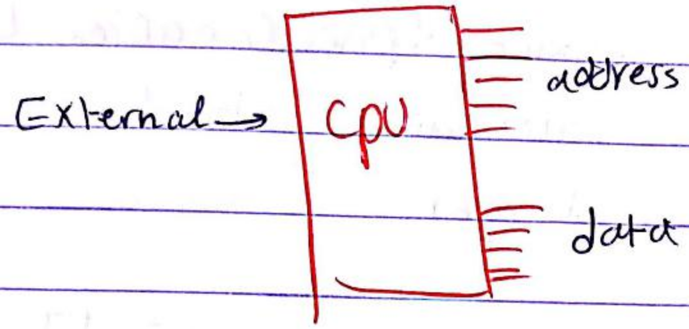
all the time

RISC is faster

on power up PC = 0 and starts execution.

* addressing mode : is How to define the operand in the instructions.

* لما تكون الـ memory external بيكون حجمها اكبر والي بكبر اكثر عدد الـ pins للـ cpu الي طالعين منها



* Reset → متى يرجع لبداية البرنامج

* interrupt → اللي بيقطع تنفيذ البرنامج .

*slide 22:

microProcessors within the same family have the •
Same Core :
Same cpu arch
Same Instruction set.
Same Instruction format and size

* moving from microcontroller to another within the same family, doesn't required learning new inst set, However, it needs learning new peripherals.
↓
prog

* Size of microcontroller chip is determined by:

1. number of pins → specified by peripherals number.
2. spacing between pins → more space, easier to interface with bus larger.

* DIP : dual in line packaging

* Working Register (Accumulator) :
holds the result of the last instruction.

* 8 bit M.C means that the word size in the Data memory is 8 bits ⇒ all variables are 8 bits ⇒ all instructions are done on 8 bits operands.

Scanned by CamScanner

هاد اللي منتحاول نحو ٥

* 8 bits M.C ; 0 ≤ all , values ≤ 255

Cimos ← don't care.

12CX / 12FX

series ← رقم ال ← Flash memory

16C(5X)

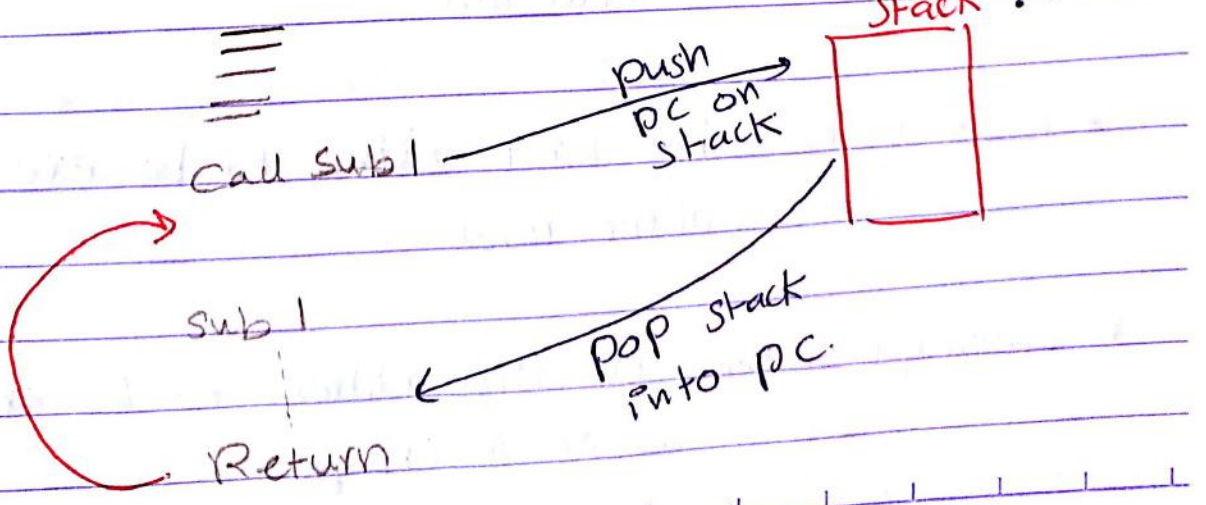
رقم ال M.C جوا

series ال

16F877(A) , 16F(87)7

advanced. ← رقم ال M.C جوا ال series اللي بقوا 16

* Stack → memory جزء من ال
بخزن فيه قيمة ال pc لما
يكون عندي call لأي معنى



Scanned by CamScanner

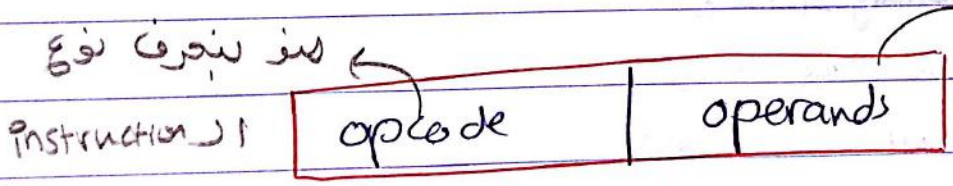
* Stack red and written automatically by Call and return.

* levels Stack: you can do 2 nested calls.

```
Call  sub1
Call  sub 2
sub1
  return
sub 2
  return
```

not nested Calls needs 1 stack level.

* Instruction word size: Size of the instruction



* Reset Vector: the first address to be executed after reset. (address 0 ...)

* Interrupt Vector: the first address to be executed after interrupt. (address 4 ...)

Scanned by CamScanner

دلحالة ازا الكونترا بس فيه Interrupt لازم يباشر بـ go to.

* more interrupt vectors Facilitates for priorities between interrupts.

* Single interrupt vector ⟹ all interrupts have same priority.

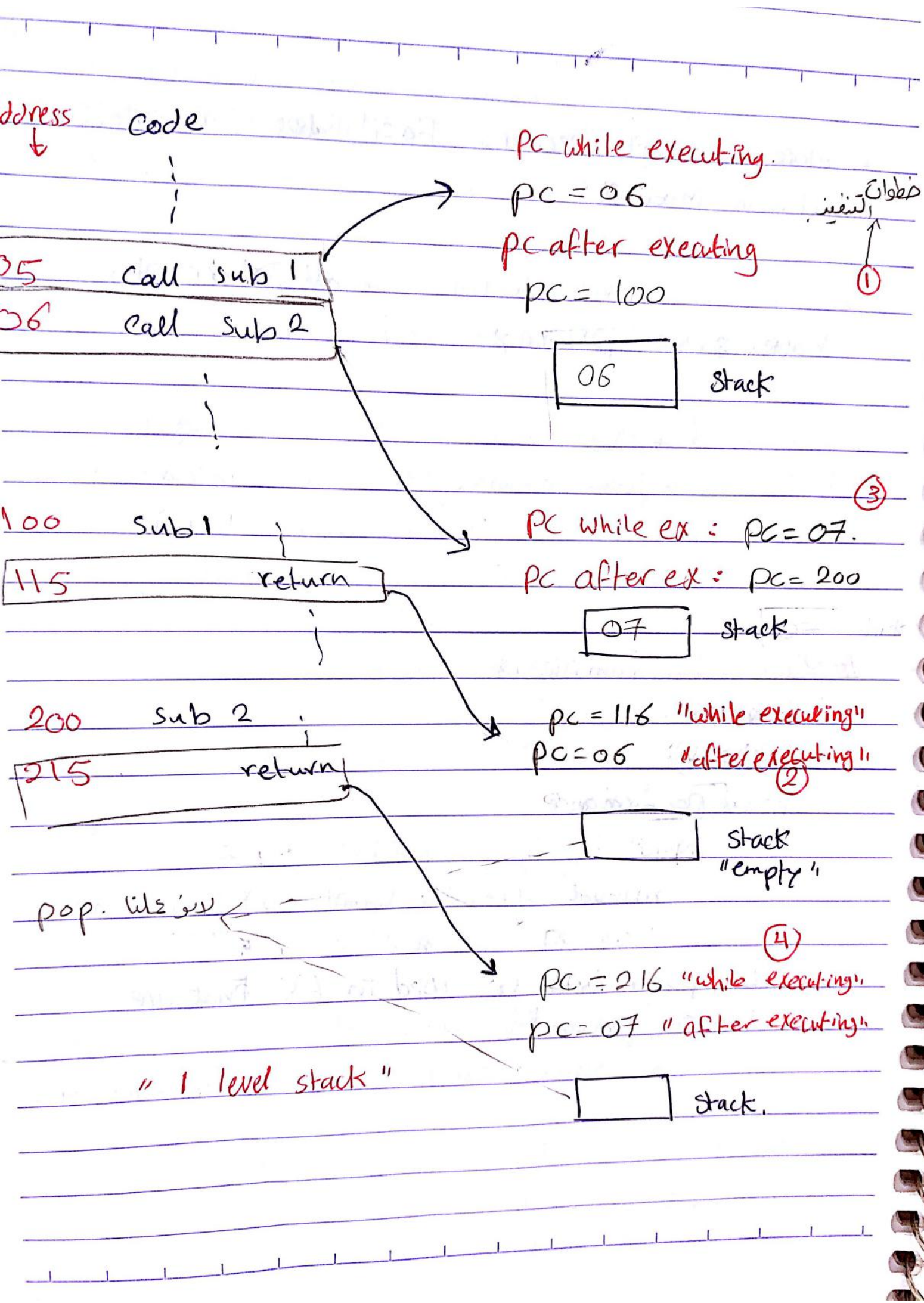
كل ما يصير عندي اكثر من interrupt بين عندي اكثر من مكانة عكن أروحلو المكانة اللي بدو حلو حسب الاولوية وأما بال single vector بروح ع مكانة واحد "فشة" اولوية.

* PIC 8 bit families :
- base line.
- mid range
- high performance.

* مش لازم اعرف من وين Family بنوفى ال manual.

L : Low power / can be used in ES that are battery powered.

Scanned by CamScanner



Scanned by CamScanner

* slide 30 :

- Instruction Register (IR): holds the fetched instruction.

* for any instructions that requires two operands, one of them comes from W-Reg.

* The second operand comes either from the instruction (literal) or from the data memory.

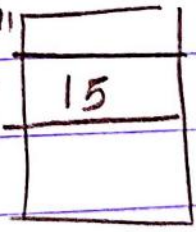
تفسير
آخر

→ the second operand value can be in the inst (literal) or its address in the instruction (direct address)

*ex:

addI 5 → literal "add the value 5"

addF 5 → address "add the value 15"



Data memory

Scanned by CamScanner

* the result will be stored either in Data Memory or in W-Reg.

* to add two variables : Var1 and Var2.

① move Var1 to W-Reg.
② add Var2.

* Status Reg : gives information about the last executed inst.
Like Carry ال Flags ال من مثلاً

Scanned by CamScanner

* Chapter 2:

16F84A → 18 pins.

* They will have 3 types of memory:

1- Data memory
2- Program memory
3- EEPROM: holds data, Perminent, not Frequently changed → holds the settings of ES.
لـ مثل الوقت والتاريخ

I/Os: [1] 8 pins Port B.

[2] 5 pins Port A.

[3] 8 bits timer.

Scanned by CamScanner

16F84A → series 16, midband Range family.
↳ 18 pins:

1. 8 pins for Port B.

2. 5 pins Port A.

3. Power supply and ground (2pins)
   VDD VSS.

active low
4. $\overline{MCLR}$ resets on 0 (1 pins)

5. 2 pins for OSC

X "Symbol"
Pin

minimum size of address bus → 1K → 1000 → 10 bit.
= = Data memory → $\lceil \log_2 68 \rceil = 7$ bit.

Scanned by CamScanner

* File Reg = data memory location.

* slide 10:

Configuration Word: 14 bits value stored in prog memory, can be written only once during prog download once written, can't be modified except by downloading the code again, it holds information about the prog.

bit 0 and bit 1 (Fosc0 and Fosc1) determines osc type that can be connected:

11	RC osc	each of them can work on specific Frequency range.
10	HS osc	
01	XT osc	
00	LP osc	

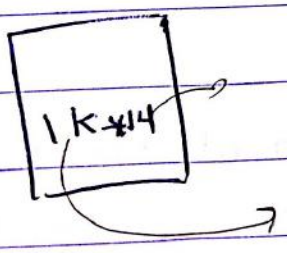
Scanned by CamScanner

* WDTE : Watch dog timer enable bit enables or disable WDT.

* WDT : resets the microcontroller on crashes.

* $\overline{PWRTE}$: if enabled (=0) on power up the MC stays in reset mode for a while. يشان يحمي ال p

* CP : Code protection.
if =0 → the code is encryped "hidden".



word size = 14 bit.

بناخد اكبر اشي يكن نكتبو

1K. هيو

Scanned by CamScanner

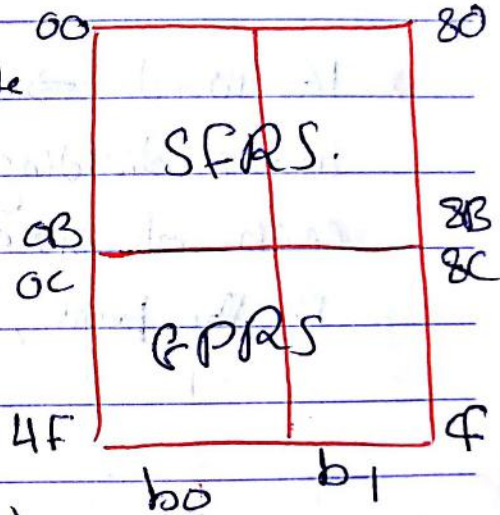
* Data memory :

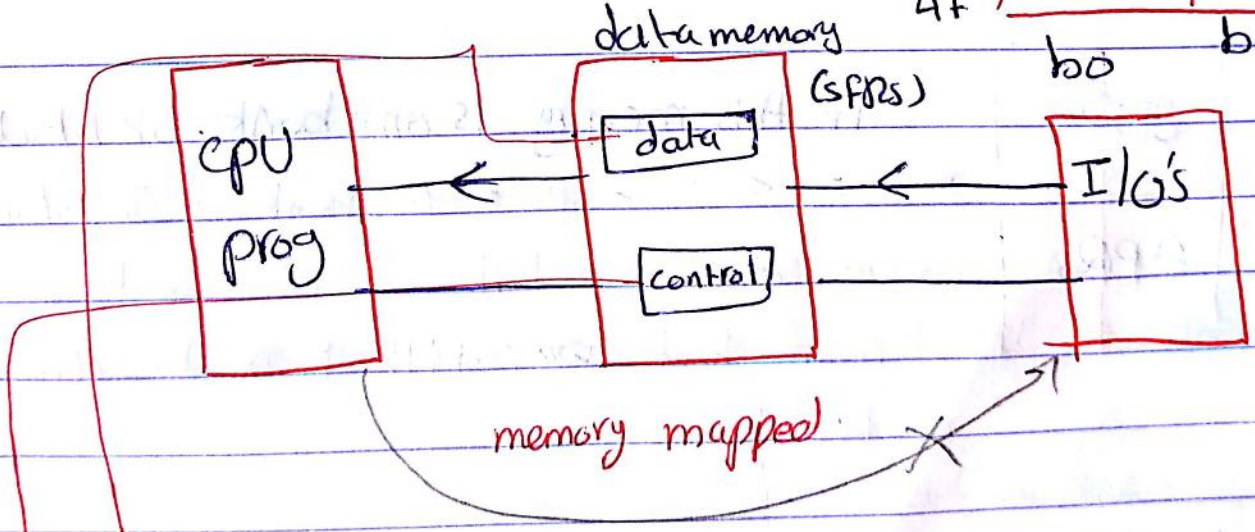
التعامل مع الـ I/Os يتم من خلال الـ SFRS

Vertically :

1. Special Function registers (SFRs)
2. General purpose registers (GPRS)

→ they have additional function beside being data memory locations.





فيه اتصال مباشر بينهم

→ data to output (output) or the entered data (input)

→ specifies settings of I/Os

Scanned by CamScanner

* GPRs: Store your own variables.

لو خزنا ال variables بدال SFRs ما بطلع error بس رح يأثر

عال settings.

* Horizantally:

2 banks, each of 80 words → total size = 160 words

79 = 4F ← من ال 00 لل 4F

* 160 words ⇒ to be addressed require $\lceil \log_2 160 \rceil = 8$ bits

after dividing data memory into two banks each of size 80 words, to address any word in the bank, we need $\lceil \log_2 80 \rceil = 7$ bits

⇒ if this memory is one bank ⇒ 8 bit address

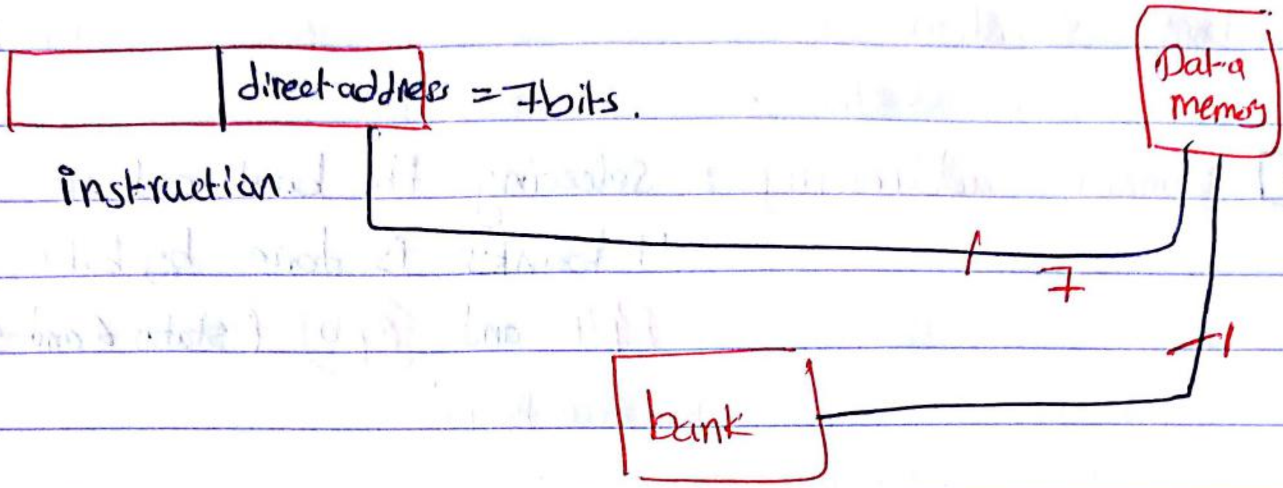
" " " " two bank ⇒ 7 bit address

we saved one bit out of the 14 bits inst but, we still have to select the bank.

* to access (read or write) any data memory location:

1. we choose the bank if not chosen
2. we access the word inside the chosen bank.

Scanned by CamScanner



* the SFRs that exist in the 2 banks are the same if modified in one bank the other is modified.

معظم ال Registers بكون ستخدم كتير وعشانه ما أضل انتقل من ال bank للثاني كبرنامج . وإذا أنا ب bank وبدي Reg موجود بال bank التاني لازم انتقله . والحالة التانية إذا ال bank الي أنا فيه تيجي.

* Addressing modes:

1) Literal addressing : the value is in the instruction

2) direct addressing : the address of the value is in the instruction.

3) Indirect addressing : the address of the value is in the SFR called File Select Register (FSR)

* bank seleetion:

(1) Direct addressing : seleeting the bank out of 4 banks is done by bits RP1 and RP0 (status 6 and 5 for two banks.

(2) Indireet addressing : 4 banks ⇒ IRP (status 7) and FSR 7 for two banks

* slide 15:

Write pseudo code to read data memory location 87_H.

$87_H \rightarrow$ 1000 0111

↳ bank 1

Code → instruction read address 87 H.

opcode to read	7 bits 87

↓ compiler

read 07.

Scanned by CamScanner

/ /

Sol:

Select bank 1 (PRO = 1)

read 07

1000 0111

ال address اللي جوا
ال inst دائماً 7bit

وبنقدر نحكي

read 87 ⇒ صحيح اخذنا اول 7bit
.1 000 0111 بس بضيف 1
عشان bank 1

* Set RP0 = 1

read location 27 of bank 1.

read address 27 H ⇒ read [1]010 0111
من ال bank غير
A7 H.

read address 87H ⇒ read [1]000 0111

7 bit

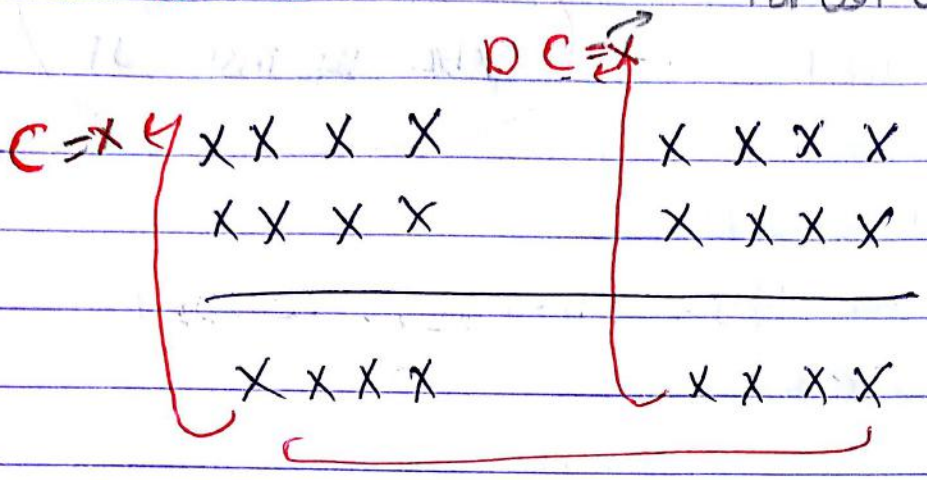
27 → ~~0~~010 0111

bank ← 1010 0111 ✓

A7

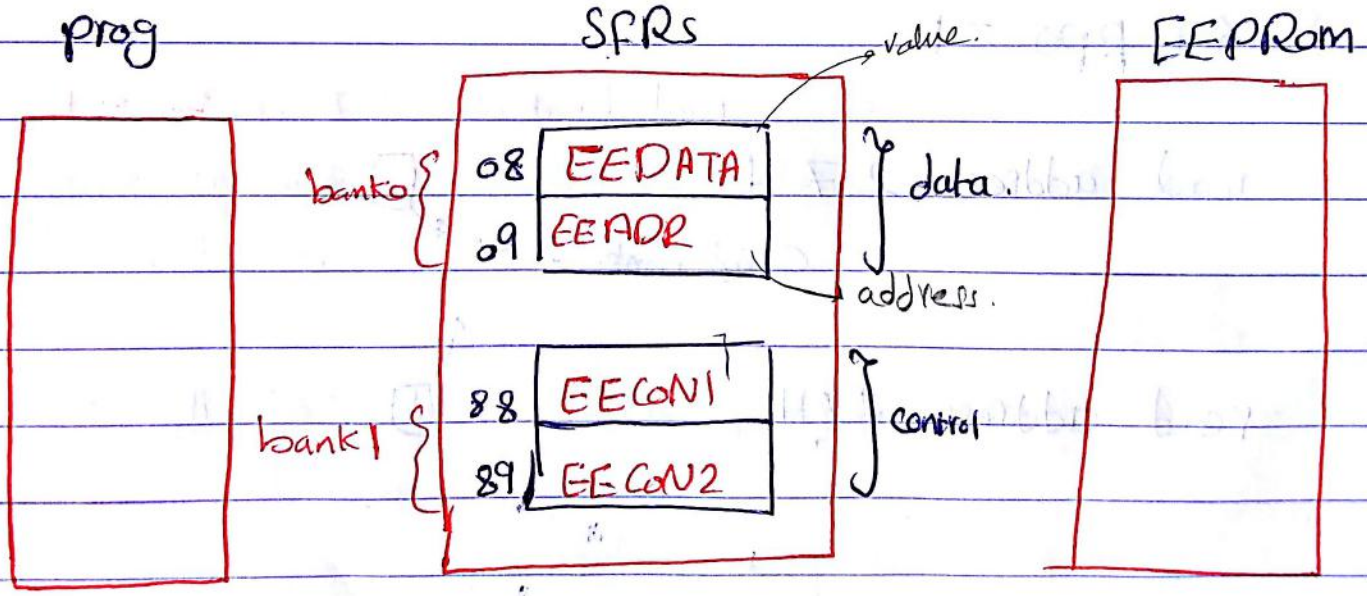
* slide 16:

4bit لأول من Carry لـ



if R=0 ⇒ Z=1

Result.



* Store value 5 in EE pRom address 15.

EEDATA = 5 , EEADR = 15

Scanned by CamScanner

/ /

* Slide 17: عملية الـ Read

store the address to read in EEADR.

set the RD in EECon1. "start reading"

EEADR موجودة بـ bank0 والـ EECon1 موجودة بـ bank1

فلازم نعمل switch للـ banks "خطوة مهمة جداً"

* The data in EEPRom at address = [EEADR] will be coppied into EEDATA.

مش بدنا نعمل switch banks تاني نرجع من الـ EECon1 للـ EEADR أو أي مكان تاني بالـ bank0.

* RD bit is set by sw, cleared by HW after reading is finished => RD = 0.

* to write data on EEPRom:

1. set EEDATA and EEADR.

2. switch banks.

3. set WREN in EECon1 (writing EEPROM is enabled)

يعني لا يعني بداية الكتابة

Scanned by CamScanner

④ write 55H then AAH in EECON2.
(to make sure that writing is not by mistake).

0101 0101
1010 1010

⑤ Set WR bit in EECON1 (start writing)

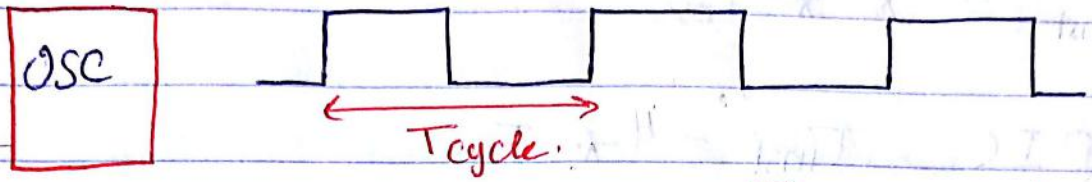
⑥ after writing finishes ⇒ EEIF = 1 in EECON1

* WR set by SW cleared by HW.

لو بدي أخبيا ال Procegur تقبرا ستغني عن خطوة H3

* عكن يجيلنا جوة instruction وتقلنا سيئ مفهوم الخلط.
الخلط اللي تقل لل Clear لل WR أو لل RD.
← يعني لل bit اللي cleared by HW.

Scanned by CamScanner



جزء بيولد clock

generation

$F = \frac{1}{T_{cycle}}$ $F_{OSC} = \frac{1}{T_{OSC}}$

* delay by:

(1) SW: Code1 → loop 10000 → loop do nothing

nothing

go to loop.

Code 2

(2) Hw: timer: timer 0

* Instruction cycle: the time that is enough to either fetch or execute single instruction.

⇒ to execute one inst, we need 2 T ins for fetch and then execute.

Fetch time = execution time.

Scanned by CamScanner

* $T_{inst} = X * T_{osc}$ (X → Factor)

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

*ex:

$F_{osc} = 4\,MHz$.

$\Rightarrow T_{osc} = \frac{1}{4\,MHz} = 0.25\,\mu s$

$\Rightarrow T_{inst} = 4 * 0.25 = 1\,\mu s$.

*ex:

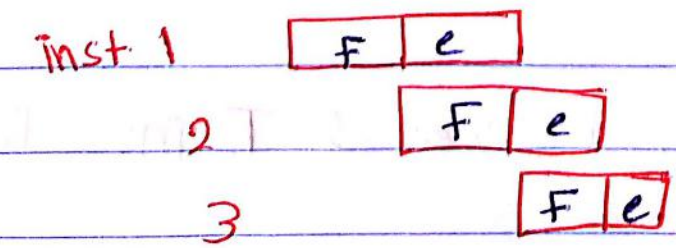
given a prog composed of 10 inst, $F_{osc} = 4\,MHz$.
How long does it take to execute this prog?

each instruction takes $2\,T_{inst} = 2\,\mu s$.

10 instruction $\Rightarrow 10 * 2\,\mu s$

$= 20\,\mu s$ (without pipelining)

with pipelining $\Rightarrow 11\,\mu s$ — 2 µs (أول تعليمة) والباقي وراها (9) تعليمات (1 µs)



Scanned by CamScanner

* time with pipelining $= \frac{1}{2}$ time without pipelining.

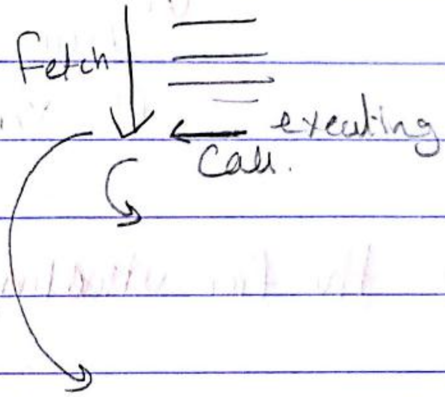
* pipelining fails when the fetched inst is not the one that will be executed.

This required one more T_{inst} to fetch the right inst.

وانا بال call بكون على الـ fetch للـ inst

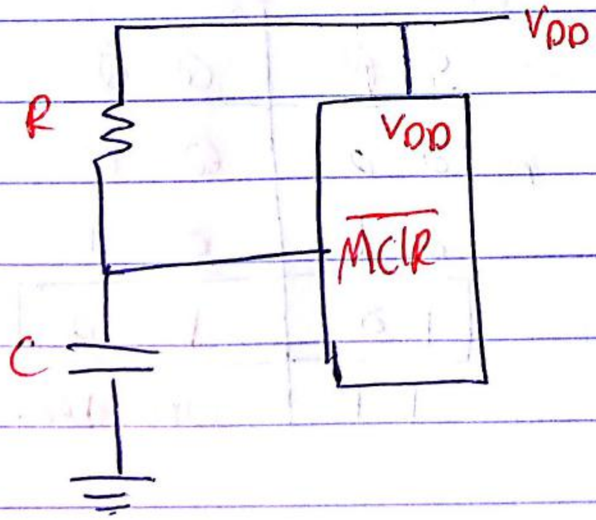
الي وراها وانا بكون بدي أروح ع inst

تانية فبنعمل flush للـ inst الأولى



* To keep M.C in Reset mode on power up:

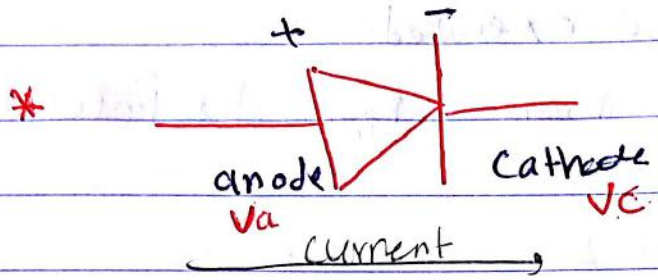
[I] external (بنوصل circuits خارجية)



on power up (VDD) ⇒ after the capacitor gets charged $\overline{MCLR} = 1$ ⇒ exits from reset and it takes the according to $\tau = RC$.

(2) Internal (code بال)

$\overline{RWRTE} = 0 \Longrightarrow$ delay on power up.

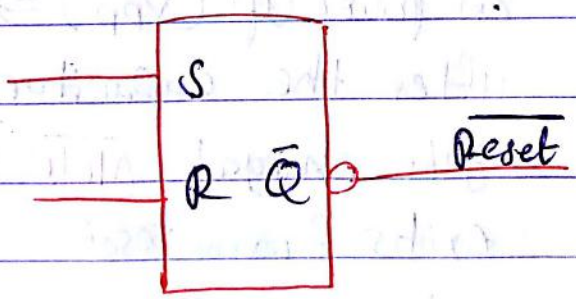


if $V_A > V_c$ + threshold.

* the free wheeling diode: used to discharge the capacitor very fast.

لو ما ضفنا الـ diode الـ cap. صيقوّل حتى يفرّغ فجأة
وقت حتى يطفي الجهاز.

* slide 23:



S	R	Q	Q̄
0	0	Q	Q̄
0	1	0	1
1	0	1	0
1	1	unstable.	

الـ output بتبقى عكس على Q̄

Scanned by CamScanner

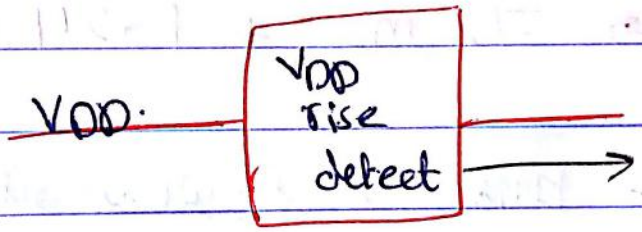
* Reset happens when S=1 and R=0.

1) $\overline{MCLR} = 0$

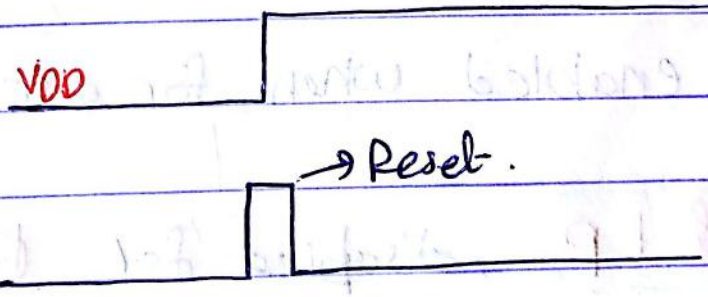
2) WDT and not in sleep mode.
   ↳ ما بشتغل بال sleep mode.

3) on power up

الـ inputs تكون الـ OR gate اللي متوصلة على S.



اول ما يصير حدث rising edge.
بتكون الـ Vo فترة قصيرة بحيث S=1 بعدين رجعت لـ 0



Scanned by CamScanner

10 bits counter ← يعدّ من 0 → 1023 → 0

بيرجع لل 0 وبهاي اللحظة بيعطينا 1

1024 cycle for output to be 1.

3) on power up : exits from reset mode after 1024 cycle from internal RC OSC + 1024 cycle from external OSC

typical value

72 ms + 1024 * 1/Tosc.

((الأشياء من الرسمة ال bubble الموجودة من PWRT))

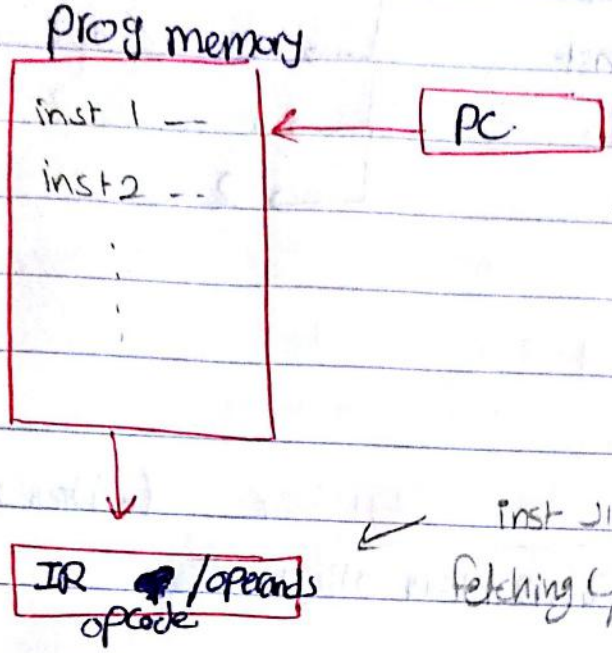
* enable OST:

by default enabled when for osc types

XT, Hs, & LP, disabled for RC OSC

من ال Configration word

* Chapter 4 :



HLL → Assembly → machine code

modify.

Assemble / compile → للتأكد من عدم وجود syntax error والحصول على machine code.

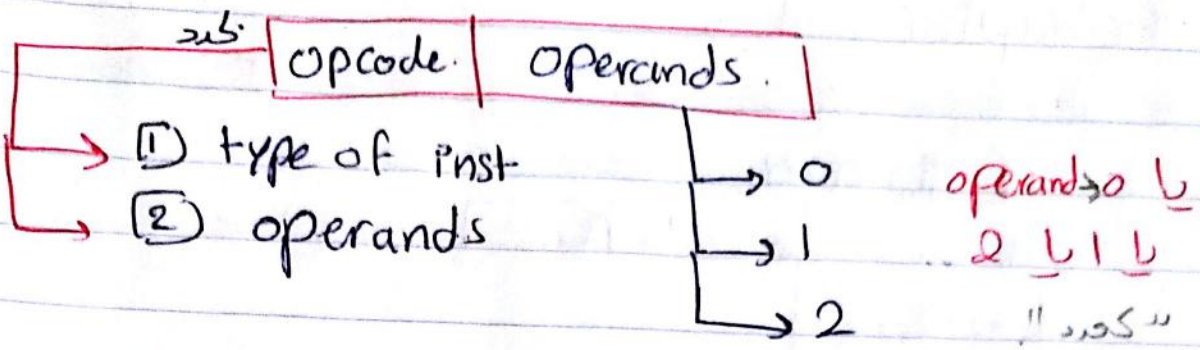
Simulate → للتأكد من عدم وجود أنواع الأخطاء الأخرى (logical , runtime...)

* Destination bit (d bit)

= 0 → destination in W-Reg

= 1 → = = File-Reg → DATA MEMORY

Scanned by CamScanner



* Categories of inst :

1. Byte orintied file Register (بجيبلي الـ address)
   8 bit. بشتغل على DATA MEM بشتغل بال

2. bit orintied file Register (بجيبلي الـ address ورقم الـ bit
   اللي بدي اشتغل عليها)

3. literal inst : no Data memory access
   The value is in the inst

4. Control inst : call, go to, return ---

DATA movement inst
ثلاثة أنواع " بينقلو الداتا
من مكان لمكان "

Scanned by CamScanner

* types of operands :

1) 7 bit file-Reg address (f).

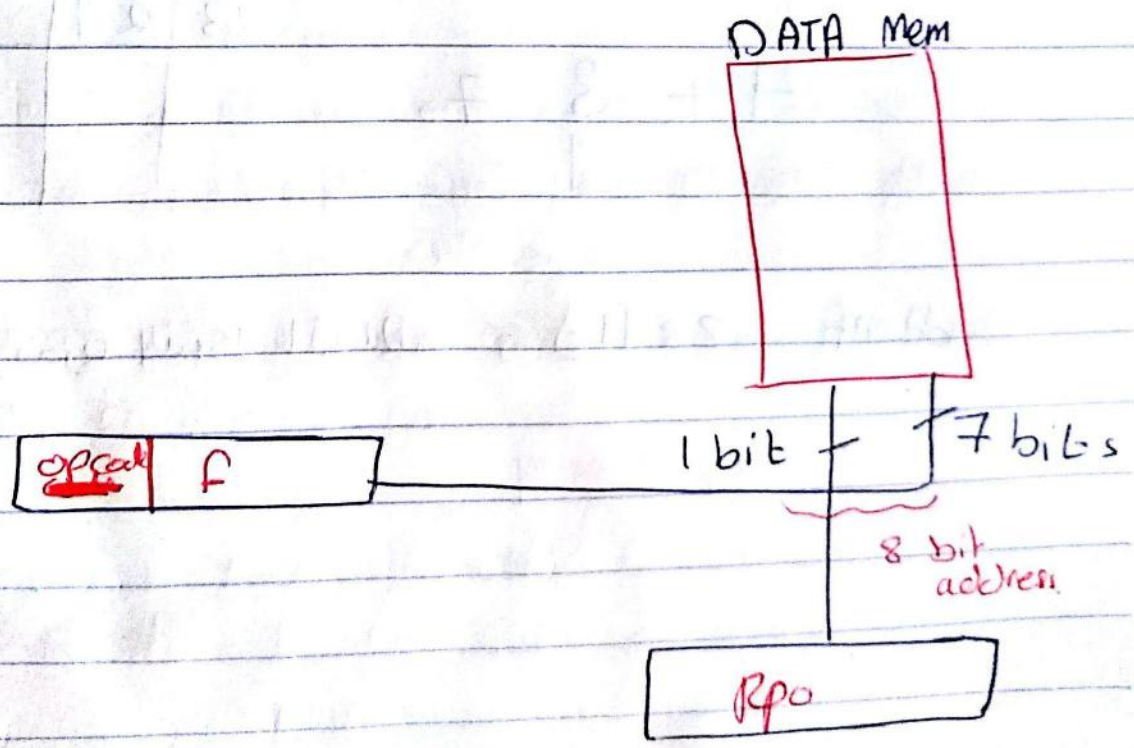
2) 3 bits bit location (b) → متى أحدد رقم ال bit
من 0 - 7 بالبنري 3 bits.
111 - 000

3) 1 bit destination bit (d)

4) 8 bits literal (k)

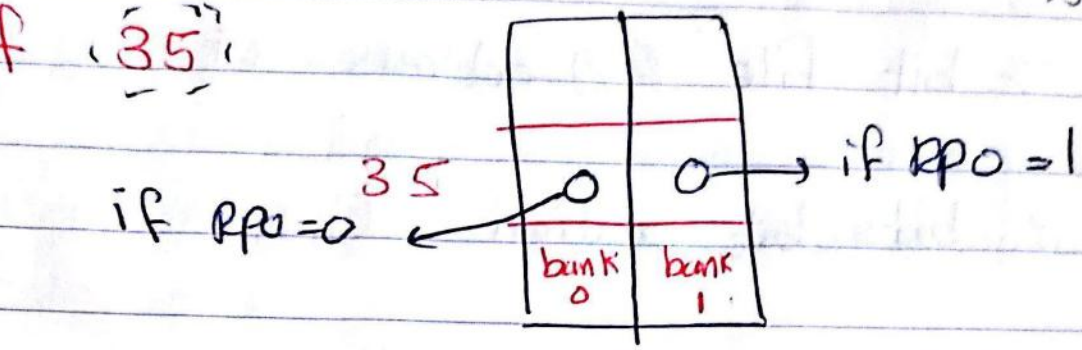
5) 11 bits instruction address (k)

↳ Call S→ 11 bits.



* clrw ○ operand w [0 0] بتخلي الـ Reg w تساوي صفر.

* clrf "35"
f



* addwf (add working to file Reg) "2 operand"

addwf 22,[0]

ع نخزن النتيجة بالـ Reg w

w [47]

22	3
23	29

القيمة الجديدة اللي ← 4 + 3 = 7 → القيمة اللي جوا الـ Reg w
ع أخذناها بالـ Reg w ↓ القيمة اللي جوا الـ address رقم 23

addwf 23,[1] → ع نخزن النتيجة بالـ address رقم 23

7 + 2 = 9

Scanned by CamScanner

* bcf f, b.

bank 0 "clear" bcf 03, 5 ← RP0 رقم 5 في bit

03 ← Status Reg.

5 ← RP0

bank 1 bsf 03, 5 "set"

* addlw (add literal to working)

addlw 3. ⇒ 7 + 3 = 10 ⇒ W

3 + 7 = 10

* all literal instruction store in W Reg.

* 6 bits يكون كل opcode الـ fixed size

opcode = $\lceil \log_2 55 \rceil$ = 6 bits

Inst عدد

~ 3 أو 4 أو 6 يكون يمكن variable حسب

Scanned by CamScanner

* بالنسبة للـ opcode الـ Control بياخد أول 3bit ازا كان Valid تتكمل الخطوات التانية للتنفيذ أما لو كان invalid بياخد الـ bit اللي وراها منصير 4 bit وهكذا ..

opcode1 = 101 } X هون لو أخد أول 3bit من opcode 2 صيكون valid يعني

opcode2 = 1011

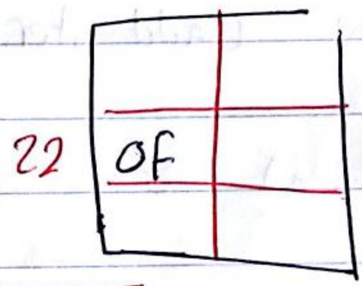
ما صيطلع عال bit الرابعة فصينفذ الاول والتاني بنفس الاشي.

* Comf F,d

Comf 22,0

0F → F0

"1's complement"



W = F0

* الـ Status Affected بتتغير بغض النظر عن مكان التخزين.

2 additions: ADDWF F,d [F]+[W]

ADDLW K

2 subtraction: SUBWF F,d [F] − [W]

SUBLW k [k] − [W]

دائماً اللي بينطرح هو الـ W

دائماً بالـ W Reg.

[ ] → content.

* من ال C Flag نعرف القيمة positive او negative

C = 0 → negative
C = 1 → positive

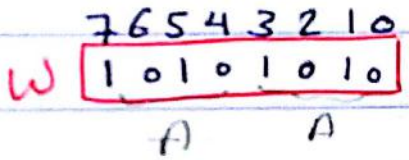
* logical Instructions:

masking

* ex:

- Complement the odd bits in w Reg.

XORLW AA



- Set the least significant 4 bits

IORLW 0F

- clear the even bits.

ANDLW AA

Scanned by CamScanner

* 0 XOR X = X

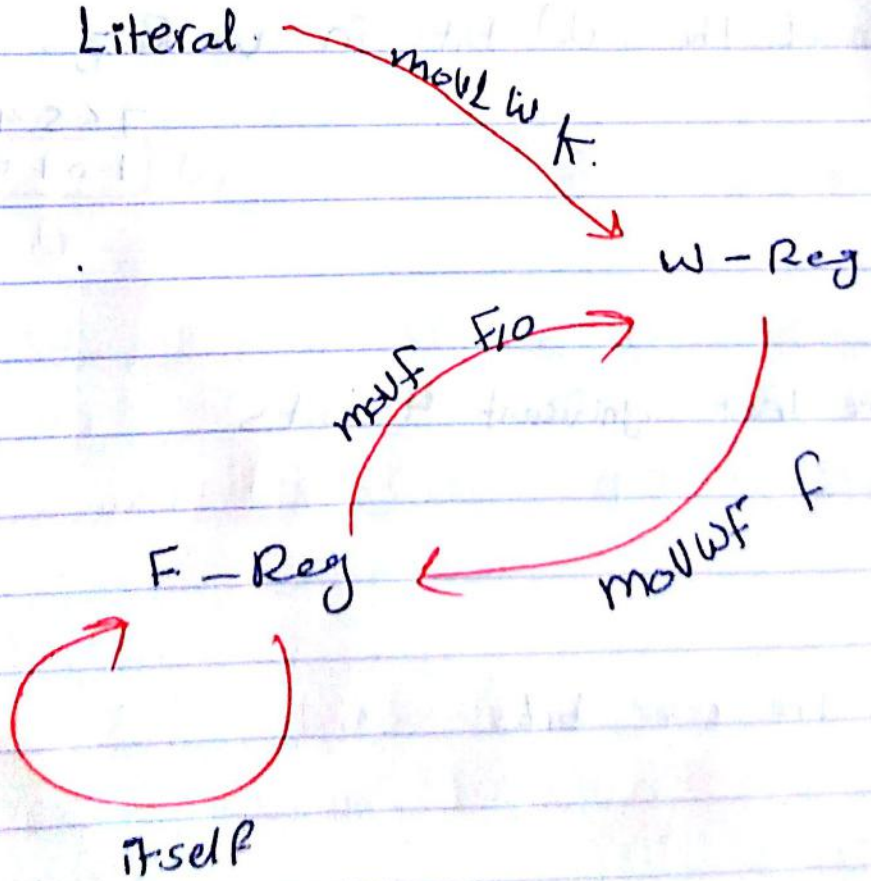
1 XOR X = $\bar{X}$

0 OR X = X

1 OR X = 1

0 AND X = 0

1 AND X = X



movf f,1 عشان اعرف القيمة
صفر ولا لا
بها اثر بتاثر ولا

Scanned by CamScanner

RP0=0

* to initialize any data memory location with a value :

MOVLW 5
movwf 22.

} حطيت القيمة 5 بال address رقم 22.

* to move the value in 22 to 23.

movf 22,0
movwf 23.

* SWAPF ← تبدل 4 bit مع 4 bit

Scanned by CamScanner

* Control Instructions :

* DECFSZ f, d : → skip يعني بروح على طول

it decrements F and if the result = 0.

⟹ don't execute the next Instruction

* INCFSZ f, d → عكس اللي فوق

* BTFSS f, b :

it cheeks bit #b in address F and if = 1

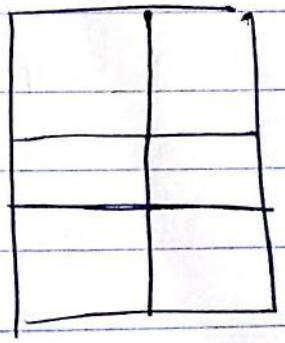
⟹ skip next instruction.

DECFSZ 22, 1

inst 1 → will be executed if

inst 2 initially [22] ≠ 1

→ will not be executed if [22] = 1



* in the four conditional instruction, they take.

2 T_{inst} in case skip happened.

Scanned by CamScanner

Incfsz 22, 1

Inst 1 → will be executed if [22] ≠ FF (1-)

Inst 2 → will not be executed if [22] = ff.

* btfss 03 (status Reg), 2 → Z bit. "checks the result of last Inst if it zero or not" يعني بشيك على اي بت Z bit

These Four Instructions will be used in Conditional Statments and Loops.

Call k → pc → push on stack. بنقل شغلتين pc ← k. وال

Go to k → pc ← k

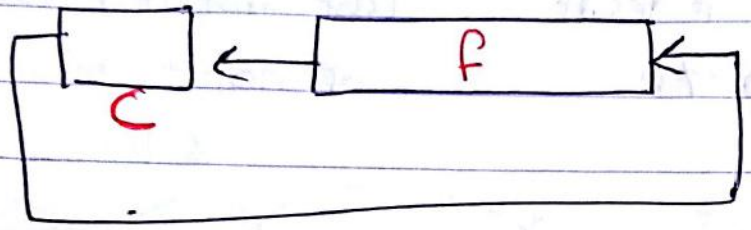
(No return)

Scanned by CamScanner

CLRF f → ما بتاخد d

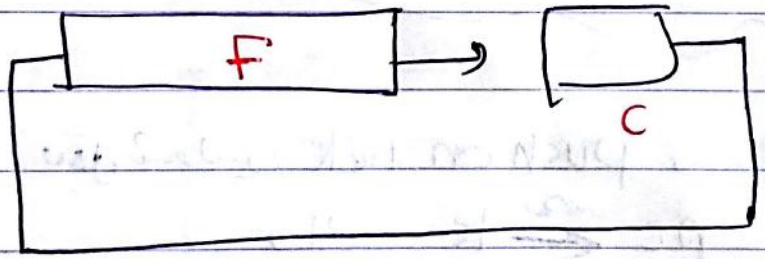
NOP → بنستخدمها لل Delay.

RLf F,d → مثل خلية ضرب



*2 if C=0
*2+1 if C=1

RRF F,d



نفس الاشي "Integer division".

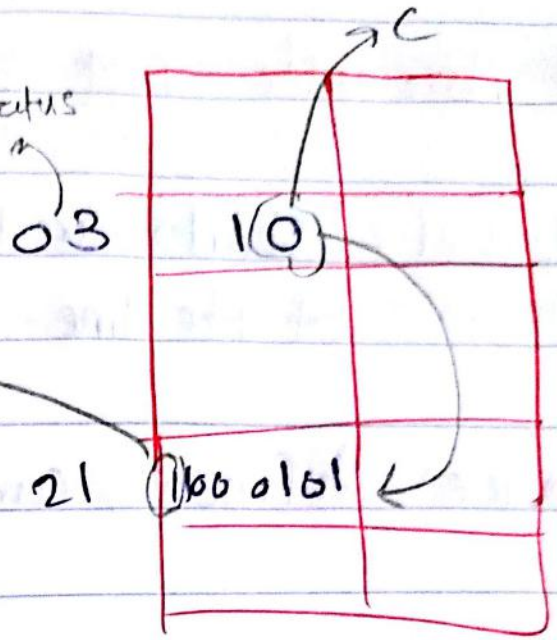
* multiply [22] by 8. $8 = 2^{3}$ → shift 3 times

```
BCF STATUS, C
RLF 22,1
BCF STATUS,C
RLF 22,1
BCF STATUS,C
RLF 22,1
```

عشان أضمن C=0 في الثلاث مرات عشان ما يجع 1.

Scanned by CamScanner

* RLF 21, 1 ← d bit لتحديد مكان التخزين

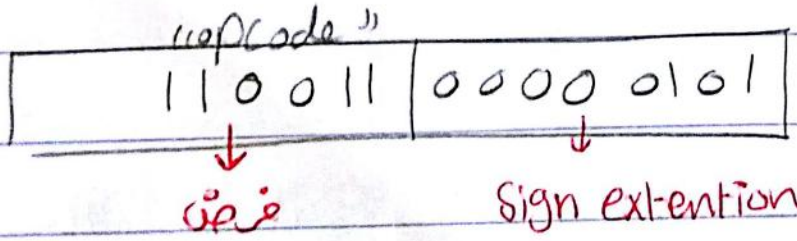


C = 1

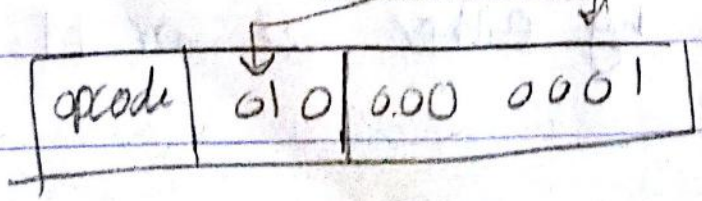
[21] = 1000 1010

* ADDLW 5

encoding.



* bsf 1, 2 (f, b)



Scanned by CamScanner

* Assembler Details :

* Label : Starts with letter at the very begining of the line. "Case sensitive"

once defined, can be used as operand.

Call [Label] → 11 bits
↳ will be replaced by the address of the first Inst comes after it.

zero bits in prog mem

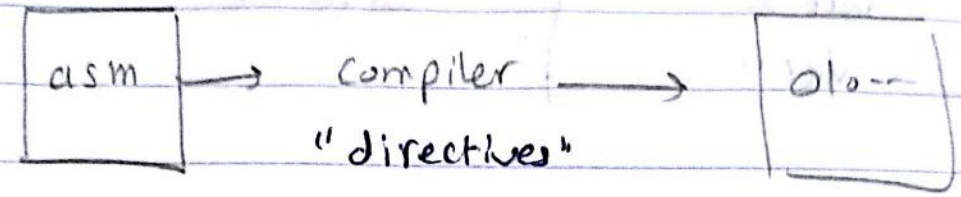
'Label';

Hexadecimal → H'32' , 0X32 , 32.
If the Valles starts with char "A" - "F"
You should precede it by either 0X or H.

Scanned by CamScanner

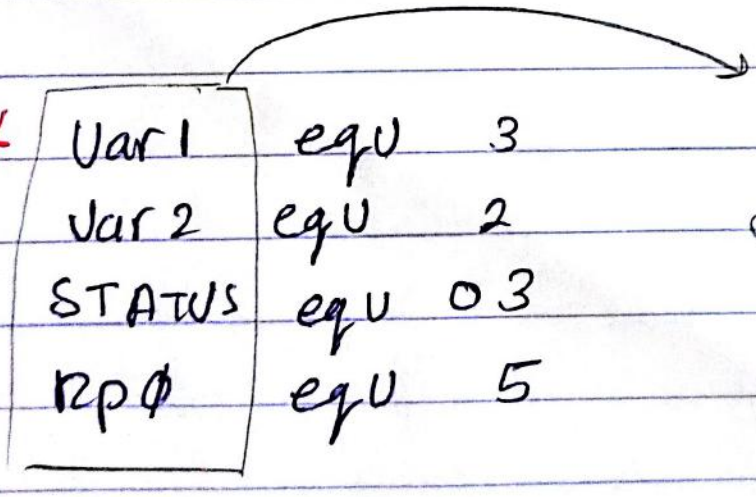
"address" Prog mem موجودين بال

* directives : they give information to the compiler at execution time, they are not existing.



* org 05 } It tells the compiler that this
05 ← movlw 0XA Inst will be stored at
06 Inst 1 address 05.
07 Inst 2

org 04. لازم اعمل Interrupt دلوقتي بالكود تكتبي *
ودلوقتي لازم أعمل org 00 بكل الأحوال.



* Labels جاية
وقبل استخدم بال
الرقم.

bsf STATUS, RP0

bsf Var2, Var2 ✓

Scanned by CamScanner

* C block OX22
Var1
Var2
Var3

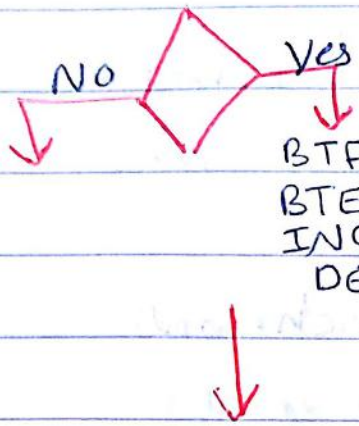
3 equate Statment.

endC → بتوقف تحويل البرنامج من Assembly لـ machine.

Scanned by CamScanner

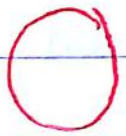
* Chapter 5 : Flowcharts :

Processing



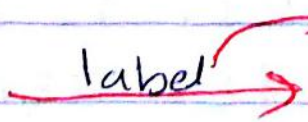
Condition

State diagrams.

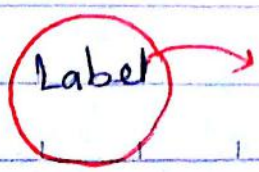


States.

labels :



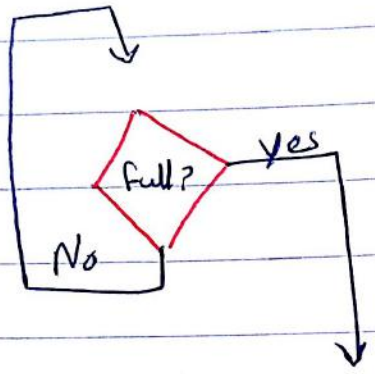
Condition to move from current state to the next state.



Name of state.

Scanned by CamScanner

* لازم يكون في 2 Labels عالشكل العلاقة بينهم AND



synchronous.
بضل أمشّيك full أو لا.
حتى إمشي بقا لحتى أضا بالـ
state diagram
support asynchronous.
في إمتى بقطعلي التمشيك.

* Conditional branch : go to if condition
in pic → skip if condition.

* if bit #2 in address 23 = 0
clrw
clrf 22 → دائماً تنفذ "else هيك"

Sol1: btfss 23,2
لو صفر clrw
clrf 22
إذا كان skip
clrw
في حالة
bit #2 = 0.

Sol 2:
movlw 04 → 0000 0100
ANDwf 23,0 → if bit #2 in
btfs_ STATUS,2 23 = 0
Z=1 else Z=0

Scanned by CamScanner

* If bit # 2 in address 23 = 0.

```
clrf  21
clrf  22
clrf  23
```
} block 1

```
else
clrf  24
clrf  25
clrf  26
```
} block 2.

```
&1 1:
btfss  23, 2
goto  block1
goto  block2
block1  clrf 21
        clrf 22
        clrf 23
goto NEXT ; "skip block2"
block2  clrf 24
        clrf 25
        clrf 26
NEXT
```

```
&1 2:
 btfss  23, 2
 goto  block1
 block2  clrf 24
         clrf 25
         clrf 26
 goto NEXT
 block1  clrf 21
         clrf 22
         clrf 23
 NEXT
```

* دائماً بنكتب if else 3 goto stat بشكل على

* ازا حطينا call بدون return بتتعبى ال stack.

* for counter = 50 ; counter ≥ 0 ; counter --

```
counter  EQU  20
```
→ ال address اللي فيه قيمة ال counter

```
movlw  D'50'
```
→ تحقق القيمة بالسؤال Decimal.

```
Loop  DECFSZ  counter , 1
      goto  Loop
```
→ بعمل Skip عن ال goto لما تصير قيمة ال counter صفر.

* لو حطينا DECFSZ counter, 0 هاي Loop infinit لانو مكونة نقلت كل التخزين لل W Reg وال W Reg قيمة ال counter فيه constant "50".

!! راعي تطابق ال labels بالبرنامج من حيث الكتابة سواء capital أو small.

* ملاحظة مهمة:

هاد لا يعتبر برنامج كامل لازم نعمل select bank و ORG 00 وال END مهم جداً كتابتهم لما يطلب برنامج كامل.

* نفس المثال السابق بس بدي أنفذ مجموعة من الـ Inst "block" ←

① Loop DECFSZ Counter, 1
clrf 24
clrf 25
clrf 26
goto Loop.

} wrong solution.
لأنو ما كرمل قيمة الـ Counter
صفر صيفل skip لأول inst
ويكمل عالي بعدها وينقص واحد
وصيكل FF ← يعني infinite loop

② Loop clrf 24
clrf 25
clrf 26.
DECFSZ Counter, 1
goto Loop.

} ✓ executed at least once.
هاي "Do while"

لو بدنا الـ Counter يعشي من 50 لـ 5 مش لصفر يعني 45 قيمة بدي ياه
١) بلش الـ Counter من قيمة 45 للصفر
٢) خلي بعد الـ DECFSZ جملة if تشيك على قيمة الـ Count إذا صارت 5 أو لا.
٣) خلي DECF وراها جملة تشيك عالقيمة

/

Scanned by CamScanner

* example 1 slide 9 :

BY defult all interrupts disable.
عشان هيك ما حط ORG 04 وحط ORG06 عشان
يبينلنا انو بلش كتابة من address رقم 6
وما في مشكلة بهالحالة نكتب عال address رقم 4

1 # of Instructions?

9 instructions.

2 Size of the prog?

9 x 14 bit.

3 movf 0x11, 0 كم bit ? "as operands"

0x11 → 7 bit

0 → single bit

goto DONE ✗ قيمتها 13 يعني D لو عدينا
بيطلع رقم ال DONE 13

goto D ✓

بس ازا ضفت أو حذفت inst
رح تصير غلط

STATUS → 7bit
بأشر على
8 bit
"as operand"

Scanned by CamScanner

* example 2 :

#include "P16F84A.INC" → من خلالها بقدر استخدم ال Registers بأسمائهم.

خانة عشرات ← CountH

خانة آحاد ← CountL

CountH	CountL	
2	7	→ مثلاً.
	6	
	5	قبل ما نطرح بنعمل check للقيمة
	4	اذا صارت صفر بنتقل عال CountH
	3	
	2	
	1	
1	0	
	FF	
	⋮	
0	0	
	FF	
	⋮	
00	00	

صارت القيمة 0.

Scanned by CamScanner

* Subroutines → function بالأغلب يبدأ بـ Label
وينتهي بـ return أو retlw k

retlw k → moul w k
return

* To call the subroutine ; we use the call Instruction, the k operand should equal either the Label of the subroutine or the address of the first instruction in the subroutine.

Call k ← بتعمل شغلتين → Stack ← push pc
pc ← k
} 2 Tinst

return ↷ pc ← pop Stack } 2Tinst

retlw k ↷ w ← k
pc ← pop Stack
} 2 Tinst

اي امر بغير في الـ pc بياخد 2Tinst وغالبي الـ inst بتاعة تمشي both of them

Scanned by CamScanner

* Passing parameters to Subroutines and returning values from the Subroutine is done through Data memory.

int V = Sum(5,3) ⟶

```
        movlw   3
        movwf   20
        movLw   5
        movwf   21
        Call    sum.
        movLw   9
        mov:
        call: sum
Sum     MovF    20,0
        ADDWf   21,0
        Return.
```

لو بدي أجمع 9 مع 11 مش 3 و5 بكتب بنفس الـ first قبل أول call

* Delay

- Hw : timer 0.
- Sw : NOP → No operation. code.

```
*   Loop1
        NOP
    goto Loop1
```

اعمل delay 256 مرة

```
*   Loop1
        Loop2
            NOP
            goto Loop2
        goto Loop1
```

256 * 256

أطول Delay

```
*   Loop1
        call sub
        goto Loop1
    sub
        Loop2
            goto Loop2
        return.
```

طريقة أخرى

Scanned by CamScanner

/ /

$* \; delay = T_{inst} * \# T_{inst}$

$= \frac{4}{F_{osc}} *$ trace the code to get it

Delay out of

* Single Loop : Initialization delay + all iterations except the last one + Last iteration.

* example 1 slide 20 :

$F_{osc} = 800 kHz$.

$delay = \frac{4}{800 \, kHz} * \# T_{inst}$

$= 5 \mu s * \# T_{inst}$

movlw, movwf / nop, nop, DECBZ, goto

$\# T_{inst} = (1 + 1) + 199 (1 + 1 + 1 + 2)$

عدد لفات حسب ال counter اللي فيه 200 وبنقص آخر iteration

nop, nop, DECBZ

$+ \underbrace{(1 + 1 + 2)}_{\text{last iteration}} = 2 + 199 * 5 + 4 = 1001$

Counter = 1

بعمل DEC بيصير صفر بنعمل skip عن ال goto

$delay = 5 \mu s * 1001 = 5.005 \, ms$

* If this code is a subroutine, we add 2 Tinst for call and 2 Tinst for return

total delay = 2 * 5 µs + 2 * 5 µs + 5.005 ms

= 5.025 ms

* write a subroutine that generates 10 ms delay given that Fosc = 1 MHz.

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

delay = Tinst * # of Tinst

10 ms = 4 µs * # of Tinst

$$\#\text{ of } T_{inst} = \frac{10\,ms}{4\,\mu s} = 2500$$

Scanned by CamScanner

تكملة الكود.

* Sub

```
movlw  X
movwf  21
Loop
    nop
    nop
    nop
    nop
DECFSZ 21او1
goto  loop
return
```

$$250 = \overset{\text{Call}}{2} + \overset{\text{with}}{2} + (X-1) * 7 + 8$$

$X = 356.4$ → غلط بطلع أكبر من 256 عشان صح بنعدل عالكود.

لو ضعنا كان nop 2 بتفل نفس الحكاية، لو صار عدد الـ nop 7 بطلع $X = 249.5$ فبنضط جوا الـ X الـ 249 فقط لحطينا فيه 250 بطلع بردو أكبر من 250.

Scanned by CamScanner

$2 + 2 + 248 * 10 + 11 = 15 + 2480$

$= 2495$

$2500 - 2495 = 5 \rightarrow$ nop

Loop بنضيفهم فوق الـ

↓

Sub

nop

nop

nop

nop

nop

movlw 249

⋮

* exercise:

modify the subroutine to give 4 ms delay

5.025 ms الـ ع

* nested loops :

initialization +

first external iteration

(all internal iterations except last internal iteration)

+ all external iteration except last external iteration

(all internal iterations except last internal iteration)

+ last external iteration

(all internal iterations except last internal iteration).

* example 2 slide 21 :

$F_{osc} = 4\,MHz \rightarrow T_{inst} = 1\,\mu s$ ← $\frac{4}{4\,MHz}$

$10\,ms = T_{inst} * \#\,of\ cycles$

$\#\,of\ cycles = \frac{10\,ms}{1\,\mu s} = 10{,}000\ cycle$

nested loop إذا عندي مرتين goto و Decfsz عندي

countH = 13 → 12 → ... → 1 → 0

countL = 250 → 0 (بعد أول لفة)

call initi

$2 + 5 + 249 * (1+2) + 5 + (11) * (255*3+5) + (255*3+6)$

(11) = countH − 1 = 12 − 1

5 = last internal iteration

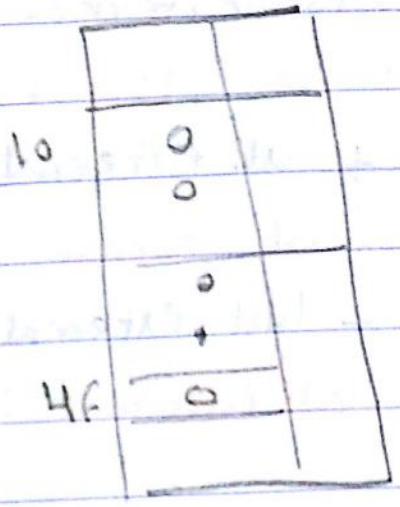
(255) = 256 − 1

Scanned by CamScanner

*ex: prog mem لانو بتقدر تتنقل ع pc بتقدر تتنقل ع

write a code to clear data memory locations

0X10 ⟶ 0X4f.




```
clrf f
```

```
clrf 10
clrf 11
clrf 12
  .
  .
  .
clrf 0X4f
```

* to deal with list of addresses, it is better to use indirect addressing:

1. store the address in FSR.

2. in place of the f operand, we write either 0X00 or INDF.

* When the ALU sees the 0x00 or INDF operand it understands that the real address is in FSR.

Scanned by CamScanner

* DATA MEM

INDF	INDF 80

* write a code to complement the address 0x25 using indirect addressing.

```
MOVLW   25
MOVWF   FSR
COMF    INDF ,1
        أو
        00
```

↓ Comf 25,1 زي

* Solution of ex:

```
        MOVLW   0x10.
        MOVWF   FSR
Loop    CLRF    INDF
        INCF    FSR ,1
        MOVF    FSR ,0
        SUBLW   0x50
        BTFSS   STATUS ,Z
        goto    Loop
```

غلط نحكي clrf FSR لانو حيمسح الـ Reg

clrf INDF ⇒ زي clrf 10.

INC INDF ,1 ⇓ غلط زي INCF 10,1 ↓ رح نعتبرها صحيحة.

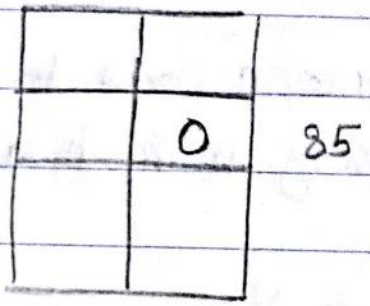
4F → 50

Compare with 50

عشان لما يوصل بدي أطلع

* INDF is not a physical register

* movlw 0x85
  movwf FSR
  clrf INDF.



*ex slide 25:

```
movf   0x10, w
ADDwf  0x11, 1
Addwf  0x12, 1
  :
  :
Addwf  0x1F, 0
movwf  0x20
```

using direct addressing

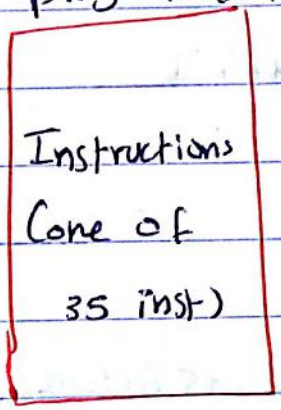
دائماً بال indirect addressing بنحط ال address بال FSR وبتعامل مع ال INDF.

Scanned by CamScanner

* Look up table

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

Cout << array [2] ; 4 يطبع

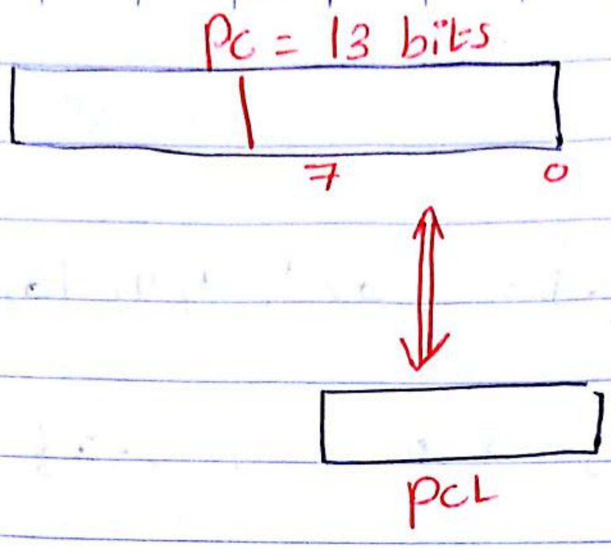


Look up table : is a method to define array in a prog subroutine.

```
array   ADDWF   PCL, 1
        retlw   0
        retlw   1
        retlw   4
        retlw   9
        retlw   16
        retlw   25
```

array of 6 elements

* عدد الـ retlw بنعرف "



اي واحد بتعدل
بتعدل الثاني
syconized *

```
movlw 2          ──→  w = array[2]
call  array
movlw 4
call  array
array  ADDWF PCL,1      ──→  goto pc + w
PCL →  retlw 0
       retlw 1
       retlw 4
       ⋮
       retlw 10
       ⋮
       movlw 5
       return
```

بتمشي عال inst كعدد بيني pc+4

بتمشي 4 inst

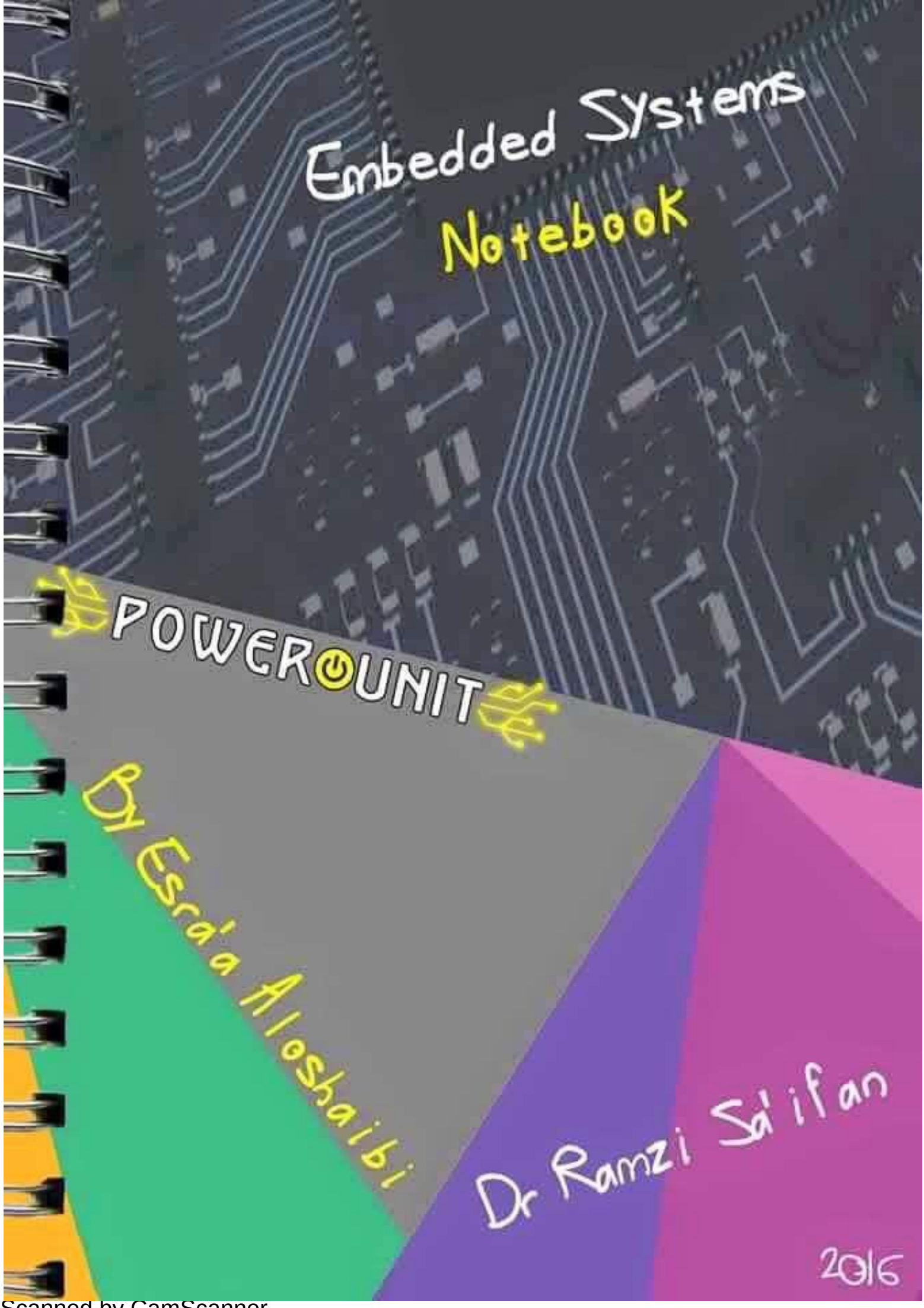
هون صفرت الـ pcl

بصير 2.

لو اعطانا 2 بصير هون

لح يرجع فوق و يحط ال array[4]

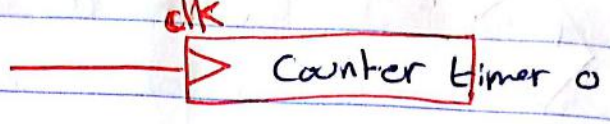
اي اشي بعدل عال pc بياخذ 2T inst لانو بيحتاج يعمل flush و fetch من جديد



Scanned by CamScanner

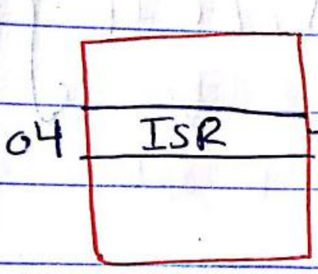
* Chapter 6

timers →



Counter 8 bits → بعد → 0 — 255 → 0

في هاي المرحلة في Flag بتصير قيمتو 1 يعني بقاطع تنفيذ البرنامج
((interrupt)). ⇒ 04



interrupt is a subroutine stored in address 04 and called by HW

Call ISR ✗

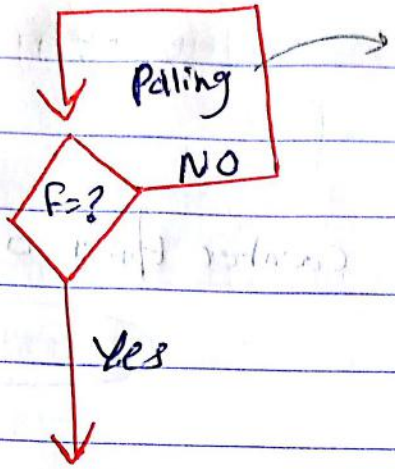
* The ISR has higher priority than regular code

interrupt VS Polling.

⇓

Keep checking

بدي أظل أشيك اذا ضغطه عد أو لا من خلال التشييك في
ال Play



Check BTFSS EECON1, EEIF } Polling Code.
goto check.

.interrupt يعني لا يوجد

* maskable: Can be disable.

* non-maskable: can't be disable.

In pic, all interrupts are maskable.

Scanned by CamScanner

* for an interrupt to happen :

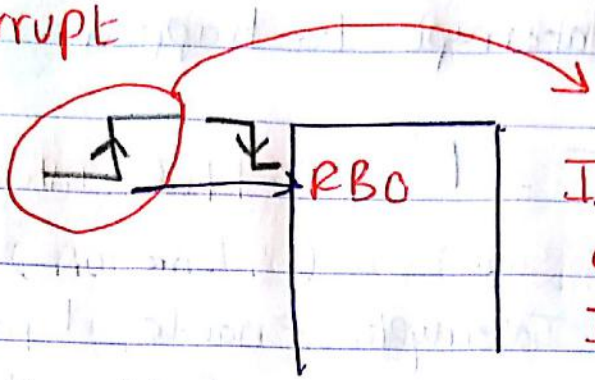
1. GIE = 1 "Global Interrupt Enable"
   - set by sw.
2. Local Interrupt enable = 1, there are 4 types of interrupt, each of them has enable bit.

3. flag = 1, for each of the 4 interrupt types there is a flag bit. it is set by Hw.

flag can be set even if the interrupts are disabled.

Scanned by CamScanner

1) * external interrupt



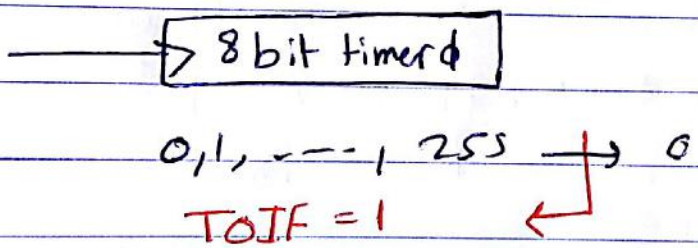
أنا بحدد بشتغل عالـ positive edge أو الـ negative

INTF = 1
GIE = 1
INTE = 1
} interrupt

لو حصل صارين بالـ interrupt ما كانوا سبب الـ flag.

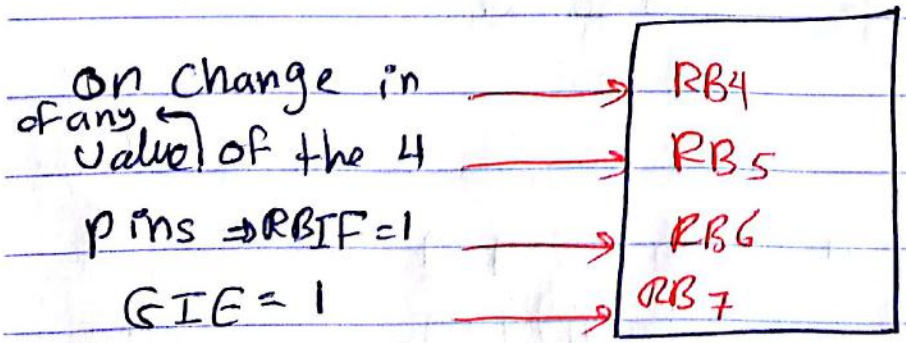
إذا حدث الإشارة positive بيجيني flag = 1 يعني positive edge.

2) * Timer 0 overflow interrupt.



T0IF = 1
GIE = 1
T0IE = 1

3) * port B change interrupt.



مجرد ما تغيرت قيمة واحد فيهم بقوم على الأقل بيعمل Interrupt.

Scanned by CamScanner

4) * EEPROM write Complete interrupt.

EEIF=1 , GEIE=1 , GIE=1

لما كنا نخلص كتابة كان في flag تتغير قيمتها.

* all flags are cleared on Reset except RBIF.

* If the PIC is in sleep mode and the local interrupt is enabled when the flag=1
→ the PIC wakes up even if GIE=0.

* to deal with interrupts : بنتعامل مع الـ bit
1- GIE bit. (INTCON(7)) → bank 0 and bank 1

2- 4 local enable bits. } 1. INTCON except EEIF in EECON1 bank1

3- 4 Flag bits. }

4- 1 bit for ↓ ↑ of external interrupt. (option Reg (61)) bank 1

Scanned by CamScanner

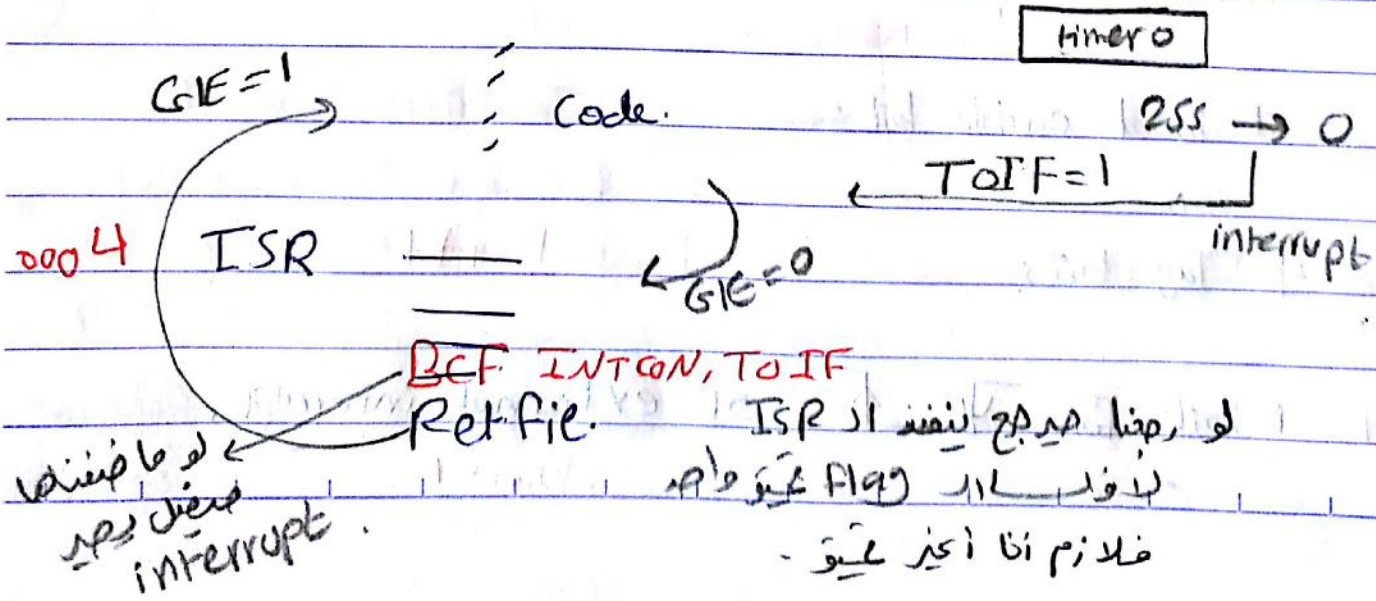
الـ Flags الـ HW هو اللي بيتحكم فيهم اما باقي انا بغير فيهم بالـ code

* clear GIE (Automatic) : to prevent any other interrupt from interrupting the current interrupt.

لازم الـ interrupt ينتهي بـ RETFIE هو اللي بيرجع قيمة الـ GIE = 1 بترجعلي في الـ code.

RETFIE → 1) BSF INTCON, GIE
→ pc ← pop stack

* BSF INTCON, GIE
BSF INTCON, T0IE



Scanned by CamScanner

* flag set by HW cleared by SW

slide 12 :

4) ___

لازم نقل عالفلاغ قبل 5

Clear the flag of the INT that caused the INT.

5) ___

```
Temp  EQU  22
ISR   movf  Temp
      movf  Temp,0
      Retfie
```

حتى أحافظ ع قيم الـ Registers عند حدوث INT الـ

```
        00x15
ISR     mouf    0X15,0
        movwf   Temp.
        :
        :
        -

mouF    Temp,0
movwf   0x15
retfie
```

تأثر على Z

```
movF    STATUS,0
movwf   Temp
        ;
        ;

moUF    Temp,0
mouwf   STATUS
```

لو كانت قيمة ال STATUS صفر صيحير ال Z Flag بمعنى واحد. وأنا عايزي أأثر عليه بدي به أخزن قيمة ال STATUS عشان ممكن بستخدم ال SWAPF.

تأثر على Z

Scanned by CamScanner

* 2 Types of INT

INTE → clrw

ToIE → clrf 25

```
ISR             btfsc   INTCON, INTF
                Call    ext_interrupt
                btfsc   INTCON, T0IF
                Call    timer0 interrupt
                retfie

ext_interrupt   clrw
                bcf INTCON , INTF
                return

timer0interrupt:    clrf    25
                    bcf     INTCON, T0IF.
                return.
```

هون أعطيتك INTE أولوية أعلى

* to use counter as timer :

1. we should know the Frequency

1MHZ → 256 cycles F=1

256 µs ⇒ F=1

2. we should be able to initialize it

156 to counter بالبدء ال 100 ms بعد F=1 حتى يبدأ

TMR0 01

↓ SFR ↓ address

MOVLW D'156' } 156 تخزين قيمة

MOVWF TMR0 } timer بال

Freq = F → prescale by 8 → Freq. = F/8

Count every 8 cycles.

prescale: is Frog division to give larger delay.

Scanned by CamScanner

PSA = 0 ⇒ prescaled ⇒ by the value $2^{PS_2 PS_1 PS_0 + 1}$

= 1 ⇒ not-prescaled

if PSA = 0 and $PS_2 PS_1 PS_0 = 100$

⇒ prescale $= 2^{4+1} = 2^5 = 32$

SFRs

جدول ال 2 reg

نتعامل معهم لما بدي نتعامل مع ال timer

PS = 0 ⇒ timer0 is prescaled with prescale

$= 2^{PS_2 PS_1 PS_0 + 1}$

PS = 1 ⇒ WDT is prescaled with prescale

$= 2^{PS_2 PS_1 PS_0}$

Scanned by CamScanner

* timer ϕ delay For TOIF to be equal 1

$$delay = \frac{4}{Fosc} * Prescale * (256 - N)$$

TMR ϕ

كل أديش بعدّ — Prescale → OPTION_REG.

كم مرة صعدّ — (256 − N)

* example :

Write a piece of code to give 5 ms delay using timer ϕ given that Fosc = 4MHz

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

$$5 * 10^{-3} = \frac{4}{4 * 10^{6}} * pre * X$$

$$\frac{2 * 10^{4}}{4} = pre * X$$

$$5 * 10^{3} = pre * X.$$

بفرض ال pre لانو الو قيم محددة

assume pre = 4

X = 1250 X (wrong)

255 أكبر قيمة ممكن تطلعا

assume pre = 32.

X = 156.25 ≈ 156

N = 256 − 156

= 100

Scanned by CamScanner

$TMR\phi = 100$.

option_Reg = xx0x 0100 = 04

internal clock.

يعني علامة prescale بجعلها 0.

فهاي بتكون don't care.

```
MOVLW   D'100'
MOVWF   TMRΦ

bsF     STATUS, RPΦ    → bank selection.

MOVLW   04
MOVWF   OPTION_REG.
```

* if $F_{osc} = 4MHZ$

max delay of $TMR\phi = \frac{4}{4*10^6} * 256 * 256$.   [N=0]

= 65.536 ms.

* Write a code to clear W-Reg after 1 s. Using timer0 given that F_{osc} = 4 MHZ.

delay = counter * $T\phi$ delay.

50 ← فرضت أو أي رقم المهم ال To delay

يطلع أقل أو يساوي 65.536ms.

1 second = 50 * $T\phi$ delay

To delay = 20 ms.

Scanned by CamScanner

$$\text{To delay} = \frac{4}{F_{osc}} * pre * (256 - N)$$

$$20 * 10^{-3} = \frac{4}{4 * 10^{6}} * pre * X$$

$$pre * X = 20{,}000$$

assume pre = 128 $\Rightarrow$ X = 156.25 $\approx$ 156 $\rightarrow$ N = 100

256 − 156

TMR0 = 100.

OPTION_REG = XX0X 0110 = 06

```
# include  PIC16F84A.INC
Counter    EQU  30
           ORG  00
           goto start
           ORG  04
           goto ISR.
start      movlw  D'50'      } Counter
           movwf  counter    }
           bsf    INTCON, GIE   } Enable Interrupt
           bsf    INTCON, TOIE  }
```

Scanned by CamScanner

```
movlw   D'100'
movwf   TMR0
bsf     STATUS,RP0
movlw   0x06
movwf   OPTION-REG
bcf     STATUS, RP0
```
→ to make timer0 overflow after 20 ms.

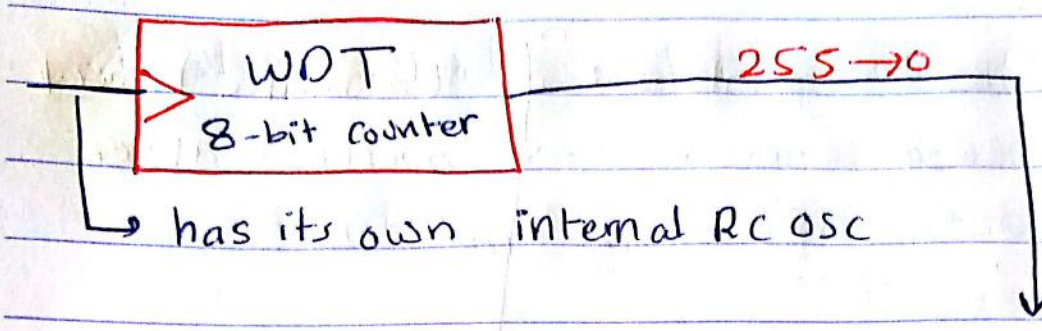
```
Loop  goto  Loop
```
→ infinit loop

طالما مفيش الـ intR timer0 overflow
بوديني ع 04 و الـ 04 بتوديني
عالـ ISR

```
ISR.  BCF     INTCON,T0IF
      movlw   D'100'
      movwf   TMR0
```
ازا ما صفرتها يرجع رح يبدا من الصفر
مش من الـ100 لـ 256.

```
      DECFSZ  counter,1
      retfie
      clrw
      movlw   D'50'
      movwf   counter
```
ازا ما صفيناها رح يبدا الـ counter من صفر

```
      retfie
      end
```
لما يرجع.



Wakeup (if in sleep mode)

Reset (if not in sleep mode) and WDTE = 1 "from conf word"

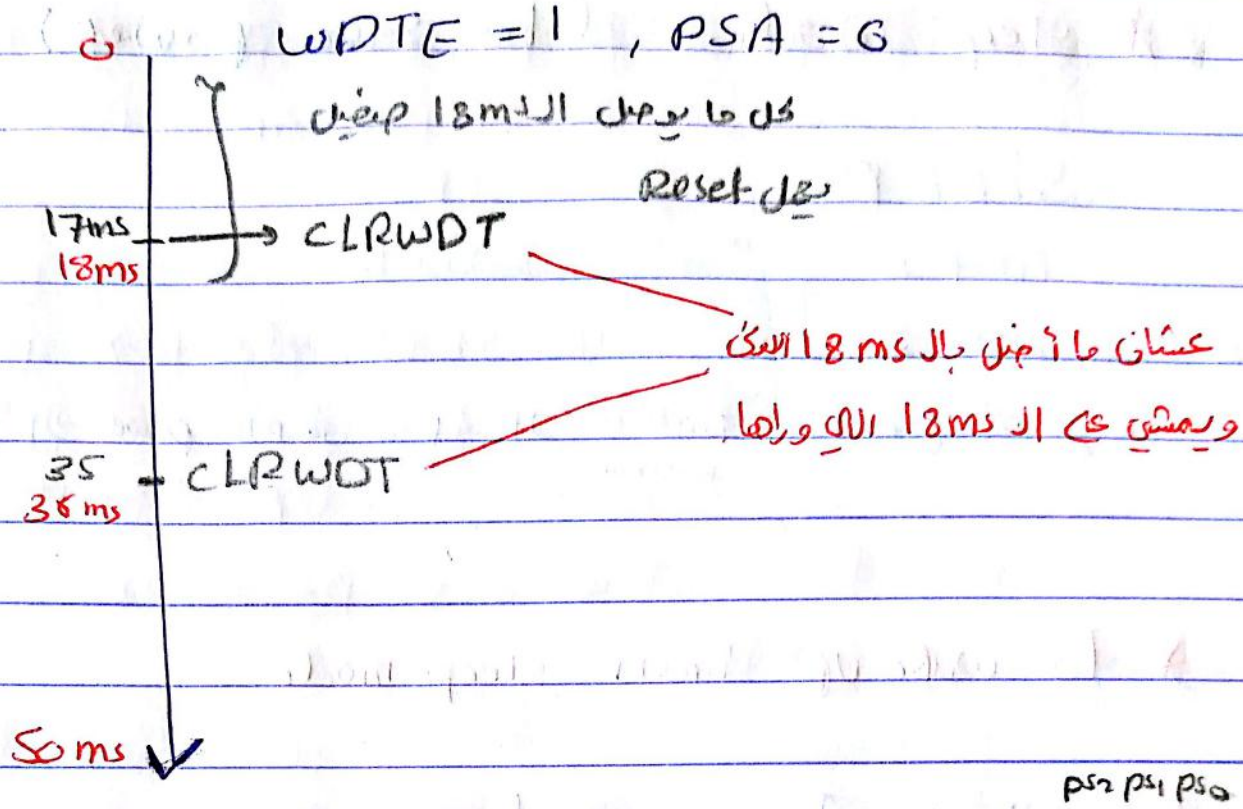
* WDT can't be initialized ; However, I can clear it using CLRWDT.

* WDT after 256 cycles of its internal RC osc, it resets.
256 cycles = 18 ms if not prescaled.
if prescaled (PSA = 1) ⇒ the time = prescale * 18 ms.

"prescale" بدون 18ms بالوضع الطبيعي كل
وانا خلص عد بقل Reset او wakeup حسب ال sleep mode

كيف بيعرف انو صار crashes ؟ مابكون في برنامج بيأخذ 100ms وأكثر
أكثر من هيك بيعرف انو في اشي غلط فبيعمل reset لحالو

Scanned by CamScanner



$$\text{max WDT delay} = 18ms * 128 \quad (2^{PS2\ PS1\ PS0})$$

$$= 2.3 \text{ seconds.}$$

Scanned by CamScanner

* sleep mode. (to save power).

SLEEP

inst 1 } ما متنفذ

inst 2

* ما تنزل ال power للصفر عشان ال Data memory ما يضيع اللي فيه.

* to wake up from sleep mode

1) WDT if enabled ⇒ maximum sleep time
if WDTE=1 ⇒ 2.3 S

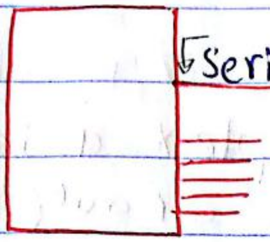
2) $\overline{MCLR} = 0$

3) interrupt : Flag =1 and its enable =1
regardless GIE. → =0 → PC+1 كل شيء ما صار
=1 → عند code ال INT

Timer 0 can't wakeup M.C (oscillator switched off)

Scanned by CamScanner

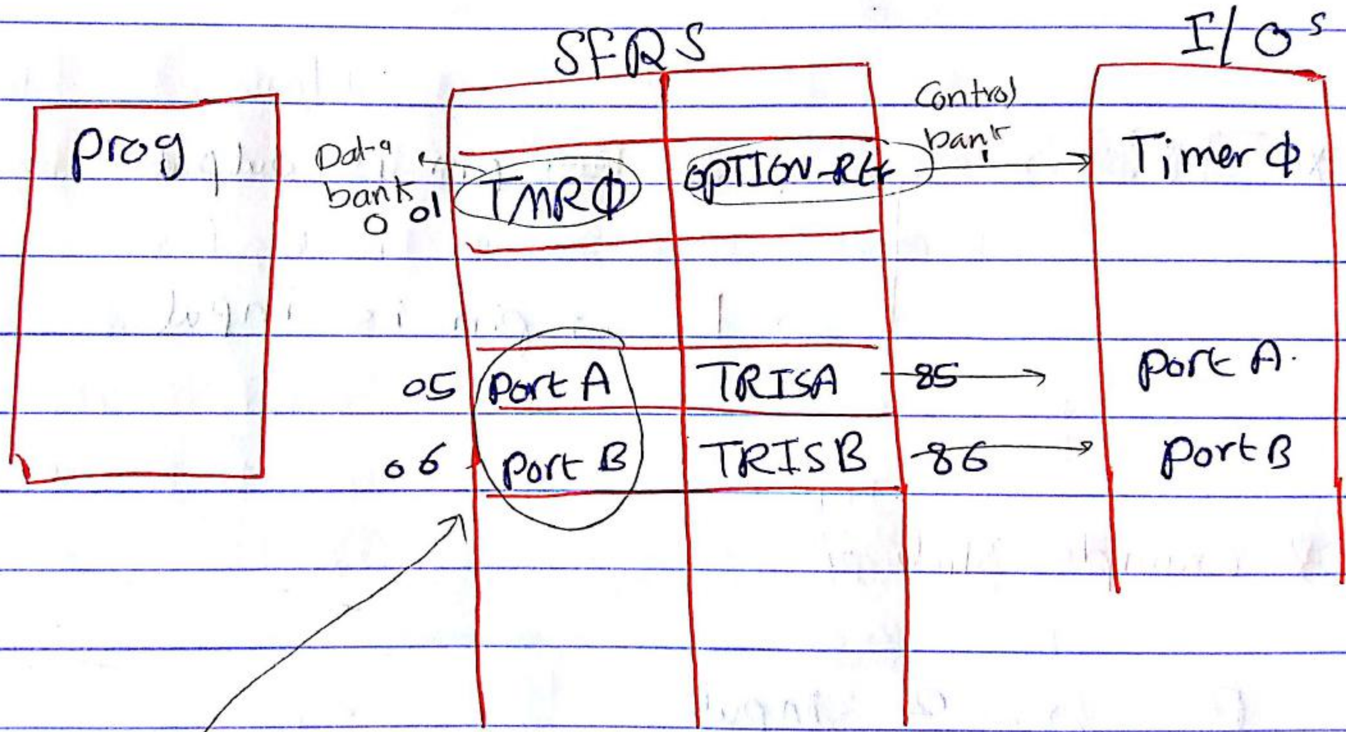
* chapter 3 :



Serial "one bit at time"

} multiple bits at a time. "Parallel" → مش منيح للمسافات البعيدة فمش شرط الـ Data rate الي متكون اعلى "مضروب بـ factor" من الـ serial

يعني مش شرط يكون ٥ مرات أسرع "لانو ٥ سلاك"



* Port data Registers : Input: it holds the enterd data

output: You write in them the data You want to output.

_ / _ /

* Control : they determine which pins are input and which pins are output.

* all port pins are half duplex. as input or output.

* You can configure pins independently.

يعني ازا port A بحوّلو لـ input مو شرط باقي الـ ports يكونو input

* TRIS :
  - → 0 ⇒ pin is output
  - → 1 → pin is input.

* example slide 7:

B
0 – 2 : input
3 – 4 : output
5 – 7 : input

TRISB = (111)(00)(111) → E7

BSF STATUS, RP0
movlw B'1110 0111' → بفضل نكتبها Binary مش Hexa.
movwf TRISB

طولما نتعامل مع الـ ports لازم اعمل switch banks.

أي قيمة بحطها بال portB ما بتغير عال TRIS يعني لو
انا مبرمجة TRISB كله صفار يعني output
وحطيت بـ portB 0xAA ← 1010 1010

لو اعتبرنا هدول LEDs
اشي مضوي واشي ما مضوي
بناءً على 1010 1010
بس ما تأثر انو input او output

slide 9:

* on external interrupt ↑ لازم اعمل هاي الاشياء

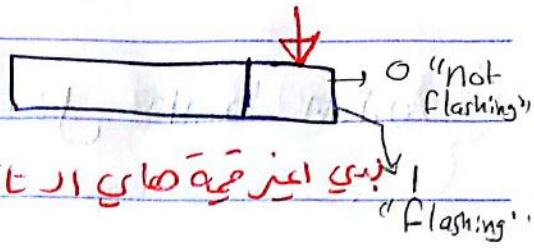
- GIE in INTCON=1
- INTE in INCON=1
- TRISB(0)=1 → لازم تكون RB0 input [illegible] interrupt
- option6=1

LED on RB1 ⇒ TRISB(1) = 0

1 second ⇒ delay

subratine 0.5 second → بناديها مرتين

* on Interrupt



بدي اغير قيمة هاي الـ bit

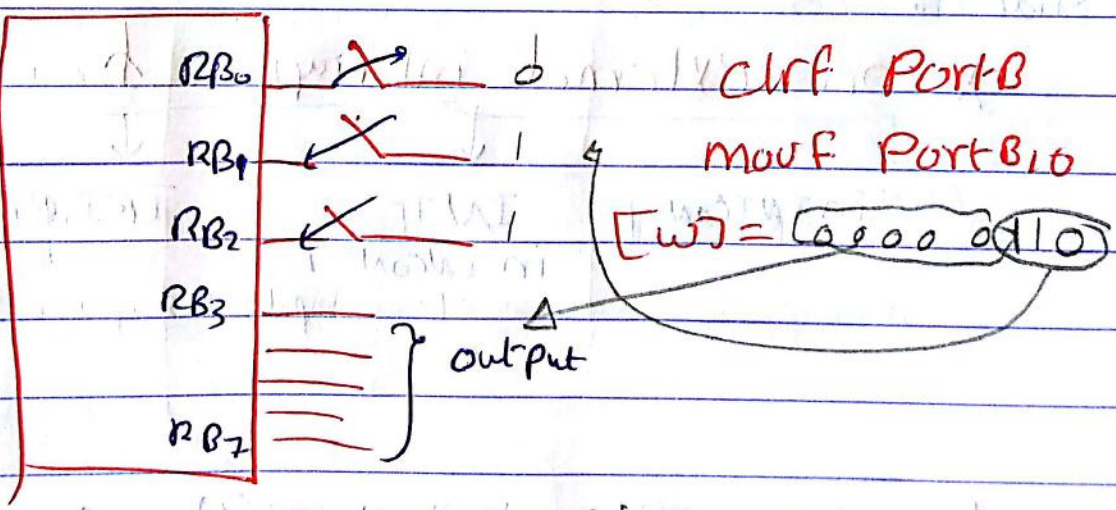
Flash = Flash XOR 01

او بعمل Complement لـ Flash لانو ما بهمني القيمة اللي فيه بغير عادي لو اللي كانت اكثر اول bit تنعكس.

CLRF PortB

⇓

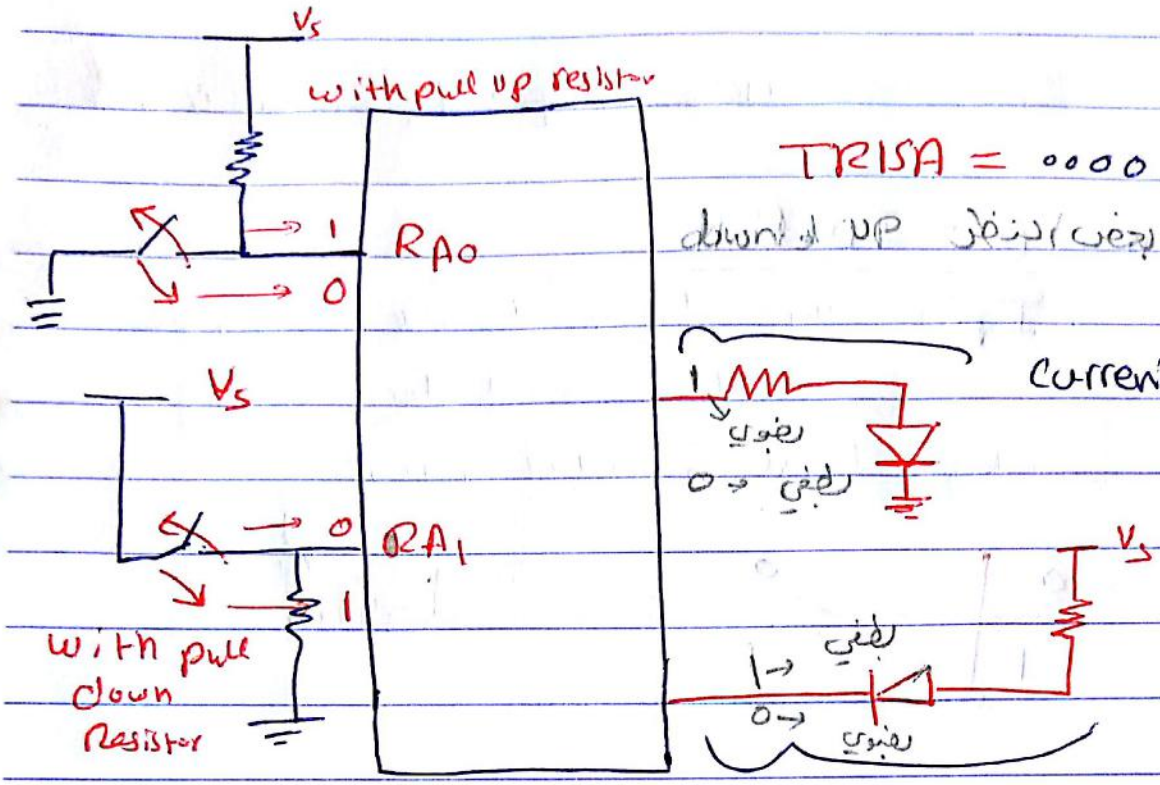
everything will be written according to the instruction except the input bits because they will be overwritten based on the inputs.



الـ clear بتأثر على الـ output bits

الـ HW تاع الـ ports حسب طلبك.

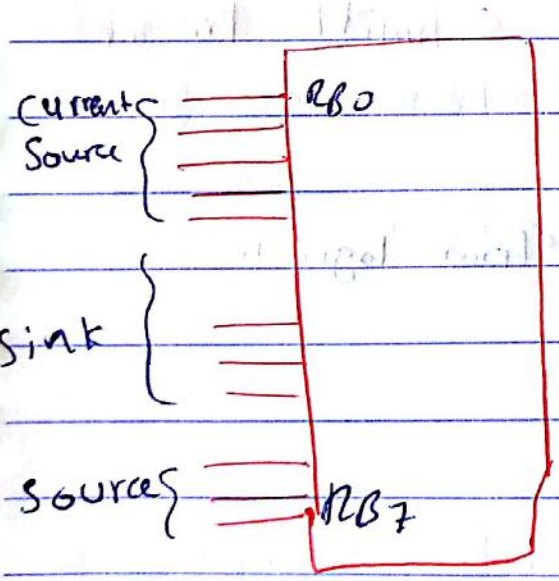
شوف Slide 18 من



TRISA = 0000 0011

input يعني ريزيستر pull up

current sink.



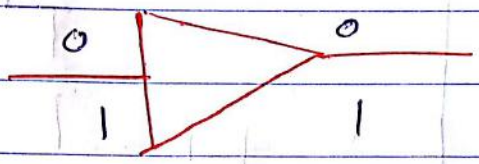
عشان يضوي كلهم 0

```
movlw   B' 1100 0111'
movwf   PortB.
```

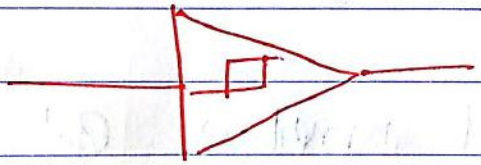
Scanned by CamScanner

*slide 25:

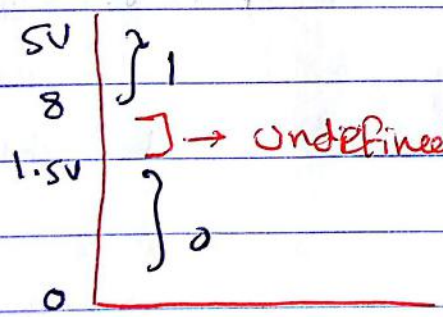
$$R = \frac{V_{OH} - V_D - I_D * R_{OH}}{I_D}$$



buffer.



Schmitt trigger.

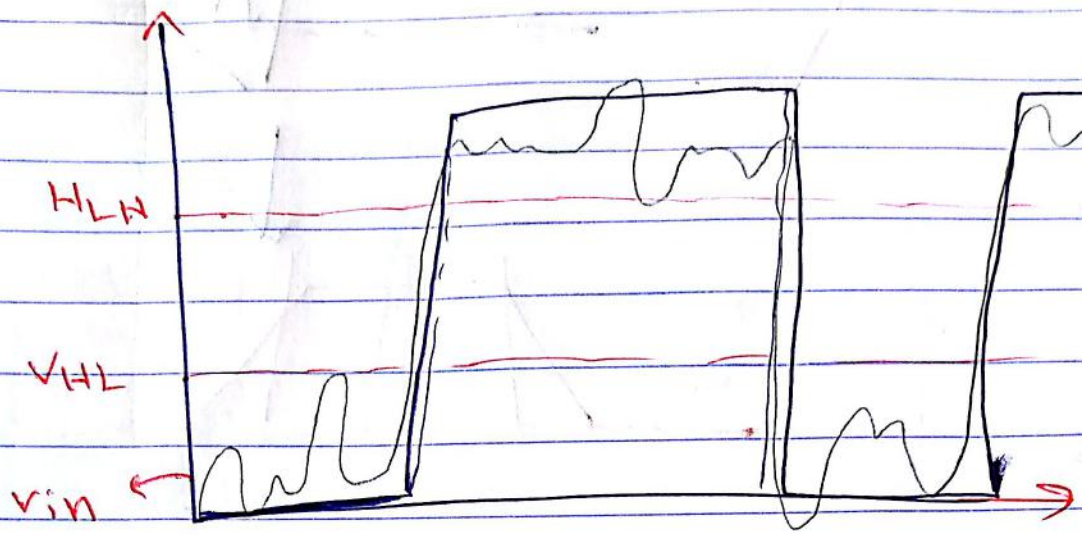


"from logic 1.

Two thresholds : rising thresholds (V_{LH})
falling thresholds (V_{HL})

$V_{LH} > V_{HL}$

Scanned by CamScanner

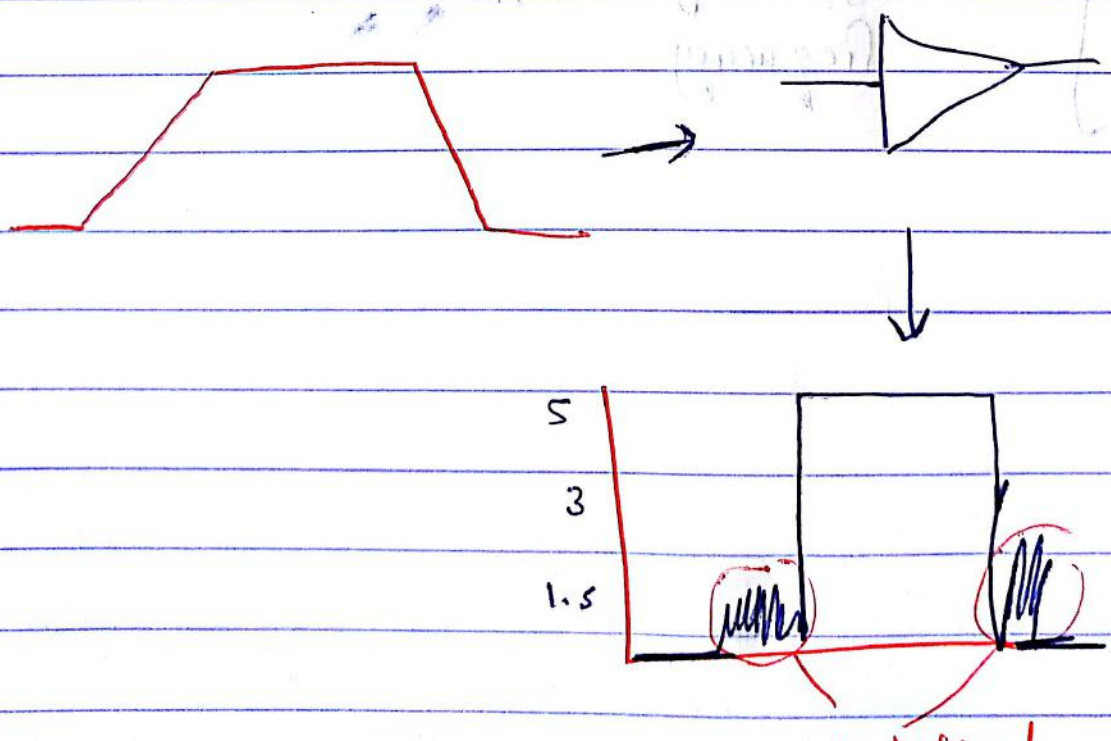


كان الـ input مزبغ صار كايو digital

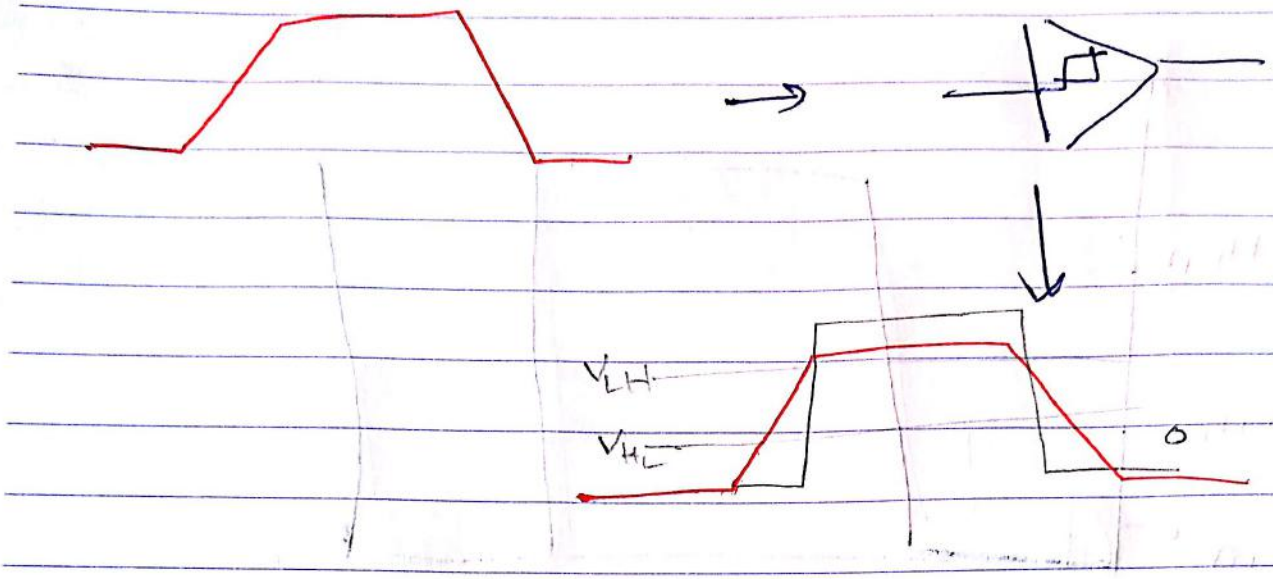
*Benefits of Schmit trigger:

1. Cancle noise and signal fluctuations.

2. Fast switching



Scanned by CamScanner



* The Oscillator:

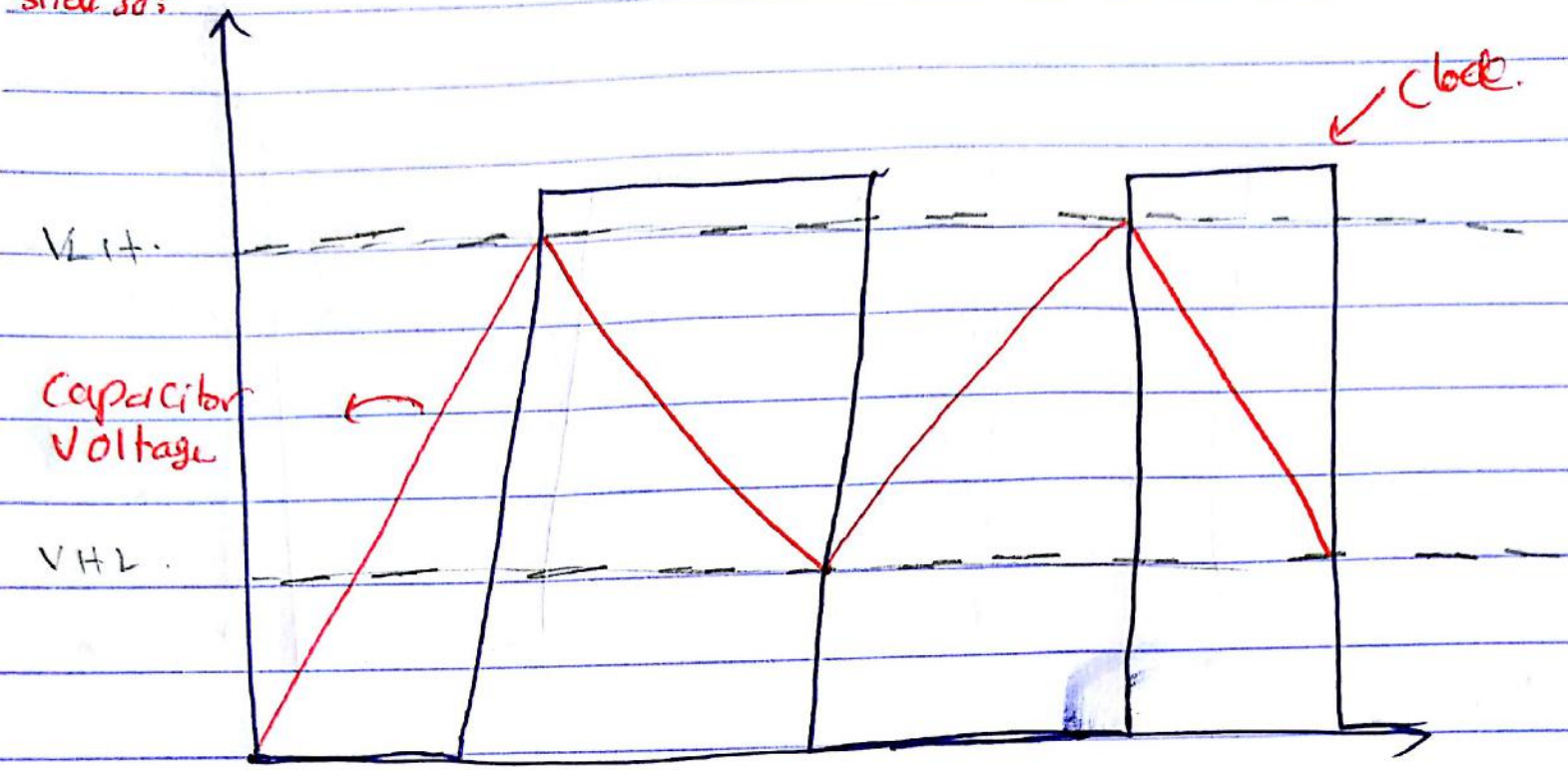
OSC types controlled by : Config word <1:0>

RC
XT
HS
LB

} of type crystal, the difference is in frequency.

Scanned by CamScanner

Slide 38:



crystal → more accurate.

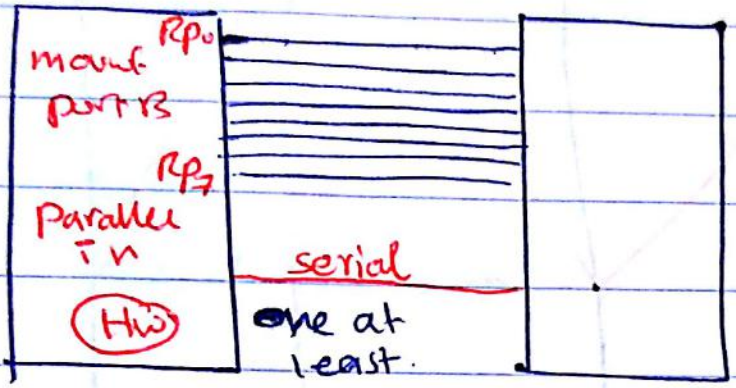
LP → low power

ما تقدر استخدمها في اشي اكبر من 4MHZ.

Scanned by CamScanner

* Chapter 10 :

Starting with Serial



typically the parallel Port data rate is not 8 times faster than serial.

* there is a cross line interference between the parallel port pins :

1. the data rates will be smaller

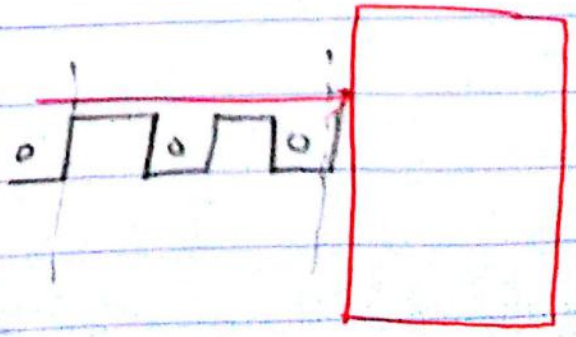
2. the parallel port is not suitable for long distances

* serial communication

Two approaches to solve these challenges :

1. synchronous :

two lines are used, one for data and the other one for clock signals



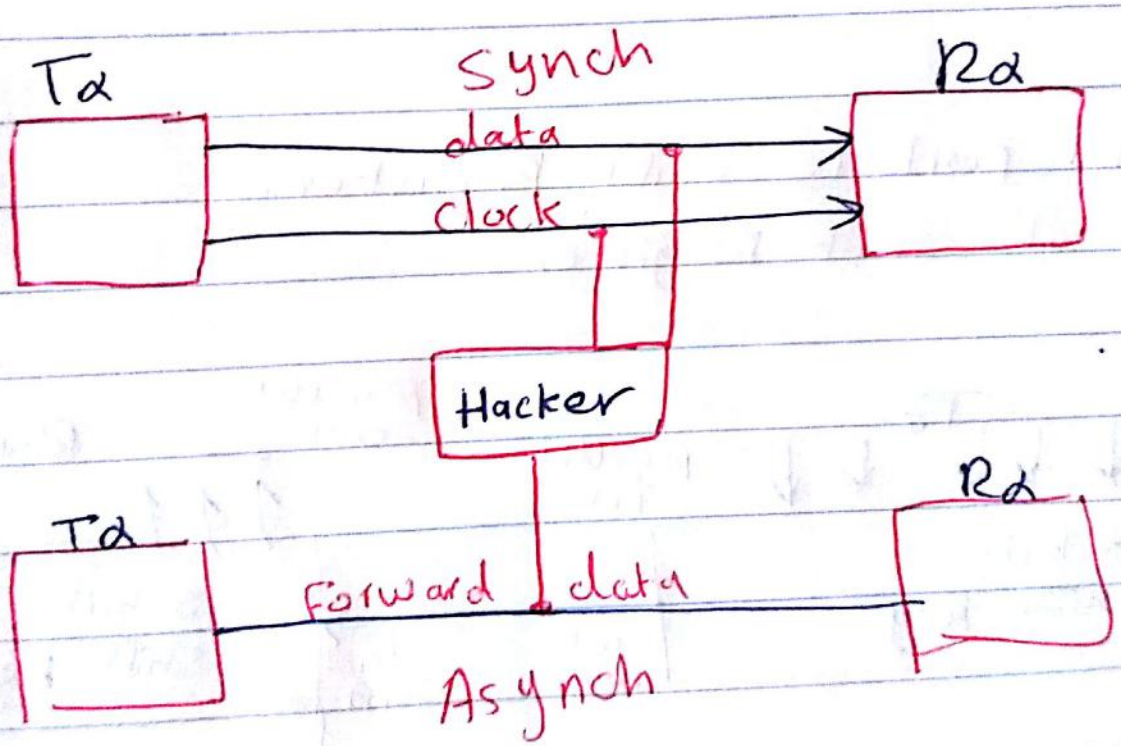
Scanned by CamScanner

[2] Asynchronous:

Single line is used for data, However Both ends should work on the same frequency.

* to know when to start and when to stop we will need 2 bits to tell us.

slide 5 → no clock signal is sent.



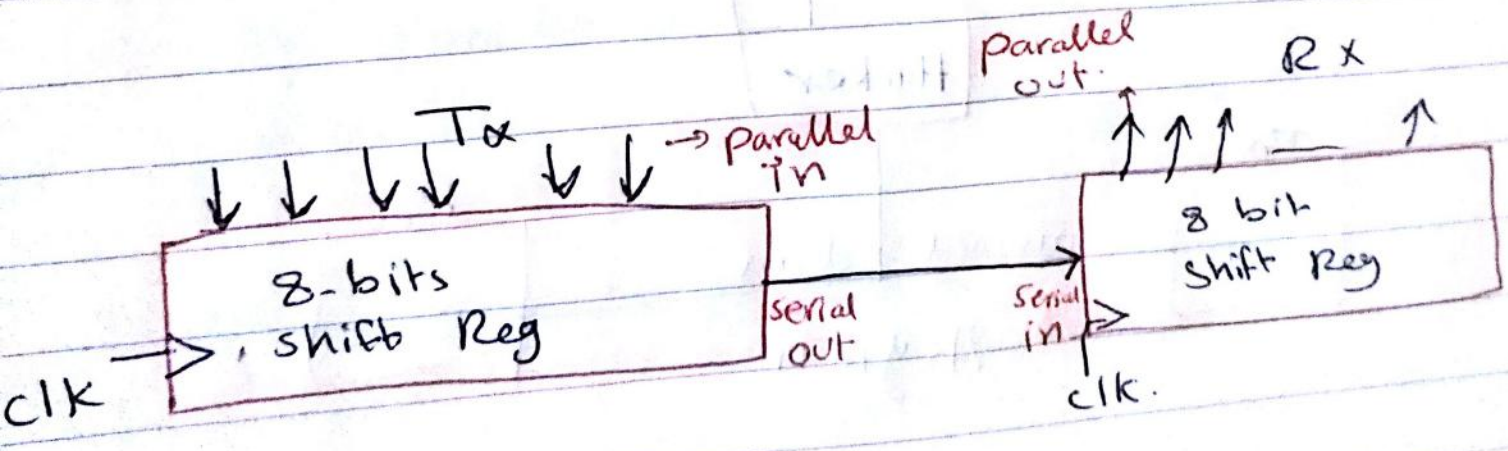
Scanned by CamScanner

* In synchronous serial port, for the reciever to understand the received signal, it should know the protocal used in transmission :-

1. Stop bit
2. Start bit
3. Parity
4. data rate
5. Encoding

→ these taken offline

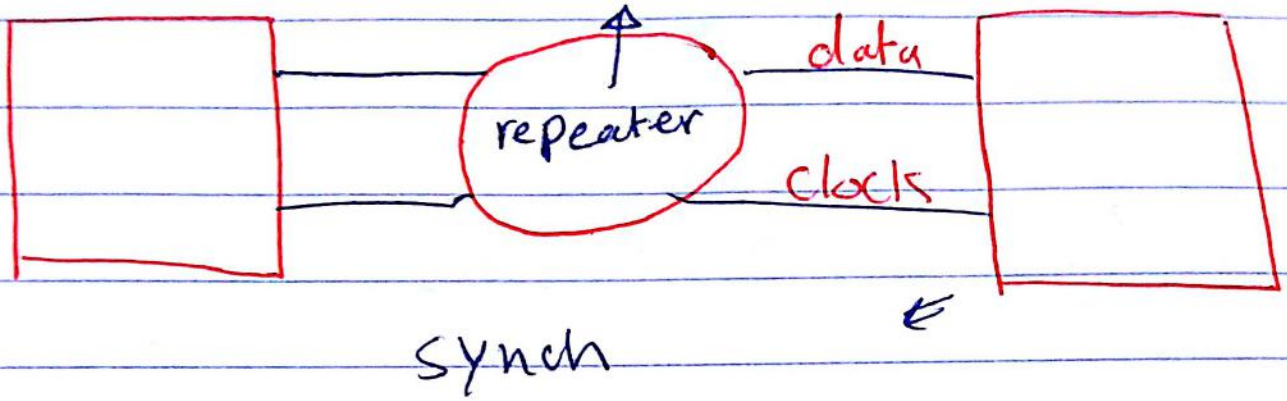
Serial port is shift Register.
8-bit shift Register.



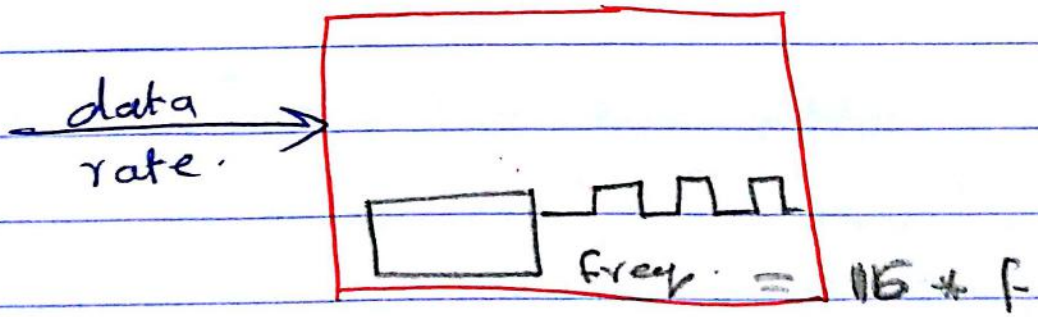
Scanned by CamScanner

* it is not necessary the one who generate the clock be a sender but it will be the master

if the distance is long



synch



Scanned by CamScanner

16F84A → 2 banks (data) — 1 bit for bank selection

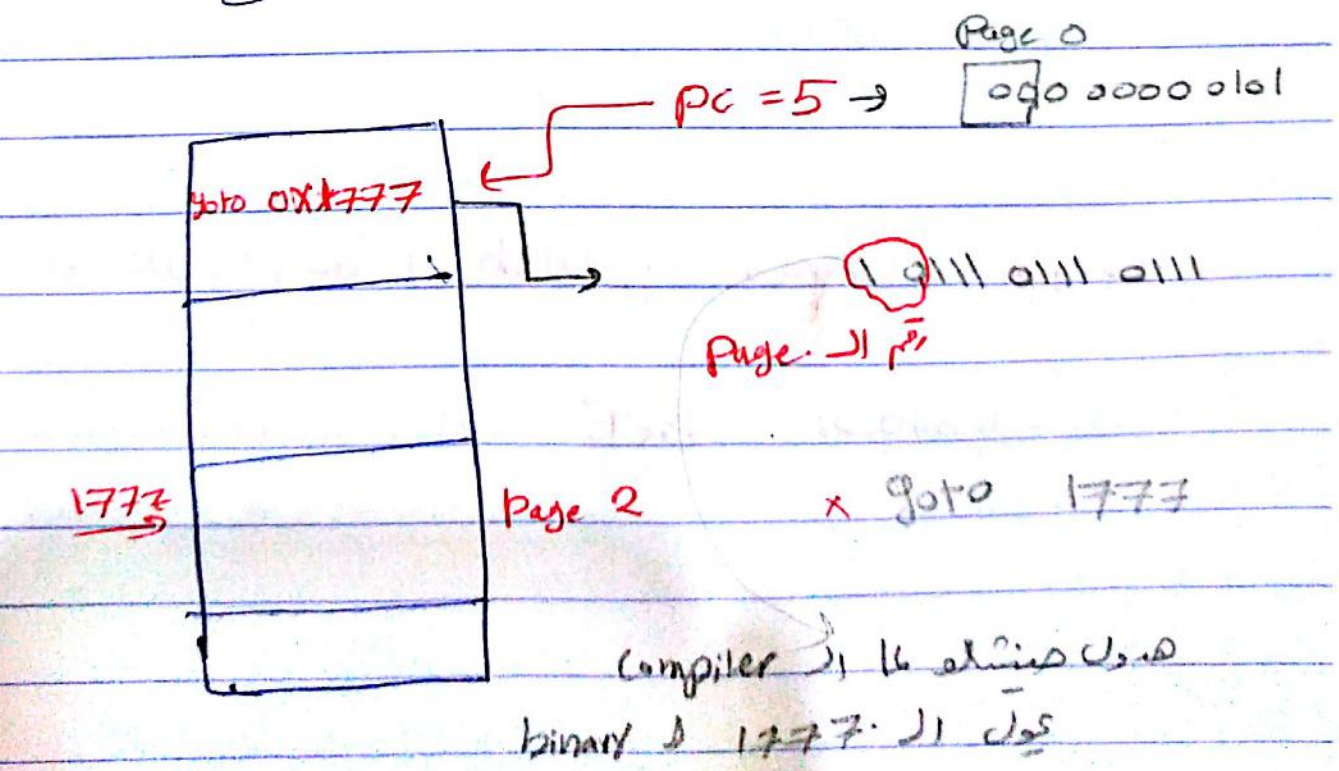
→ 1024 word (prog) — minimum 10 bits for address, However 13 was use. (PC)

PIC16F877A → 4 banks for data memory — 2 bits for bank selection

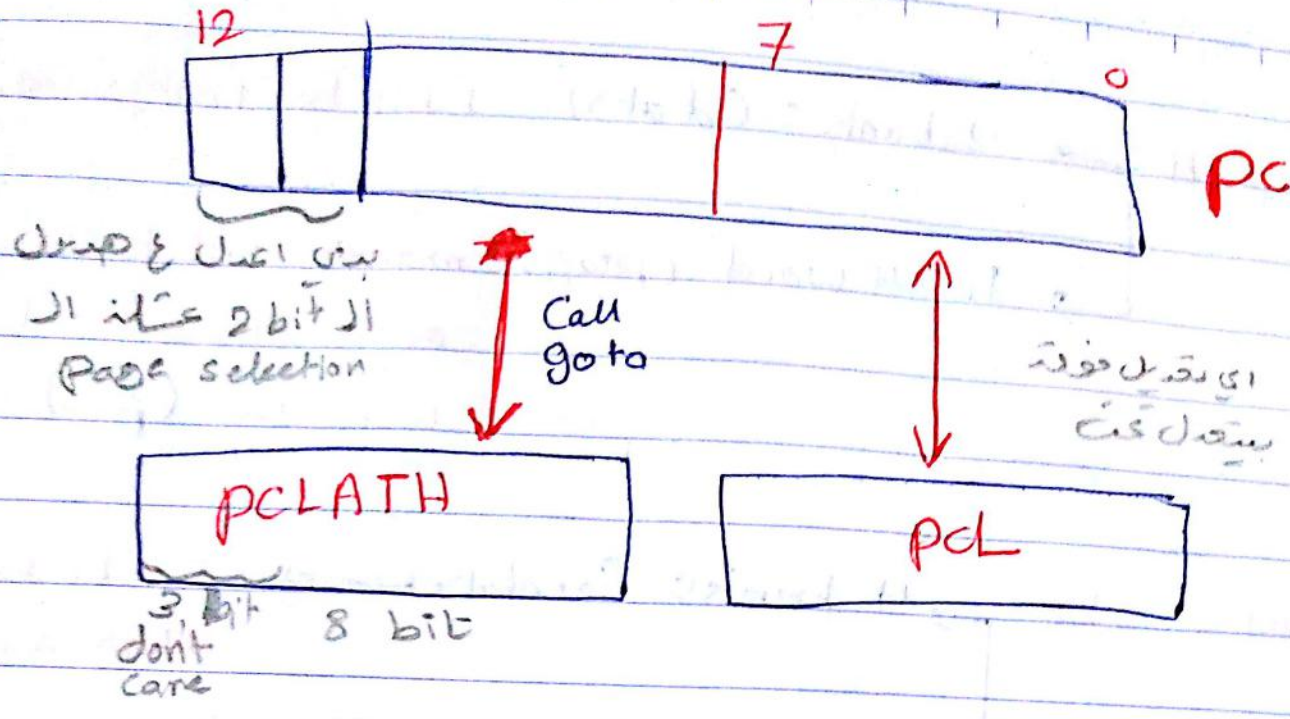
→ 4 pages, each of size 2k words

→ 8k words → 13 bit address (minimum)

Call k / Go to k → 11 bits inst address



Scanned by CamScanner



..we will use page selection. by PCLATch<4:3>

bsF PCLATch, 4 ← الكتابة الي قبل

bcF PCLATch, 3

goto FFF.

لو التعديل على ال pclATch بدّل عال pc مباشرة

كانه ما تنفذت ال bcF وال goto

* 4 banks selection:

direct address: RP1 and RP0 (status <6:5>)

indirect address: IRP (status 7) and FSR(7)

ex:

Write code to read data memory location 0x185 to W-Reg.

bank 3 | 1 1 | 000 0101

address in bank 3

direct:

```
bsF   status, RP1
bsF   status, RP0
movF  0x05,0
```

indirect:

```
bsF   status, IRP
bsF   FSR,7
movlw 0x05
movwF FSR
movF  INDF,0
```

FSR ← 7 bit ← overwriting

Scanned by CamScanner

Right Sol:

```
bsf   Status, IRP
movlw 0x85
movwf FSR
movf  INDF, 0
```

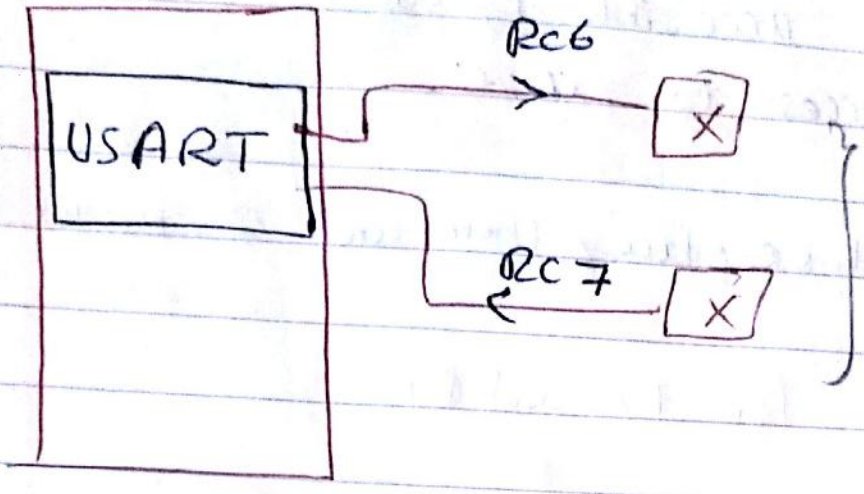
* additional Interrupt Registers :

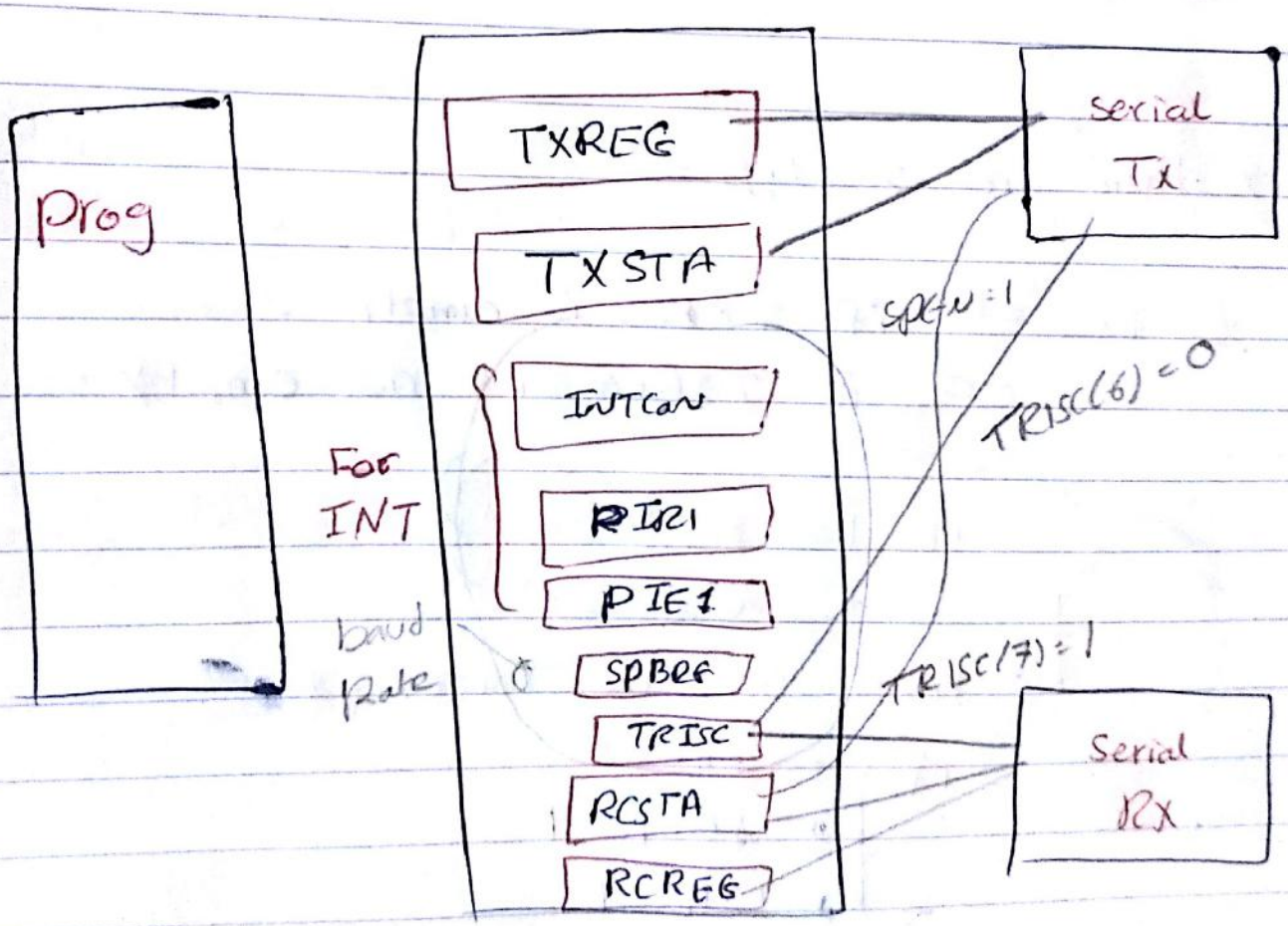
PIR1 ⟶ Flags , PIE1 ⟶ enables

وعنا اسمي جميع المعد PEIE لازم يكون 1 حتى يشتغل

الكود ديه عن interrupt

Scanned by CamScanner





Scanned by CamScanner

* TSR is not accessible by sw.
we have access to TXREG.

For Tx to take place, there are 2 enables:

1. SPEN = 1 for Tx and Rx
2. TXEN = 1 for Tx only.

* there are 2 Flags:

1. TXIF = 1 if TXREG is empty.
   = 0 if TXREG is not empty.

TXIF is read only

↳ may generate interrupt

if ↳ GIE = 1
↳ PEIE = 1
↳ TXIE = 1

Scanned by CamScanner

(2) TRMT = 1 when shift Reg. is empty. (TX is over)
= 0 " " " " not empty.

* اللي بينتهي بـ F بيعمل Interrupt.

* baud Rate = F(SPBRG, BRFH bit, Fosc).

PAE → مش موجود بالـ INTCON<6>
غلط هاي الإجابة محطوطة بالسلايدات

* RSR is not accessible by S.W.
* RCREG is accessible by SW
* RCREG is a queue of 2 levels (FIFO)

* when one byte is received, it moves automatically to RCREG and Reception continues.

* when the stop bit of the third byte is received before reading the previous two bytes ⇒ OERR = 1

* OERR is read only bit.

OERR = 1 →
- (1) Reception stops
- (2) 3rd byte will be lost

* RCIF = 1 if there is at least one byte in RCREG ⟹ interrupt if GIE = 1
PEIE = 1
RCIE = 1

movf RCREG,0 ⟹ بتقرأ وبتفضي

ما بتقدر نحدد لو أي byte بدنا نقرأ لأنو FIFO.

* For RX to take place ⟹ 2 enables → 1. SPEN = 1
2. CREN = 1

* FERR = 1 if the recieved stop bit = 0

لازم نقرأ الـ bit قبل الـ byte لأنو لو قرأت الـ byte بالأول

صير الـ bit يأشر على الـ byte اللي بدها.

Scanned by CamScanner

Function.

* Baud rate = F (Fosc , SPBRG, BRGH)

$$= \frac{F_{osc}}{16 * (1 + SPBRG)}$$

BRGH=1 ← 16

BRGH=0 ← 64

*example Slide 33 :

PIE X
PIRI X
INTCON X

} INTR لانو ما منستخدم
ع نقل polling

$$9.6 * 10^3 = \frac{20 * 10^6}{\{16 \text{ or } 64\} * (1 + SPBRG)}$$

يا هاي يا هاي بناخذ ⇐ (16 أو 64)
بنجرب وحدة واذا
طلع اكبر من 255
صح واذا لا بنجرب
الثاني.

بدنا ناخذ ال 64 يعني BRGH=0

واذا الثنتين صح
بنختار اي واحد
عادي

$$SPBRG = \frac{20 * 10^6}{9.6 * 10^3 * 64} - 1$$

= 31

TXSTA = x0100 x0xix

TXREG =

TRISC = x0xx xxxx

SPBRG = 31

RCSTA = 1xxx xxxx

Baud Rate ⇒ # of bits per second.

PRGH ← default ما حكى عنها لانه الـ
تكون صفر.

اول مرة ما غلط check عال TXIF
لانه فاضي

```
movF   FSR,W
sublw  0x43
btFSS  STATUS,Z
```

هل الـ value وصل للـ 0x43
اذا اه بطلع اذا لا برجع عال
TX

Scanned by CamScanner

`bcf PIR1, TX1F` → ضفّ لألو

Cleared by HW.

لو بدي أعمله clear بخط byte بـ TX ولو بدي أبعت

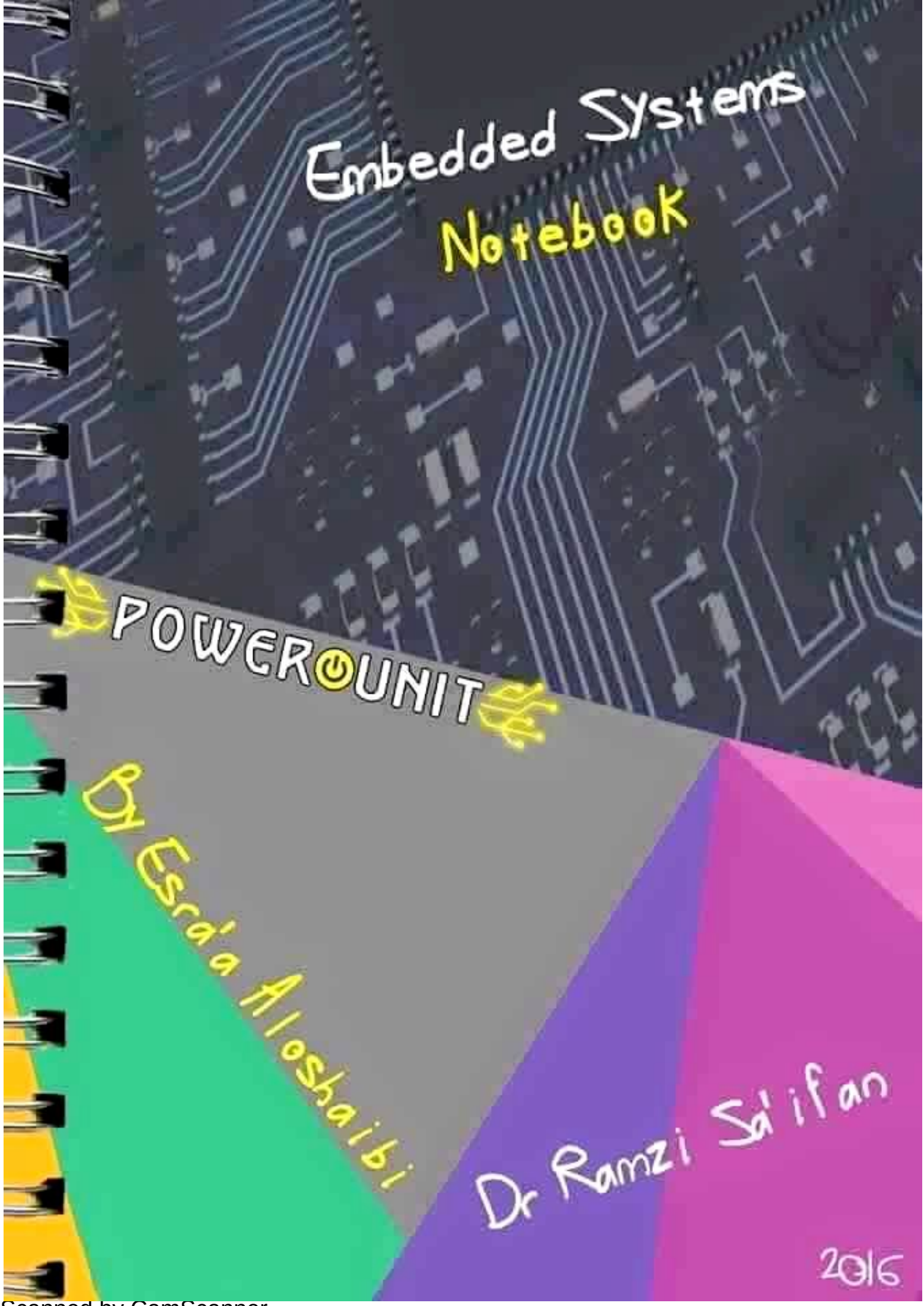
register

بخل clear لـ TX

enable

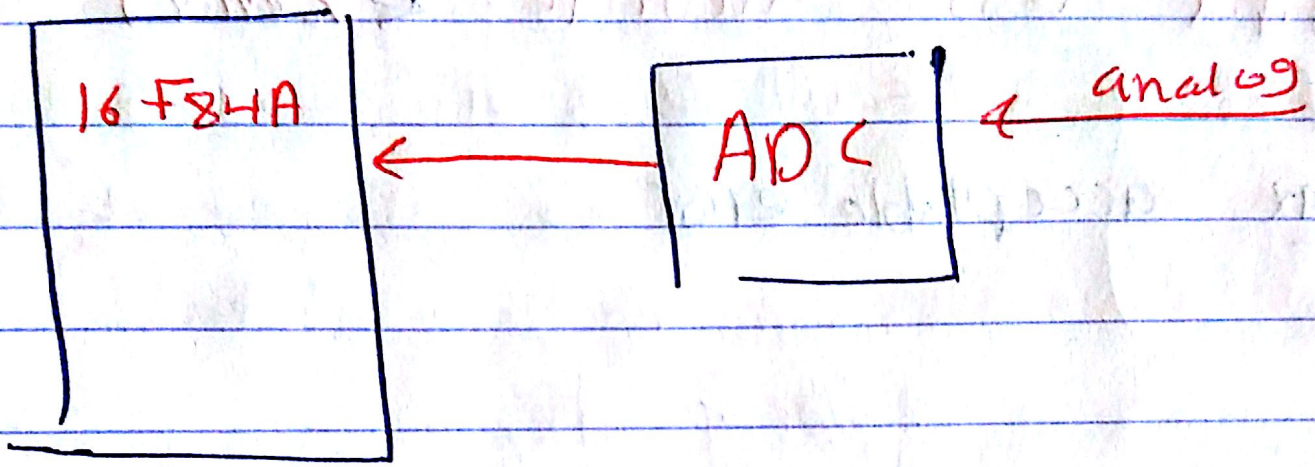
✱ chapter 11 :

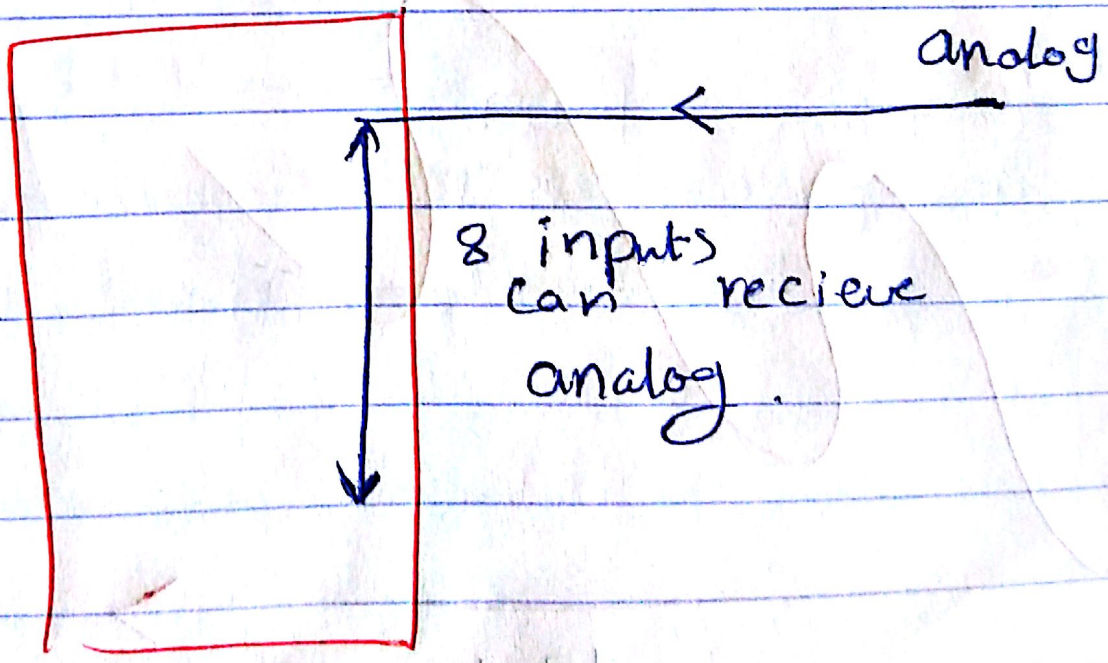
Scanned by CamScanner



Scanned by CamScanner

* chapter 11 :





Scanned by CamScanner

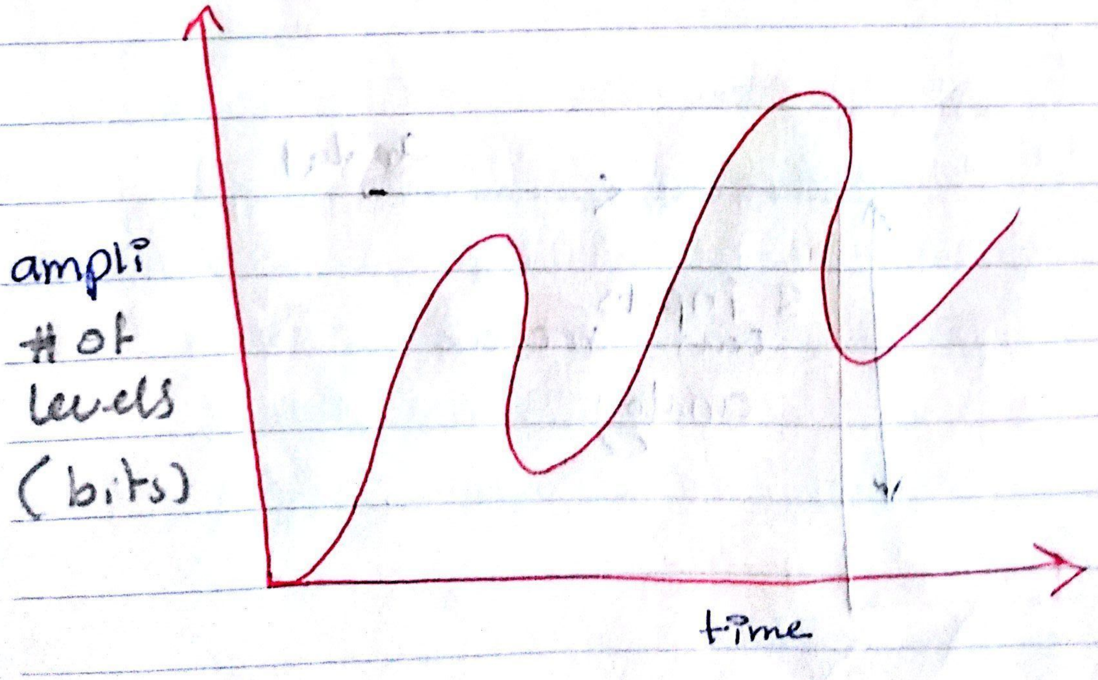
* analog: Can take Unlimited # of Values in Specific Range

* digital: Can take limited # of Values in Specific Range.

بدون الـ Range ، التنين بياخد Unlimited Values.

analog more accurate → loss }
digital less accurate. ← % error } عند التحويل

max acceptable error → اذا تخطيناه بضر ... في الـ data



ADC steps:

1) Sampling:
Reading the analog input.
Sampling rate: every How long you read a sample.

more samples ⟹ ⊕ more accurate.
⊖ more power
⊖ more storage.
⊖ ~~[illegible]~~ faster ADC.
⊖ more cost.

minimum sampling rate ≥ 2 * f_{max}.

ex:
You bought an ADC with maximum sampling rate = 100k sample/second.

heighst signal frequency can be converted.

$\frac{100kHz}{2} = 50kHz$.

> 50kHz ⟹ aliasing

Scanned by CamScanner

② Quantization:

Rounding the sampled analog value to the closest acceptable digital value.

more bits (digital levels) →
- ⊕ more accurate
- ⊖ more storage
- ⊖ Faster ADC
- ⊖ more power

V_{REF}^{+}
V_{REF}^{-}
} الـ Range اللي بكوّن بينو اي اشي برا حدود الـ range بنعمله clipping

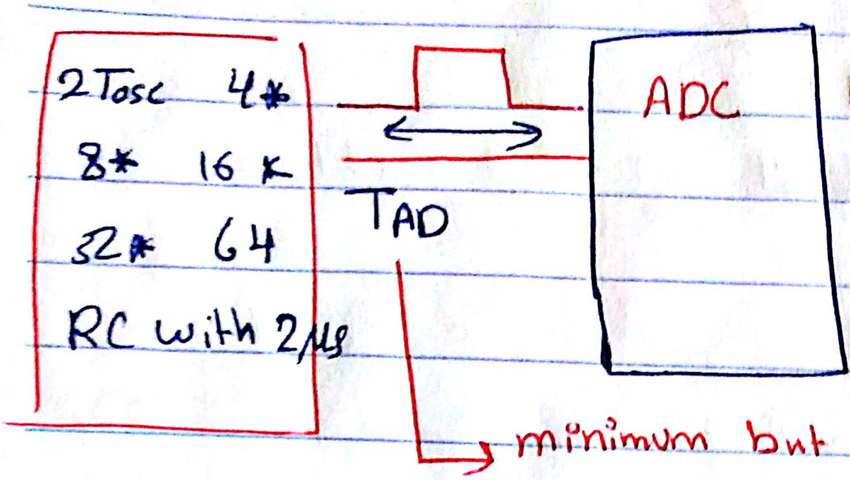
ازا استخنينا عن الـ Vref صار less accurate.

Scanned by CamScanner

11/4

Scanned by CamScanner

* 10 bits ADC , Sampling time = 7.6 τ.



10 bits conversion (Quantization) require 12 T_{AD}

PCF6 bits → يحددلي مين من ال inputs
digital or analog

GO / $\overline{DONE}$ = 1 ⟹ Start conversion

⟹ become 0 when conversion complets

ال Result افضل تكون right justified عشان الصفر اللي عالشمال

ما بأثر وهاد تكون مهم بنتيجة 8 bit بنخليها left justified.

Scanned by CamScanner

10 0000 0001 = 513

right justified → 1

left justified → 512

نتيجة أقرب عشان هيك نفضل ال left justified

حالة قراءة 8 bit

ADC Code steps :

configuration (ADCON0, ADCON1, interrupt, TRIS)

delay " for sampling " You have to write code for it.

GO/DONE = 1 ← هون بنكتبش code delay

↳ takes time but no need for delay code

GO/DONE = 0 ← لحالها

ADIF = 1

اما ال sampling ما عندنا اشي يقلي انو خلصت عشان هيك بنكتبلو code.

to configrate ADON , Go/Done in single step:

movLW B'00000 0101'
movwf ADCON0

} X

delay ملحق لازم فيش

*ex :

what are the required settings to read single analog input and interval voltage references?

PCFG = 1110 من ال table.

CHS2:0 = 000

TRISA(0) = '1'

C/R

→ # of analog inputs

→ # of external voltage refrences

Single analog input = 1/0.

Scanned by CamScanner

قيمة Rs بتعتمد على قيمة V_{DD}

* Sampling circuit (acqusition circuit):

Sampling time = 7.6 RC

Sampling time = 2 µS for op Amp settling time

$$+ 7.6 * (R_S + R_{IC} + R_{SS}) * C_{Hold}$$

R_S: analog input — R_{IC}: internal R_{IF} — R_{SS}: switch dependent on V_{DD} — C_{Hold}: internal

Temp coef (0.05 µS per °C > 25)

كل هاي للعادلة عشانه لغرف أدبش مع نفط الـ delay

$$= 2\,\mu S + 7.6 (R_S + R_{IC} + R_{SS}) * C_{Hold}$$

$$+ \boxed{(T - 25) * 0.05\,\mu S}$$

if $T < 25 \Rightarrow$ Temp coef = 0

Scanned by CamScanner

Example slide 36

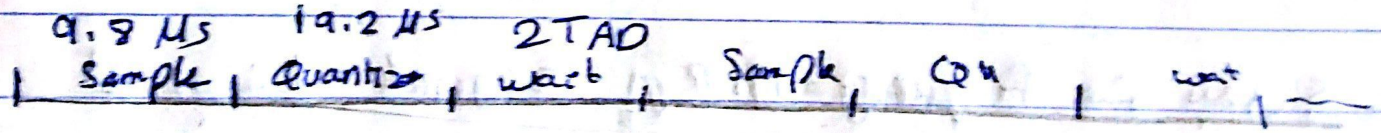
Conversion (Quantization) time = 12 * T_{AD}

= 12 * 1.6 μs

= 19.2 μs

total Conversion time for one sample =

9.8 + 19.2 = 29 μs



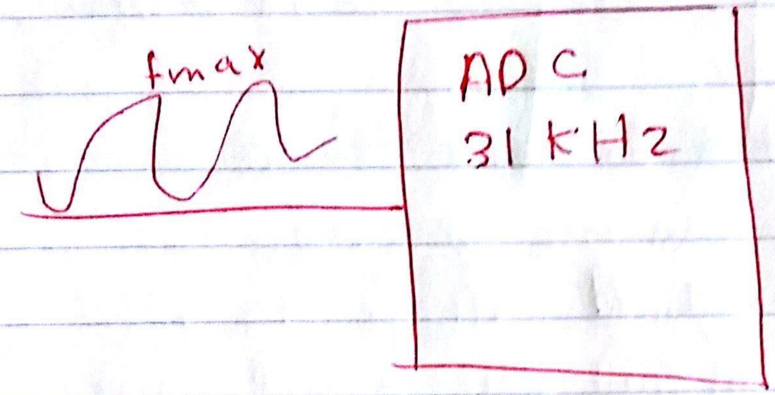
* for repeated samples, we should wait 2 T_{AD} between two samples

⇒ total time for one sample if we assume repeated sampling = 29 + 2 * 1.6 μs = 32.2 μs

Scanned by CamScanner

$\Rightarrow$ sampling rate $= \dfrac{1}{32.2 \times 10^{-6}} = 31\ kHz$

what is the max freq that can be connected to this ADC ?



$= \frac{1}{2}$ sampling rate $= 15.5\ kHz$

لو اجتنا signal بـ frequency = 17kHz شو بنعمل ؟

① بنجيب external ADC بس هالشي غالي .

② if we are intersted in 5 bits conversion

$\Rightarrow$ result less accurate. $\dfrac{V_r}{2^{n+1}}$ كانت 10 صارت 5 ال error تزداد

دايمًا ال ADC بجيبلنا 10bit بس انا مهتمة بأول 5 وهدول بيطلعو بعد حوالي 7 cycle هال 5 التانيين بدهم كمان

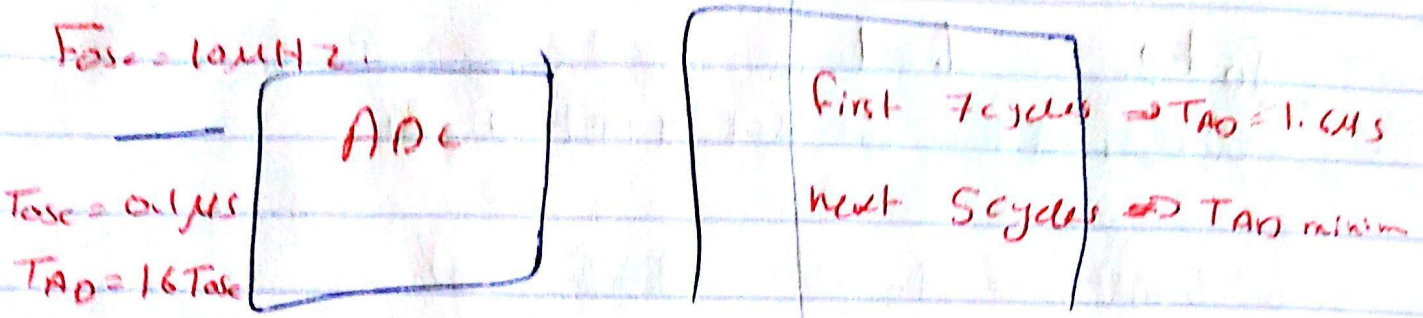
$T_{AD} = 1.6\mu s$ لـ 7cycle بيعطيني اول 5cycle .

T_{AD} minimum لـ 5cycle

Scanned by CamScanner

Conv time

19.2



$T_{AD\,min} = 2\,T_{osc} = 0.2\,\mu s.$

$7 * 1.6\mu s + 5 * 0.2$

$= 12.2\mu s$

وقرنا 7μs على حساب الـ accuracy

الـ 5 least sign bit الأخيرة

* Total conversion time = 9.8 + 12.2 + 3.2

= 24.2 μs.

* Sampling rate = $\frac{1}{24.2 * 10^{-6}}$ = 41.3

* max freq = 20.65 KHz

لازم حتى نقدر نشتغل الـ 17KHz

go to $-1

اطلع لل inst السابقة لو 1 أكثر انزل للي تحت

لو 5-$ اطلع لهناك قبل 5 inst

كأنها label يعني :a

*Chapter 8:

The keypad is 12 push buttons:
arranged and read in specific ways!

* reading the pressed push button is in two stages:

1. We read row # → make row pins input
   make column pins output
   output 0 on column pins.

   TRISB

   movlw B'11110000'
   movwf TRISB
   clrf PortB

   movf PortB, 0 ←
   movf row-index

2. We read column # → make row pins output
   make column pins input
   output 0 on row pins.

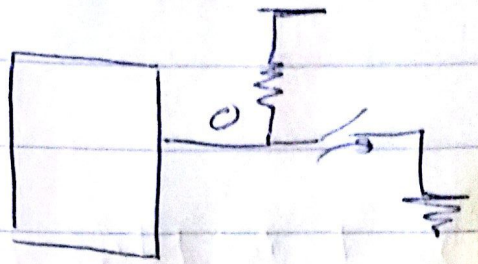
   movf PortB, 0 ←
   movwf column-index

   movlw B'0000 1110' (output / input)
   movwf TRISB
   clrf PortB

   will clear only the output pins

Scanned by CamScanner

* slide 7:



Case1 → row index = 111 0 0000, 3 (single)

Column index = 0000 1010, 1 (single zero)

apply code slide 13

W = 1, 3

Value = 3 + row index + Column index

= 10

```
movlw   B'1111 0000'
movwf   TRISB
bcf     STATUS, RP0
clrf    PORTB
movf    PORTB, 0
movwf   Row index
bsf     STATUS, RP0
movlw   B'0000 1110'
bcf     STATUS, RP0
clrf    PORTB, 0
movwf   Column Index
```

Case 2 → row index = 1110 0000, 3

Column index = 0000 0110, 0

W = 0, 3

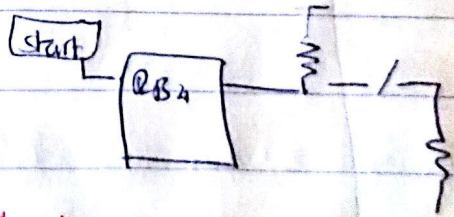
Value = 9

slide 9:

ISR

=

```
State XOR1
Loop btfsc  PORTB, 4
     goto   Loop
     bcf    INTCON, RBIF
     retfie
```



check botton released or not

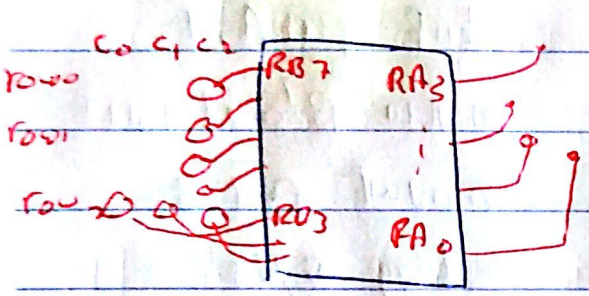
else interrupt happens twice "when pressed and when released"

Scanned by CamScanner

* To clear RBIF → doesn't lose the value on Reset.
(" Port B change interrupt flag to be cleared ")

```
movf PortB, 0
BCF INTCON, RBIF
```

example slide 10:



6 lines ← row

⇒ rowindex = 101 0000

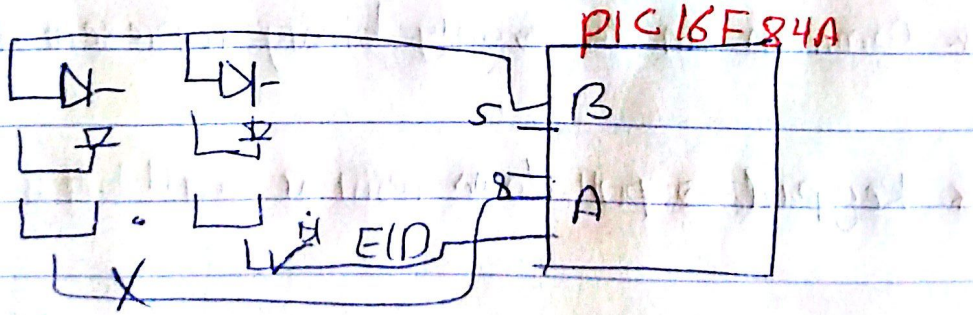
⇒ columnindex = 0000 1100 ?

⇒ W-Reg = 2, 1, 5

column ← row ← value

column ←

*slide 16:



If we send 5 → movlw B'0110 1101'
both will display 5 → movf PortB
we need solution ⇒ one on other off

Scanned by CamScanner

25 ⇄ 52 → 0101 1011 → loop bsf portA,0
bcf portA,1
movlw B'0110 1101'
movwf portB
delay 5ms
bcf portA,0
bsf portA,1
movlw B'0101 1011'
movwf portB
delay 5ms
goto Loop.

* if only for 1 second show 25
we add counter (loop داخل)
movlw D'100'
movwf counter

Decfsz counter,1 } after delay
goto Loop

Slide 18 → use decoder to save one extra bit.

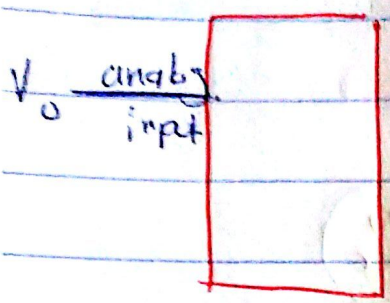
LED * common anode → only binary representation

* keypad → pull down resistor → all btfss ↑ change → btfsc

* portB interrupt : we can't read columns then rows because never portB change interrupt will happen (rows are output they must be input)

Scanned by CamScanner

*Slide 28

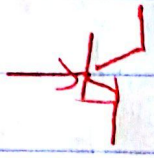


$$V_o = 5 * \frac{R_{LDR}}{10k + R_{LDR}}$$

must connect on A/D if V_o down more light more R.

* optical object sensor:

1. IR LED, when current passes through it, IR light is emitted.

2. Optical transistor 

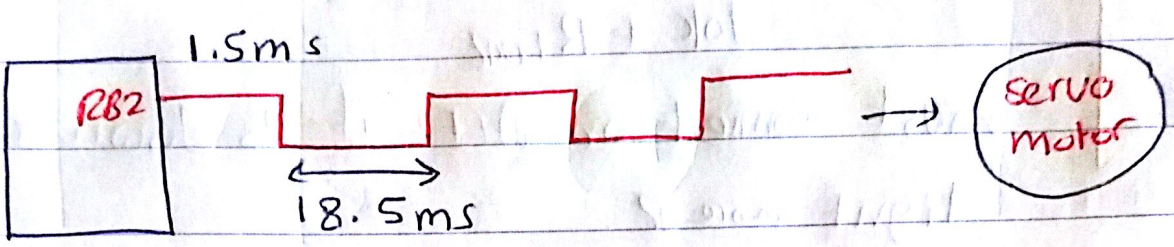
3. plastic housing: passes only useful light (IR)

* it works in two ways:

   1. cutting light
   2. reflecting light.

Scanned by CamScanner

* Servo motor $0 \rightarrow 180^\circ$



F = 50 Hz

* the angle is dependent on pulse width

to rotate $35^\circ \rightarrow 1.25 + 35 * \left(\frac{1.5 - 1.25}{90}\right)$ per degree

```
Loop    BSF  portB, 2
        Call delay1.5ms

        BCF  portB, 2
        Call delay18.5ms

        goto Loop
```

Scanned by CamScanner

/ /

* ما بصير أشبك الـ pic دغري مع الـ motor ← ما هيشتغل
فبخلي الـ motor يوخذ external voltage والـ pic يشغل الـ switch

motor → inductive → ما افضل الاسم يظل شغال power ما بخلي
شوي "بفصلش البرجة"

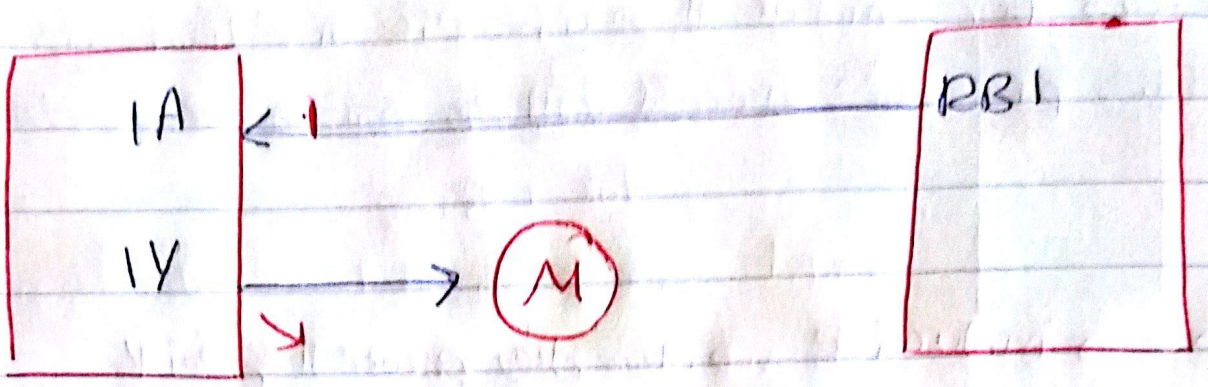
Slide 42 → ازا سكرنا 2 switches بنفس الجهة بصير عندي
short circuit ، بنحرق الـ motor
ولو سكرنا تنين جنب بعض ما بمشيش current.

L 293D ↴

2 Motors 2 directions
4 motors 1 direction

A → input
↓
بتكون جاي من الـ pic

Y → output
↓
عليها شبك الـ motor



external Voltage source

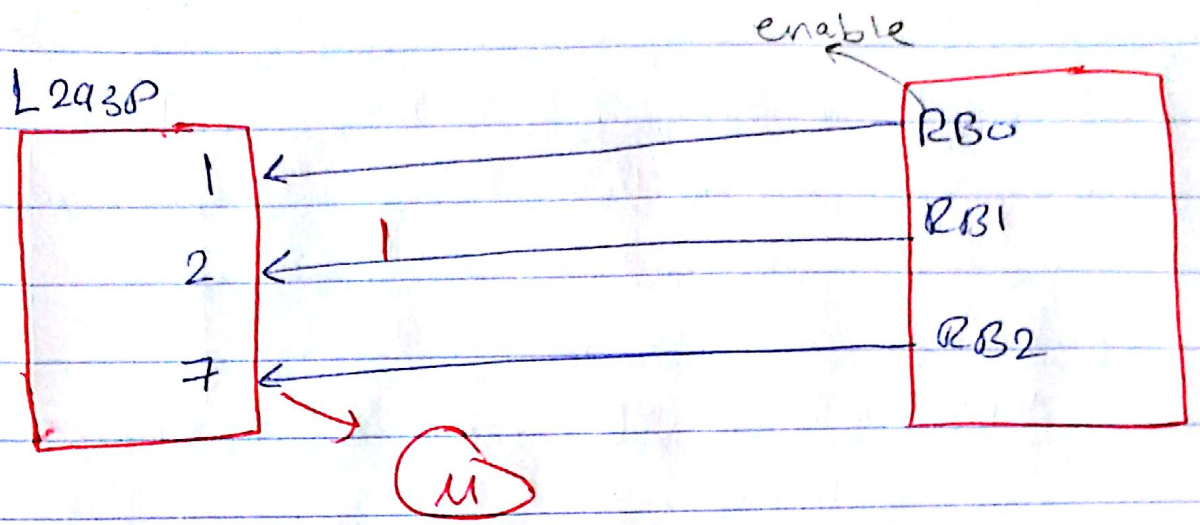
* 2 V_{cc} → V_{LS} : if input is high
⇒ will be represented as V_{LS}
→ V_{OS}.

$V_{LS} < V_{OS}$

* if $V_{OS} = 0$ all connected motors are off.

V_{LS} → motors هي اللي داخل على

Scanned by CamScanner



* write code to let the current pass up down

```
BSF   portB,1
Bcf   portB,2
```

لو ما بدي أشغل الـ motor

اوديت 1,1

```
Bcf   portB,0
```
→ RB0 → enable

mosfet / BJT → switch بشغل من خلالهم الـ PIC

* $R_{Prot} = 1K\Omega$, max current of diode = 20mA , $V_D = 0.3V$

$V_{max} = 20mA * K\Omega + 5.3$ → $5 + 0.3$ ← V_{DD}

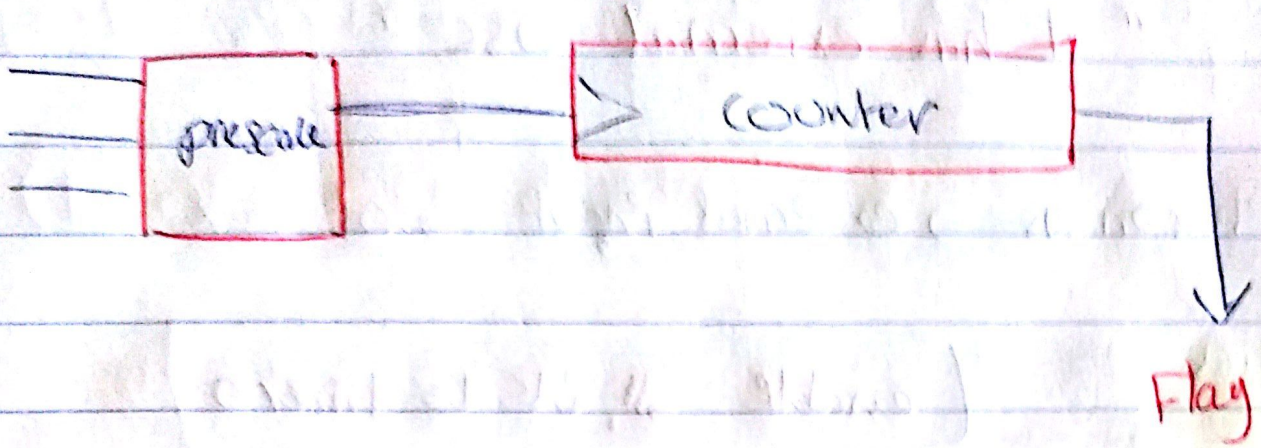
$= 20 + 5.3$

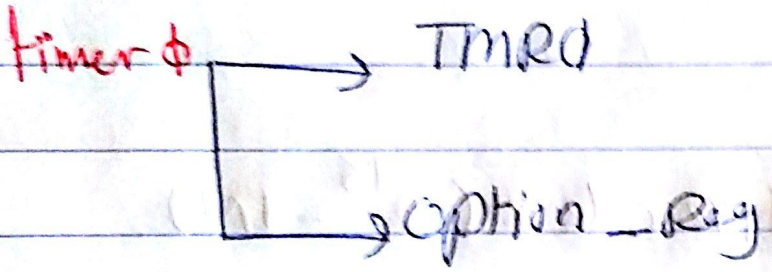
$= 25.3$

$V_{min} = 20.3V$

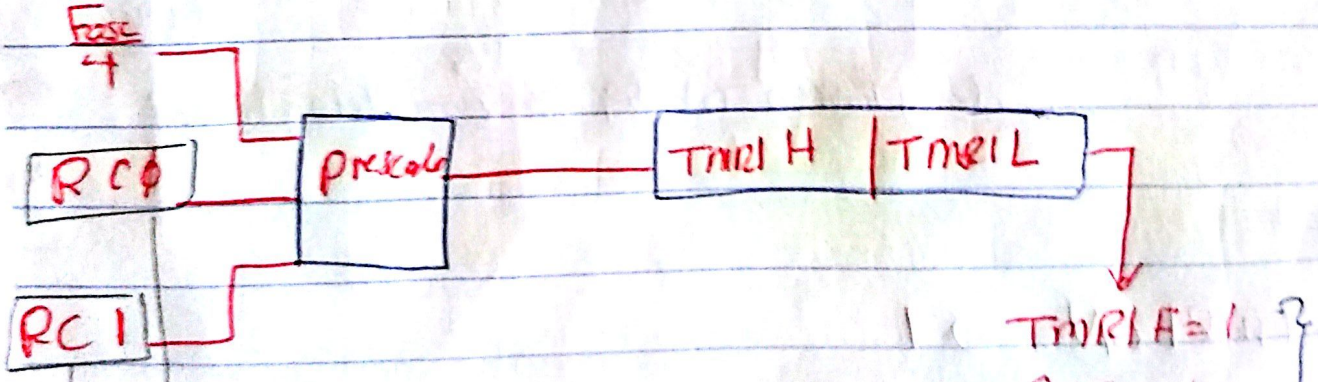
Scanned by CamScanner

* chapter 9:





timer 1 → 16 bit (0 — $(2^{16}-1)$)



Scanned by CamScanner

timer 1 → keeps counting in sleep mode

"it has external osc"

synchoronous → external input → o mud

enable الى كان لو timer o

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

= (8 * To delay) ($\frac{4}{Fosc}$ * 256 * 256)

delay timer o بيبقى 8 ← delay timer 1

TMR1 ON → 1

يشتغل

Timer 2 → 8 bits counter 0 → PR2

بعدّ من ال 0 لحد ما يوصل لقيمة PR2 يعني مش ل 255

when timer2 = PR2 → timer 2 starts counting again , from 0

في فرق بين timer0 و timer 2 بقيم ال prescale الممكنة.

* ex :

if postscale = 5 and PR2 = 100 then when TMR2 = 100 5 times ⟹ TMR2IF = 1

يعني بصير يعد لل 100 "5 مرات" مش يصير ال flag=1.

$$TMR2\ delay = \frac{4}{Fosc} * prescale * postscale * (PR_2 + 1)$$

معنا منش initial value لانو صيك صيك مبخلو Reset كا شيادي

مع ال PR2.

* Suppose TMR2 has initial value

delay of TmR2 = $\frac{4}{Fosc}$ * prescale * (PR2 - N + 1)

+ $\frac{4}{Fosc}$ * prescale * (postscale (-1) * (PR2 + 1)

هاد القانون مع بعض في حالات ميكون ناقص

PR2 < initial ⇐

TMR2 = 100 } → صيد من 100 لـ 255
PR2 = 50 ويرجع 0 ويعد لـ 50
يعني أخد 155 + 50 = 205 ×

* max delay of TMR2 = $\frac{4}{Fosc}$ * 16 * 16 * 256 (PR2+1)

= $\frac{4}{Fosc}$ * 2^{16}

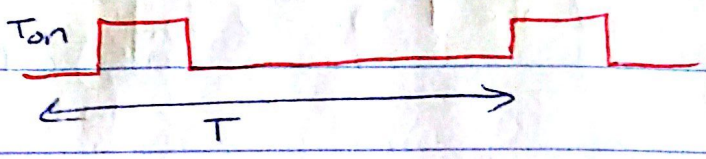
max delay of timer0 = max delay of timer2

postscale = 4

Scanned by CamScanner

CCP

- Capture
- Compare
- pwm "pulse with modulation"



Servomotor

Capture: when an event happens ⇒ record the time

Compare: When the time equal value ⇒ do an event.

* CCP can work in one of these modes at a time

يشتغل بأكثر من واحد مع بعض

CCP1RL } Data.
CCP1RH }
CCP1CON } control.

Capture → RC2 → لازم يكون Input

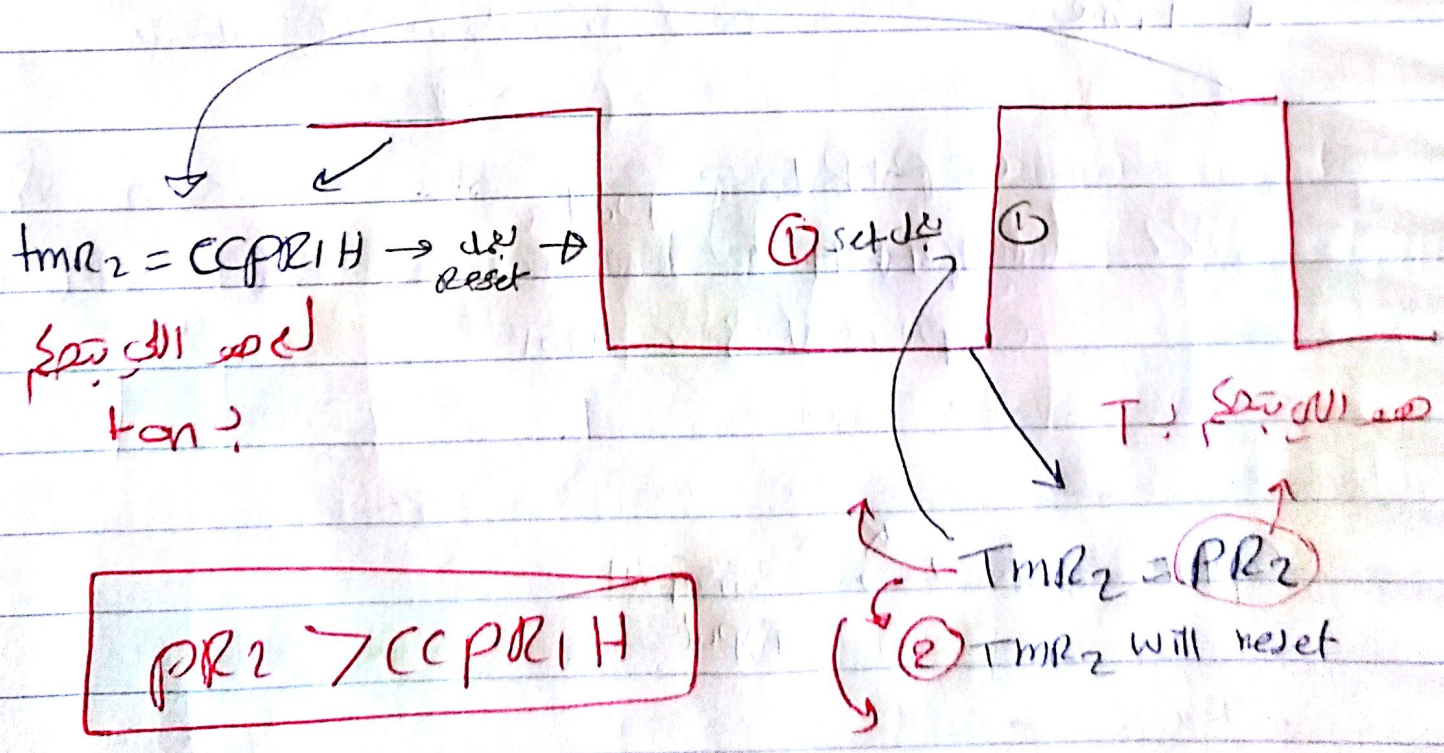
Compare → RC2 → لازم يكون output

↳ بالحالة الأربعة فيها الـ Flag = 1.

pwm

S	R	Q
0	0	Q
0	1	0
1	0	1
1	1	unstable

لما بتساوى مع TMR2 بعمل CCPR1H R=1 بعمل PR2 ولما بتساوى مع بعمل S=1



لهي اللي بتحكم ton :

Scanned by CamScanner

CCPR1L ينتقل الى ← CCPR1H

TMR2 = PR2 لما

```
movlw   D'100'
movwf   CCPR1H
        CCPR1L
```

```
movlw   D'200'
movwf   CCPR1H
        CCPR1L
```

} ton > هون من اللي فوقة

هاد لما أول مرة أكتب القيمة 100 وتساوى قيمة TMR2 مع PR2 حتنزل القيمة الموجودة بـ CCPR1L وتنكتب محل القيمة اللي بـ CCPR1H نزال 200 حتنكتب مرة على cycle وحدة بعدين بنكتب القيم اللي بدنا إياها "200" بـ CCPR1L

مقارنة

* 10 bits Value : CCPR1L || CCP1CON<5:4>

بحالة ال PWM

Note1 → 2 bits internal counter. كأنه prescale

بدي اقارن 8 bit مع 10 bit مش منطقي فبضيفوا تيجي → timer2

فايدة ال 2 bits تعوض Note1 اللي بنضيفها لـ timer2

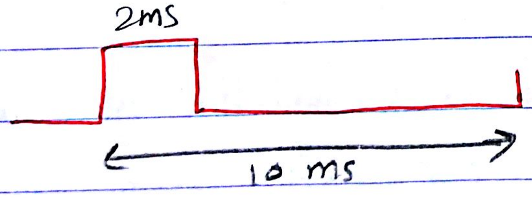
CCPR1H + CCP1CON

8 bit   2 bit

لما يخلص ال cycle نغير على قيمة ال CCPR1L هاد لتتغير

فبتبين نتيجته بال next cycle

* write a code to generate PWM signal with ton = 2 ms and T = 10 ms if Fosc = 1 MHz



$t_{on} = \text{Value (10 bits)} \times 10^{-6} \times \text{prescale}$ (10^{-6} ↓ T_{osc})

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

$\boxed{V = 125}$

لازم نستخدم نفس قيمة ال prescale تاعت التنتين

$T = (PR_2 + 1) \times (10^{-6} \times 4 \times 16)$

$10 \times 10^{-3} = 64 \times 10^{-6} PR_2 + 64 \times 10^{-6}$

$\boxed{PR_2 = 155}$

Scanned by CamScanner

Code → $PR_2 = 155$

CCP1CON = 0f

CCPR1L = 0001111

10bits

$V = 125 = 0001111101$

TMR2CON = xxxx 1 1x

on خليه بتكون 16 prescale

disable يكون pwm mode بال postscale ال

انتهت المادة

بالتوفيق ☺

Scanned by CamScanner