EMBEDDED

DR.EYAD JAFAR

BY:REEM MUIN

POWERUNIT

What is an Embedded System?

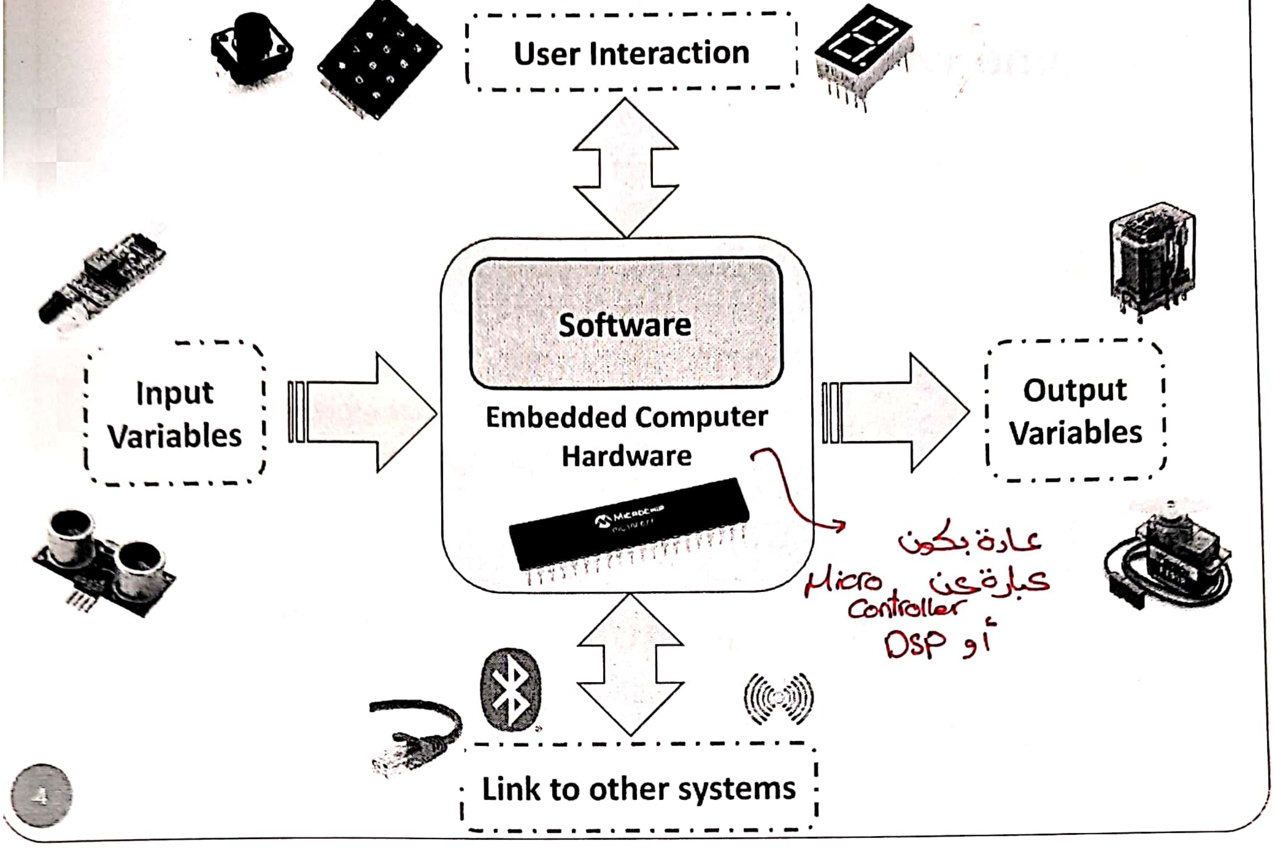
← مصمم ليسوي task محددة وصعب تعديلو

- An *embedded system* is a computer system that is
  - designed to perform one or a few dedicated functions often with real-time computing constraints
  - *embedded as part of a complete* device often including hardware and mechanical parts.

← Embedded عكسها

- By contrast, a general-purpose computer, such as a personal computer, is designed to be flexible and to meet a wide range of end-user needs.

The Essence of Embedded Systems



User Interaction

Input Variables

Software

Embedded Computer Hardware

Output Variables

Link to other systems

عادة بكون عبارة عن Micro Controller أو DSP

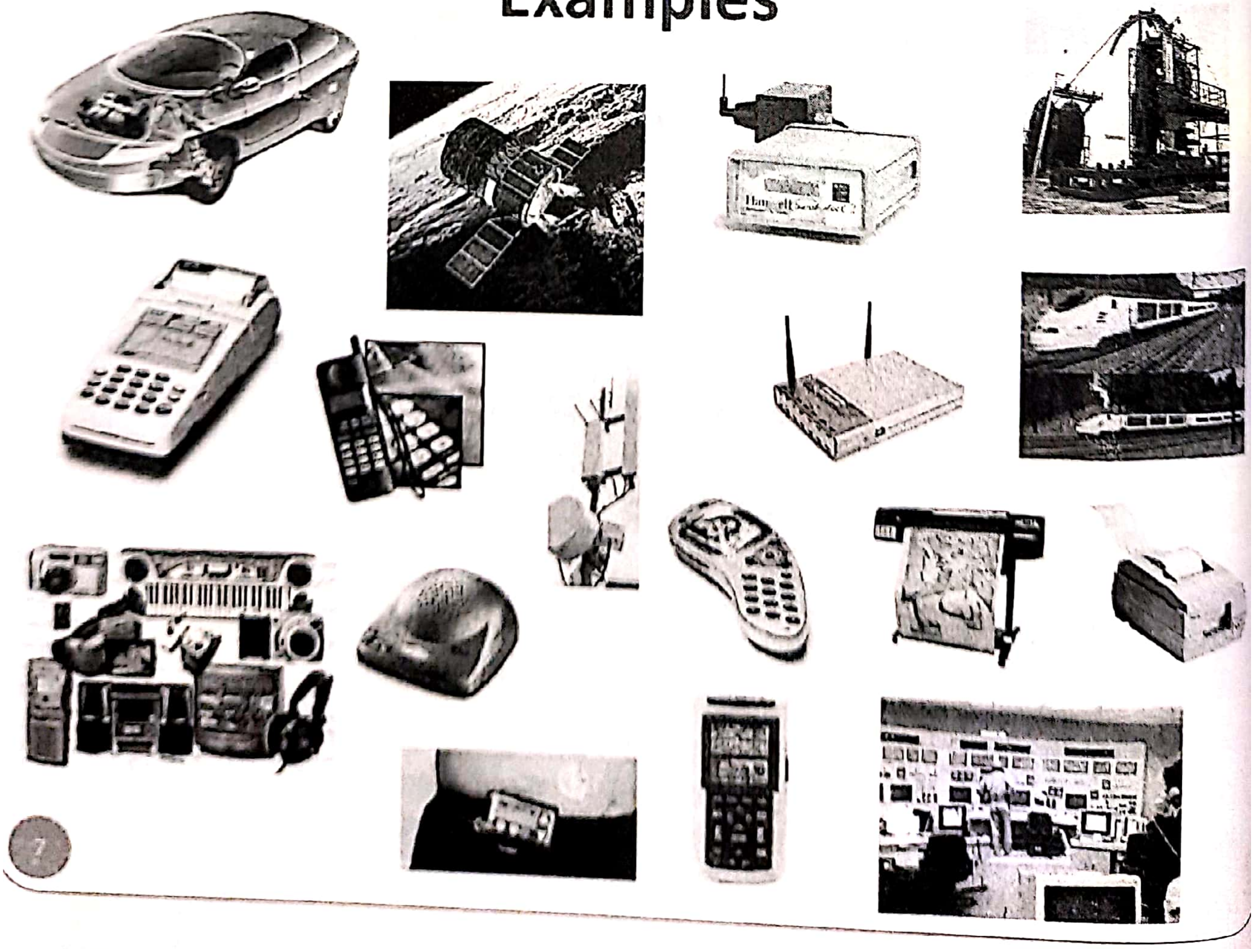
The Essence of Embedded Systems

- **Characteristics** : تتشارك في خصائص
  - *Software driven*
  - *Reliable*
  - *Real-time control system* ← تشتغل بالوقت الحقيقي
  - *Microcontroller or DSP based* → hardware ال
  - *Autonomous / human interactive / network interactive*
    ماحد بيتدخل بعملهم زي الروبوت ←
  - *Operate on diverse input variables and in diverse environments*

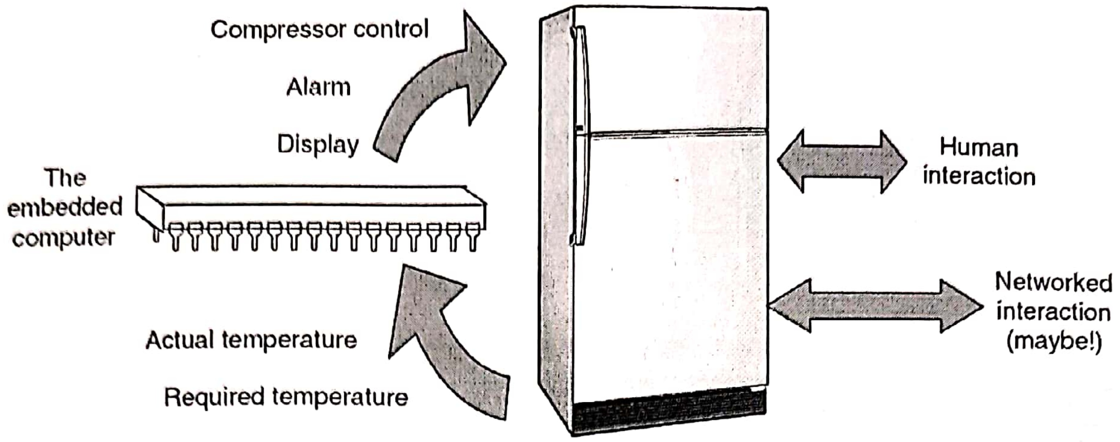
Examples

- Automotive
- Avionics/Aerospace/Defence
- Industrial Automation
- Telecommunications
- Consumer Electronics & Intelligent Homes & Retail (Thin Clients/POS)
- Scientific & Medical Equipment
- Computer peripherals

Examples



Examples



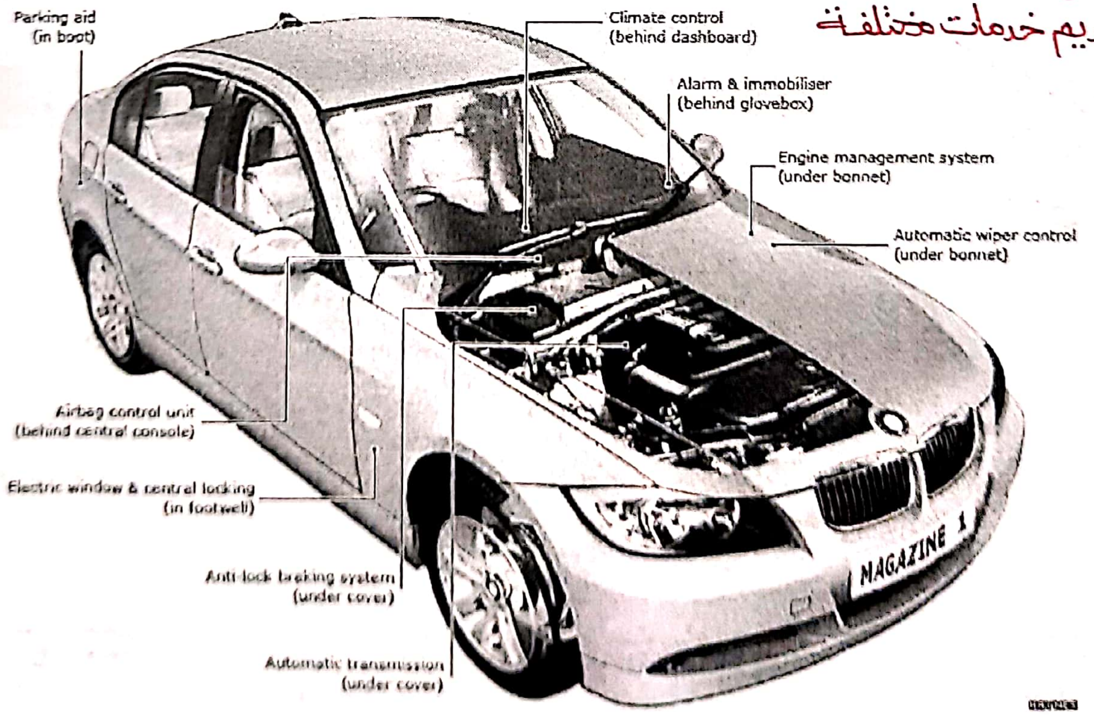
- The refrigerator is required to maintain low temperature by reading the current value and controlling the compressor accordingly

عملية التحكم في درجة الحرارة تكون حسب المستخدم عن طريق

Embedded system

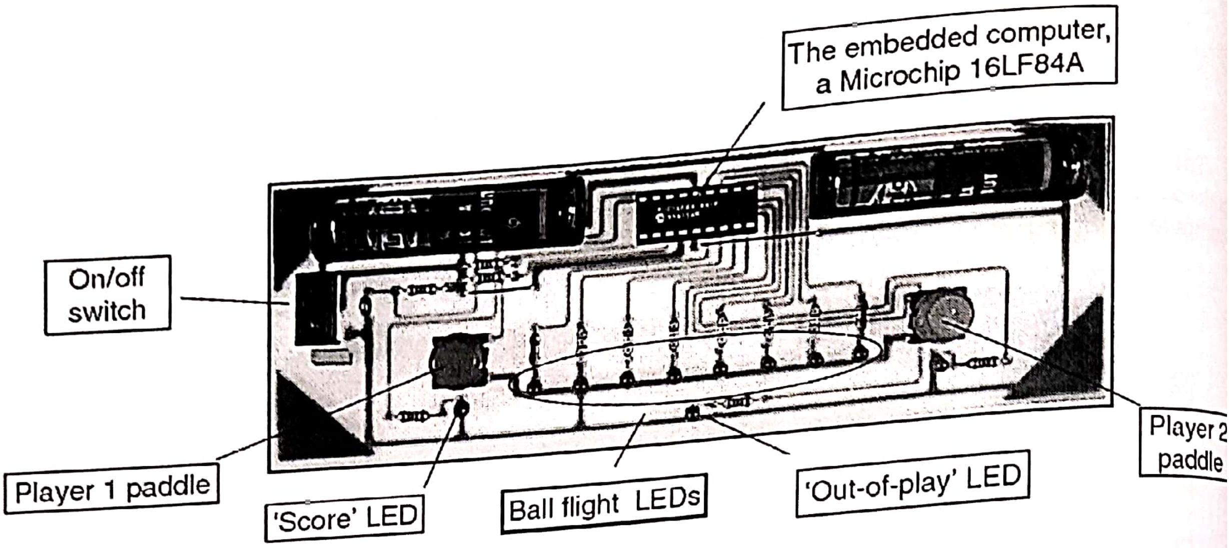
Examples

كمبيوتر داخل السيارة متصل
بكثير من الـ Sensors
لتقديم خدمات مختلفة



- Different sensors in the car door produce signals that are of great importance when integrated with the rest of the car functionality

Examples

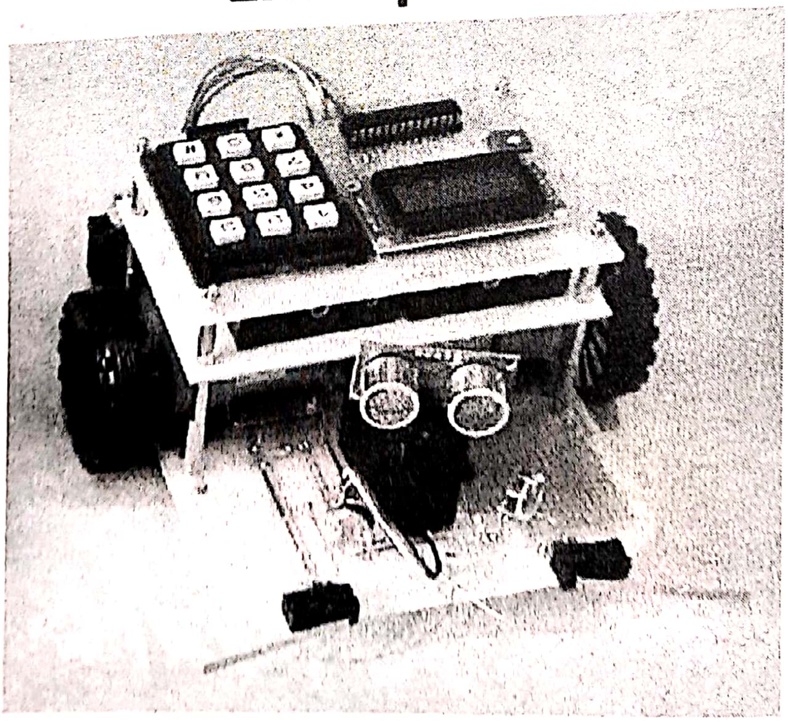


- The Electronic 'ping-pong'

لعبة تنس الطاولة

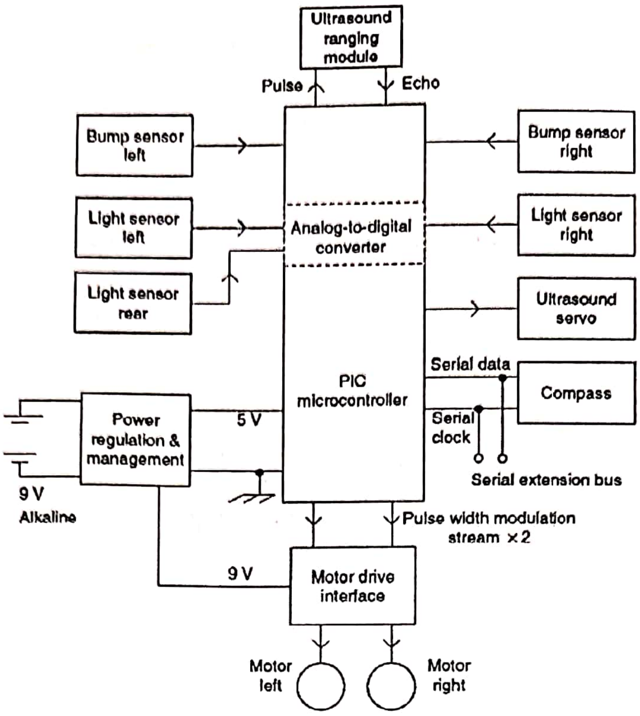
Examples

روبوت بمشي
و بشوف لوحيخبط
باشي وهيك .



- The Derbot Autonomous Guided Vehicle
- More sensors and powerful microcontroller

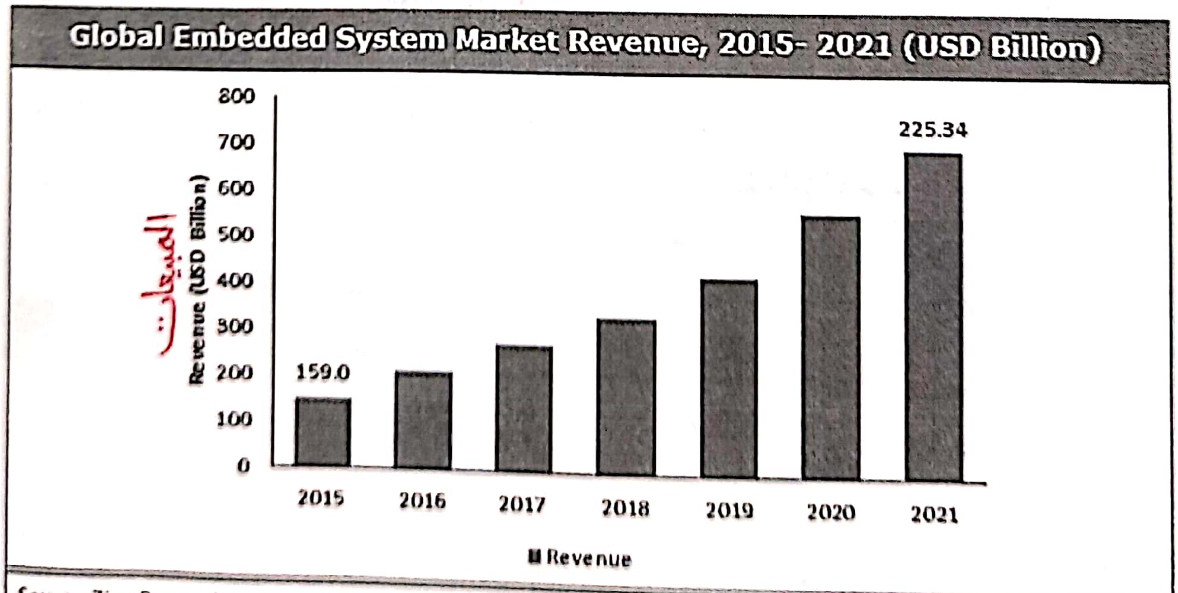
Examples



- The Derbot Autonomous Guided Vehicle

Embedded Systems Market

↳ موجودة في كل المجالات

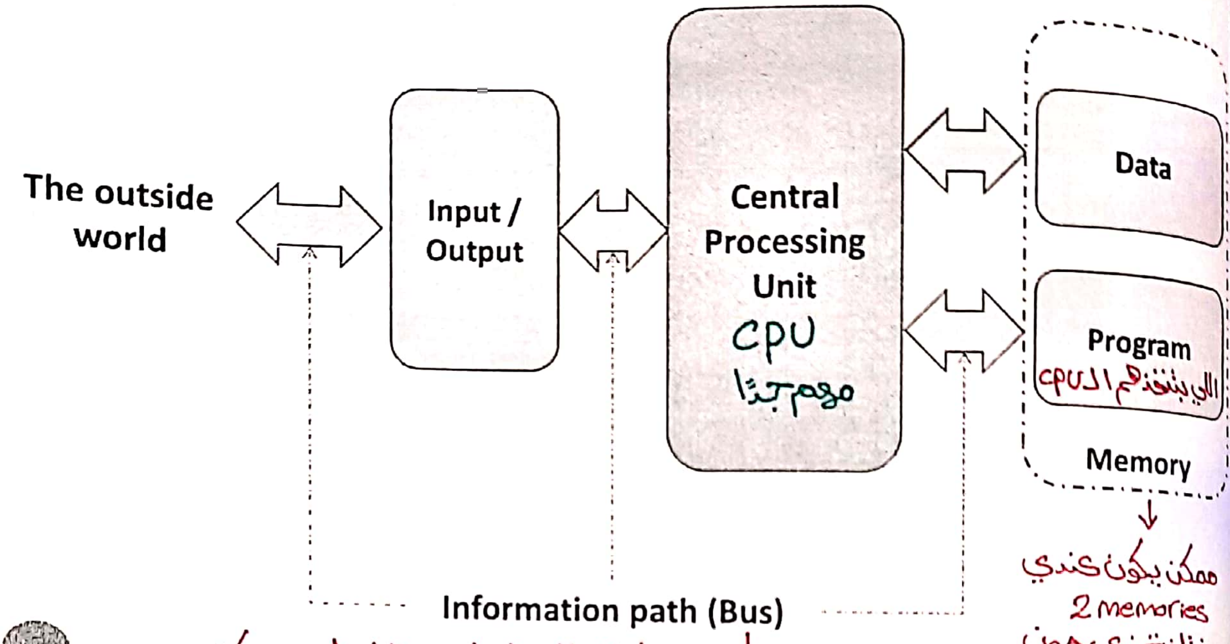


Source: Zion Research Analysis 2016

" مفاهيم نحتاجها "

Some Computer Essentials

- **Elements of a Computer**



ممكن يكون عندي
2 memories
مختلفة زي هون
مش شرط وحدة .

← بتربط العالم الخارجي بالداخلي وممكن
يحمل data أو address أو control

2 wires تحمل
بالعادة
read write

Some Computer Essentials

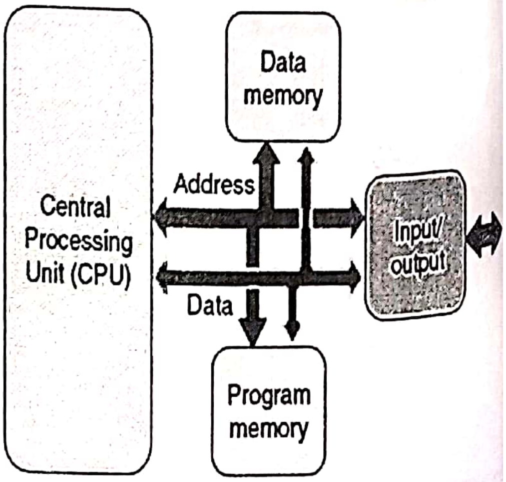
(**Memory Organization**)

① **The Von Neumann Architecture** → هي الطريقة الشائعة

- One address bus and one data bus → بتشاركوا كلهم بنفس الـ bus
- I/O may be also connected to these busses
- Simple and logical architecture, however / رخيص
  - Same memory width ① for instruction and data ?!
  - Shared busses ?! ②

المشكلة الثانية ← إنه دايما نفس الـ width وهيك بصير تقييد لأنه الـ width الـ component لازم يساوي width الـ bus.

مدامه مشترك
ممنوع أكثر من 2 يستخدمه فما بسمح
لـ other components تدخل بينهم .



مشكلتها : اللي بده يستخدم الـ Bus
وبلاقيه مشغول لازم يستنى فحبصير
ازدحام Bus contention
وحبصير في Delay

Some Computer Essentials

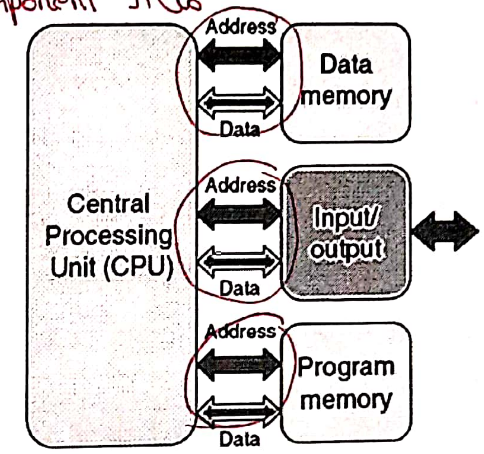
Memory Organization

② The Harvard Architecture

- Separate address and data bus for program memory and data memory بتخصص مجموعة من الBusses لمجموعة من ال Component
- More flexibility; (ما بفيدني)
  - Different memory width
  - Simultaneous access of data and program memories
- Complex ?!

مشكلته غالي ومعقد لكن

بخفف من ال Shared bus



الMicrocontrollers اللي حنشوفهم حيكونوا غالبًا hybrid يعني بجمعوا الطريقتين.

Some Computer Essentials

Instruction Sets

- Every CPU has a set of instructions that it can recognize and execute
- There are different approaches in designing instructions for the CPU in attempt to speed up program execution
  - CISC (Complex Instruction Set Computers) → حاول تحط أكبر عدد ونوع من الinstructions
    - Many instructions and addressing modes
    - Instructions have different levels of complexity (different size and execution time) »الهدف يسهلوا البرمجة للمستخدم«
    - Relatively slow بطيئ نسبيا
    - Shorter programs → لأنه في complex instruction
  - RISC (Reduced Instruction Set Computers) → أقل وأبسط عدد من ال instruction بحجم محدد من أجل تقليل التكلفة.
    - Few instructions and addressing modes
    - Simple instructions of fixed size
    - Relatively fast سريع نسبيًا
    - Longer programs

Ex: / Intel AMD

Ex: MIPS / ARM

Some Computer Essentials

- **Memory Types**
  - ① **Volatile** → متطاير
    - Holds its contents as long as power is ON زي RAM
    - Used as temporary *storage to hold data* مؤقت دا
    - Easy to write
    - RAM

مجرد ما أطفي ال Power بتروح ال data

* ال Program في ال Embedded تخزن على Non-Volatile Memory.

  - ② **Non-volatile** → غير متلاشي
    - Retains its values on power out
    - More difficult to write in terms of time and power
    - In embedded systems, it is *usually used to store programs*
    - ROM / FLASH / EEPROM

لما أطفي ال system بتفضل موجودة ال data

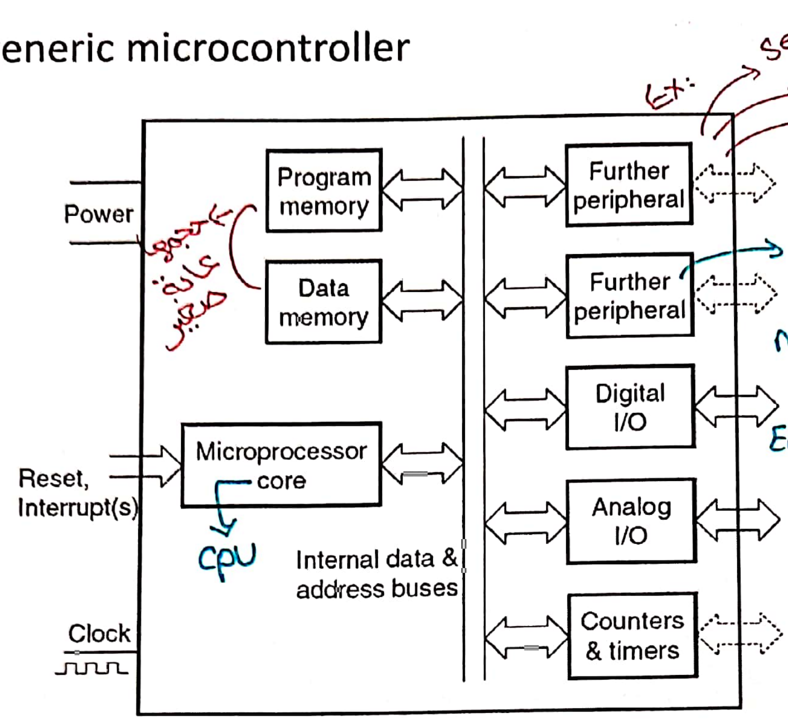
هي Microprocessors بس بقدرات أقل (أبسط) ← يفضل استخدامها في تصنيع وبناء ال Embedded

Microprocessors and Microcontrollers

- **First microprocessors in the 1970s**
  - The computer CPU on a single chip
  - Initially, memory and I/O interfacing outside the CPU
  - As technology evolved, the microprocessor became more self-contained, powerful, and faster → خاصة في صناعة ال transistor
- **A special category of microprocessors emerged**
  - Microcontrollers لتسهل على بناء ال Embedded System
  - Intended for control purposes → مخصص لأغراض التحكم
  - *No high computational power, huge memories, or high speed is required*
  - *Has excellent I/O capabilities* → input/output أحط كتير
  - Small, low cost, and self contained → يحتوي على additional components داخلها

Microprocessors and Microcontrollers

- A generic microcontroller

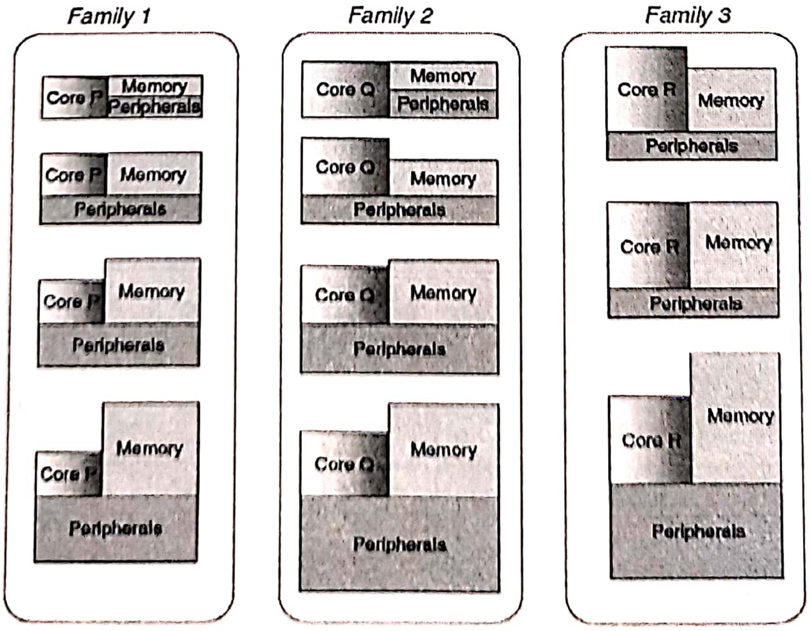


→ devices اضافية داخل الـ Microcontrollers لتسهل علي بناء الـ Embedded System

---

Microprocessors and Microcontrollers

- Microcontroller Families
  كل Family فيها مجموعة من الموديلات بخصائص نفسها مشتركة
  - Different families with <u>*each family built around the same core*</u> → CPU نفس
  - Family members differ in *memory size* and *peripheral capabilities* →



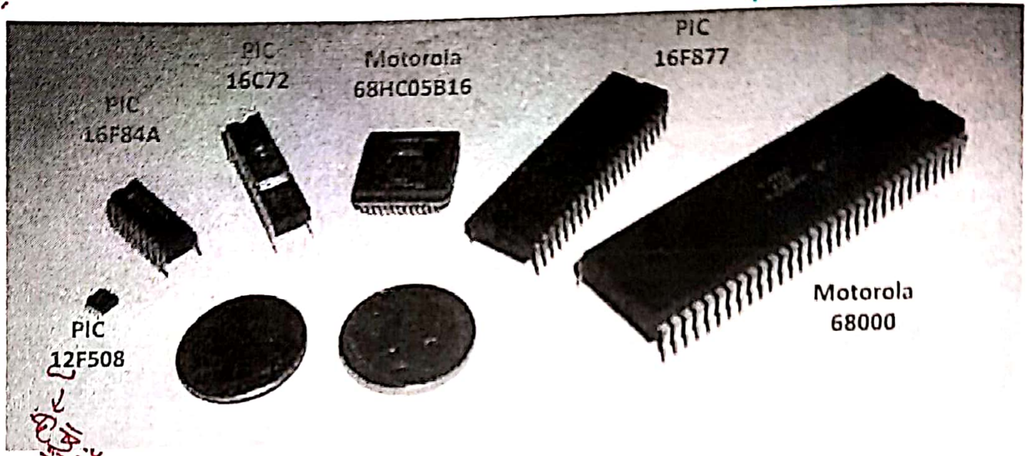
ممكن يختلف حجم الـ Memory أو عدد الـ Peripheral capabilities

Microprocessors and Microcontrollers

- Microcontroller Packaging
  - Plastic packaging
  - Pins for I/O, clock, communication, and Power.
  - The number of pins usually determines يحدد the size of the chip

Interface بين ال Circuit اللي داخل ال microcontroller وبين العالم الخارجي.

لـ كل ما زاد عدد ال Pins معناها قدرة ال input / output أعلى غالبًا.



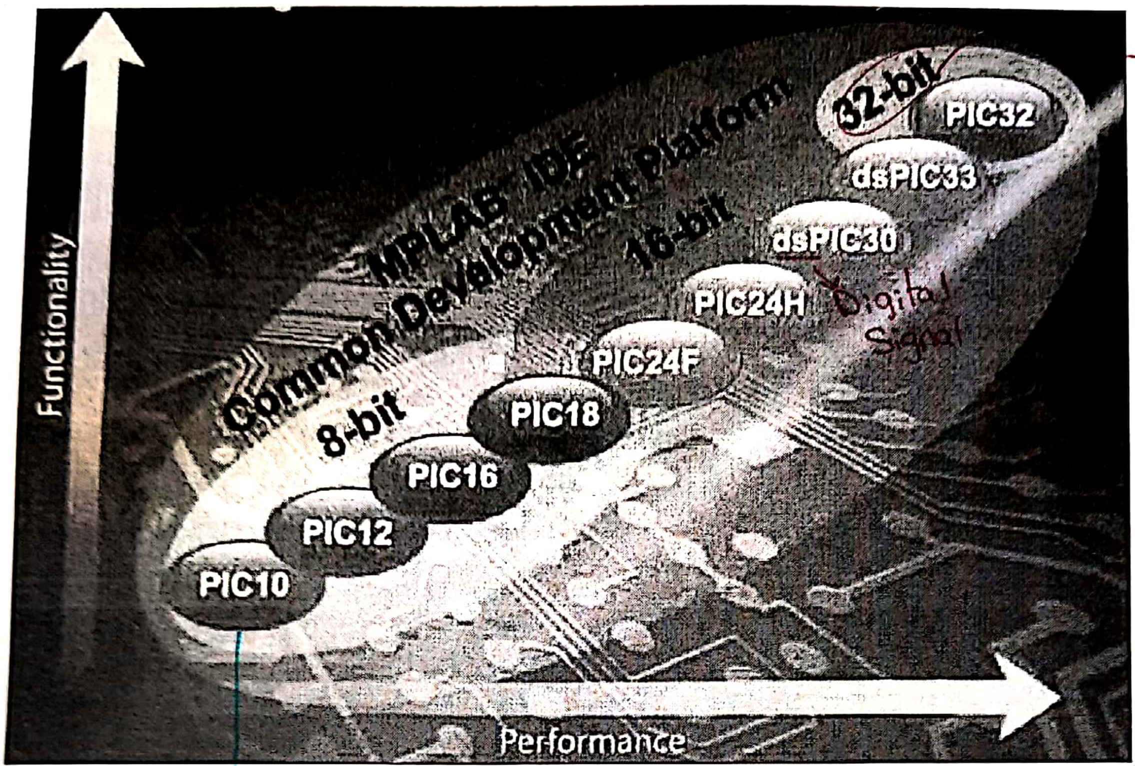
أصغر الأشكال غالبًا

شركة Microchip and the PIC Microcontrollers

- Peripheral Interface Controller (PIC) was originally a design by General Instruments intended for simple control applications
- In the late 1970s, GI introduced PIC® 1650 and 1655
  - Standalone design self contain
  - RISC with 30 instructions
  - Single working register (accumulator)
  - Many attractive features
- PIC was sold to Microchip

تم بيعو إلى

Microchip and the PIC Microcontrollers



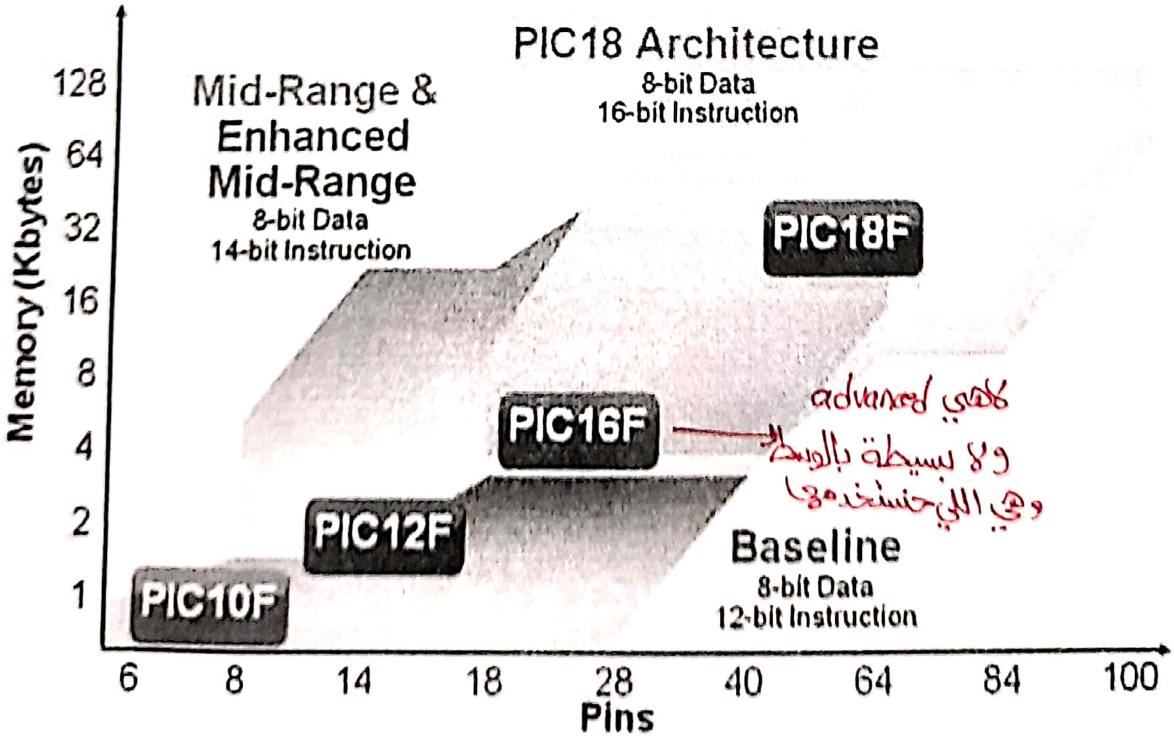
Digital Signal

حجم ال data الي بتقدر ال CPU تتعامل معها (data width

لا لو بدي أجمع رقمين لازم حجم الرقم يكون 32- bit وهكذا.

كل وحدة عبارة عن عائلات من ال Microcontrollers الي بتصنعها ال Microprocessor .

Microchip and the PIC Microcontrollers



لا هي advanced ولا بسيطة بالوسط وهي الي حنستخدمها

Microchip and the PIC Microcontrollers

- **PIC Families**

PIC Family	Stack Size (words)	Instruction Word Size	No. of Instructions	Interrupt Vectors
12CX/12FX	2	12- or 14-bit	33	None
16C5X/16F5X	2	12-bit	33	None
16CX/16FX	8	14-bit	35	1
17CX	16	16-bit	58	4
18CX/18FX	32	16-bit	75	2

رقم ال model

سنركز على 16F بال course

حرف C يعني

- Example: the 16C84 was the first of its kind built using CMOS technology. It was later reissued as 16F84A incorporating flash memory and other technological features

الحروف تعني عن مميزات في هذا النوع.

Microchip and the PIC Microcontrollers

- **PIC Characteristics**
  - Low-cost
  - Self-contained
  - 8-bit
  - Harvard architecture
  - RISC → عدد inst قليل
  - Pipelined
  - Single accumulator (the working or W register)
  - Fixed reset and interrupt vectors

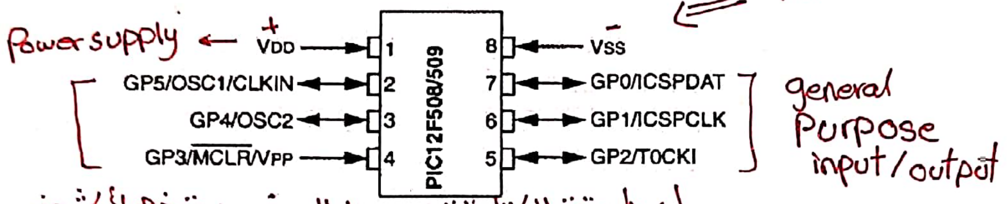
مثال سريع

The PIC 12 Series

- PIC 12F508/509 → أصغر وأبسط Pic microcontrollers
- The smallest and simplest PIC

Very Small

Pinout



لـ طريقة الكتابة تعني هذا الـ pin يستخدم لأكثر من وظيفة

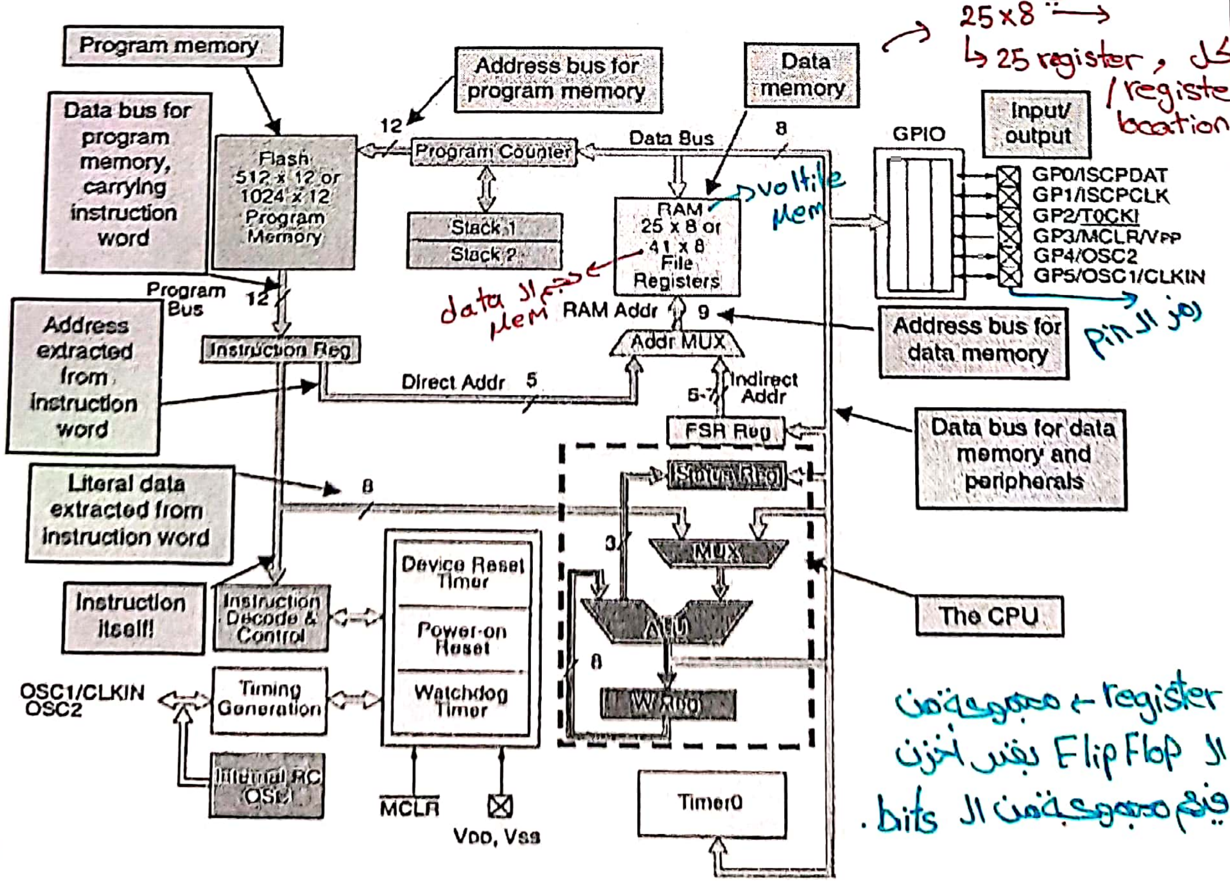
Key

V_{DD}:	Power supply	V_{SS}:	Ground
V_{PP}:	Programming voltage input	MCLR:	Master clear
OSC1, OSC2:	Oscillator pins	CLKIN:	External clock input
GP0 to GP5:	General-Purpose input/output pins (bidirectional except GP3)		
CSPDAT:	In-Circuit Serial Programming™ data pin.		
CSPCLK:	In-Circuit Serial Programming™ clock pin.		

طريقة التوصيل بين الـ harvard و الـ Von Neumann

ولكن مبدايا هو harvard .

The PIC 12 Series Architecture



25x8 تعني →
↳ 25 register , وكل register 8 bits / location

register ← مجموعة من الـ Flip Flop بنفس تخزن فيه مجموعة من الـ bits .

Summary

- *An embedded system has one or more computers embedded within it that perform control operations*
- *A microcontroller is at the heart of embedded systems. It is basically a microprocessor with extended I/O capabilities*
- *Microchip is one of the popular vendors for a large variety of microcontrollers with different features*

Introducing the PIC 16 Series and the 16F84A

Chapter 2
Sections 1-8

Dr. Iyad Jafar

Outline

- Overview of the PIC 16 Series
- An Architecture Overview of the 16F84A
- The 16F84A Memory Organization
- Memory Addressing
- Some Issues of Timing
- Power-up and Reset
- The 16F84A On-chip Reset Circuit

Overview of the PIC 16 Series

- The PIC 16 series is classified as a mid range microcontroller

لع مش بسيطة ولا Advanced
وسط.

- The series has different members all built around the same core and instruction set, but with different memory, I/O features, and package size

Some members of the PIC 16 Series family

Device number	No. of pins*	Clock speed	Memory (K = Kbytes, i.e. 1024 bytes)	Peripherals/special featur
16F84A	18	DC to 20 MHz	1K program memory, 68 bytes RAM, 64 bytes EEPROM	1 8-bit timer 1 5-bit parallel port 1 8-bit parallel port
16LF84A	As above	As above	As above	As above, with extended supply voltage range
16F84A-04	As above	DC to 4 MHz	As above	As above
16F873A	28	DC to 20 MHz	4K program memory 192 bytes RAM, 128 bytes EEPROM	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 5 10-bit ADC channels, 2 analog comparators
16F874A	40	DC to 20 MHz	4K program memory 192 bytes RAM, 128 bytes EEPROM	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 8 10-bit ADC channels, 2 analog comparators
16F876A	28	DC to 20 MHz	8K program memory 368 bytes RAM, 256 bytes EEPROM	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 5 10-bit ADC channels, 2 analog comparators
16F877A	40	DC to 20 MHz	8K program memory 368 bytes RAM, 256 bytes EEPROM	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM modules, 2 serial communication modules, 8 10-bit ADC channels, 2 analog comparators

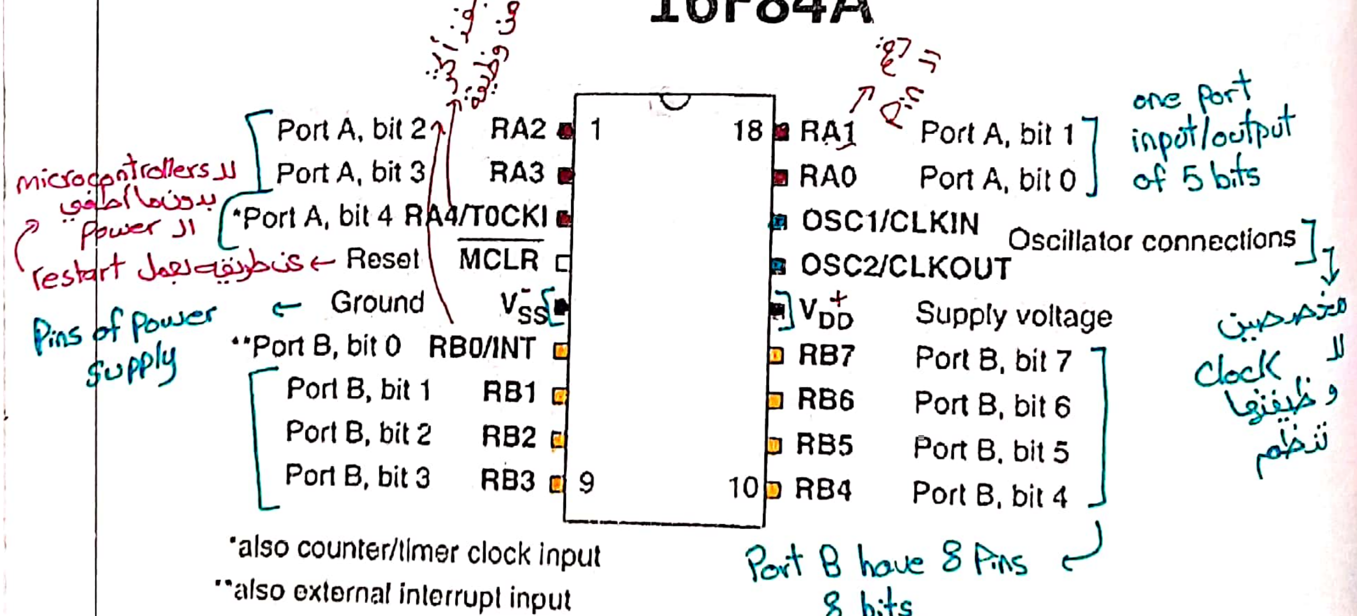
*For DIP package only.

حنركز عهاد الموديل

بتشتغل ع Voltage أعلى

2 I/o Port

An Architecture Overview of the 16F84A



نفس الـ Pin تنفذ أكثر من وظيفة

رقم الـ Pin

one port input/output of 5 bits

microcontrollers للـ بدون ما اطفي الـ Power

طريقة لعمل restart

Pins of Power Supply

مخصصين للـ clock وظيفتها تنظم

Port B have 8 Pins 8 bits

*also counter/timer clock input

**also external interrupt input

- 18 Pins / DC to 20MHz / 1K program Memory/ 68 Bytes of RAM / 64 Bytes of EEPROM / 1 8-bit Timer / 1 5-bit Parallel Port / 1 8-bit Parallel Port

1024 x 14

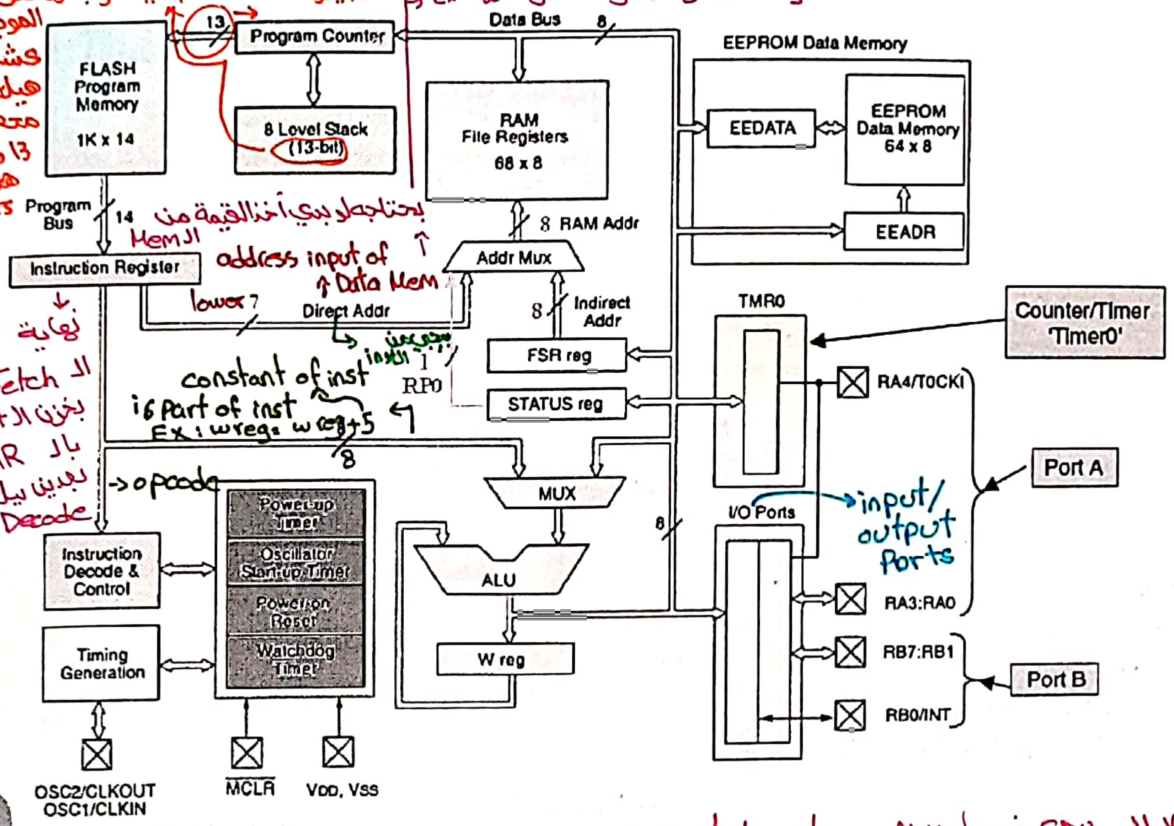
Flash

Port A

Port B

لازم نشوف شكلها من جوا عشان نعرف كيف نبرمجها .

An Architecture Overview of the 16F84A



لازم يكون ١٠ ولكن الرقم اللي مكتوب هون لكل الموديل عشان هيك محطوط 13 و لا هو ١٠ bits

→ Ex: w reg= w reg + M [address]

بحتاجها بس بدي أخذ القيمة من ال Mem

address input of Data Mem

lower 7

بيجي من ال inst

constant of inst is part of inst
Ex: wreg= w reg+5

نهاية ال Fetch بخزن ال inst بال IR

بعدين بيلش Decode

-> opcode

→ input/output Ports

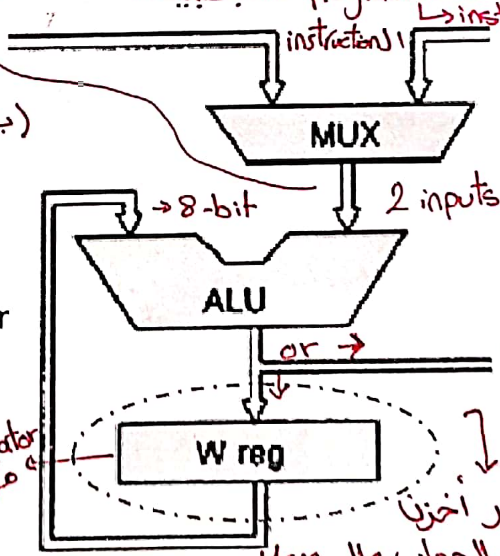
* ال Cpu فيها ALU و Control unit *

→ Arithmetic and logic unit

The PIC 16F84A ALU and Working Register

الهاردوير المسؤول عن إجراء العمليات الحسابية والمنطقية .

- *Arithmetic & Logic Unit*
  - 8-bit ALU
  - Supports 35 simple instructions (بال RISK) ← نوع
  - Input operands are
    - The working register
    - Content of some file register or a literal
  - The result is stored in Working register or in a File register
- *The Working Register*
  - Inside the CPU
  - For many instructions, it can be chosen to hold the result of the last instruction executed by the CPU



→ File registers

بيجي من ال Data memory أو ال Program Mem حسب طبيعة ال instruction

→ instructions register

→ 8-bit

2 inputs

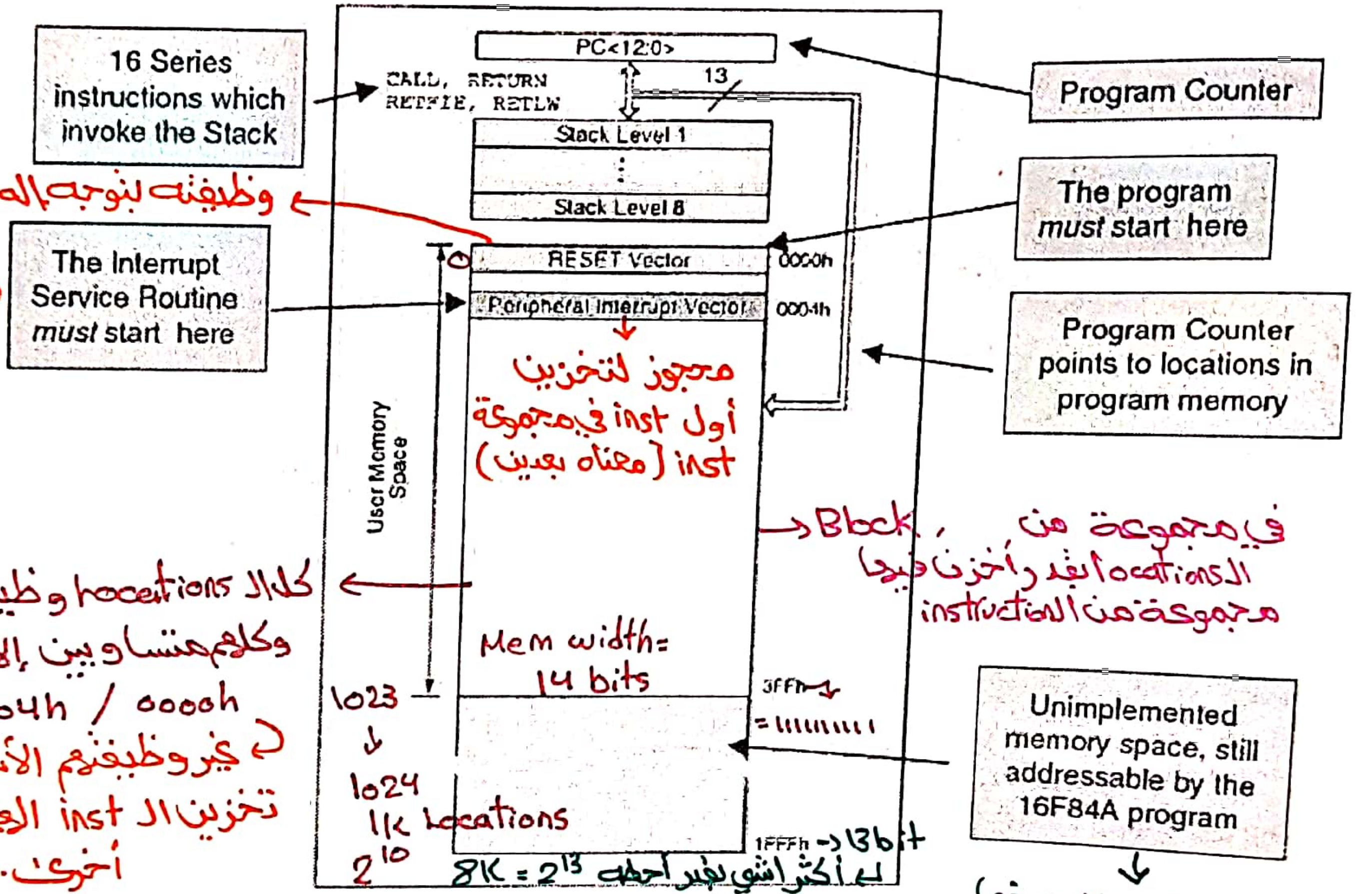
or → Data Memory

Accumulator متراكم

بقدر أخزن هاد الجواب عال Mem أو W reg ← بقدر أستخدمه لاحقاً وهي دائماً ال input لل ALU

The PIC 16F84A Memory Organization

- *Program Memory and Related Units*



16 Series instructions which invoke the Stack

CALL, RETURN RETFIE, RETLW

PC<12:0>

13

Stack Level 1

Stack Level 8

Program Counter

The program *must* start here

RESET Vector 0000h

The Interrupt Service Routine *must* start here

Peripheral Interrupt Vector 0004h

Program Counter points to locations in program memory

User Memory Space

3FFh

1FFFh

Unimplemented memory space, still addressable by the 16F84A program

وظيفته بتوجه الـ microcontroller أول ما يشتغل البرنامج ولازم نبلش فيه .

0

محجوز لتخزين أول inst في مجموعة inst (معناه بعين)

→ Block , في مجموعة من الـ locations بقدر أخزن فيها مجموعة من الـ instruction

كل الـ locations وظيفتهم يخزنوا inst وكلهم متساوين إلا 2 locations :

0000h / 0004h (special)

← غير وظيفتهم الأساسية اللي هي تخزين الـ inst الهم وظائف أخرى .

Mem width= 14 bits

1023

↓

1024

1K locations

2^{10}

3FFh = 1111111111

1FFFh → 13 bit

لـ أكثر اشي بقدر أحطه $8K = 2^{13}$

↓

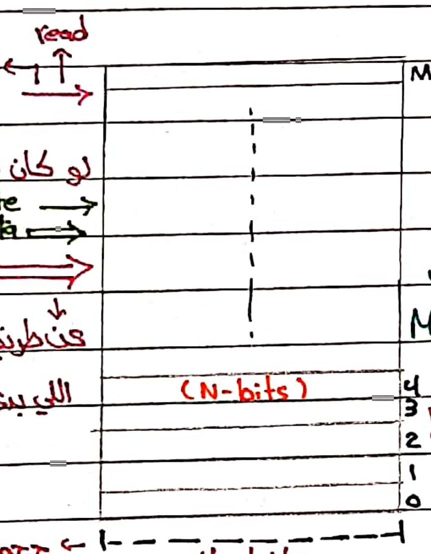
ما فيه Memory هدفها

إنهم ما بيسووا أكثر من تصميم لكل موديل بسووا تصميم واحد حسب أكبر حالة وبستخدموا اللي بدهم ياه للحالات الأصغر فبكون هاي المساحة غير مشغلة في الموديل تاعنا .

Chapter 2 :

Slide 8 :

Memory : كأنها Array فيها مجموعة من ال Locations
وكل واحد من هذه ال Locations بقدر يخزن مجموعة من ال bits.



input بقلي في read ولا write وهو 1-bit، لو كان 0 يعني ما بسوي read

write data ← لو حكتب عال Mem

Address input عن طريقه بعطي رقم ال location اللي بدي أحكي معاه.

*هاي ال Memory حتى أقدر أستفيد منها كـ Processor لازم يكون عندي قدرة أسوي عليها عمليتين رئيسيات اللي هم عملية القراءة من ال Memory والكتابة على ال Memory

* (Load, read/write, store)

read data ← output بطلعلي ال data اللي قريتها

0 → رقم ال location بنسميه : Memory Address

N-bits (memory width) ← حجم ال data اللي تخزن في ال location.

*لو ال cpu بها تقرأ من ال Memory لازم : 1- لازم تحدد رقم ال location اللي بدها تقرأ منه. 2- تخبر ال Memory إنها بدها تسوي read operation. 3- ال Mem تفهم الكلام وتتجاوب مع الطلب وتروح لل location المطلوب وتقرأ ال content تجبه وترجعه لل cpu

وهاي الخطوات حتى تصير لازم يكون في interface بين ال Memory والعالم الخارجي وال cpu

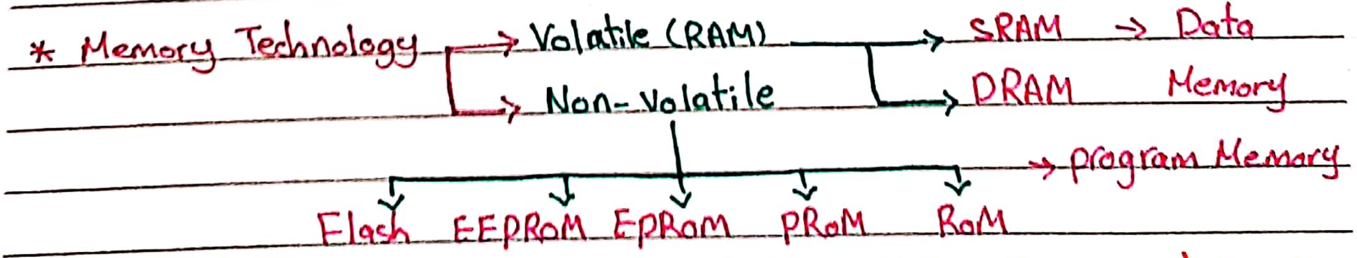
*لو ال cpu بدها تكتب عال Memory : 1- لازم أحدد رقم ال location. 2- لازم أخبر ال Memory إنه بدي أسوي write.

*في معظم ال Memories ال data write/read عادة ما بكونوا منفصلات " ⟷ "

*ال Memory تحتوي M Locations و حجم ال location = N ، Ex: 40 x 8
↓ ↓
M N

*حجم ال address input = $\lceil \log_2 M \rceil = n$

*حجم ال output = N bits



*ال Microcontrollers اللي حنشتغل معها فيها : SRAM / EEPROM / Flash

The PIC 16F84A Memory Organization

- *Program Memory*
  - *1K x 14 Bits* ← Location
  - *Address range 0000H – 03FFH*
  - *Flash (nonvolatile)* → سريعة / density عالية
  - *10000 erase/write cycles* → without any errors → أكثر من هيك ممكن تعمل بس مش مضمون من ال errors.
  - *Location 0000H is reserved for the reset vector*
  - *Location 0004H is reserved for the Interrupt Vector*
- **Program Counter** → reg /IP(Instruction Pointer)
  يخزن قيمة معينة
  - *Holds the address of the instruction to be executed (next instruction)*
- *Stack* → Small memory → addesses بخزن مؤقت، مربوط في ال PC → يخزن ال value في PC بشكل مؤقت
  - *8 levels (each is 13 bits)*
  - *SRAM (volatile)*
  - *Used to store/load the return address with instruction like CALL, RETURN,RETFIE, and RETLW (interrupts and subroutines)*
- ***Instruction Register***
  - *Holds the instruction being executed* → يخزن ال Inst اللي قاعد بنفذها الآن.

سريعة
↑
SRAM غالبًا
↑
Volatile ←
كشان لو طفي ال system يرجعلي من البداية مش يكمل من حد ماوقف.

Volatile/
SRAM

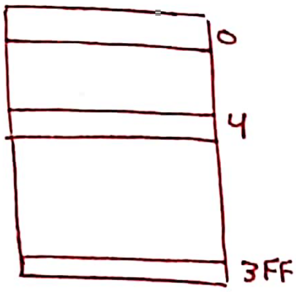
Volatile ←
↓
غالبًا
SRAM

The PIC 16F84A Memory Organization

- ***The Configuration Word*** تكوين →
  انا بس أغير ال Value الموجودة في ال configuration word لازم أرجع للكومبيوتر وأنزل الكود بعدين مرة ثانية على ال Microcontroller.
  - A special part of the program memory
  - Allows the user to configure different features of the microcontroller at the time of program download and is not accessible within the program or while it is running → مابقدر أقراها أو أعدلها أو أوصلها أثناء تنفيذ البرنامج.

R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	$\overline{\text{PWRTE}}$	WDTE	FOSC1	FOSC0
bit13													bit0

bit 13-4	CP: Code Protection bit 1 = Code protection disabled 0 = All program memory is code protected → ماحد بقدر يقراه
bit 3	$\overline{\text{PWRTE}}$: Power-up Timer Enable bit 1 = Power-up Timer is disabled 0 = Power-up Timer is enabled
bit 2	WDTE: Watchdog Timer Enable bit 1 = WDT enabled 0 = WDT disabled
bit 1-0	FOSC1:FOSC0: Oscillator Selection bits ← نوع ال circuit اللي بنعطيني ال Clock 11 = RC oscillator → RC Circuits 10 = HS oscillator 01 = XT oscillator 00 = LP oscillator (10, 01, 00) Cristals



هاي اللي بشوفها و لكن في بعد آخر location

اللي هي 1FFF كملها locations ولكن أنا ك user

ما بقدر استخدمهم لأخزن ال data و لكن وظيفتهم : 2007H ← تخزن ال configuration word

وظيفتها بقدر أخزن فيها معلومات لتحكم بميزات مختلفة لل microcontrollers

وفي locations مستخدمة للتواصل مع الجهاز المحمول (الكومبيوتر)

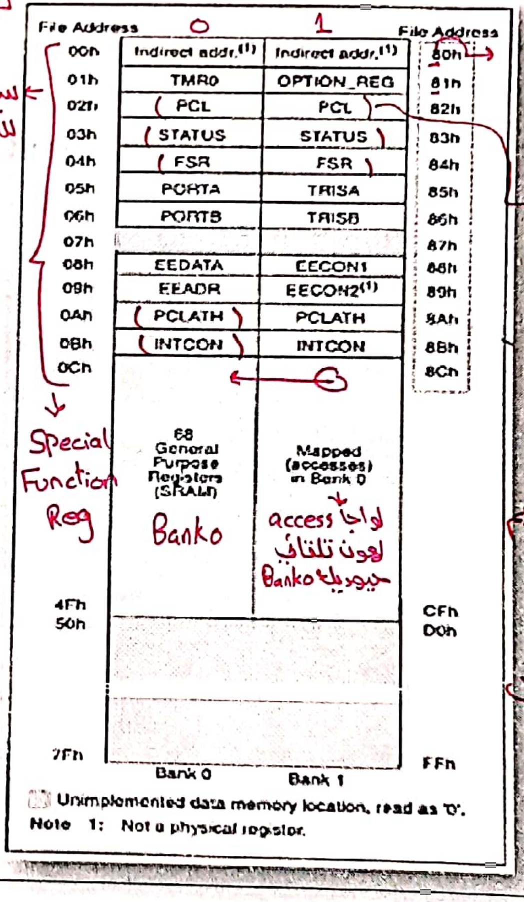
The PIC 16F84A Memory Organization

(File Registers / RAM / Data Memory) ↲

Data Memory and Special Function Registers (SFRs)

مستخدمين للتحكم في ال Microcontroller.

- *SRAM (volatile)*
- *Banked addressing* → مشروحة بالدفتر
- ***Special Function Registers SFRs***
  - *Locations 01H-0BH in bank 0 and 81H-8BH in bank 1*
  - *Used to communicate with I/O and control the microcontroller operation*
  - *Some of them hold I/O data*
- ***General Purpose Registers***
  - *Addresses 0CH – 4FH (68 Bytes)*
  - *Used for storing general data*



File Address	0	1	File Address
00h	Indirect addr.(1)	Indirect addr.(1)	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h			87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2(1)	89h
0Ah	PCLATH	PCLATH	8Ah
0Bh	INTCON	INTCON	8Bh
0Ch	68 General Purpose Registers (SRAM) Bank0	Mapped (accesses) in Bank 0 لو اجا access لهون تلقائي حيوديك Bank0	8Ch
4Fh			CFh
50h			D0h
7Fh			FFh
	Bank 0	Bank 1	

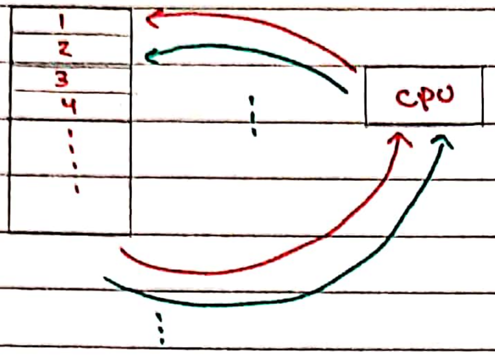
▨ Unimplemented data memory location, read as '0'.

Note 1: Not a physical register.

1 000 0000 ↳ Most sig bit → بحدد باي Bank

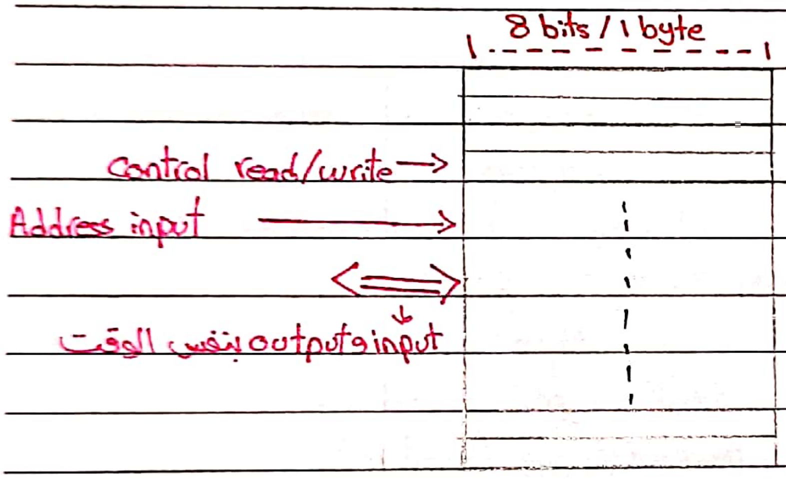
الفكرة هون انه ال PCL فعليا موجود ب Bank0 بس ممكن توصلها ب 02h أو 82h يعني بقدر أوصلها بغض النظر عن رقم ال Bank ومعمول هيك لأنه مستخدمين بكثرة.

Slide 9:



الخطوات الرئيسية لتنفيذ الـ inst: ←
Fetch, Decode, Execute
بستخلص الـ cpu تنفيذهم لازم تكون متذكرة وين وصلت بالبرنامج وهذا ما ينفذه الـ Program counter.

Slide 11: Data Memory / File Registers / RAM.



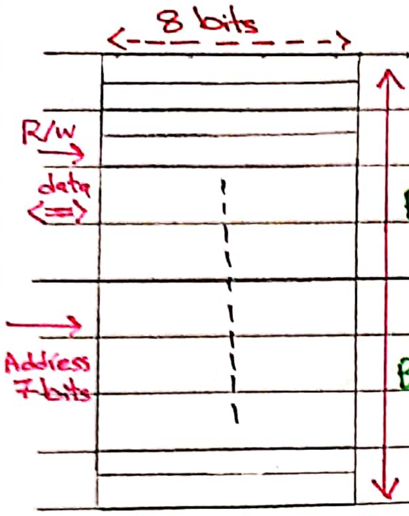
*هاي الـ Mem مخصصة لتخزين الـ Data اللي أنا بتعامل معها خلال تنفيذ البرنامج.

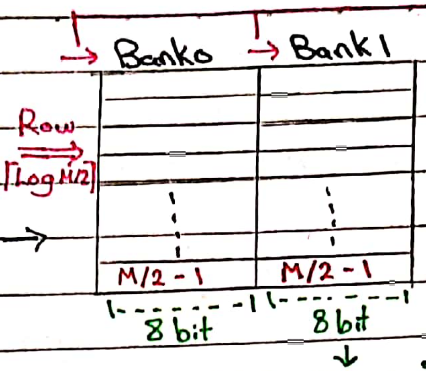
68 Registers/Location ↓

هو بالأصل أكثر من 68 ولكن هذا الرقم بيسمى GPR (general porpose Register) أنا كـ user بقدر أستخدمهم لأخزن شو ما بدي وفي 22 Register آخرين يسموا SFR (Special Function Registers) بقدر أتعامل معهم وأقرأ وأكتب عليهم بس الـ Value اللي بيحتويها مميزة والها معنى والها وظيفة في الـ microcontroller. إذاً الإجمالي 90 registers تقريباً.

*بما إنهم 90 registers فالـ Address input $= 7\lceil \log_2 90 \rceil$. (2^7)

∴ Address input = 7 bits





Log # of Banks
↓
فمعنى عندي 2 Banks
يحتاج 1 bit

<≠> 8 bit

مستقلين عن بعض ←

بالأصل يحتاج 7 bit بس عشان التصميم شامل كل ال models فبكون 8 bit .

68 register GPR

22 register SFR

* ال PIC 16 بتستخدم طريقة شوي مختلفة لترتيب ال locations في ال Data Memory

* الطريقة هي تقسيم ال locations الموجودة إلى مجموعات وهذه المجموعات يتم ترتيبها بطريقة 2D وكل مجموعة تسمى Bank تحتوي نصف ال locations التي كانت .

* ال Address بهذه الطريقة أصبح مختلف قليلاً فيجب إدخال رقم الصف ورقم العمود .

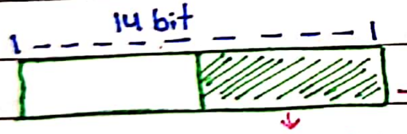
* عدد ال bits اللي احتجنا في ال 1D هم نفس عدد ال bits اللي احتجناهم في ال 2D ⇐

* one Bank ⇒ $\# bits = Log_2 M$

* Two Banks ⇒ $\# bits = Log_2 M/Banks + Log_2 Banks$

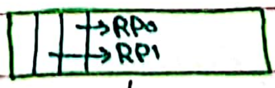
* الهدف من تصميمها بهذا الشكل هو : له علاقة بتصميم ال instructions .

* في ال PIC Microcontrollers حجم ال Instruction ثابت ، جزء منهم يستخدم ك Memory Address إذا كانت ال instructions بتحكي مع ال Memory



* ولكن كان تصميم ال Memory لكامل العائلة وعدم جعل الجزء المستخدم من ال instruction للMemory Address متغير فقسموا ال Banks بحيث يثبت ال Address field على 7 bits وتخزين جزء من ال address داخل ال instruction وما تبقى من ال address يخزن في مكان آخر ال User بتحكم فيه :

Register فيه مجموعة من ال bits ال لها علاقة في اختيار ال Banks ←



The PIC 16F84A Memory Organization

Special Function Registers (SFRs)

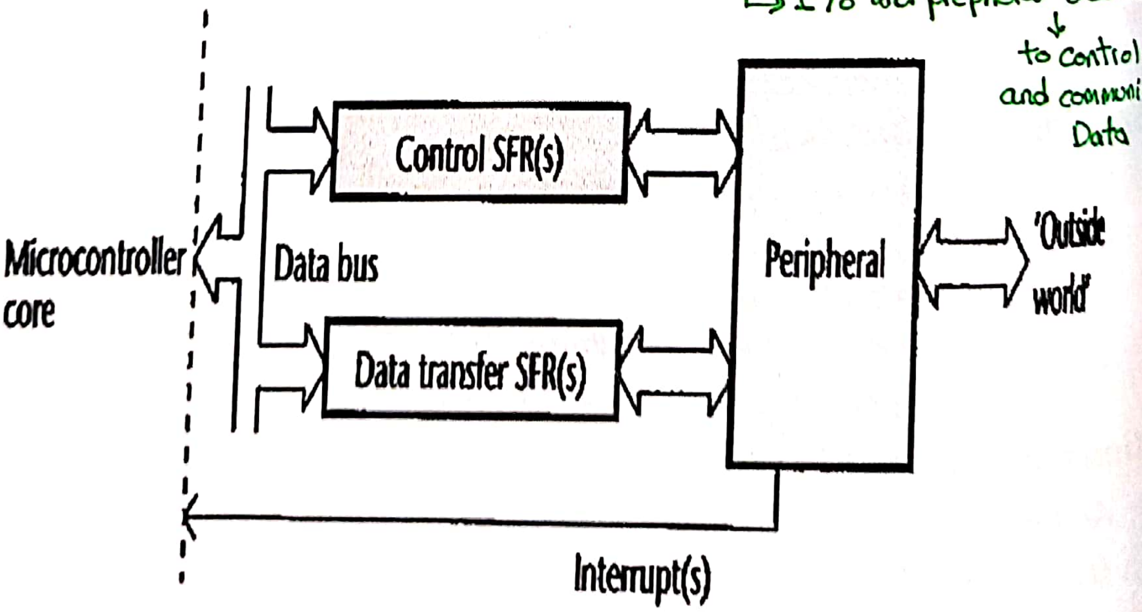
Address	Bank 0	Bank 1	Address
00h	INDF	←	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	←	82h
03h	STATUS	←	83h
04h	FSR	←	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h
07h	Unimplemented	←	87h
08h	EEDATA	EECON1	88h
09h	EEADR	EECON2	89h
0Ah	PCLATH	←	8Ah
0Bh	INTCON	←	8Bh
0Ch - 4Fh	GPR	←	8Ch - CFh

INDF : Data memory contents by indirect addressing
TMR0 : Timer counter
PCL : Low order 8 bits of program counter
STATUS : Flag of calculation result
FSR : Indirect data memory address pointer
PORTA : PORTA DATA I/O
PORTB : PORTB DATA I/O
EEDATA : Dtata for EEPROM
EEADR : Address for EEPROM
PCLATH : Write buffer for upper 5 bits of the program counte
INTCON : Interruption control
OPTIN_REG : Mode set
TRISA : Mode set for PORTA
TRISB : Mode set for PORTB
EECON1 : Control Register for EEPROM
EECON2 : Write protection Register for EEPROM

The PIC 16F84A Memory Organization

- *Special Function Registers (SFRs) interacting with peripherals*

File registers → GPRs
File registers → SFRs
SFRs → MC-related → بتحكم بال operation بشكل عام
SFRs → I/o and prepheral Devices → to control and communicat Data



(العودة لسلايد 6 عليها شرح)

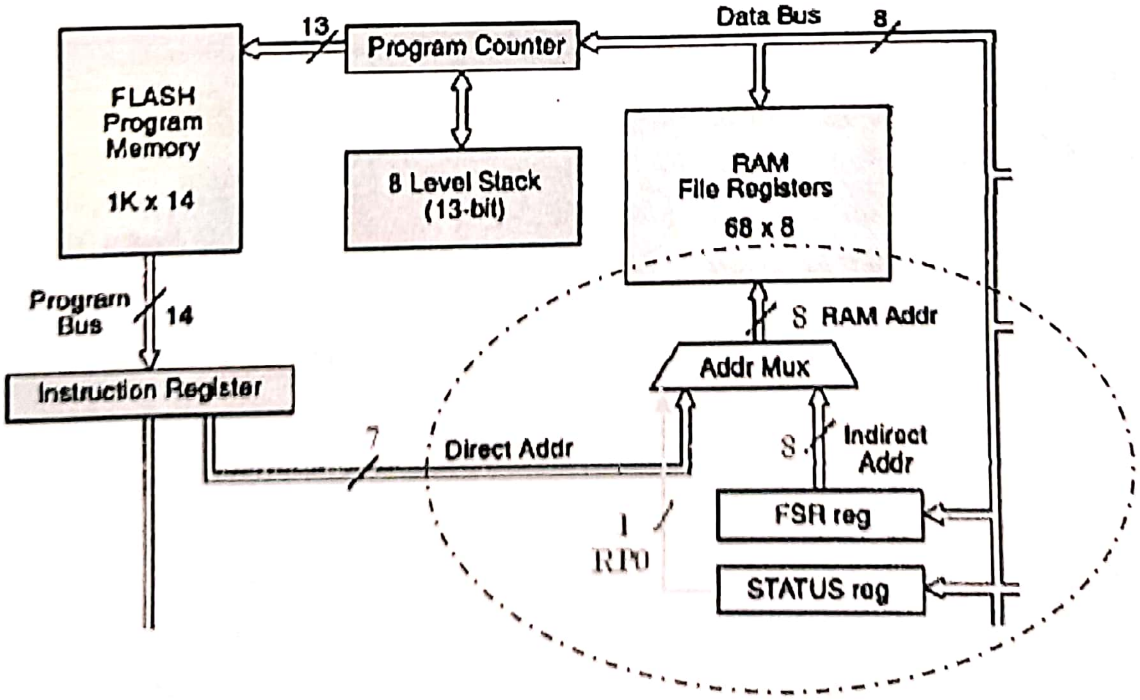
The PIC 16F84A Memory Organization

- *Data Memory Addressing*
  - For PIC 16F84A, the address of any memory location (File Register is 8 bits
    - One bit is used to select the bank
    - Seven bits to select a location in the bank
  - Bank selection is done through using bits 5 and 6 of the STATUS registers (RP0 and RP1)
  - For the 16F84A, only RP0 is needed since we have two banks
  - In general, two forms to address the RAM (File Registers)
    - **Direct addressing –** the 7-bit address is part of the instruction
    - **Indirect addressing**
      - the 7-bit address is loaded in lower 7 bits of the *File Select Register* (FSR , 04H)
      - Bank selection is done using the most significant bit of FSR and the IRP bit in the STATUS register

---

The PIC 16F84A Memory Organization

- *Data Memory Addressing*



لدخل ال register بيستخدموا Microcontrollers لأسوي عمليات مختلفة أو آخذ بعض المعلومات عن ال microcontroller

باله 2 Addresses لأنه موجود بال 2 Banks لأنه مهم. هو فعلياً مو داخل ال Mem ولكنه برسمه جوا لأنه من وجهة نظر ال address إنه جزء منه.

The PIC 16F84A Memory Organization

- ***The STATUS Register (03H, 83H)***

بس هي بنحتاجها لتحديد ال Bank

Bank Selection bit

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{\text{TO}}$	$\overline{\text{PD}}$	Z	DC	C
bit 7							bit 0

↳ indirect

↳ Direct addressing

Flag

ممكن نسميهم Flag أو bit

R/W - x

read write

you can

قيمتها لا bit الأولية لما اشتغل ال Power

bit 7-6 **Unimplemented:** Maintain as '0' → ما بحتاجهم في PIC 164A

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 **$\overline{\text{TO}}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3 **$\overline{\text{PD}}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2 **Z**: Zero bit → هل اخر عملية كانت نتيجتها 0 ؟
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

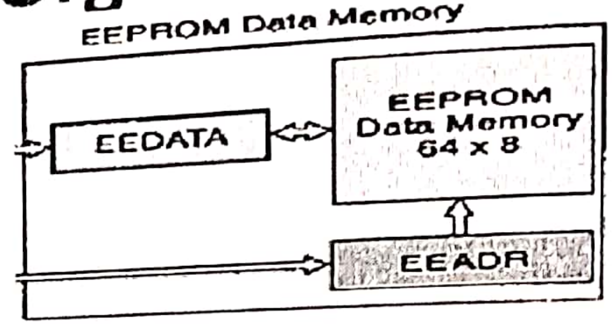
bit 1 **DC**: Digit carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{\text{borrow}}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{\text{borrow}}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

The PIC 16F84A Memory Organization

- ***Data Related***
  - ***EEPROM Data Memory*** ← القراءة والكتابة منها بطيئة بستخدموها للتخزين ال data اللي ما بدي تروح أول ما نطفي ال Power



    - *64 bytes Non-volatile*
    - *10 000 000 erase/write cycles*
    - *Used to store data that is likely to be needed for long term*
    - *Operation is controlled through EEDATA (08H), EEADR (09H), EECON1 (88H), and EECON2 (89H) SFRs* → 4 Registers → (وسيط)
      ↳ بحتاج وسيط مشان ي SRAM
    - ***To read a location***
      - *store the address in EEADR and set the RD bit in EECON1* ↳ Read bit
      - *data is copied to EEDATA register*
    - ***To write to a location*** ← عملية ال write طويلة حماية لل data عشان ما يصير كتابة بالصدفة
      - *data and address are placed in EEDATA and EEADR, respectively*
      - *enable writing by setting the WREN bit in EECON1 SFR* ← لسا ما صير كتابة ↳ write Enable bit
      - *store 55H then AAH in EECON2* ← لسا برضه
      - *commit writing by enabling the WR bit* EECON1 ← هون بلش كتابة
      - *Once the write is done, the EEIF flag is set in EECON1.*

وظيفته يحكيلي هل خلصت عملية الكتابة أو لا

The PIC 16F84A Memory Organization

- ***The EECON1 Register (88H)***

U-0	U-0	U-0	R/W-0	R/W-x	R/W-0	R/S-0	R/S-0
—	—	—	EEIF	WRERR	WREN	WR	RD
bit 7							bit 0

بقرأعل set وبرجع 0 لما تخلص علية الكتابة.

معناها set يعني أنا Programmers بقرأعل set يعني أسوي قيمته 1 وال System بيرجع قيمته لصفر لما تخلص علية القراءة.

bit 7-5 **Unimplemented:** Read as '0'

bit 4 **EEIF:** EEPROM Write Operation Interrupt Flag bit
1 = The write operation completed (must be cleared in software)
0 = The write operation is not complete or has not been started

bit 3 **WRERR:** EEPROM Error Flag bit
1 = A write operation is prematurely terminated
(any $\overline{\text{MCLR}}$ Reset or any WDT Reset during normal operation)
0 = The write operation completed

bit 2 **WREN:** EEPROM Write Enable bit
1 = Allows write cycles
0 = Inhibits write to the EEPROM

bit 1 **WR:** Write Control bit
1 = Initiates a write cycle. The bit is cleared by hardware once write is complete. The WR bit can only be set (not cleared) in software.
0 = Write cycle to the EEPROM is complete

bit 0 **RD:** Read Control bit
1 = Initiates an EEPROM read RD is cleared in hardware. The RD bit can only be set (not cleared) in software.
0 = Does not initiate an EEPROM read

هي اللي بتخلي ال system يشتغل

Some Issues of Timing

- **The Clock** → مهمة

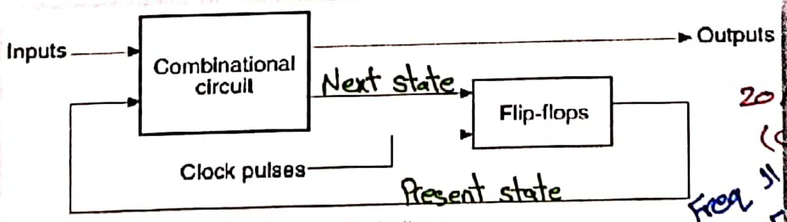
منال input بطلع ← ال output / ما فيها ذاكرة

فيها ذاكرة و Flip-Flop أو register وهكذا

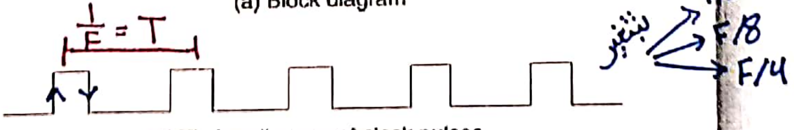
  - The microcontroller is made up of combinational and sequential logic. Thus, it requires a clock !

Synch--

  - Clock – a continuously running fixed frequency logic square wave
  - Timers, counters, serial communication functions are also dependent on the clock
  - Operating frequency has direct impact on power consumption → أعلى Freq في ال component التي هي ال cpu داخل ال Pic = 5 MHz لإنه أعلى Freq لكل ال microcontroller هي 20 MHz ← 20/4=5 (ربع ال Freq الأساسي)
  - Every microcontroller has a range for its clock



(a) Block diagram



(b) Timing diagram of clock pulses

$F = \frac{F_{osc}}{4} \Rightarrow T_{cy} = \frac{4}{F_{osc}}$

instruction cycle

ال Period لتنفيذ ال inst

Some Issues of Timing

ال cpu بنفذ فيها ال inst

- **Instruction Cycle** →
  Fosc /4 = Finstructions
  - The main clock is divided by a fixed value (4 in the 16 series) into a lower-frequency signal
  - The cycle time of this signal is called the ***instruction cycle*** ↓ Tcy = 1/Finst = 4/Fo
  - The primary unit of time in the action of processor

Clock frequency	Instruction cycle	
	Frequency	Period
20 MHz	5 MHz	200 ns
4 MHz	1 MHz	1 µs
1 MHz	250 kHz	4 µs
32.768 kHz	8.192 kHz	122.07 µs

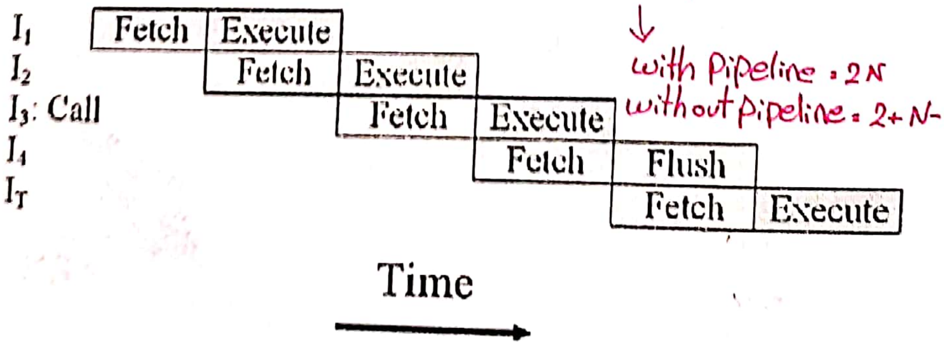
/4 →

→ Tcy = 4/Fosc or 1/Finst

↓ المقلوب

Some Issues of Timing

- **Pipelining** (زيادةعدد ال inst اللي بخلصوهم في وحدة الزمن)
  - Every instruction in the computer has to be fetched from memory and then executed. These steps are usually performed one after another
  - The CPU can be designed to fetch the next instruction while executing the current instruction. This improves performance significantly!
  - This is called *Pipelining*
  - All PIC microcontrollers implement pipelining (RISC+Harvard make it easy)



Speedup = 2
↓
with Pipeline = 2N
without Pipeline = 2+N-1

→ Time أي System ما بشتغل مباشرة لازم أعطيه
يجهز حاله .

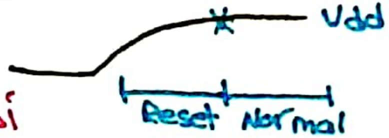
Power-up and Reset

- On power-up, the microcontroller must start to execute the program stored in the program memory from its beginning (address 0000H)

* Reset state and Normal state : ال microcontroller الها حالتين

- A specialized circuit inside the microcontroller detects this and is responsible for putting the microcontroller in the ***reset state***:
  - the program counter is set to zero PC=0000 → لأنه البرنامج تبعي اللي حيبلش ينفذ موجود هون
  - the SFRs are set such that the peripherals are in safe and disabled
- <u>Another way to put the microcontroller in the reset state is to apply logic zero to the Master Clear input (MCLR)</u>
- Some reset circuit configurations

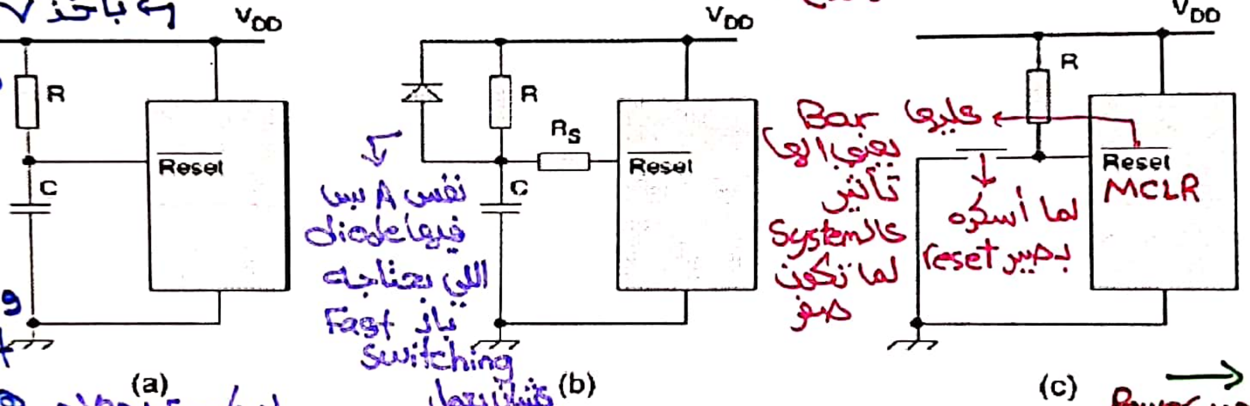
أطمئن ال Normal
وأروح ال reset



reset
بطريقتين :
Power-
MCLR

← باخذ V لل capacitor
وبدخله عال
Reset

وعشان أطول ال
reset استخدم
هذه السيركت



→ VDD < 5mv/s لو
بطولها

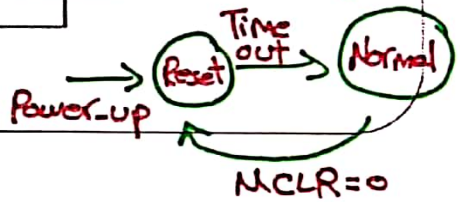
نفس A بس
فيها diode
اللي بحاجته
باز Fast
Switching
عشان يعمل
discharge

Bar
يعني إلها
تأثير
ال System
لما تكون
صفر

عليها ←

لما أسكره
بصير reset

Summary :



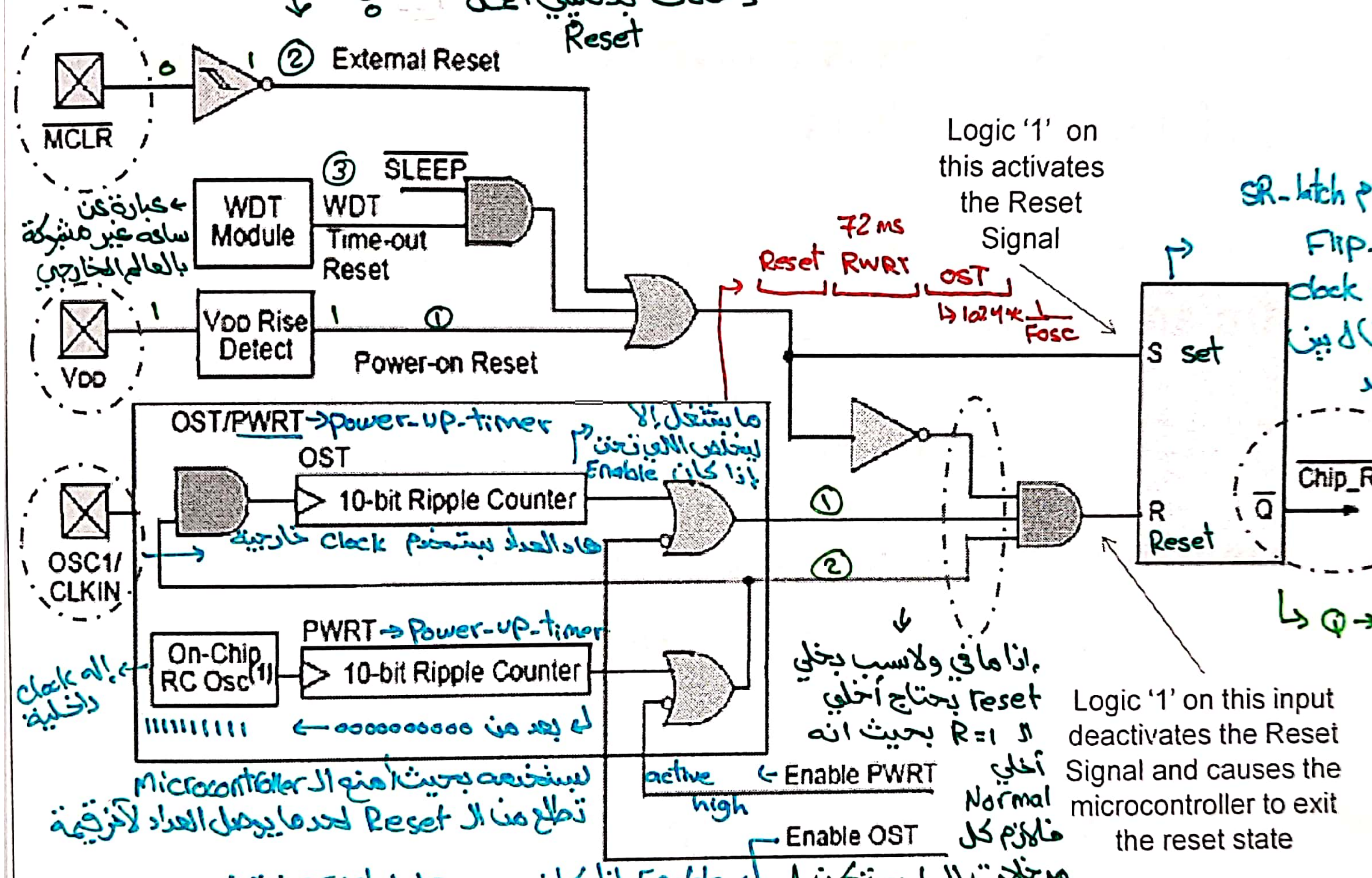
* الـmicrocontroller لازم تضلها متذكرة هي بأي state فلازم عندي storage

MCLR=0

بشو الأسباب اللي بتخلي الـMicrocontroller تنقل بين الـstates المختلفة
reset
Normal

The 16F84A on-Chip Reset Circuit

3 حالات بتخليني أعمل Reset



Logic '1' on this activates the Reset Signal

Logic '1' on this input deactivates the Reset Signal and causes the microcontroller to exit the reset state

← عبارة عن ساعة غير مشتركة بالعالم الخارجي

ما بشتغل إلا
ليخلص اللي نحن م
إذا كان Enable

هاد العداد بيستخدم clock خارجية

clock داخلية ← إله

لـ بعد من 0000000000 ← 1111111111

بيستخدم بحيث أمنع الـMicrocontroller تطلع من الـReset لحد ما يوصل العداد لآخر قيمة

← Enable إذا كان مصدرها crystal فقط
XT LP HS

إذا ما في ولا سبب بخلي reset بحتاج أخلي الـ R=1 بحيث انه أخلي Normal فلازم كل مدخلات الـand تتكون 1

مستخدم SR-latch
مو مستخدم Flip-Flop
لأنه ما به يعتمد ع clock
ما به يكون الانتقال بين
الـStates معتمد
ع clock
لأنه لسا
الـsystem
مو شغال.

لـ Q → state → 1 Reset
→ 0 Normal

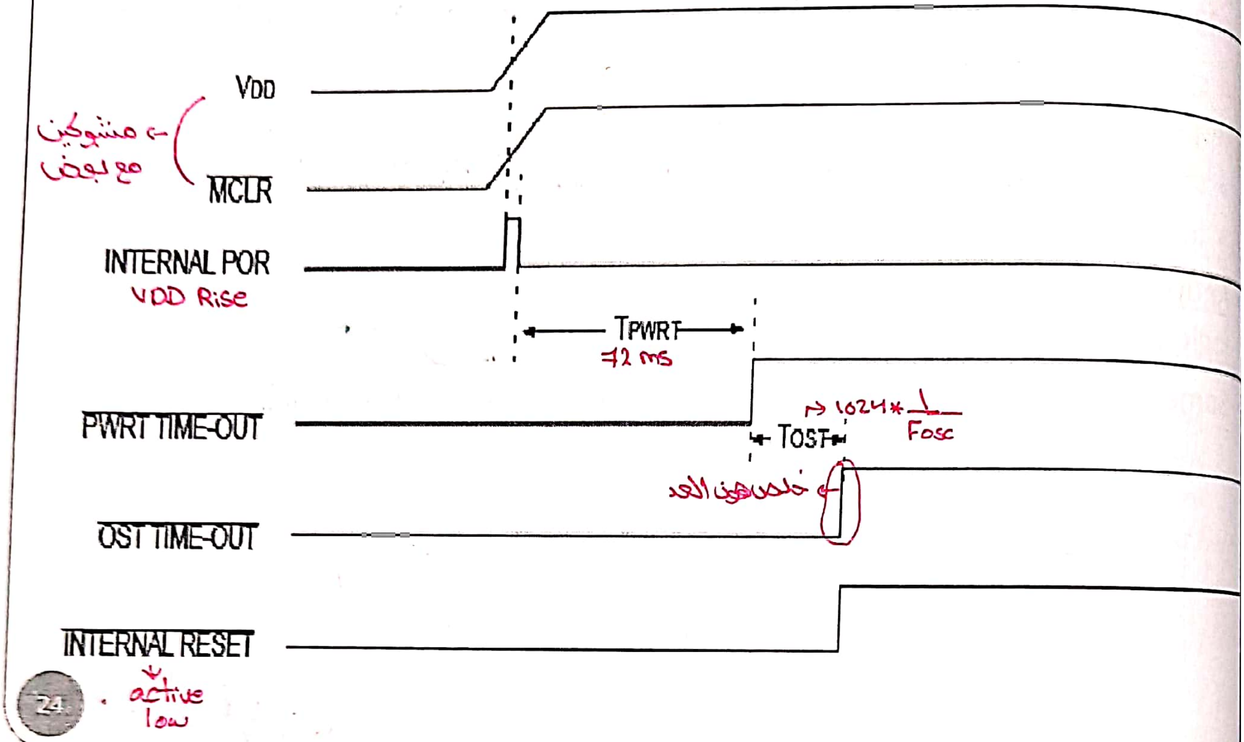
S	R	Q^+
0	0	Hold
0	1	Reset
1	0	set
1	1	Not Valid

* يستخدم المدخلين ① و ② لإطالة وقت الـReset إذا كنت محتاج ذلك

بتورجيني متى بنطلع ال microcontroller من ال Reset-state في حالات مختلفة

The 16F84A on-Chip Reset Circuit

Example on reset timing when MCLR is connected to VDD



Summary

- The PIC 16F84A series is a diverse and cost effective family of microcontrollers
- The PIC 16F84A is pipelined RISC processor with Harvard architecture
- The PIC 16F84A has three different memory types
- An important memory area is the Special Function Register area which act as link between the CPU and peripherals
- Reset operation must be understood for proper operation of the microcontroller

PAGE
DATE

Slide 19 :

* الفائدة من زيادة ال Frequency هي إني أقلل ال Time بين ال Edges بالتالي بقدر أعمل أكثر من job خلال ال Time وبذلك بتحسن ال Performance .

* السبب في وجود Maximum Frequency :

1- لإنو بستهلك نفس الطاقة في فترة زمنية أقل فال Power بتصير عالية وبتتحول الطاقة من كهربائية إلى حرارية، إذا زدنا ال Frequency عن ال Maximum وبالتالي ممكن يخرب النظام .

2- أنا أعطيت ال combinational circuits الوقت الكافي لتقوم بعملياتها وتطلع ال output وال Next state فلو زدت ال Frequency بقل ال Period time فبالتالي النتيجة اللي حصلت عليها خاطئة لإنو ما أعطيت ال combinational circuit الوقت الكافي لتعطيني النتيجة فما بصير أعلي ال Freq كثير لإنو في delay لازم أراعيها .

Slide 21 :

* الخطوات الرئيسية لتنفيذ أي Instruction :

Inst 1 F D E → 3 cycles per inst
Inst 2 F D E
Inst 3 F D E

* عدد ال cycles الكلي اللي بحتاجه = 9 cycles . (more time : 9 Msec)

* بعدين صار في ما يسمى بال Pipelining وهو تنفيذ أكثر من مهمة مختلفة بنفس الوقت :

Inst 1 F D E → 3 cycles per inst
Inst 2 F D E
Inst 3 F D E $(3 \underset{\uparrow}{+} N - 1)$ cycles

* هيك صار عدد ال cycles الكلي = 5 cycles . (less time : 5 Msec)

* ال Time to Execute the Instruction هو نفسه ولكن تحسن ال Performance نتيجة ال (overlapping) .

* Speedup = $\frac{3N}{3+N-1}$ → (قديه الأول أبطأ من الثاني)

* الـCPU بداخل ال computers بتاخد ال Prog من ال harddisk لل Mem لكن ال CPU داخل ال microcontrollers مافي harddisk فعادة بتكون مخزنة على ال Prog Mem وعادة تكون Non-volatile لحتى تضل ال micro متذكرة البرنامج.

Introduction

← أي CPU وظيفتها تنفذ وتفهم مجموعة من ال instruction محددة.

- Every computer can recognize and execute a group of instructions called the *Instruction Set*
- These instructions are represented in binary (*machine code*) لـ لغة تفهمها ال CPU
- *A program* is a sequence of instructions drawn from the instruction set and combined to perform specific operation
- To run the program:
  - It is loaded in binary format in the system memory
  - FDE ← The computer steps through every instruction and execute it
  - Execution continues unless something stops it like the end of program or an interrupt

مثل انه يخلص البرنامج أو تحدث عملية مقاطعة (interrupt)

How to Write Programs

- **Machine code**
  - Uses the binary equivalent of the instructions
  - Slow, tedious, and error-prone → ممكن أخطأ فيها بسهولة

متعبة

```
00 0111 0001 0101
```

→ Ex: w = w+13 بنعبر عنها هيك

- **Assembly**
  - Each instruction is given a *mnemonic* → نمط معين من الحروف.
  - A program called *Assembler* converts to machine code
  - Rather slow and inefficient for large and complex programs

كل processor اله لغة Assembly خاصة لكن بنفس المبدأ.

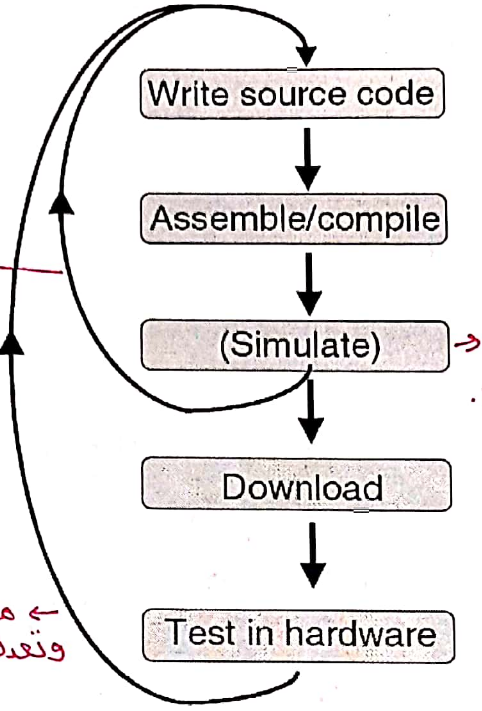
```
addw NUM, w
```

- **High-level language**
  - Use English-like commands to program
  - A program called *Compiler* converts to machine code
  - Easy !! The program could be inefficient !

← ذكي وهو سهل وبحولها لل Assembly ثم Machine code

```
for (i=0; i<10; i++) sum += a[i];
```

Program Development Process



Write source code

Assemble/compile

(Simulate)

Download

Test in hardware

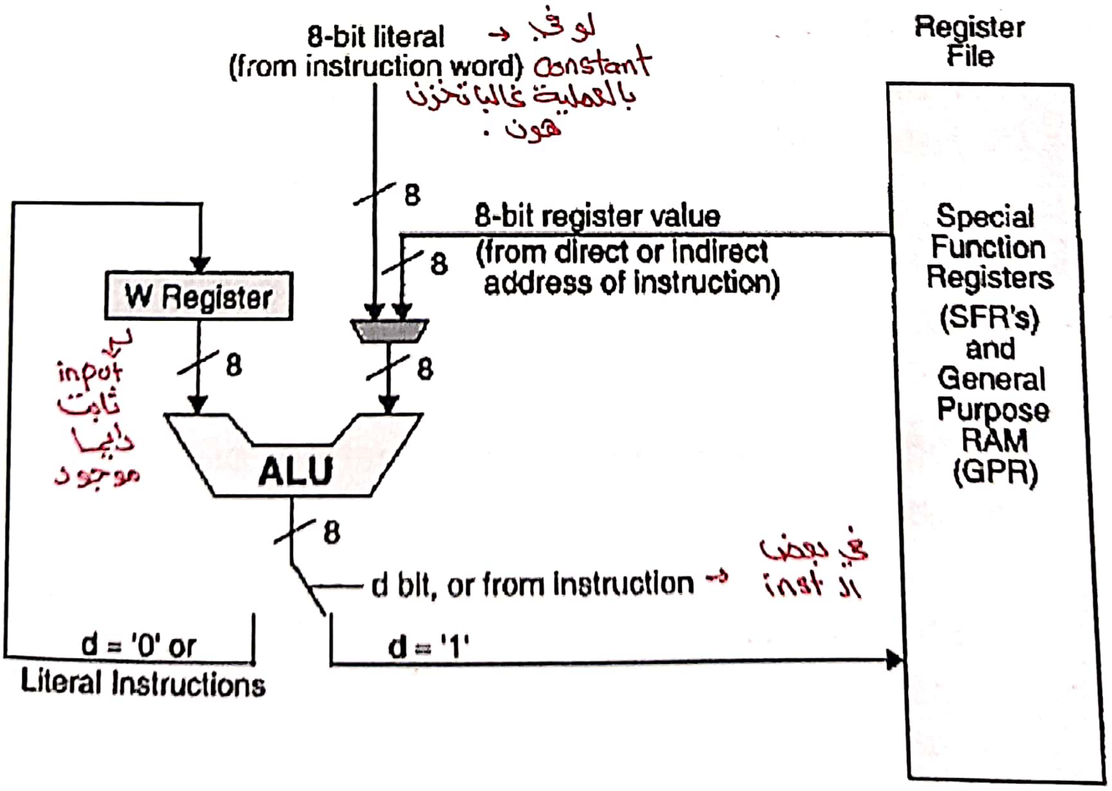
← فيحال اكتشفت
إنه عندك
logical Error

→ اختيارية ولكن الأفضل تعملها
عشان تتأكد إنه كله ماشي صح.

← ممكن تضطر ترجع تكتب
وتعدل عالبرنامج.

The PIC 16 Series Instruction Set

- The PIC 16 Series ALU



→ لو في constant
بالعملية غالبا تخزن
هون.

input
ثابت
دايما
موجود

→ في بعض
ال inst

The PIC 16 Series Instruction Set

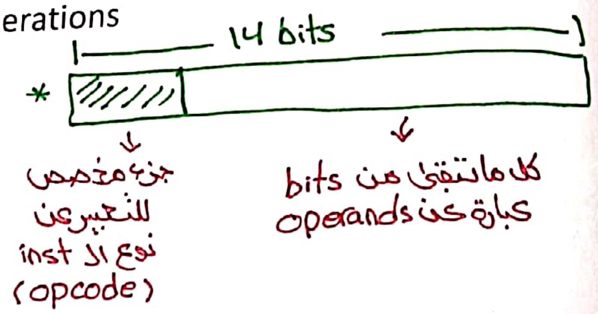
- 35 instructions represented using 14 bits!!! دائماً
- The binary code of the instruction itself is called the *Opcode*
- Most of these instruction operate/use on values called *Operands (ranging from no operands to two)*
- Three categories of instructions → بناءً على حجم ال operand ونوعه
  1. Byte-oriented file register operations → بياخذ كل ال FR
  2. Bit-oriented file register operations → بغير على bit بال FR بس
  3. Literal and control operations
     8-bit
- Type of operations
  1. Arithmetic
  2. Logic
  3. Data movement
  4. Control
  5. Miscellaneous

بثبتوا حجم ال inst عشان ال cpu تعرف كم حجم ال inst لما تقراه وما يكون صعب

ال First operand معروف ال W هناك ال Second operand

بعتمد على طبيعة ال op

عامة



The PIC 16 Series Instruction Set

- **Introduction to PIC 16 ISA**
  - Types of operands
    - A 7-bit address for a memory location in RAM (Register File) denoted by **f**
    - A 3-bit to specify a bit location within an the 8-bit data denoted by **b**
    - A 1-bit to determine the destination of the result denoted by **d**
    - A 8-bit number for literal data or 11-bit number for literal address denoted by **k**

The PIC 16 Series Instruction Set

- Examples
  - **clrw**
    - Clears the working register W
  - **clrf f**
    - Clears the memory location specified by the 7-bit address f
  - **addwf f , d**
    - Adds the contents of the working register W to the memory location with 7-bit address in f. the result is saved in *W if d =0, or in f if d = 1*
  - **bcf f , b**
    - Clears the bit in position specified by b in memory location specified by 7-bit address f
  - **addlw k**
    - Adds the content of W to the 8-bit value specified by k. The result is stored back in W

The PIC 16 Series Instruction Set

Byte-oriented File Register Operations

- Format: `op f, d`
  - `op`: operation
  - `f`: address of file or register
  - `d`: destination (0: working register, 1: file register)

Example: addwf 0X30,0

- Example:

  `addwf PORTA, 0`

Adds the contents of the working register and register `PORTA` then puts the result in the working register.

The PIC 16 Series Instruction Set

Bit-oriented File Register Operations

- Format: `op f, b`
  - `op`: operation
  - `f`: address of file or register
  - `b`: bit number, 0 through 7

- Example:

Example : bsf 0X0A,5

`bsf STATUS, 5`

Sets to 1 Bit 5 of register `STATUS`.

The PIC 16 Series Instruction Set

Literal and Control Operations

- Format: `op k`
  - `op`: operation
  - `k`: literal, an 8-bit if data or 11-bit if address

- Examples:

`addlw 5`

Adds to the working register the value 5.

`call 9`

Calls the subroutine at address 9.

The PIC 16 Series Instruction Set

Arithmetic Instructions

قديش بده حتى ينفذ وحدة
من هاي الInstruction ←

Mnemonic	Operands	Description	Cycles	Status Affected
ADDWF	f, d	Add W and f	1	C,DC,Z
COMF	f, d	Not ← 1's comp Complement f	1	Z
DECF	f, d	Decrement f	1	Z
INCF	f, d	Increment f	1	Z
SUBWF	f, d	Subtract W from f → F-W	1	C,DC,Z
ADDLW	k	Add literal and W	1	C,DC,Z
SUBLW	k	Subtract W from literal → L-W	1	C,DC,Z

d = 0 , result is stored in W
d = 1 , result is stored in F

FD E
FD E

→ ففعليا كأنه تحتاج cycle الـE يعني 1

مافي Flag خاص للـBorrow ولكن الـC هو اللي بطلع الـBorrow لو في Borrow بصير C=0 لو مافي بصير C=1 يعني تأثير عكسي

EX: 0X01 − 0X03 → C=0 we have Borrow

0X03 − 0X01 → C=1 we don't have Borrow

The PIC 16 Series Instruction Set

Logic Instructions

Ex:
XoRWF f,1
ComF f,1] → XnoR operatio

Mnemonic	Operands	Description	Cycles	Status Affected
ANDWF	f, d	AND W with f	1	Z
IORWF	f, d	Inclusive OR W with f	1	Z
XORWF	f, d	Exclusive OR W with f	1	Z
ANDLW	k	AND literal with W	1	Z
IORLW	k	Inclusive OR literal with W	1	Z
XORLW	k	Exclusive OR literal with W	1	Z

+ COMF → تعتبر logic أيضا

d = 0 , result is stored in W
d = 1 , result is stored in F

The PIC 16 Series Instruction Set

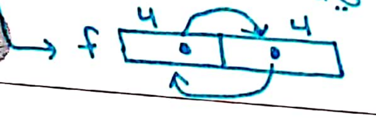
Data Movement Instructions

لـ ال instruction اللي ممكن استخدموا حتى أحرك ال Data
داخل ال microcontrollers .

Mnemonic	Operands	Description	Cycles	Status Affected
MOVF	f, d	Move f	1	Z
MOVWF	f	نشخ ← Move W to f	1	
SWAPF	f, d	Swap nibbles in f	1	
MOVLW	k	Move literal to W	1	

قيمة حرفية

Ex: MOVLW 0 → حط 0 في W
MOVLW F → حط 15 في W



d = 0 , result is stored in W
d = 1 , result is stored in F

* MOVF, MOVWF, MOVLW:
فعليا ما بتغير ال data
بناخد نسخة منها.

The PIC 16 Series Instruction Set

instruction تتحكم في مسار البرنامج .

Control Instructions

مهمة
بستخدمو
للتنقل بين
أجزاء الكو

Mnemonic	Operands	Description	Cycles	Status Affected
DECFSZ	f, d	Decrement f, Skip if 0	1 (2)	
INCFSZ	f, d	Increment f, Skip if 0	1 (2)	
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	conditional
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	
CALL	k	Call subroutine	2	
ع GOTO	k	Go to address	2	uncondition
RETFIE	-	Return from interrupt	2	
RETLW	k	Return with literal in W	2	
RETURN	-	Return from Subroutine	2	

d = 0 , result is stored in W , d = 1 , result is stored in F

The PIC 16 Series Instruction Set

ما إلها تصنيف محدد. ←

Miscellaneous Instructions

Mnemonic	Operands	Description	Cycles	Status Affected
CLRF	f	Clear f	1	Z
CLRW	-	Clear W	1	Z
NOP	-	No Operation	1	
RLF	f, d	Rotate Left f through Carry	1	C
RRF	f, d	Rotate Right f through Carry	1	C
BCF	f, b	Bit Clear f	1	
BSF	f, b	Bit Set f	1	
CLRWDT	-	Clear Watchdog Timer	1	TO',PD'
SLEEP	-	Go into standby mode	1	TO',PD'

→ بنقدر ناخذ محتويات Some File Register ونسويلها rotation by 1 bit إما يمين أو شمال وعملية ال rotation بتصير من خلال ال Carry

d = 0 , result is stored in W , d = 1 , result is stored in F

The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
bcf 0x31, 3	clear bit 3 in location 0x31	None
bsf 0x04 , 0	set bit 0 location 0x04	None
bsf STATUS , 5	set bit 5 in STATUS register to select bank 1 in memory	None
bcf STATUS , C	clear the carry bit in the status register	None
↳ التعامل مع الأسماء أسهل من الأرقام		
addlw 4	Adds 4 to working register W and store the result in back in W	C, DC, Z
addwf 0x0C, 1	Add the content of location 0x0C to W and store the result in 0CH (d =1)	C, DC , Z
sublw 10	Subtract W from 10 and put the result in W	C, DC, Z
subwf 0x3C, 0	Subtract W from contents of location 0x3C and store the result in W	C, DC, Z

الـ inst مش حساسة تجاه الأحرف كبيرة أو صغيرة لكن الأسماء (Lables
حساسين.

The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
incf 0x06, 0	Increment location 0x06 by 1 and store result in W	Z
decf TEMP, 1	Decrement location TEMP by 1 and store in TEMP	Z
comf 0x10, 1	Complement the value in location 10H and store in 0x10	Z
andlw B'11110110'	AND literal value 11110110 with W and store result in W	Z
andwf 0x33, 1	AND location 0x33 with W and store result in 0x33	Z
iorlw B'00001111'	Inclusive-or W with 00001111	Z
iorwf X1 , 0	Inclusive-or W with location X1 and store result in W	Z
xorlw B'01010101'	Exclusive-or W with 01010101	
xorwf 0x2A , 0	Exclusive-or W with location 0x2A and store result in W	Z

The PIC 16 Series Instruction Set

Examples

Instruction	Operation	Flags Affected
clrw	Clear W	Z
clrf 0x01	Clear location 0x01	Z
movlw 18	Move literal value 18 into W	NONE
movwf 0x40	Move contents of W to location 0x40	NONE
movf 0x21 , 0	Move contents of location 0x21 to W	Z
movf 0x21 , 0x33	Incorrect syntax	--
movwf 0x1B , 1	Incorrect syntax	--
swapf T1 , 1	Swap 4-bit nibbles of location T1	NONE
swapf DATA , 0	Move DATA to W, swap nibbles, no change on DATA	NONE
rlf TEMP , 1	Rotate contents of location TEMP to left by one bit position through the C flag	C
rlf 0x25 , 0	Copy contents of 0x25 to W and rotate to left by one bit position through the C flag	C

PAGE
DATE

Chapter 4

Slide 7 : Byte-oriented :

Format: op f, d → destination → 0: working reg / → 1: File reg

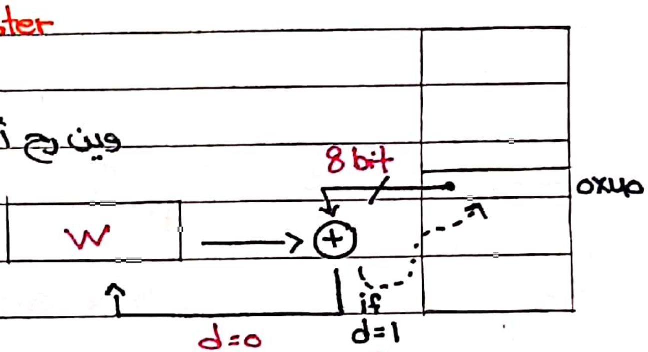
type ↲ ↳ Address of file register

Ex: ADDWF 0X40 , 0 ($\overset{f}{0X40}$, $\overset{d}{0}$)

working reg ↓ بتجمع — 0X40 ↓ اخترت ↲ وين رح أخزنه

File reg مع الreg

اللي بالRAM



* opcode	d	f
6-bits	1	7-bits

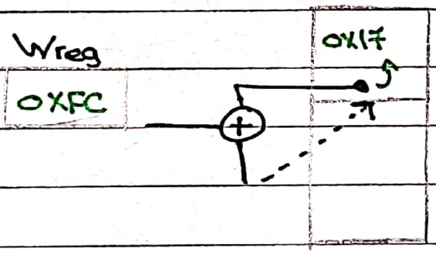
d=1 ↓ تخزن فقط في نفس ال address

ما بقدر أخزنها بمكان ثاني ← one instruction بس لأنه بحتاج أكثر من 14-bits.

Ex: ADDWF 0X42 , 1 Assume : Wreg = 0XFC / M[0X42] = 0X17

Wreg = ?? and M[0X42] = ??

Sol.



half carry

$$\begin{array}{r} 0XFC \\ +\ 0X17 \\ \hline 1\quad 13 \end{array}$$

1 ↓ Carry

13 → 0X13 ↓ ALU-output

⇒ M[0X42] = 0X13 / Wreg = 0XFC / C=1 → Status reg اللي موجود في C-bit

/ Z=0 / DC=1 → half-carry بسبب ال — نلاقي حيتحول لواحد عشان ال Carry=1

↳ 0 ≠ ALU-output لأنه ال

* C=1
DC=1
Z=0

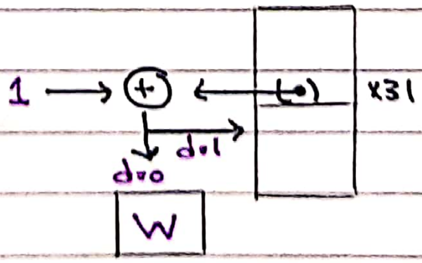
} status register نلاقي بنحدثوا.

EX: INCF 0X31 , 0 (F ↑ 0X31, d ↑ 0)

↓ increment som FR and add it +1

Assume : M[0X31] = 0XFF

$$\begin{array}{r} 0XFF \\ +\quad 1 \\ \hline 1\ 00 \end{array}$$



C = No change
Z = 1
DC = No change

New M = 0X00

M[0X31] = 0XFF

7	6	5	4	3	2	1	0

* Bit-oriented FR :

* Format : $\underset{\text{operation}}{\text{op}}$ $\underset{\text{address}}{F}$, b → bit number : 0-7

Ex : BCF F, b
bit clear FR (0) ↓ 7-bit, bit position (3-bit)

opcode	b	f
4	3	7

* فيوم التخزين بكون دايما عال FR ماعندي خيار أخزنهم عال Wreg .

Ex : BSF F, b
Set (1)

Ex : BCF 0X03, 0X5 knowing that: M [0X03] = 0XF5
↳ 1111 0101 => 1101 0101 => M [0X03] = 0XD5
bit 5 → 0

* C, Z, DC } Not affected

* Note : 0X03 → what is the Bank of this Location ?
0000 0011
↳ most sig → Bank بمثل ال

فممكن يوحيلي إنه هو في Bank0 لكن فعليا هذا الكلام غير صحيح لإنه الي بحدد ال Bank
مش الرقم هاد : 0000 0011 ، فال Bank بيأتي من ال Status Register
$\underbrace{0}_{\text{لا يخزن}}\underbrace{000\ 0011}_{f\ (7\text{-bit})}$
فلازم أتطلع عال Status reg عال RPO مشان أعرف ال Bank (f+RPO ⇒ Address)

* Note : BCF 0X03, 0X5 → Clear Status reg bit-5 : RPO (Switch Bank 0) , BSF 0X03, 0X5 → set (Switch Bank 1)
* BSF Status, RPO ← حنصير نكتبه هيك

* Note : if we have 4 Banks : RP1 (bit : 6) RPO (bit : 5) Ex : we go to Bank 2 $(10)_2$; (1 → RP1, 0 → RPO)

BCF 0X03, 0X5
BSF 0X03, 0X6

* Literal and Controls operation :

↳ constant

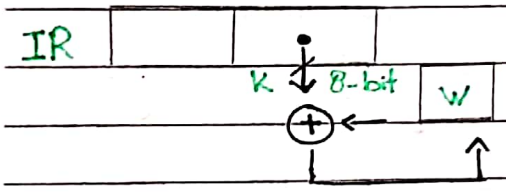
* Format : op K

operation ↳ literal, an 8-bit if data or 11-bit if address

* Ex : ADDLW K → الجواب بنخزن في ال working reg

literal ↳ working reg

opcode	k
6	8



Ex: ADDLW 0X36 , knowing W = 0XC1 ⇒ new W = 0XF7

0X36
0XC1 +
0XF7

* C = 0

Z = 0

DC = 0

Ex: IORLW 0X3D , knowing W = 0X7F ⇒ new w = 0X7F

OR ↙ ↓ 8-bit 8-bit

0X3D → 0011 1101
0X7F → 0111 1111 OR
0111 1111

* C = Not affected

Z = 0

DC = Not affected

Ex: Calculate the 2's complement of M[0X40] :

COMF 0X40 , 1

INCF 0X40, 1

Ex : NOR operation:

IORWF 0X11, 1

COMF 0X11, 1

Ex : MOVF f,1 Z → بفيدني عشان أعرف ال Z شو قيمتها

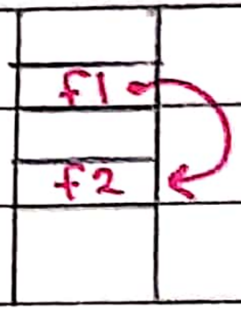
*Note: طريقة أخرى لمعرفة ال Z :

MOVLW 0
ADDWF f,0 ممكن → Z = 1 / → Z = 0

MOVLW 0XFF
AND WF f,0 ممكن → Z = 1 / → Z = 0

Slide 15 : if I want to move from f1 to f2 :

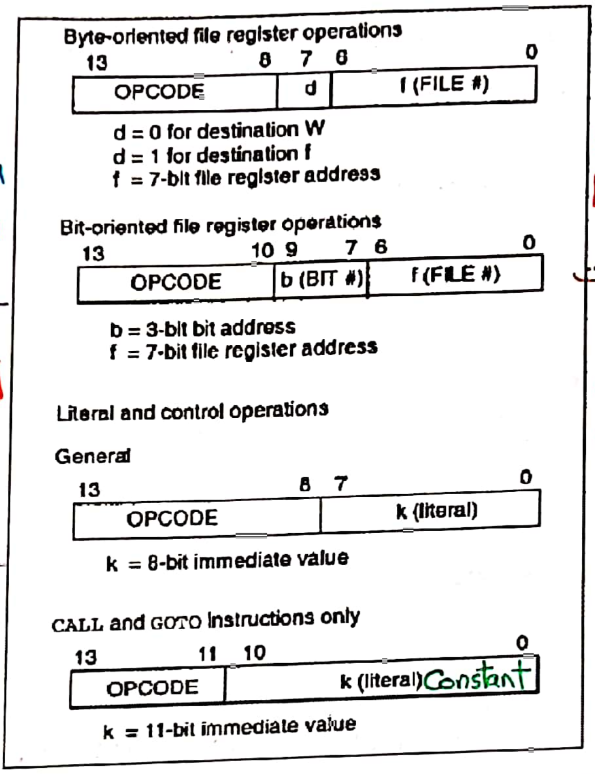
what can I do ?



```
MOVF f1,0
MOVWF f2
```

The PIC 16 Series Instruction Set Encoding

ما بقدر أحدد بأي bank الا لما أشوف ال RP0



Byte-oriented file register operations

d = 0 for destination W
d = 1 for destination f
f = 7-bit file register address

Bit-oriented file register operations

b = 3-bit bit address
f = 7-bit file register address

Literal and control operations

General

k = 8-bit immediate value

CALL and GOTO instructions only

k = 11-bit immediate value

Check Appendix A for opecode binary codes

* ال Program memory ما بنقسمهم بندخلهم 14-bits وفي مرحلة ال decod بيتم تخزين وتقسيم ال register زي ما موضح.

EX: BCF 0XDC, 6

4	3	7
0100	110	1011100

0XDC → 11011100

⇒ $(01001101011100)_2$

⇒ 0X135C

ال cpu بتميز كم حجم ال code عن طريق ال Most sig اللي بالجدول اللي بوضح ال opcode

EX: ADDWF 0X85, 1

10000101 ← بجيبه لإنه بياخذ 7-bit only

* أول شي بده يعرف شو نوع ال op : الADDWF مبرمجة تكون هيك :

opcode	d	f
6	1	7

* بعض بده يحسب بعبيهم ، في شغلات بقدر يسويها مباشرة وشغلات لأ :

opcode	d	f
000111	1	0000101
6	1	7

اللي صمم ال cpu لازم يكون أعطى binary code لكل inst موجودة

⇒ 00011110000101 ⇒ 0X0785

البرنامج اللي بحول كل inst بال code تاعي إلى binary وعادة بكون عليه إضافات بتسهل علي كمبرمج.

Assembler Details

بستخدم أرقام/حروف/ــ ولازم تبلش من أقصى اليسار

- Any assembler line may have up to four different elements

بقدر أضيف comment باستخدام ;

label — mnemonic — operand — comment

بقدر أعطي lable بسهل علي قراءة وكتابة البرنامج + بساعدني بال flowcontrol لما اتحكم بمسار البرنامج.

```
start bsf   status,5   ;select memory bank 1
      movlw B'00011000';config pattern for port A
      movwf trisa
      movlw 53
      ...
```

Ex: goto start

بدل هذا ال lable بال address ← Assembler تاع ال inst

- We can specify values in different bases in assembler programs

Radix	Example
Decimal	D'255'
Hexadecimal	H'8d' or 0x8d
Octal	O'574'
Binary	B'01011100'
ASCII	'G' or A'G'

* لو ما حطينا ال base ال Assembler بحط ال deafult اللي عنده.

Assembler Details

Assembler عبارة عن كلمات بنعبر عن Commands

- **Assembler directives** → الموجهات لل Assembler بنوجه ال Assembler حتى يسوي شيء معين خلال العملية تحويل البرنامج.
  - These are assembler-specific commands to aid the processing of assembly programs

Assembler directive	Summary of action
org	Set program origin
equ	Define an assembly constant; this allows us to assign a value to a label
cblock and endc	Define a block of variables
end	End program block
#include	Include additional source file

عالأقل بيكون عندي وحدة ← org

equate (بنريح بدل ما نكتب أرقام) ← equ

Ex: BCF 0x03,0
↓
BCF status,c
↓
Status equ 0x03
C equ 0

بنوقف عملية التحويل لل binary لا تعني إيقاف علية تنفيذ البرنامج ← cblock and endc

ضروري تكون ← end → لازم عالأقل وحدة تكون

* org → originate (حطلي نقطة مرجع لـ Mem add)

Ex: org 0x0000 → وظيفة هذا السطر انه ال assembler لما ينزل هنا ال code عال Prog Mem هاي ال inst رح تتخزن بال address 0000 والي بعدها 0001 واللي بعدها 0002، يعني كأنه نقطة المرجع 0 ويمشي عالتسلسل

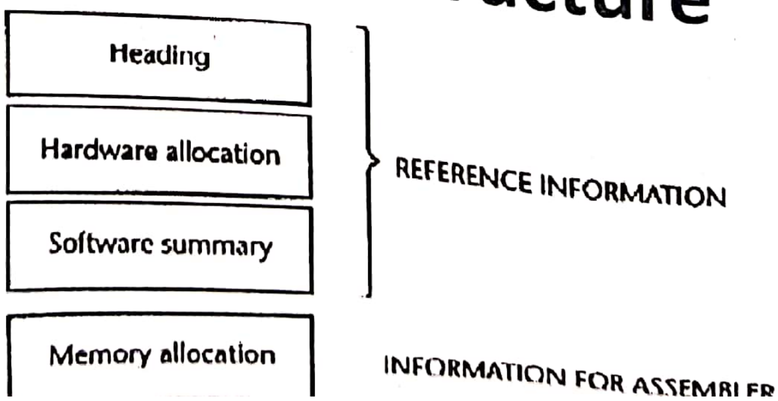
qrw
MOVF --
: --

#include → بحط كل ال lables اللي بدي أعرفهم بـ File وبجيبه بس استخدموا بدل ما أضل أعرفهم.

Program Structure

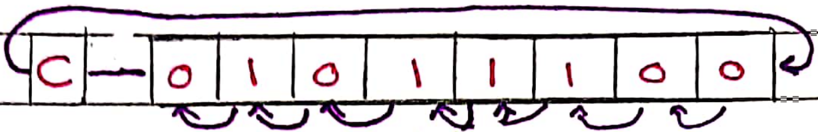
Ex:
Status equ 0x03
C equ 0
:
} بـ File

⇒ #include P16F84.inc



Slide 17 : RLF / RRF

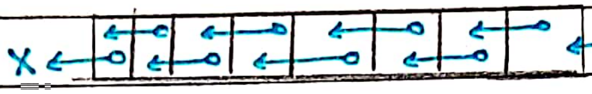
Ex : RLF



*أي FR موجود داخل ال PIC16 Series هو 8-bit .

* أنا بسوي rotation ل 9-bit مش 8-bit لأنه في ال Carry كمان .

*Note :

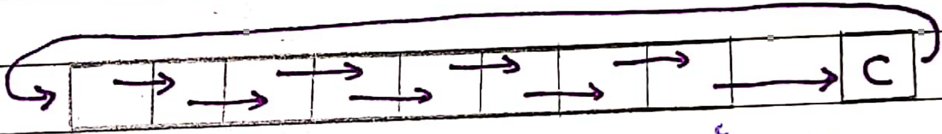


lift Shift operation ↓

فائدتها لما يكون عندي مثلاً Binary Number وأعمله هيك يعني ضربت الرقم باثنين وهنا مفيد الها لقدام ، ولكن ال shift op مو موجودة كـ inst فبستخدم ساعتها ال rotate op مع inst أخرى . وعشان هيك لازم أضمن إنه ال C=0 عشان لما يلف ويرجع لأول bit يكون 0 كأني اسويت Shift op 8 ←

EX : BCF 0x03,0 → هاي ال inst بتروح على ال Status RF اللي رقمه 3 عال bit = 0 اللي هو ال Carry وبتسويله Clear .

RLF f,1



EX : RRF f,d

* بستفيد منها اني أسوي Shift Right :

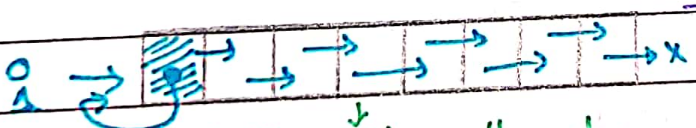
Shift Right operation logical ↓

فائدتها أقسم على 2 ←

EX : BCF 0x03,0

RRF f,1

لو كان آخر bit هو Signed bit ← فلازم أحافظ على إشارة الرقم لأنه لو كان 1 يعني سالب ، ولو 0 يعني موجب



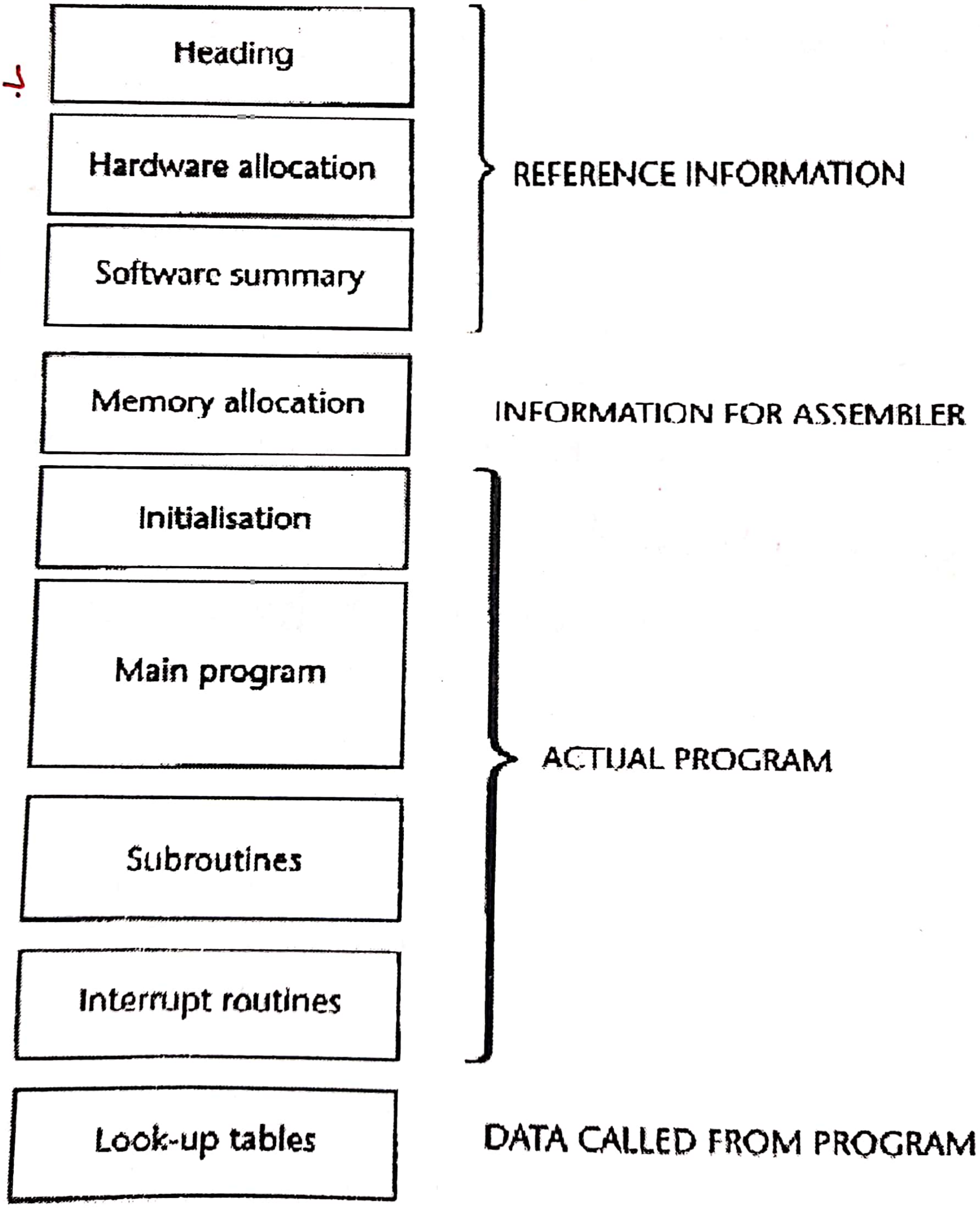
Shift right arithmetic (بكرر الإشارة)

ما بقدر أكتب هاد ال inst اللي بتمثلها ا

Program Structure

File

inc



Heading

Hardware allocation

Software summary

REFERENCE INFORMATION

Memory allocation

INFORMATION FOR ASSEMBLER

Initialisation

Main program

Subroutines

Interrupt routines

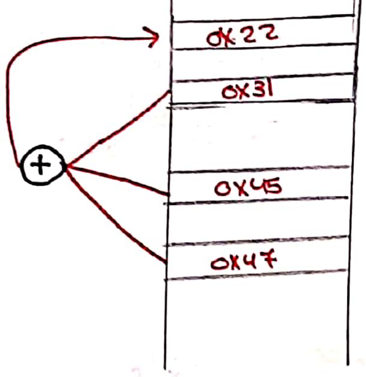
ACTUAL PROGRAM

Look-up tables

DATA CALLED FROM PROGRAM

Sample Program 1

- Write a program to add the numbers stored in locations 31H, 45H, and 47H and store the result in location 22H



①
```
MOVF   0X31,0
ADDWF  0X45,0
ADDWF  0X47,0
MOVWF  0X22
```
→ this is better than the Second:

②
```
CLRW
ADDWF  0X31,0
ADDWF  0X45,0
ADDWF  0X47,0
MOVWF  0X22
```

هيك كتبنا الـ code الرئيسي ولكن لما بدي أنزله فهذا برنامج غير كامل في أشياء لازم أضيفها ليكمل ↓

هي ADDWF → الحل ولكن ما تجمع FR مع FR بنجمع FR مع W

Sample Program 1

```
;************************* EQUATES ********************************
STATUS          equ     0x03            ; define SFRs
RP0             equ     5
;************************** VECTORS ********************************
            org     0x0000          ; reset vector
        0   goto    START
            org     0x0004
INVEC   4   goto    INVEC           ; interrupt vector
;************************* MAIN PROGRAM ****************************
START   5   bcf     STATUS , RP0    ; select bank 0
        6   movf    0x31 , 0        ; put first number in W
        7   addwf   0x45 , 0        ; add second number
        8   addwf   0x47 ,  0       ; add third number
        9   movwf   0x22            ; save result in 0x22
DONE        goto    DONE            ; endless loop
            end
```

ممكن أستغني عنهم ← لو استدعيت #include p1684a.inc

حاليا بقدر أستغني عنهم لأنه ما عندي interrupt بس من الأفضل أحطها بالبداية.

عشان أنط عنه وما أخزن عليه

endless loop

0	goto start
1	
2	
3	
4	goto INVEC
5	bcf
6	movf
	⋮

حطيتها لأضمن ولو شلتها بزبط لأنه 0x47, 0x45, 0x31 هم GPR's وهم بـ Bank0 ولو كان الـ access بـ Bank1 حيرجع لـ Bank0

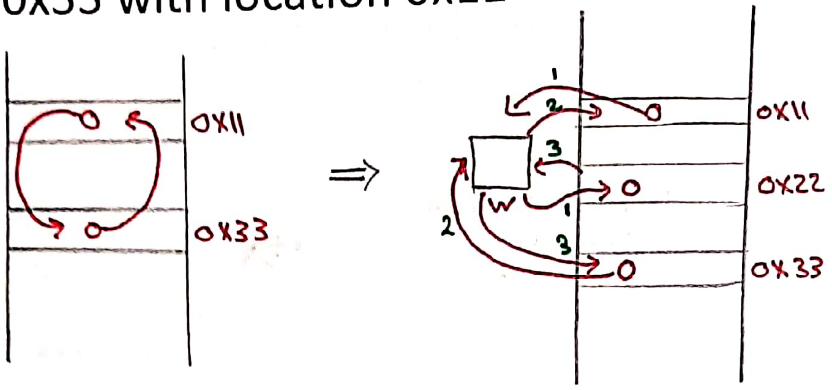
فال micro ما في نهاية فيها من نفسها فلازم نحط شيء يوقف لتنفيذ البرنامج ↑

endless loop → عشان أنهي البرنامج بالـ CPU

end → هاي للـ Assembler

Sample Program 2

- Write a program to swap the contents of locatio
  0x33 with location 0x11



*SWAP→ مابتزبط لإنهاي بتبدل بنفس ال FR . ال least مع ال most 4-bit

---

Sample Program 2

```
;************************** EQUATES **********************************
STATUS          equ     0x03                ; define SFRs
RP0             equ     5
;************************** VECTORS **********************************
                org     0x0000              ; reset vector
                goto    START
                org     0x0004
INVEC           goto    INVEC               ; interrupt vector
;************************** MAIN PROGRAM ************************
START           bcf     STATUS , RP0        ; select bank 0
                movf    0x33 , 0            ; put first number in W
                movwf   0x22                ; store the 1st number temporaril
                movf    0x11 , 0            ; get 2nd number
                movwf   0x33                ; store 2nd in place of 1st
                movf    0x22 , 0            ; get 1st number from 0x22
                movwf   0x11                ; store 1st  in place of 2nd
DONE            goto    DONE                ; endless loop
                end
```

w equ 0
F equ 1

لو بدي أسهل عحالي وأبطل أكتب 0 و 1 للدلالة على w و F

Summary

- The PIC 16F84A has 35 instructions to perform different computational and control operations
- Programs can be written using different levels of abstraction
- Using assemblers simplifies the program development process
- There exist many IDE to aid writing programs and simulate their behavior before putting them into hardware

* ممكن تنزل MPLAB وتروح عموقع اللاب وتقرأ أول 3 تجارب وتعود وتجرب.

Building Assembler Programs

Chapter 5
Sections 1-6

Dr. Iyad Jafar

Outline

- Building Structured Programs
- Conditional Branching
- Subroutines
- Generating Time Delays (Software approach) باستخدام
- Dealing with Data (Lookup tables) (Indirect addressing)
- Example Programs

Building Structured Programs

- Writing programs is not an easy task; especially with large and complex programs
- It is essential to put down a design for the program before writing the first line of code
- This involves documenting the programs flow charts and state diagrams, Meaningful comments, Pseudo code.

رسومات

Pseudo code → عبارة عن code وهي تكتبه لتعبر شو تعمل

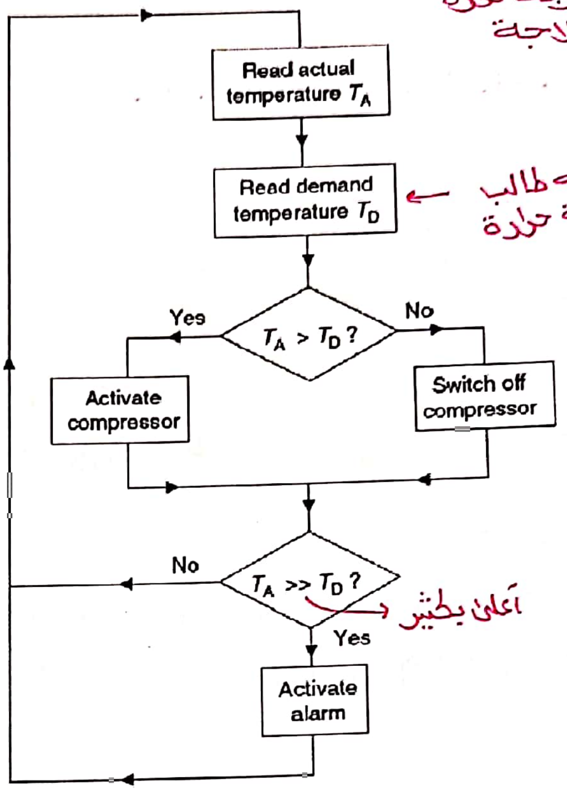
state diagrams → التنقلات بين ال State

Meaningful comments → كلام

Building Structured Programs

- **Flowcharts**
  - Rectangle for process
  - Diamond for decision

رسمة لدرجة حرارة الثلاجة



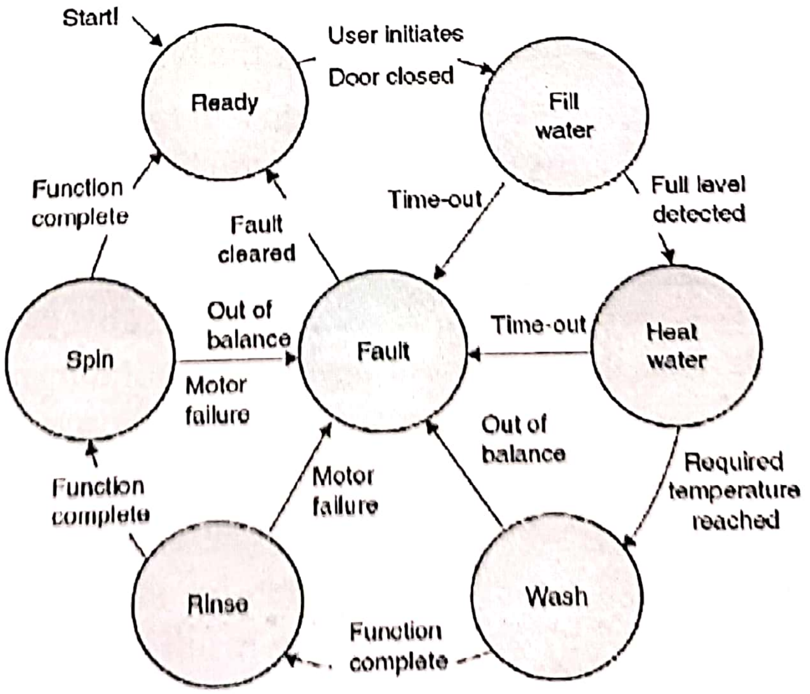
قديه طالب درجة حرارة

أعلى بكثير

Building Structured Programs

رسمة للغسالة

- State Diagrams
  - Circle for state
  - Arrow for state transition labeled with condition(s) that causes the transition



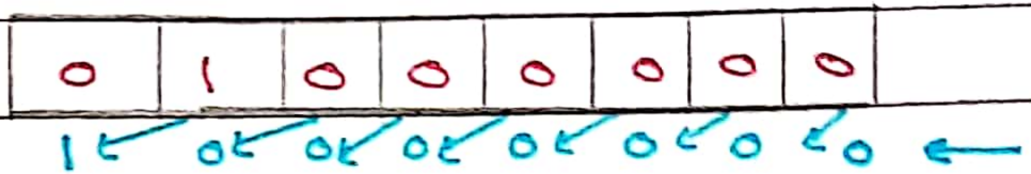
PAGE

DATE

* Note for shift left : (signed number)

بعد ما عملت shift left الرقم كان موجب

صار سالب والمشكلة هون overflow

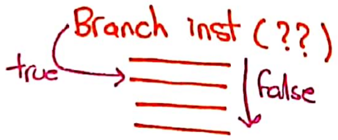


كان +64 فصار -128 لأنه 128 ما بتوسع لأمثلهم في 8-bit أكبر شي 127

Conditional Branching

- Microprocessors and microcontroller should be able to make **decisions**
- This enables them to behave according to the state of logical variables
- The PIC 16 series is not an exception ! They have ***four conditional skip*** instructions
- These instructions *test for a certain condition and skip the following instruction* if the tested condition is true !

Next one instruction just



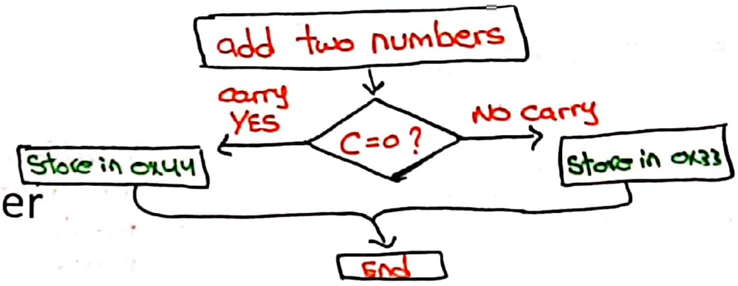
Conditional Branching

Instruction	Operation	Example
btfsc f , b	Test bit at position b in register f. skip next instruction if the bit is clear '0'	**btfsc STATUS , 5**
btfss f , b set(1)	Test bit at position b in register f. skip next instruction if the bit is set '1'	**btfss 0x21 , 1**

Instruction	Operation	Example
decfsz f , d	Decrement the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the decremented result is zero	**decfsz 0x44, 0**
incfsz f , d	Increment the contents of register f by 1 and place the result in W if d = 0 or in f if d = 1. Skip next instruction if the incremented result is zero	**incfsz 0xd1 , 1**

Conditional Branching

- **Example1:** a program to add two numbers that are stored in locations 0x11 and 0x22. If the addition results in *no* carry (C=0), the result is stored in location 0x33. otherwise (C=1), the result is stored in location 0x44



- The STATUS Register

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	TO	PD	Z	DC	C
bit 7							bit 0

Conditional Branching
Example 1

```
STATUS    equ    0x03         ; define SFRs → or #include pic16F84A.inc
          org    0x0000       ; reset vector
          goto   START
          org    0x0006 → Skip for address 4 because the interrupt
START   6 movf   0x11  , 0    ; get first number to W
        7 addwf  0x22 , 0     ; add second number
          btfsc  STATUS , 0   ; check if carry is clear
          (goto  C_SET)  C=1  ;  go to label C_Set if C==1
      C=0 movwf  0x33         ; store result in 0x33
          goto   DONE
C_SET     movwf  0x44
DONE      goto   DONE         ; endless loop
          end
```

لو استخدمنا btfss رح يتغير ال address تاع ال mov

لو كان C=0 عشان أمنعه ينفذ ال C_SET بحطوها.

Conditional Branching

- **Example 2:** Write a program that multiplies the content of location 0x30 by 10.

```
* movf 0x30,0
  addwf 0x30,0
  addwf 0x30,0
    :
```

بكررها 9 مرات لحد ما يصير x10
ولكن هذا الموضوع inefficient
ومش منطقي

```
for(i=0; i<10; i++) {     → loop counter
    x= x+1
}
```

* الdecrement أفضل اشي بلغة assemply ← بعد من 9 لا 0

```
COUNTER   equ   0x20
 org   0x0000
  movlw   D'9'   → Decimal number
  movwf COUNTER → 0x20
  movf 0x30,0
repeat  addwf 0x30,0
  decfsz COUNTER,   1→ لازم ننتبه عليها ما بنخزنها بالw
  goto  repeat      =0 بنخزنها بالf
  movwf 0x30
DONE   goto DONE
```

+9 0000 1001
-9 → 1111 0111

```
movlw  0XF7   → -9
movwf  COUNTER
movf   0x30, w
L1 addwf 0x30,w
  incfsz COUNTER,F
  goto L1
  movwf 0x30
    :
```

لو كنت حكيتها ك inc مش dec لازم أحط بال Counter = -9

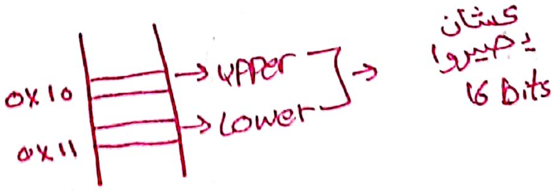
أو أحط '247' ← 256-9

① carry 0000 0000

Conditional Branching

- **Example 3:** The upper and lower bytes of a 16-bit counter are stored in locations COUNTH and COUNTL, respectively. Write a program to decrement the counter until it is zero. Decrementing the counter is allowed if the counter is initially non zero.

باخذ الـ lower وبظل أطرح فيه لحد ما يوصل صفر لو وصل بطرح من الـ upp والـ low واحد وبرجع أكمل لحد ما يصير الاثنين صفر.



Conditional Branching
Example 2

```
COUNTL          equ       0x10          ; lower byte of counter in 0x10
COUNTH          equ       0x11          ; upper byte of counter in 0x11
                #include "P16F84A.INC"
                org       0x0000
START           movf      COUNTL , F    ; check if the both locations are zeros
                btfss     STATUS  Z     ; if so, then finish
                goto      DEC_COUNTL    ; if COUNTL is not zero, decrement it
                movf      COUNTH , F    ; if it is zero check COUNTH
                btfsc     STATUS ,Z
                goto      DONE          ; if both are zeros, then DONE
                decf      COUNTH, F
DEC_COUNTL      decf      COUNTL,F
                goto      START
DONE            goto      DONE          ; program gets here if both are zeros
                end
```

COUNTL → low

COUNTH → high

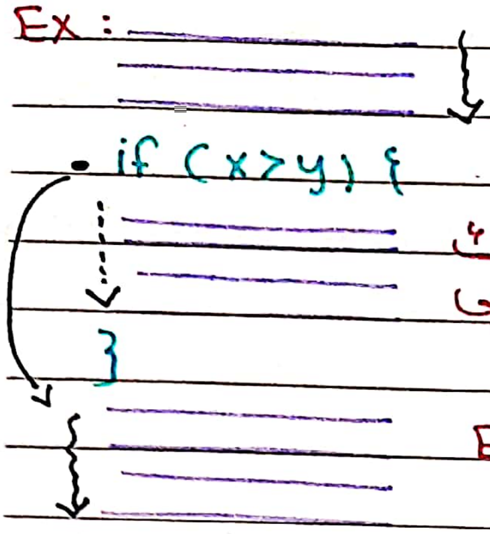
0

0

→ 10 → 01   00 → 11   8 → 7

Chapter 5

Slide 6: Conditional Branching :



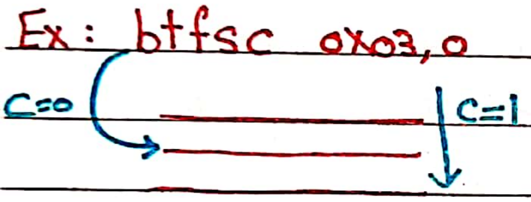
كنا نستخدم هكذا دائماً في لغة ← البرمجة إن كنا نريد التحقق من شيء فلو كانت جملة الـ f صحيحة أنفذ ما داخلها وإن لم تكن أنفذ ما بعدها.

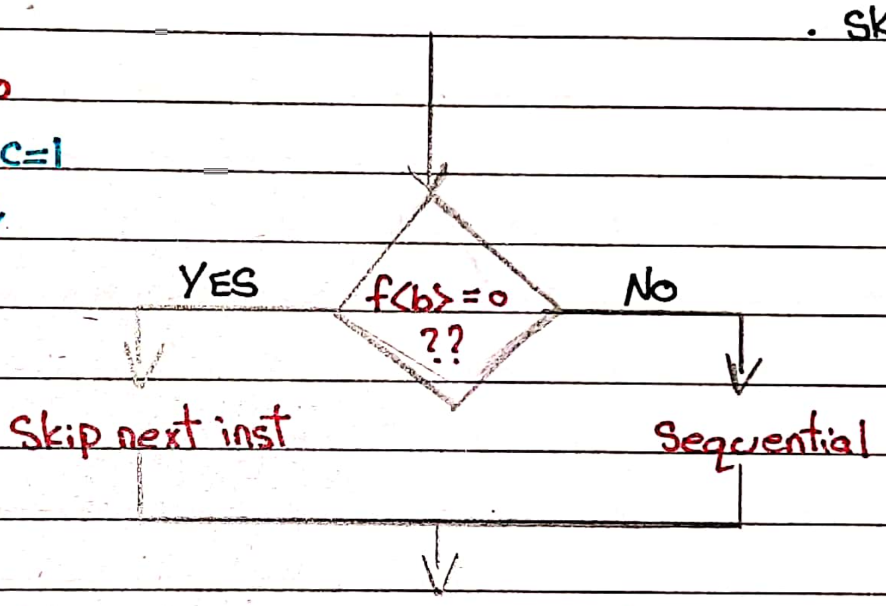
Ex: if, for, while, switch

Slide 7 :

* btfsc → if clear (0) ⇒ هنا الـ inst بنروح ع f معينة بنشوف bit معينة فيها إذا كان صفر أو لا ، لو كان صفر بنسوي skip لو لأ ما بنسوي skip .

bit test file skip





* الـ btfss نفس المبدأ ولكن بنفحص لو الـ bit = 1 أو لأ.

* Slide 7 :

* decfsz → skip if zero ⇒
decrement, file reg

معناها إذا نتيجة طرح 1 من ال FR كانت تساوي صفر رح يعمل ال skip next inst ولو لا مارح يعمل skip.

* incfsz → skip if zero ⇒
increment, file reg

معناها إذا نتيجة جمع 1 لل FR كانت تساوي صفر رح يعمل ال skip next inst ولو لا مارح يعمل skip.

* ال incfsz ممكن أسويها لـ value معينة وتطلع صفر بإنها تكون :

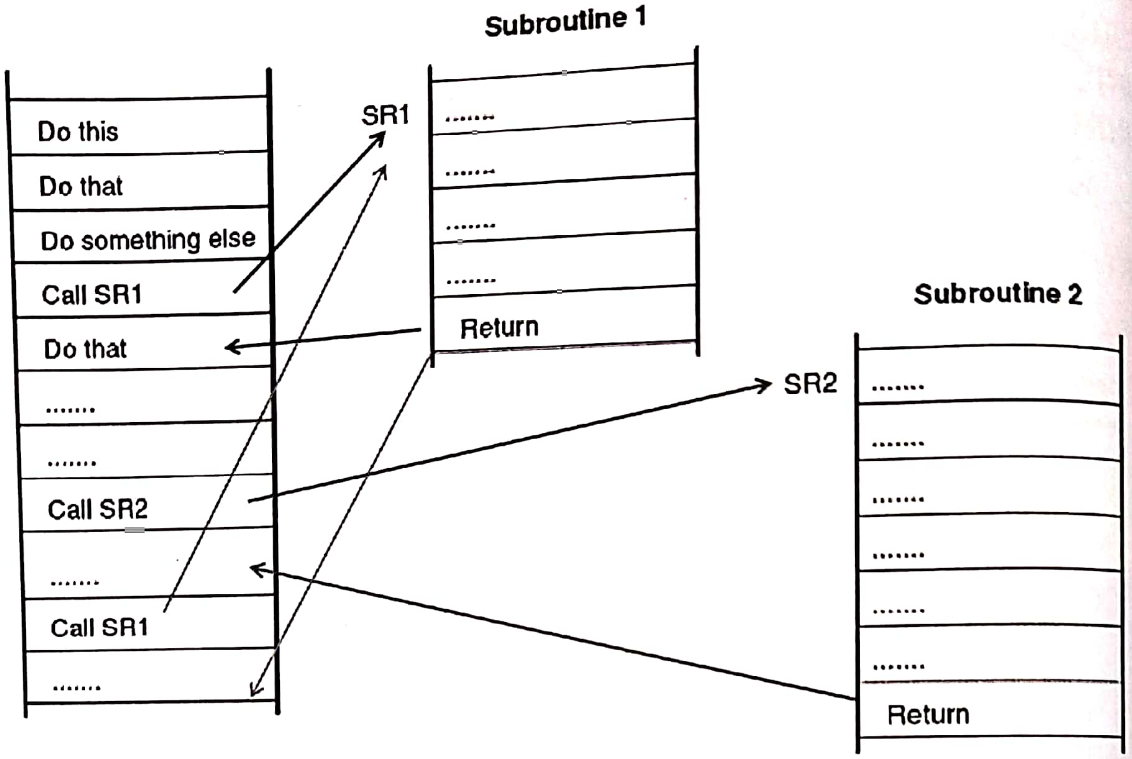
① Signed num ← تكون قيمة سالبة لما أزيدها كذا عدد معين من المرات توصل صفر.

② 1111 1111 ← لو عملتله inc رح يصير 0000 0000 وال carry رح يعمل وماحيتخزن.

Subroutines → Functions نفس مفهوم ال methods أو ال

- In many cases, we need to use a block of code in a program in different places
- Instead of writing the code wherever it is needed, we can use *subroutines/functions/procedures*
  - Block of code saved in memory and can be called/used from anywhere in the program → وبنناديه قد ما بدك
  - When a subroutine is called, execution moves to place where the subroutine is stored
  - Once the subroutine is executed, execution resumes from where it was before calling the subroutine

Subroutines


Subroutine 1

Subroutine 2

Do this
Do that
Do something else
Call SR1
Do that
....
....
Call SR2
....
Call SR1
....

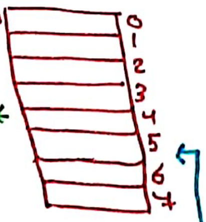
SR1
....
....
....
....
Return

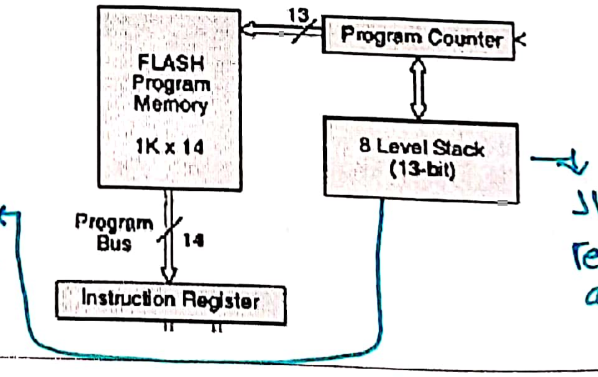
SR2
....
....
....
....
....
....
Return

Subroutines

- The ***program counter*** holds the address of the instruction to be executed
- In order to call a subroutine, the program counter has to be loaded with the address of the subroutine
- Before that, the current value of the PC is saved in ***stack*** to assure that the main program can continue execution from the following instruction once the execution of the subroutine is over

← Mem صغيرة بنخزن
8 Location وبنقدر أخزن
فيهم Values
* آخر Value خزنتها بالStack هو
أول Value بنطلعه





يخزن ال return address

Subroutines

- In PIC, to invoke a subroutine we use the **CALL** instruction followed by the address of the subroutine
- The address is usually *specified by a symbolic label in the program*

بأشر عال inst أسهل ما أكتب الها address لأول inst في ال Subroutines.

- To exit a subroutine and return to the main program, we use the ***RETURN*** or ***RETLW*** instructions

حنشوف شو فائدتها للأمام

Subroutines

* 4X3 = 4+4+4 or 3+3+3+3

- **Example 4:** Write a program that uses a subroutine to multiply the contents of locations 0x30 and 0x31 and then return the result in the working register.

→ Code of Subroutines

```
MuIT    CLRW
L1      ADDWF 0X30, W
        DEC FSZ  0X31, F
        Goto     L1
        Return
```

ممكن بدالها أعمل
DECF 0X31,F
MOVF 0X30,W
ولكن زعب أول أسهل

زي مثال القيمة 10x بحيس تقريب..

Subroutines - Example

```
STATUS      equ     0x03        ; define SFRs
            org     0x0000      ; reset vector
            goto    START
            org     0x0005
START       ........
            movlw   0x15        ; pass the first number
            movwf   0x30
            movlw   0x09        ; pass the second number
            movwf   0x31
            call    multiply    ; call the subroutine
            ........
            movlw   0x05        ; pass the first number
            movwf   0x30
            movlw   0x04        ; pass the second number
            movwf   0x31
            call    multiply    ; call the subroutine
            ........
DONE        goto    DONE        ; endless loop
```

← خزنت الأرقام اللي بدي أضربهم أول بأول.

Example - Continued

```
multiply    clrw
Repeat      addwf  0x30, 0    ; repeated  addition
            decfsz 0x31, 1    ; counter
            goto   repeat
            return

            end
```

← بسوي dec بالأول بعدين بسوي ال test

بعد ال clrw بعمل if لو كان 0x30=0 مايعمل شي ولو كان 0x31=0 برضه مايعمل اشي.

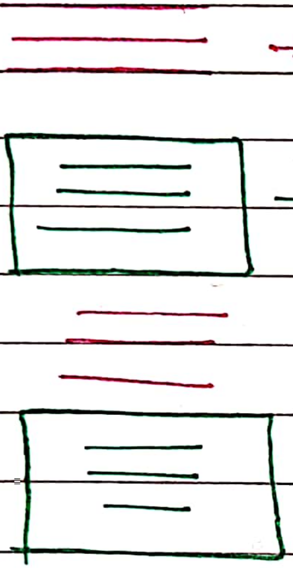
what if one of the numbers is zero?

الحل :

* M[0x30]=0 ← عادي وبطلع النتيجة صح بس بضيع وقت

* M[0x31]=0 ← لو كان ال counter لما يوصل لل dec حيصير بالماينس فبيجمع 256 مرة فهذا كلام غلط

* Slide 13 : The concept of Subroutines :



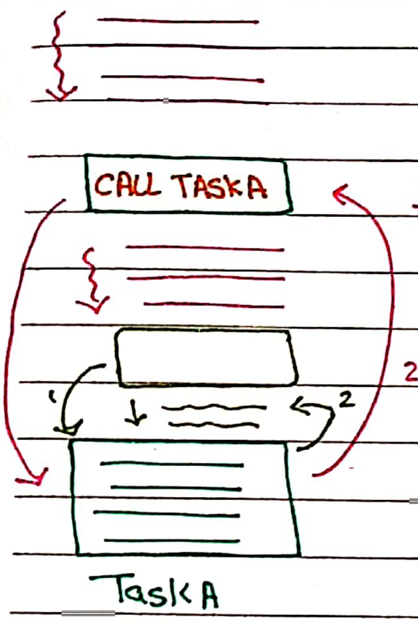
→ instructions → تقوم بعمليات معينة

→ Task A → مجموعة inst تقوم بوظائف مخصصة محددة

→ رجعت احتجت Task A

* روح ألاحظ إنه عندي كثير تكرار لإني بستخدم مجموعة Task A أكثر من مرة وهذا الكلام غير منطقي لإنه بستهلك Mem كثير.

الحل : أخلي Task A بمكان معين بال Mem inst وأصير أناديه كل ما أحتاجه.



* نحتاج أيضاً في الـ Hardware آلية تمكننا من الذهاب لـ Task A والرجوع لتنفيذ باقي الـ inst لكي يسير البرنامج بشكل صحيح.

* احتجت taskA بدي أروح أنفذها وأرجع أكمل باقي الـ inst ← والي استفدته إني حطيت taskA بمكان واحد فقط دون تكرار.

* الـ Code المسمى بـ Task A هي الـ Subroutines والـ Subroutines هي مجموعة من الـ inst مخزنة في الـ Mem كل ما أحتاجها بروحلها.

* من الـ PC بعرف وين وصلت بتنفيذ البرنامج
لـ Program counter

* لازم أغير الـ PC إذا كنت بدي أتنقل بين الـ Subroutions

* الـ CALL هي الـ inst اللي بتنادي الـ Subroutines : فهي بتاخذ الـ address تبع أول inst في الـ Subroutines وتخزنها في الـ PC وبس أخلص الـ Subroutines بدي أرجع الـ inst الرئيسية وهي (return) ← بترجعني للمكان الي لازم أكمل فيه بتغيير الـ PC

* الـ Return address أنا خسرته لما عملت CALL فعشان أقدر أرجعه لازم أكون محافظة عليه أو مخزنته بمكان عشان أقدر أرجعه للـ PC و الـ Stack هو reg يخزن الـ Mem address الـ inst بشكل مؤقت مثل الـ Return address.

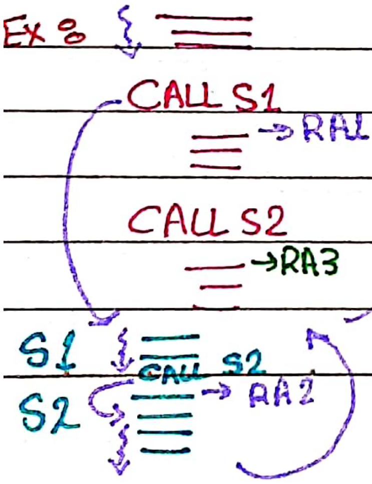
* CALL → ① Stores Return address in stack.
↳ ② Load PC with subroutines address.

* Return → Loads the return address into the PC.

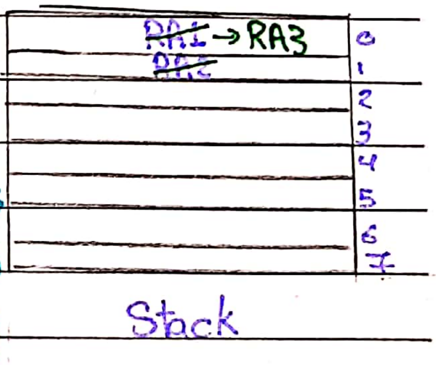
* الفائدة من استخدام الـ Subroutines : التقليل من التكرار و إعادة استخدام الـ code أكثر من مرة وهو ما يسمى بـ Code Reuse. والـ Modularity والتي تعني تقسيم البرنامج لأكثر من Subroutines والفائدة منه أنه بيسهل عملية الـ Code وتوزيع المهام.

Slide 15 : لماذا الـ Stack يحوي 8 locations

Ex :



← ففي أكثر من 1 level لأنه ممكن يصير عندي Subroutines داخل الـ Subroutines عشان ما أخسر أي RA أثناء التنفيذ.
(Nested subroutines)



PAGE

DATE

* ال Stack يحتمل 8 level فقط.

* EX: Main $\xrightarrow{0}$ S1 $\xrightarrow{1}$ S1 $\xrightarrow{2}$ S2 $\xrightarrow{3}$ S3 $\xrightarrow{4}$ S4 $\xrightarrow{5}$ S5 $\xrightarrow{6}$ S6 $\xrightarrow{7}$ S7

→ S8 ← لازم أخزنه بس ما ضل شي محل بال Stack فرح يرجع يأشر على 0

وهنا عادي بس لما أصير أرجع حيرجع للكل إلا من S1 إلى ال Main لأني خسرت ال value لأول RA.

فلازم أنا أنتبه كمبرمج إنه عدد ال Nested calls ما يتعدى 8.

* ال Stack أيضًا بيستخدم مع ال interrupts وسيشرح ذلك في CH6.

Slide 16: Example: * بنفضل نكتب ال Subroutines قبل ال main *

main ______ 0x0000
______ 1
______ 2
______ 3

CALL DELAY 4 → Machine code of this inst
______ 5

100	001 1111 0000

↳ opcode → من ال Sheet

→ address 11 bits

ok
go to ok

DELAY ______ 0x01F0

*when the CALL inst is executed, what we stored in stack? Return Address
⇒ 0x0005

END

* لازم نحط Return

لإنه لازم ال Subroutines تنتهي ب Return

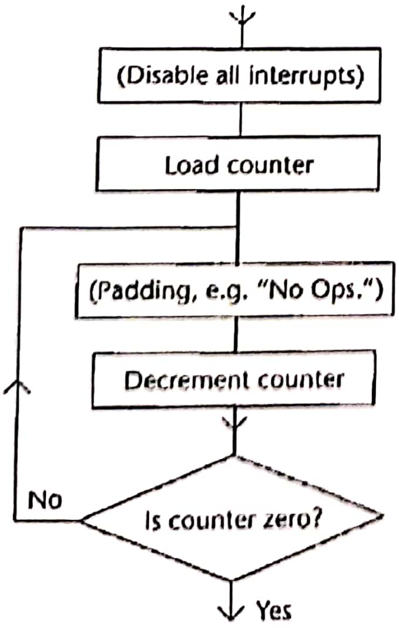
Generating Time Delays → نحتاجها في كثير من الحالات

- In many applications, it is required to delay the execution of some block of code; i.e. a time delay!
- In most microcontrollers this can be done by
  - Software → زي ماعلنا تنفيذ inst
  - Hardware (Timers) → ب CH6
- To generate time delay using software, *let the microcontroller execute non useful instructions* for certain number of times!
- If we know the clock frequency and the cycles to execute each instruction we can generate different delays

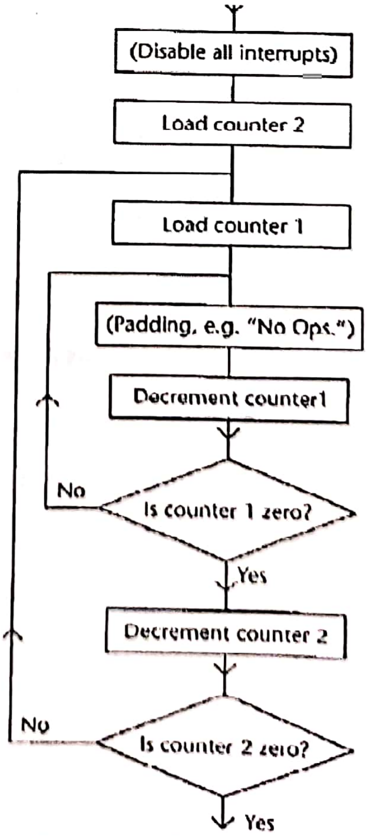
$$Delay = \#cycles \text{ x } clock\ cycle\ time$$
$$= \#\text{cycles x } 4 / F_{osc}$$

Generating Time Delays

- Structure of Delay Loops



One loop for small delays



Nested loops for large delays

Generating Time Delays

- **Example 5:** Determine the time required to execute the following code. Assume the clock frequency is 800KHz.

```
             1  movlw    D'200' ; initialize counter
             1  movwf    COUNTER
del      1   1  nop                ; main loop for delay
         1   1  nop
         2   1  decfsz   COUNTER, F
         0   2  goto     del
```

لفائدة إني بقدر استخدمها أكثر من مكان
وأيضًا في generate multiple يعني لو بدي
50 Msec بناديه 10 مرات وخلص

Subroutines لو بتكون ← return

- What if this code to be used as a subroutine??!!

Time = $(1+1 + \underset{200-1}{199}(5) + 4) * \frac{4}{800*10^3}$ = 5.005 msec

Call ↩ ②+1+1+② ↩ return

CALL DELAY ≡ CALL DELAY

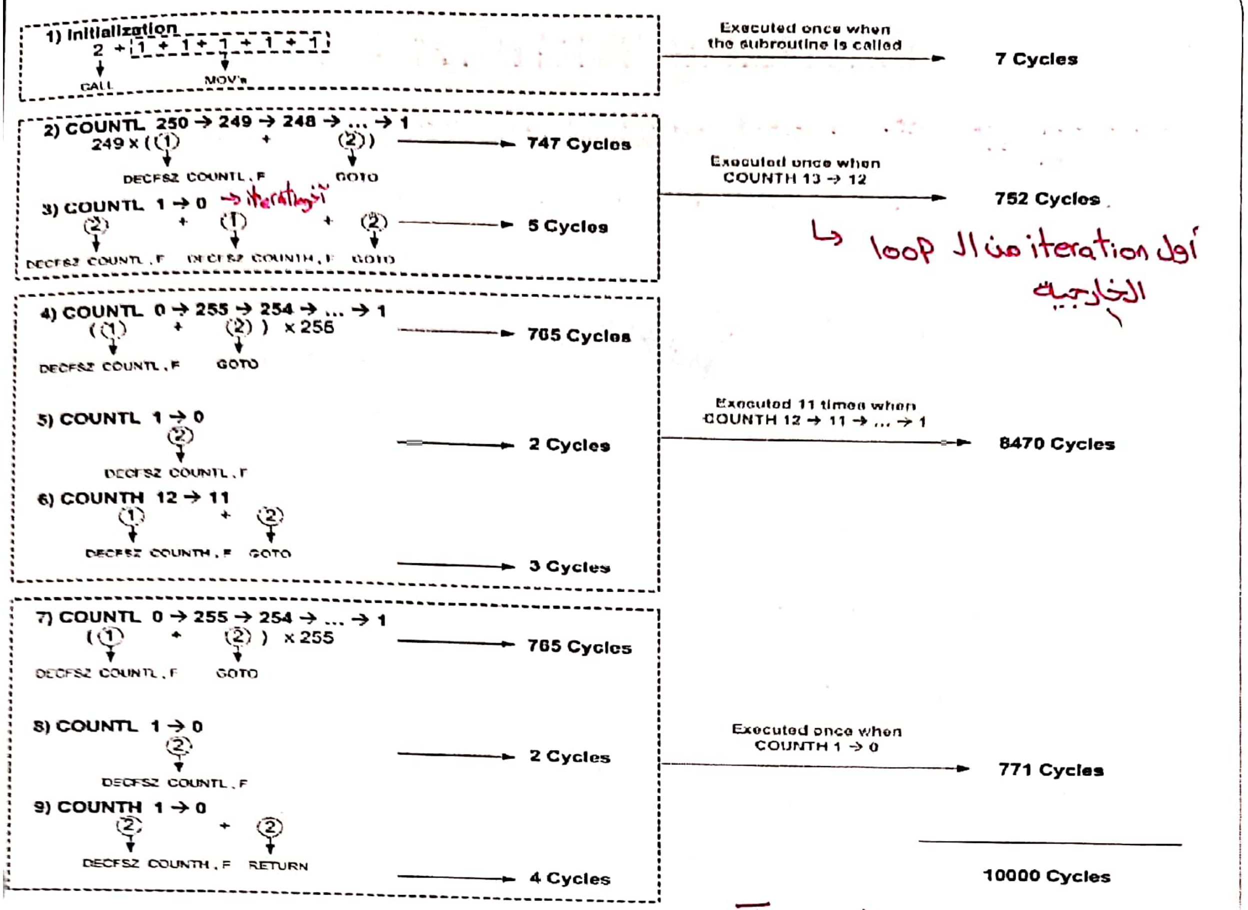
Generating Time Delays

- **Example 6:** Analyze the following subroutine and show how it can be used to generate a delay of 10 ms exactly including the call instruction. Assume 4 MHz clock frequency

```
TenMs  nop                         ; beginning of subroutine
       movlw   D'13'
       movwf   COUNTH → outer loop
       movlw   D'250'
       movwf   COUNTL → Inner loop
Ten1   decfsz  COUNTL , F          ; inner loop   255 → 0
       goto    Ten1                               0 → 255 → ---→ 0
       decfsz  COUNTH , F          ; outer loop
       goto    Ten1
       return
```

عم ترجع هون يعني كم تتكرر 256 من 0 → 255

Scanned with CamScanner



1) Initialization
2 + 1 + 1 + 1 + 1 + 1
CALL MOV's
→ Executed once when the subroutine is called → **7 Cycles**

2) COUNTL 250 → 249 → 248 → ... → 1
249 x (① + ②) → **747 Cycles**
DECFSZ COUNTL , F GOTO

3) COUNTL 1 → 0 → iteration آخر
② + ① + ② → **5 Cycles**
DECFSZ COUNTL , F DECFSZ COUNTH , F GOTO

→ Executed once when COUNTH 13 → 12 → **752 Cycles**

↳ أول iteration من ال loop الخارجية

4) COUNTL 0 → 255 → 254 → ... → 1
(① + ②) x 256 → **765 Cycles**
DECFSZ COUNTL , F GOTO

5) COUNTL 1 → 0
② → **2 Cycles**
DECFSZ COUNTL , F

6) COUNTH 12 → 11
① + ② → **3 Cycles**
DECFSZ COUNTH , F GOTO

→ Executed 11 times when COUNTH 12 → 11 → ... → 1 → **8470 Cycles**

7) COUNTL 0 → 255 → 254 → ... → 1
(① + ②) x 255 → **765 Cycles**
DECFSZ COUNTL , F GOTO

8) COUNTL 1 → 0
② → **2 Cycles**
DECFSZ COUNTL , F

9) COUNTH 1 → 0
② + ② → **4 Cycles**
DECFSZ COUNTH , F RETURN

→ Executed once when COUNTH 1 → 0 → **771 Cycles**

10000 Cycles

Time = 10000 * 1 μsec = 10ms

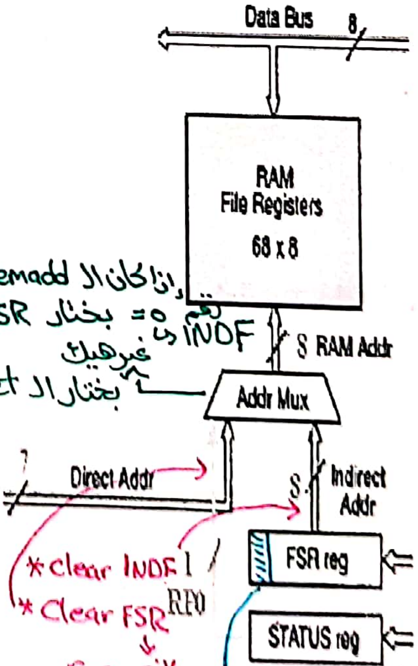
* movwf 0x40 → | | f: 100 0000 | → Direct addressing → ال address يخزن كجزء من ال inst

Working with Data

Indirect Addressing → How the cpu addresses the Data Mem or FR

- Direct addressing is capable of accessing single bytes of data
- Working with list of values using direct addressing is inconvenient since the address is part of the instruction
- Instead, we can use indirect addressing where
  - The File Select Register FSR register acts as a pointer to data location.
  - The FSR can be incremented or decremented to change the address
- The value stored in FSR is used to address the memory whenever the INDF (0x00) register is accessed in an instruction
- This forces the CPU to use the FSR register to address memory

ما بناخذ رقم 0 لأنه ما فيه اشي
بنفهم انه بدنا نسوي indirect address



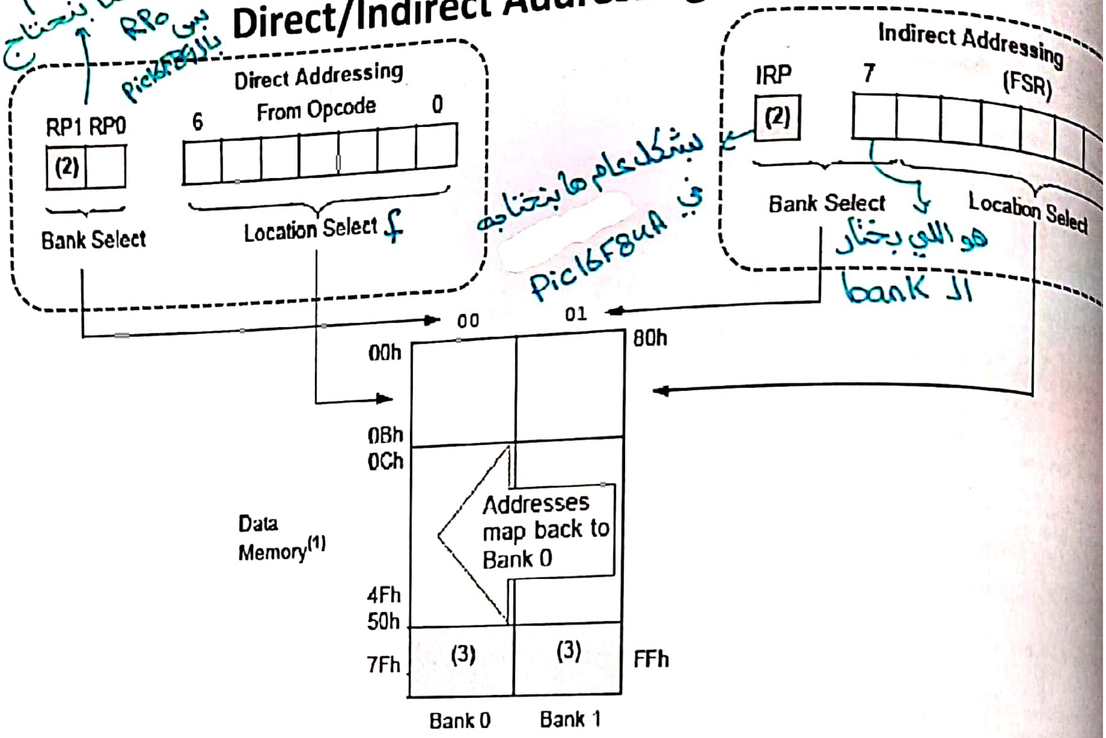
اذا كان ال Memadd رقم 0 = INDF بختار FSR
غيرهيك بختار ال direct

* clear INDF
* Clear FSR
لأنه reg رقمه 4 مش 0

في ال indirect ال most sig من FSR هو ال bank bit Selection

Working with Data

Direct/Indirect Addressing in 16F84A



بشكل عام بس احنا بنحتاج بس RP0 بال Pic16F84A

بشكل عام ها بنحتاجه في Pic16F84A

هو اللي بختار ال bank

Note 1: For memory map detail, see Figure 2-2.

2: Maintain as clear for upward compatibility with future products.

3: Not implemented.

Working with Data

- **Example 7: A program to add the values found locations 0x10 throug 0x1F and stores the result in 0x20**

```
STATUS      equu    0x03        ; define SFRs
FSR         equ     0x04
INDF        equ     0x00
RESULT      equ     0x20
N           equ     D'15'
COUNTER     equ     0x21
            org     0x0000      ; reset vector
            goto    START
            org     0x0005
START       movlw   N
            movwf   COUNTER     ; initialize counter
            movlw   0x11
            movwf   FSR         ; initialize FSR as a pointer
            movf    0x10 , W
LOOP        addwf   INDF , W    ; get 1st number in W
            incf    FSR , F     ;  add using indirect addressing
            decfsz  COUNTER , F ;  point to next location
            goto    LOOP        ; decrement counter
            movwf   RESULT
DONE        goto    DONE
            end
```

UP موجودين بال 2 Banks فما بحتاج أقلب عال Bank

بنا نعدل عال code ليصير يجمع $0+v_1+v_2+\dots+v_{16}$ (16 ad

حيصير عال code كالتالي ←

D'15' → D'16'

0x11 → 0x10

بقدر أشيل هدول لانو ما بستخدم interrupt

بتشال وبدالها clr W or movlw 0

تعادل addwf 0x11,w بأول مرة وهكذا

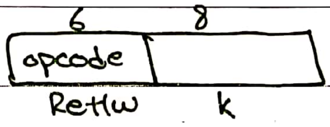
انتبه لازم تخزنها عال F لو كتبت w حيضل ال FSR نفسه

Working with Data

Look-up Tables →data المتعلقة بالتعامل مع ال

- A look-up table is a block of data held in the program memory that the program accesses and uses
- The **movlw** instruction allows us to embed one byte within the instruction and use ! How about a look-up table ?
- In PIC, look-up tables are defined as a subroutine inside which is a group of **retlw** instructions
- The **retlw** instruction is similar to the return instruction; however, it has one operand which is an 8-bit literal that is placed in **W** after the subroutine returns
- In order to choose one of the **retlw** instructions in the look-up table, the program counter is modified to point to the desired instruction by changing the value in the **PCL** register **(0x02)**
- The **PCL** register holds the lower 8 bits of the program counter

∴ RETLW D'10' →



Working with Data

- **Example 8: A subroutine to implement a look-up table for the squares of numbers 0 through 5. To compute the square, place the number is stored in W before calling the subroutine SQR_TABLE.**

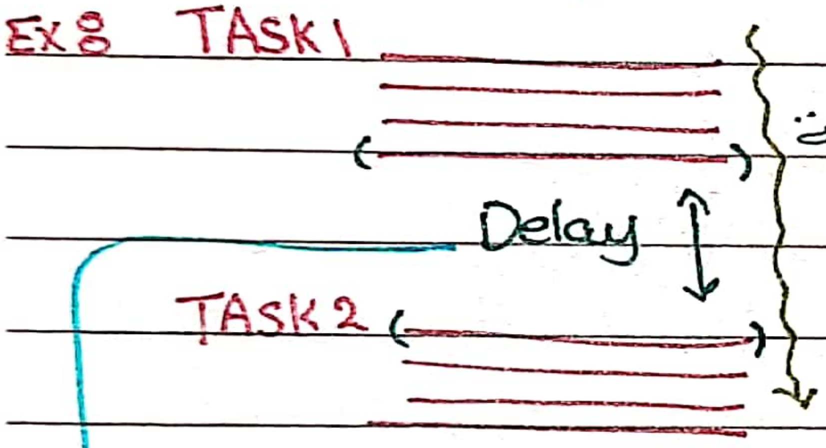
```
SQR_TABLE    addwf   PCL , 1 ; modify the PCL to point the
                             ; required instruction
             retlw   D'0'    ;  square value of 0
             retlw   D'1'    ;  square value of 1
             retlw   D'4'    ;  square value of 2
             retlw   D'9'    ;  square value of 3
             retlw   D'16'   ;  square value of 4
             retlw   D'25'   ;  square value of 5
```

; Remember that the PC always points to the instruction to be executed

Summary

- Building complex programs requires putting down it requirements and design
- Programs tends to execute instructions sequentially unless branching or subroutines are used
- A subroutine is piece of code that can be called from anywhere inside the program
- A simple way to generate time delays is to use delay loops

Slide 20 : Concept of time delay.



* لما ينفذ البرنامج تبعي داخل ال System حيمشي هيك ←

* اكن حسب طبيعة البرنامج تبعي مطلوب مني تأخير بداية ال TASK الثانية لبعض الوقت بعد تنفيذ الأولى. ←

وهنا هو موضوع ال Time Delay

* تأخير ال inst داخل ال cpu لا يأخذ زمن صفر أكيد.

* أي inst بنفذها تحتاج Some time وبال sheet بنبين كل inst كم cycle بدها

ومعظمهم 1 cycle وفي 2 cycles وفي إما 1 أو 2.

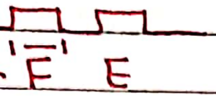
* بحتاج هون إني أحط مجموعة من ال inst والمفروض ما يأثروا على ال state تاعت البرنامج ←

لأنهم مو جزء من ال TASK المطلوب تنفيذها وإحداث تغيرات فيها هي فقط للتأخير.

PAGE

DATE

*Note :



→ cycle time

1 cycle ⇒ $T_{cy} = \frac{4}{F_{osc}}$

* inst to generating time delays :

* Nop → ideal بناخذ 1 cycle وما بتأثر عال FR أو WR وما إلها ذلك فهي

↳ 1 cycle → 1 Tcy → if Fosc = 4MHz → Tcy = 1 Msec

* فرضًا بدي أأخر TASK2 عن TASK1 بالخلف مقدار 5 Msec

↳ Time = # cycles * Tcy = # Nop * 1 Msec

⇒ # Nop inst = 5 Nop inst

*مش ضروري دايمًا Nop ولكن احنا حاليًا بنستخدموا.

* فرضًا بدي الـ delay = 500 Msec عافتراض Fosc = 4MHz

↳ وهذا الكلام مش منطقي → # Nop = 500 inst ⇒

وضخم فالحل نعمل for loop أو إني استخدم غير الـ Nop ولكن أضمن إنه اللي استخدمه ما يأثر عالبرنامج.

⇒ Loops : TASK 1

movlw N → 1cycle

movwf counter → 1 cycle

delay decfsz counter, F → 1 cycles

goto delay → 2cycles

TASK2

1 cycle بناخذ ← sequential : False

لو كملت

2 cycles بناخذ ← True : لو كانت =0

وعملت skip لها ← الشغل عليها دايمًا

هون لازم أعلم أول وهم بالبداية ← initialization لـ Some FR اسمه counter وبدي أسوي عليه N-1 decrement ليوصل صفر. هون ما في Nop بس ممكن نعمل code نستخدم في Nop.

→ Time = #cycles * Tcy

#cycles = overhead cycles + # iterations * cycles per interations

↳ الـ inst اللي ما بنتكرر بس بحتاج أحطها لأسوي الـ delay مثل :

movlw N

movwf counter

↳ تكرارات

↳ الـ cycle لكل تكرار

* أي inst بتغير مسار البرنامج بتاخذ 2 cycles ولو ما بتغير مسار البرنامج بناخذ 1 cycle

* بعدين آخر decfsz رح تكون تاخذ 2 cycles لإنها صارت True والـ goto وظيفتها حتكون 0 cycle. وهنا حيتكرر مرة وحدة بس اللي هي آخر iteration.

decfsz ، goto

$$\#cycles = \underbrace{1+1}_{overhead} + (1+2)(N-1) + 1*(2+0)$$

الـ iteration كلها ما عدا آخر وحدة

آخر iteration

→ decfsz

Scanned with CamScanner

$500 * 10^{-6} = [2 + (N-1)(3) + 2] \times 10^{-6} \Rightarrow N = 166.333$ بدي أوجد N :

$\downarrow$ Delay = 5 Msec $\downarrow$ Tcy = 1 Msec

$\downarrow$ عدد المرات اللي حنتكرر فيها ال loop هي يا إما 166 أو 167

delay > 500 Msec / delay < 500 Msec

$\downarrow$ بشوي $\downarrow$ بشوي

⇒ Note : Delay = 1000 Msec] في المثال السابق

Fosc = 4 MHz ⇒ Tcy = 1 Msec

$\Rightarrow 1000\ \text{Msec} = 2 + (N-1) * 3 + 2 \Rightarrow N = \underline{333}$

حخزن هالقد في Counter أول ما أبلش .

* أكبر Value بقدر أحطها في 8 bit هي 255 فما حقدر أمثل 333 ب 8 bit

* Time = # cycles * Tcy → if i want to increase the time:

- ↳ Tcy → بس هاي صعبة ألعب فيها
- ↳ cycles → أغير بعدد ال cycles
  - ① Put Nop inside the loop
  - ② Nested Loops

①

```
         Movlw N           1
         Movwf counter     1
delay    Nop               1 -> أول    1 -> آخر iteration
         Nop               1           1
         decfsz counter,F  1           2 -> بطلت ال loop فاضية .
         goto delay        2           0
                           N-1         1
```

⇒ Time = # cycles * Tcy / # cycles = overhead + # iteration * cycles / iteration

$\# cycles = 1 + 1 + (N-1)(1+1+1+2) + 1(1+1+2+0)$

$\Rightarrow 1000 * 10^{-6} = (6 + (N-1)(5)) * Tcy \Rightarrow N = 199.8 \rightarrow 199$ ↳ 200] → وهنول الرقمين بقدر أخزنهم ب 8 bit

↳ أي هنيم أفضل بعتمد عالبرنامج ومرات بحتاج أجيب رقم بالزبط بدون فواصل وبجيبه عن طريق تغير أو إضافة عالكود .

② Nested Loops :   Ex : Fosc = 8 MHz → Tcy = 0.5 μsec

```
      movlw  D'20'             1
      movwf  count_outer       1
LX    movlw  D'30'             1
      movwf  count_inner       1
L1    decfsz count_inner       داخلية loop → 1  or  2
        goto  L1                             2       0
      decfsz count_outer    1 2
        goto  LX            2 0
```

$$1+1+\underbrace{\left[(1+1+(1+2)*29+(2+0+1+2)*1)\right]}_{\text{هذا الكلام يتكرر ١٩ مرة}}+\underbrace{\left[1+1+(1+2)\times 29+2+0+2+0\right]}_{\text{تكرر آخر مرة}}$$

$= 1881 \text{ cycles} \Rightarrow \text{Time} = 1881 * 0.5 * 10^{-6} = 940.5\ \mu\text{sec}$

↳ قريب من 1000 ← بقعد أعدل وأجرب لو بدي لأوصل 1000

* مهم : بأي loop عدد الـ cycles هو نفسه لكل الـ iterations ما عدا آخر iteration

لإنه بصيرفيها القرار, اذا الـ loop حتعمل Skip أو لأ.

* Note :

```
L   decfsz X,F   1
    goto L       2
```

→ #cycles = (1+2) x N → بالتقريب

↳ #cycles = (1+2)(N-1)+(2+0)*1 → بالزبط

Nop → لو حطيت Nop هون حيصير دائما 1+2 أو 2+1 فيعني نفس الحسبة وهنا بيرجعلك وانت بتكتب الكود كيف تكتبه لتسهل عليك الحسبة

Slide 25 8

Ex 8 clear all locations 0X20 to 0X2F → write group of inst to make this.

* Clrf 0X20
Clrf 0X21 →
⋮
clrf 0X2F

هذه طريقة سطحية ولا نستخدم ← الأفضل يكتب على شكل loop

```
    movlw  D'16'
    movwf  CNT
L   Clrf   (Variable!)
    decfsz CNT,F
    goto   L
```

ليش الـ address اللي لازم أحطه هون ؟ لو فكرت فقط بالـ direct address ما حينفع لأنه الـ address لو خزنته بالـ inst بالـ direct address حيصير ثابت ما حيتغير وأنا بدي ياه Variable فهون بيجي دور الـ indirect addressing

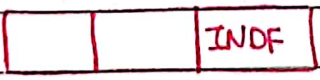
FSR ← file select Register : أنا كـ programer بقدر أخزن Value جوا هاد الـ FSR بحيث إنه بطريقة معينة بقدر أفهم الـ CPU إنه ما تاخذ الـ address اللي بتخزن بالـ inst خذي الـ address اللي موجود داخل الـ FSR ، وما دام الـ address صار بإمكاني أخزنه بـ Register بقدر إغيره جوا الـ loop .

⇒ movlw D'16'
movwf CNT
movlw 0X20
movwf FSR
L clrf F~~S~~R → INDF →
Incf FSR,F → لأني حغيره ليه

في مشكلة بكون هون بإنه هاي الـ inst لما تتحول لـ machine code رح تتحول لـ opcode معين وح يتخزن هون ↶ ومعنى هذا

decfsz CNT,F
goto L

الكلام إنه بسوي الـ clear للـ FSR نفسه بس أنا ما بدي هيك ، فبدي طريقة تفهم الـ CPU إنو بدنا ناخذ محتويات الـ FSR ونستخدمها كـ Memory address وهون بيجي دور الـ INDF reg ⇆ 0X00 0X80 هو فعليا مش reg حقيقي هو عبارة عن رقم



← حتفهم الـ CPU إني حستخدم محتويات الـ FSR حتى أسوي Memory address

* indirect addressing : هي طريقة بتمكني إني أغير الـ Memory address اللي عم بتعامل معه في الـ data Memory ، إني أخلي هذا الـ address ← Variable وهذا الكلام بحتاجه لما أسوي access على list of Numbers

PAGE
DATE

*Slide 25 :

Addressing : كيف أشكال الaddress تبع الLocation اللي بدي أحكي معاه بال data Memory منوجهة نظر الـ pic16 .

Direct Addressing : اللي كنا نستخدمه من البداية؛ أي inst بنحكي مع الـ data Memory في pic16 micro... فعليا بنحط الaddress اللي بدها تحكي معاه داخل الـ inst نفسها. والaddress لو تخزن بال inst بكون ثابت ما بقدر أغيره .

InDirect Addressing : هي اللي وظيفتها تحللي المشكلة لو بدي أغير أو أبدل عال address .

* Movwf INDF → معناها إنه cpu تروح على الregister FSR واستخدمي محتوياته كـ Memory address فهو لو كان الFSR فيه 0X20 فهاي الinst تقابل Movwf 0x20 .

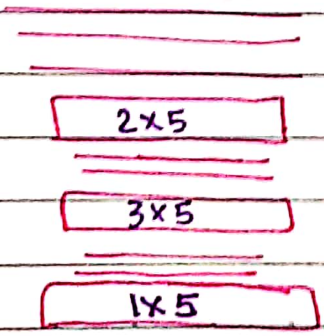
* Movwf FSR → هاي الinst عبارة عن move لل WR على FR في direct والaddress ال FSR هو 0x04 فهاي ال address اللي بحكي معاه مش 0x00 بالتالي يكون direct address لأنه الحالة الوحيدة اللي بتفهم ال cpu إنها indirect بال address 0x00 .

Slide27 :

```
movf   0x10, w
addwf  0x11,w
addwf  0x12,w
addwf  0x13,w
   ⋮
addwf  0X1F,w
movwf  0X20
```

→ هون لو فكرنا بطريقة
سطحية ولكن هيك حزن
(add inst 15)
فالحل يكون بال loop
وال indirect

Slide 28 : نتخيل إنه بدنا نكتب برنامج للPic microcontroller وهذا البرنامج بسوي شغلات كثير ومن الشغلات اللي بحتاجها هذا البرنامج حتى ينفذ في أماكن متعددة أو في مكان واحد يحتاج يحسب حاصل ضرب رقم معين في 5 :



→ بحتاج إني أضرب أرقام معينة في الرقم 5

→

* إذا بعرف إنه الأرقام اللي حضربها بـ 5 معروفة وبـ range بقدر أستخدم الـ Subroutines

Ex : 0	0
1	5
②	⑩
3	15
④	⑳
5	25

← بدل ما أحتاج إني كل ما بدي أضرب رقم بـ 5 أنادي Subroutine جواها for loop وتسوي العملية وترجعلي الجواب. وأنا بعرف range الأرقام اللي عندي فليوما أسوي table زي هاد ← ولو احتجت ناتج ضرب رقم منهم في 5 بكل بساطة بروح كرقم السطر بالجدول. (حجم الـ table معروف عندي)

وهنا ما يسمى في look up table : بسوي mapping من input value لـ output value

* بدي أخزن الـ table ثم أعمل access للـ table .

- كيف أخزن الـ table :

• أفضل مكان لتخزين الـ table في الـ data Memory لأنها data .

• بحتاج أخزن one column اللي هو الـ output column فقط .

↑ باشر هون

```
Movlw  0    ← FSR
movwf  0x10    +0
movlw  D'5'
movwf  0x11    +1
movlw  D'10'
movwf  0x12    +2
movlw  D'15'
movwf  0x13    +3
movlw  D'20'   +4
movwf  0x14
movlw  D'25'   +5
movwf  0x15
```

← خزنت column الـ output هيك خزنت الـ table

Storing ← (12 inst)

* هنا الكلام لطيف لو كانت الـ table صغيرة، لكن لو كبرت بصير الموضوع صعب، وهاي الطريقة ما بنشرح وبنتعمل لإنه حجم الـ Data Mem مو كبير كفاية لأخزن كل هنا. الطريقة في الصفحة التالية.

accessing (5 inst)

↑

```
Movlw   0x10
Movwf   FSR
movf    0x30,w
addwf   FSR,F
movf    INDF,w
```

عشان يوخذ إنه بدي قيمة FSR ← movf INDF,w

↑

- كيف أعمل الـ access للـ table :

• لوحد حكالي بدي نستخدم هنا الـ table حتى تحسبلي ناتج ضرب الرقم الموجود في location ← 0x30 في 5 . فعليًا ما بعرف الرقم اللي بـ 0x30 لسا.

PAGE
DATE

* الـ look up table في الـ microcontroller تعرف على شكل Subroutines ولكن مختلف عن اللي أخذناه، مكتوب بطريقة سهلة و إذا أسوي عليه access برضه بطريقة سهلة.

* الـ Subroutines في الـ look up table هي عبارة عن مجموعة من special return inst

* retlw → بناخذ الـ Value اللي موجودة بالـ Stack كـ address وبتحطها بالـ PC وبتكمل execute هناك، زي الـ return، لكنها بتسوي عملية إضافية يعني بترجع عالـ main prog وبنفس الوقت بتخزن الـ Value بالـ working register والـ Value بتحددها إنت.

MUIT5

0 →	RETLW D'0'
1 →	RETLW D'5'
2 →	RETLW D'10'
3 →	RETLW D'15'
4 →	RETLW D'20'
5 →	RETLW D'25'

(Storing)

- الـ look up table → عبارة عن Subroutine ببساطة مكون من مجموعة من retlw بحيث كل Entry في الـ table في إلها 1 inst.
- لما أنادي هنا الـ Subroutine بدي إياه يرجعلي وحدة من الـ Values الموجودة بناء على الـ input Value.
- تخزنوا على شكل inst في الـ inst Mem وهذا منطقي لإنوا أكبر من الـ Data Mem.
- بسوي access على هذا الـ Table عن طريق إني أعمل CALL للـ Subroutine:

بدي تجيبلي حاصل ضرب 3*5 واللي هي 15 تخزن في WR ←	Movlw D'3'
(Access = 2)	CALL MUIT5

or

Movf 0X30,w ← هون بس متغير الـ Value
CALL MUIT5

- كيف بدي بس أدخل عالـ Subroutine أنط لأوصل الـ 3 ؟ لازم أغير من مسار البرنامج عند تنفيذ الـ Subroutine. واللي بتحكم بمسار البرنامج هو الـ Programm counter:

MUIT5	(addwf PCL,F)
PC ⇒ +0 0 →	RETLW D'0'
+1 1 →	RETLW D'5'
+2 2 →	RETLW D'10'
+3 3 →	RETLW D'15'
+4 4 →	RETLW D'20'
+4 5 →	RETLW D'25'

دائمًا بأشر عالـ Next inst

* فعليا أنا بجمع الـ PC الـ Value اللي بمررها للـ Subroutine
→ PC = PC + W

* addwf PC,F → ما بقدر أكتبه لإنه الـ PC مش مخزن بالـ data Mem مخزن بالـ inst Mem.

* PCL → Lower 8 bits of Program counter :

→ (Storing = 7)

(7 + 2 = 9) → مقارنة باللي قبل 17

← 10 Bit →
| | PCL |

Pc ↳ مخزن بالـ data Mem

→

Scanned with CamScanner

- من اللحظة اللي ببلش فيها بهاي الـ inst = (movlw D'3') للحظة اللي برجع فيها بعاد السيناريو كم cycle بحتاج ؟ ⇐

- أي inst بتغير مسار البرنامج بتاخذ 2 وأي inst ما بتغير مسار الـ inst بتاخذ 1

$$\Rightarrow \underset{\text{Movlw D'3'}}{1} + \underset{\text{CALL MUIT5}}{2} + \underset{\text{addwf PCL,F}}{2} + \underset{\text{RETLW D'15'}}{2} = 7 \text{ cycles}$$

فعليا أخذت 2 cycles لإنها بتغير الـ PC

*أي inst بتغير أو بتكتب عالـ PC بتاخذ 2 cycles حتى لو ما غيرت مسار البرنامج . (مهمة جدًا)

* بدك تنتبه لما تكتب : addwf PCL,F

لـ هاي لازم تكون F لإنو بدي أغير الـ PCL لو كتبت W ما رح يعدل على الـ PC .

* أكبر table بقدر أخزنها في الـ Program Mem حجمها 256 بايت ببلش تخزين

الـ table من $\underbrace{XX}_{\text{upper Bit of الـ PC}}\ \underbrace{00000000}_{\text{lower 8 bit of PC اللي بستخدمو}}$ ⟵ XX 0000 0000 → XX 1111 1111

* الـ look up table مش ضروري دايما يكون mapping لعلاقات رياضية عادي ممكن يكون هيك :

0 → w

1 → α

3 → !

⋮

Working with Time: Interrupts, Counters, and Timers

Chapter 6

Dr. Iyad Jafar

Outline

- Introduction
- Interrupts
- Timer/Counter → Timer 0 module
- Watchdog Timer
- Sleep Mode
- Summary

---

* بستخدموا ال Polling مرات لما ما يفرق معهم ال Time أو ما بدهم يكتبوا Code لل interrupts أو التشغلين .

Introduction

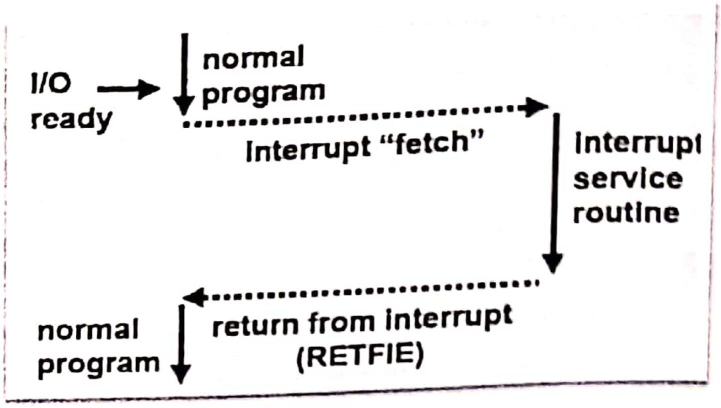
- Microcontroller should be able to deal with time
  - Respond in a timely manner to external events → الاستجابة في الوقت المناسب
  - حساب Measure time between events
  - Generate time-based activity → duration of some activity

قياس

- لهذا الغرض For this purpose, microcontrollers are usually provided with timers and support interrupts

Interrupts

- An interrupt is an event that causes the microcontroller t halt the normal flow of the program and execute anothe program called the interrupt service routine (ISR)



- Interrupts can be thought of as *hardware-initiated subroutine calls*
- Usually, interrupts are generated by I/O devices such as timers or external devices

Interrupts vs. Polling

- **Advantages**
  - Immediate response to I/O service request
  - Normal execution continues until it is known that I/O service is needed

ما بضيع وقت وأنا بسأل

- **Disadvantages**
  - Coding complexity for interrupt service routines
  - Extra hardware needed
  - Processor's interrupt system I/O device must generate an interrupt request

additional hardware

General Hardware Structure for Interrupts

events → تقاطع عمل ال cpu داخلي أو خارجي → →

- Interrupts sources can be *external and internal*

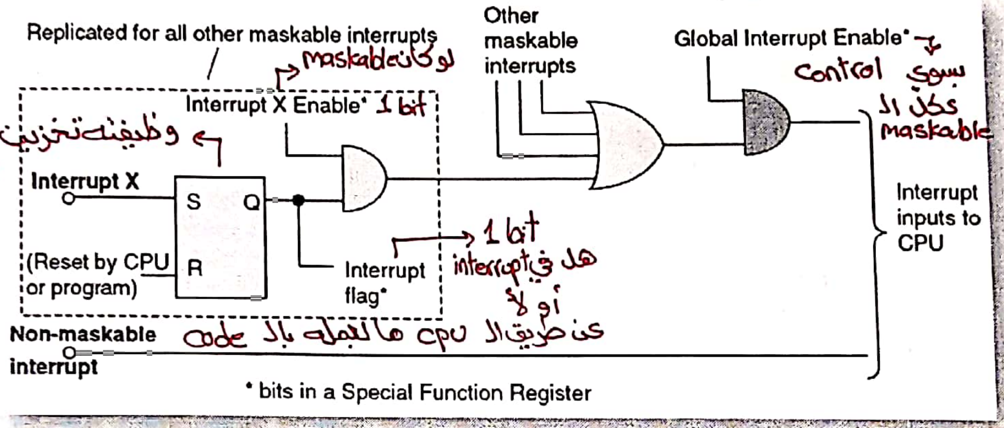
MC

- Two types of interrupts : *maskable* and *non-maskable*
  - Maskable can be enabled/disabled by setting/clearing some bits → أنا ك programer بقدر أحدد هل هذا ال interrupts مسموح
  - Non-maskable interrupts can not be disabled and they always interrupt the CPU → ما بقدر أهمل فيها IRQ

  الو أنه بيقاطع ال cpu أثناء تنفيذ ال code الأصلي

- Usually, each interrupt has a flag (a bit) that is set whenever the interrupt occurs

* ال pic الي بتعامل معها ال interrupt فيها maskable *



Ex:

← IRQ

لا ممكن أهمله لو كان disabled وآخذ فيه لو كان enabled

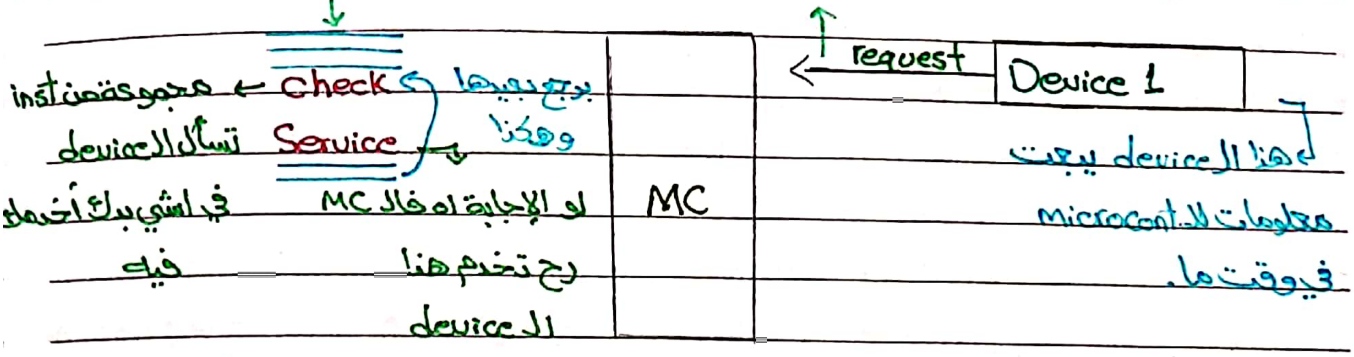
لـ شكل مبسط لل hardware الي بحتاجه.

The 16F84A Interrupt Structure

- **Sources of interrupts**
  - ***External interrupt*** INT
    - The only external interrupt input
    - The input is multiplexed with RB0 pin of port B
    - It is edge triggered
  - ***Timer overflow interrupt***
    - It is an internal interrupt that occurs when the 8-bit timer overflows
  - ***Port B on change interrupt***
    - An interrupt occurs when a change is detected on any of the upper 4 bits of port B
  - ***EEPROM write complete interrupt***
    - Internal interrupt that occurs when EEPROM finishes writing

Slide3 :

ال MC لما توصلها هاي الإشارة من ال device1 بتجاوب معاه وبتروح تسوي شي معين أنا ببرمجه.



مجموعة من inst ← Check

تسأل ال device Service

برجع بعيدها وهكذا

لو الإجابة اه فال MC رح تخدم هنا ال device

في اشي بدك أخدمك فيه

هنا ال device يبعت معلومات للـ microcont في وقت ما.

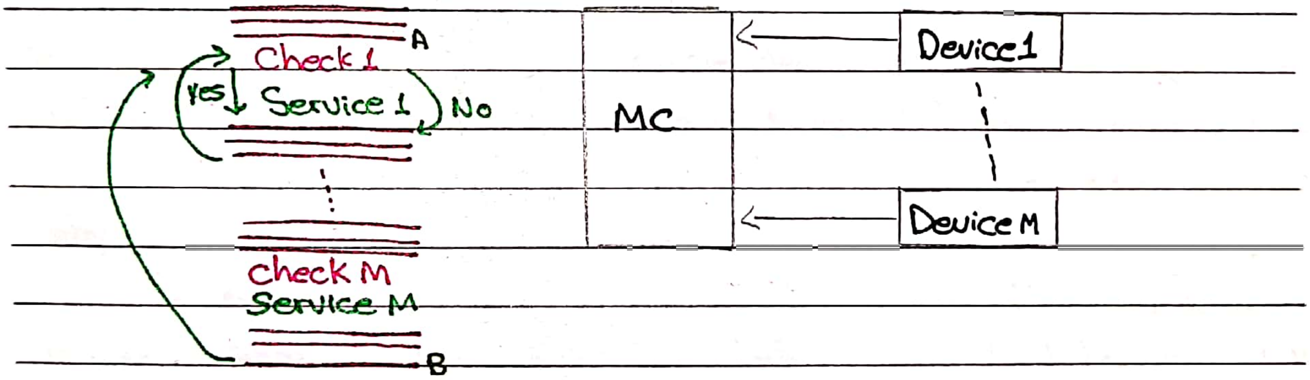
عادة ال Microcontroller بتكون موجودة في Embedded system و في ال Embedded system في component أخرى. وظيفة ال Microcontroller إنها نتخاطب مع هاي الأجهزة وتتحكم فيها.

ما مجموعة من ال inst لو لأ ما بتخدم وبتضل تكرر ال check كل فترة

* ال microcontroller ما بتعرف متى ال Device بتحتاج الخدمة منها فعشان هيك كل فترة زمنية لازم تضل تسأله محتاج الخدمة أو لأ بشكل متكرر.

* هذه الطريقة في تخاطب ال MC مع ال Devices تسمى : Polling

* لو كان عندي أكثر من Device بروح بنفس المبدأ : ←



* Polling : Simple and logical (بسيطة ومنطقية) ← ميزاته

* مشكلة ال Polling : * Delayed Response من ال MC ،

request ← ما رح يقدر يشوفه إلا بنهاية الخط الزهري وخليك في delay كثير كبير ممكن يخسرني ال request ، وكل ما زاد عدد ال devices بضيع الوقت أكثر وأكثر لو ما كان محتاج خدمة من ال MC

* wastes MC time

* ممكن هاي الطريقة لل interfacing نستخدمها في بعض المرات لو كانت هاي المشاكل تأثيره مو مهمة في برنامجي.. البديل لل Polling هو ال interrupts

PAGE
DATE

Slide 4 :

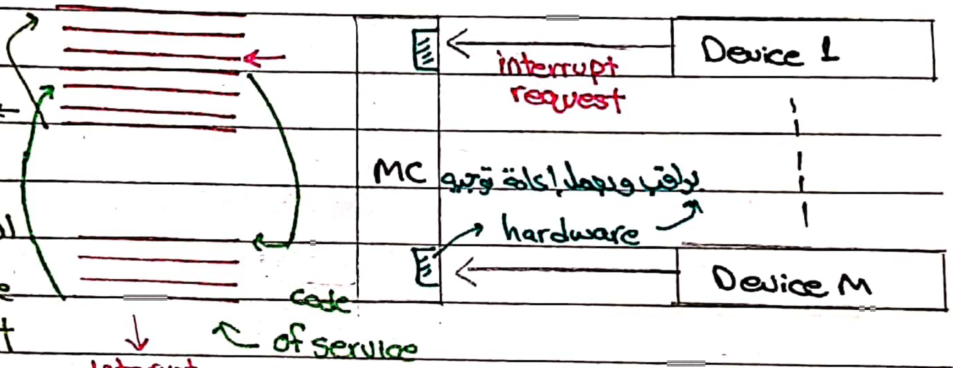
* البديل اللي ممكن استخدمه لحل مشكلات ال Polling هو ال interrupts

* Ex : نفرض الدكتور في المحاضرة بشرح ، كيف ممكن ياخذ أسئلة من الطلاب :

① كل خمس دقائق بوقف شرح وبلف ع الطلاب واحد واحد وبسألهم لو عندهم سؤال وهنا مضيعة للوقت (Polling)

② بيتم الشرح خلال المحاضرة واللي عنده سؤال برفع إيده وبسأل ، بقاطع الدكتور (interrupts) الدكتور بتجاوب معاه بحيث إذا انه سأل عن فكرة والدكتور بشرح عن فكرة معينة وما بده يوقفها بكمل شرحه بعدين بجاوب سؤاله وبرجع بكمل الشرح . وهيك ما بضيع الوقت وتتم الإجابة في أقرب وقت ممكن .

* interrupts :



ال ← Main وبتنفذ بشكل مستمر وبحمم ال Code بطريقة لما ال Device يبعت Interrupt request لل MC ليقاطعها فال MC إذا وصلتها المقاطعة عند inst3 مثلاً فبتكمل تنفذيها وبعدين بتجاوب ال interrupt request وهكذا!.

* ما بضيع وقت وانا بسأل ال microcontroller إذا ال device بده خدمة أولاً وبقلل ال Power ويقدم الخدمة بأسرع وقت ممكن .

* التأخير اللي ممكن يصير لحد ما تنفذ الخدمة بسيط وقليل جداً مقارنة بال Polling

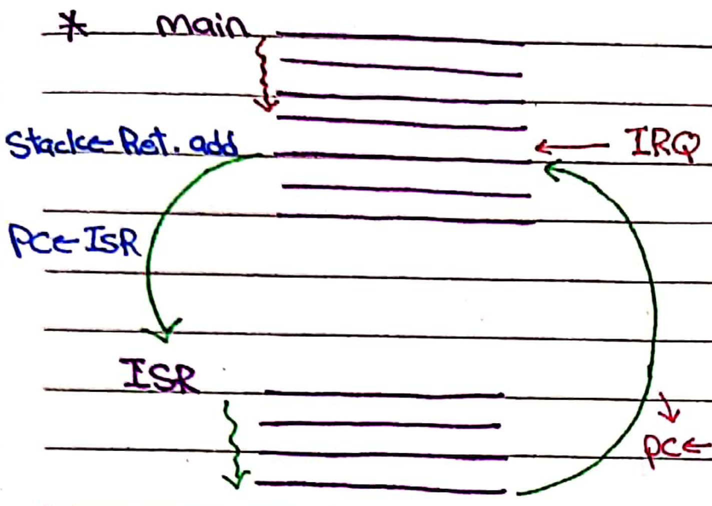
* ال hardware (interrupt circuit) وظيفته يراقب ال external sources المشبوكة مع ال MC وإذا كان في request منهم يقاطع ال MC ويوقف تنفيذ ال prog الموجود في ال MC و يعمل إعادة توجيه لل Code تاع ال Service

* interrupts : ① Save MC Time
② Provides immediate respons → مش صفر ولكنه زمن قليل

* Disadvantages of interrupts : ① complex / additional hardware
② additional coding complexity

* تفرق عن ال Subroutine لإنه مافي Call inst اللي بغير مسار البرنامج هو hardware event ولكن كلاهما يغير مسار البرنامج .

* أنا اللي بكتب ال Code ونسقه:



الـ cpu رح توقف لنها وصلت بعد ما تتخلص → IRQ
تنفيذ الـ current instruction

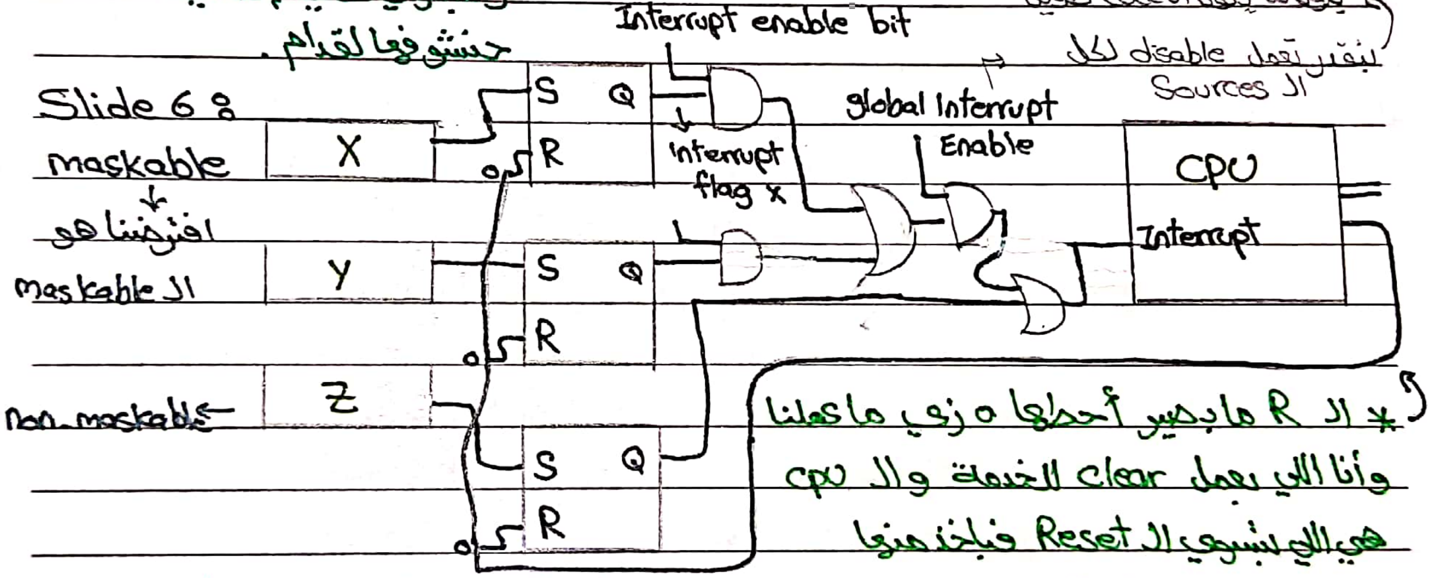
*كيف ممكن نغير مسار البرنامج المكان اللي فيه
الـ ISR بالنسبة للـ Pic1684A ؟
المفروض تروح الـ PC وتحط فيها الـ address
للـ ISR
PC← Ret add from Stack

*كيف أرجع المكان اللي وقفت عنده ؟ لازم في
الـ PC أحط الـ return address واللي هو
بكون حطيته في الـ Stack

(RETFIE)
Special inst
نشتغل نفس الـ return
ونسوي عملية إضافية
حنشوفها لقدام.

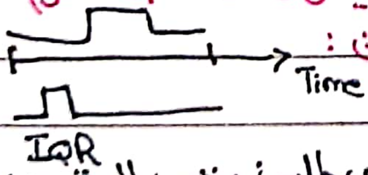
يعني ما يكون الـ code طويل
بنقدر نعمل disable لكل
الـ Sources

Slide 6 :



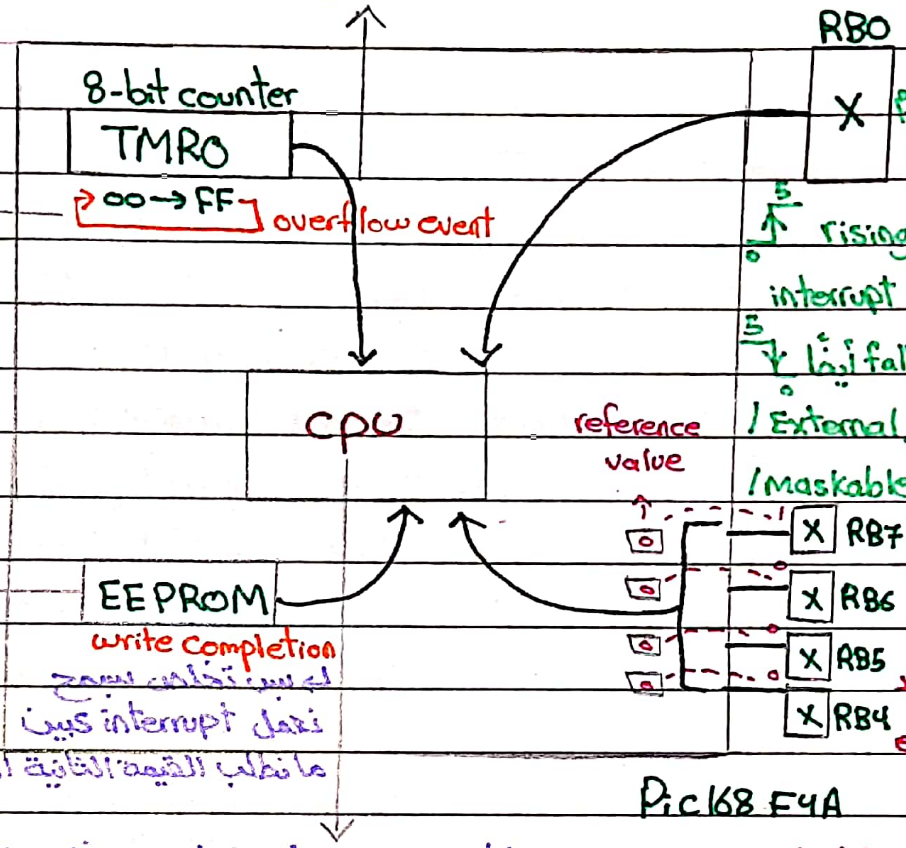
* الـ R ما بصير أحطها 0 زي ما عملنا
وأنا اللي بعمل clear للخدمة والـ cpu
هي اللي بتسوي الـ Reset فباخذ منها
وبوصلها للـ R

لـ لأعرف مصدر الـ interrupt
وأخزن الـ request
عشان بعدين لو كان الـ Req قليل
وما حيقدر يشوفه الـ cpu فلازم
يكون مخزن :



*لو كان عندي طلبين بنفس الوقت
حنعرف لقدام مين ننفذه أول.

Slide 7 : بستخدمه لأسري interrupt عال cpu / internal / Maskable



RB0 (Port B, Pin 0)

هنا الـ Pin بيستخدم مصدر للـ interrupt

* اذا نشغت عليه rising bit بستخدم هال event الـ interrupt

* اذا نشغت عليه falling bit أيضاً

/ External / edg triggerd interrupt / Maskable

بسيوصل FF يرجع لـ 00

Mem غير متطايرة non-volatile بستخدمها لأخزن Data بشكل دائم / Internal / Maskable

لما بس تخلص بسمح نعمل interrupt كبين ما نطلب القيمة الثانية الي نكتبها لإنها بتطول

بضل بشكل متواصل يقارن الـ ref value مع الـ RB ولو كان واحد عالأقل بختلف ف event للاختلاف بستخدمه للـ interrupt / External / Maskable

الـ component المسؤولة عن تنفيذ الـ inst ، can be interrupted بمصدر أو أكثر أو ولا مصدر.

* في حالة الـ Pic168F4A عندي 4 مصادر للـ interrupt :

① Timer overflow interrupt (internal / Maskable)

② EEPROM write complete interrupt (internal / Maskable)

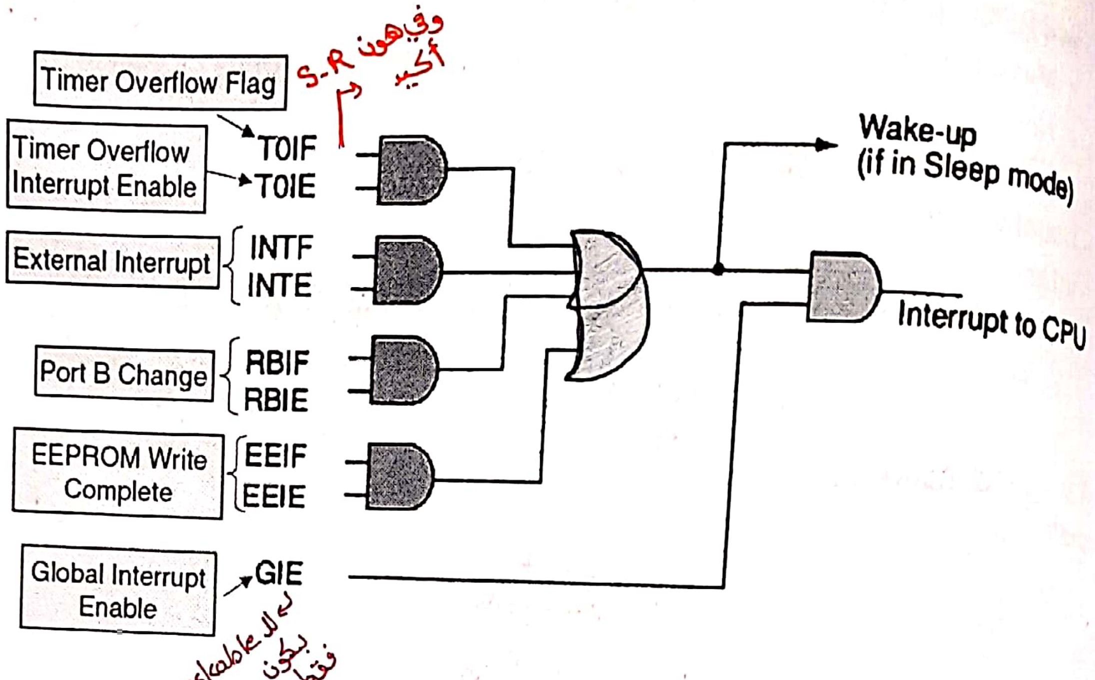
③ External interrupt (External / Maskable)

④ Port B on change interrupt (External / Maskable)

* كل الـ interrupt Source في الـ PIC168F4A هم Maskable .

The 16F84A Interrupt Structure

- Interrupt Hardware Structure



S-R وفي هون أكيد

maskable الـ بكون فقط

No non-maskable interrupts in 16F84A

الـ EEIF موموجود لأنه الـ Flag تاعها موجود بالـ register :

EECON1 ← هوموجود الـ EEIF

↳ R/W-0

/ interrupt flag

The 16F84A Interrupt Structure

SFR

- The INTCON Register → لازم أشوفه أنا كـ Programer

ما بخاف كموضوع الـ Bank لأنه موجود عالثنين ← INTCON REGISTER (ADDRESS 0Bh, 8Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

بعتبر أقواه فأكثر كلية

أول ما أشغل الـ System بتكون القيمة 0 (disable)

لو أخليه zero أول ما أبلش

dont care 0/1

bit 7 GIE: Global Interrupt Enable bit
1 = Enables all unmasked interrupts
0 = Disables all interrupts

bit 6 EEIE: EE Write Complete Interrupt Enable bit
1 = Enables the EE Write Complete interrupts
0 = Disables the EE Write Complete interrupt

bit 5 T0IE: TMR0 Overflow Interrupt Enable bit
1 = Enables the TMR0 interrupt
0 = Disables the TMR0 interrupt

bit 4 INTE: RB0/INT External Interrupt Enable bit
1 = Enables the RB0/INT external interrupt
0 = Disables the RB0/INT external interrupt

bit 3 RBIE: RB Port Change Interrupt Enable bit
1 = Enables the RB port change interrupt
0 = Disables the RB port change interrupt

bit 2 T0IF: TMR0 Overflow Interrupt Flag bit
1 = TMR0 register has overflowed (must be cleared in software)
0 = TMR0 register did not overflow

bit 1 INTF: RB0/INT External Interrupt Flag bit
1 = The RB0/INT external interrupt occurred (must be cleared in software)
0 = The RB0/INT external interrupt did not occur

bit 0 RBIF: RB Port Change Interrupt Flag bit
1 = At least one of the RB7:RB4 pins changed state (must be cleared in software)
0 = None of the RB7:RB4 pins have changed state

*BSF INTCON, T0IE

*BSF INTCON, INTE

enable لأنه صفر

لو في أكثر من interrupt مثلاً هي كما للم

هيك سويت بالبداية

*BSF INTCON , GIE

نفس الشي عملته enable

Bank1 موجود ب

The 16F84A Interrupt Structure

- **The Option Register (81H) – interrupt related bit**

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 — $\overline{\text{RBPU}}$: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 — INTEDG: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

العلاقة بال interrupt

bit 5 — T0CS: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 — T0SE: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 — PSA: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 — PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

Select the transition type on input RB0/INT that will cause an interrupt

عشان أتأكد إني ب Bank1

م

BSF status, RP0 INT

BCF option-R, INTEDG

بخليني أشوف وأحدد هل ال interrupt بكون عال rising أو عال Falling

وال deafult ال 1 ← rising

EX8 BSF INTCON, INTE

ل BSF INTCON, GIE → لما يخلص تنفيذها, إذا هلا عندي rising عال RB حيصير interrupt، لو كان بدي يكون عال falling بكتب قبل : فوق بالأخضر

The 16F84A Interrupt Structure

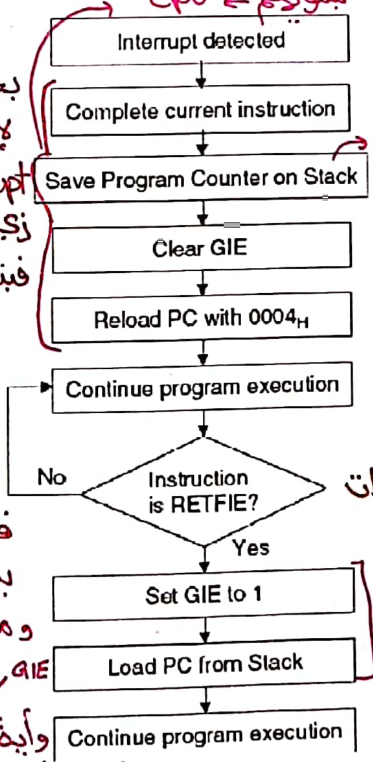
- **Interrupt Operation**

سكوت الباقي كبين ما أنفذ اللي عندي

*RETFIE :

بحطها هي مش return لإنه لما يصير عندي interrupt بصير أحداث معينة زي ما بالرسم التوضيحي هاد فبنستخدمها عشان تحط RA ← PC ويرضه تخلي ال GIE=1 زي ما كان قبل ما ننفذ أول interrupt فلو استخدمنا return بس كان ما قدرنا نرجع GIE=1 ومش منطق أكتب ورا كل inst BSF INTCON, GIE فلهيك بنستخدم RETFIE وأيضا ما بحط return قبل ال

لإنه ← *لما بدي interrupt يقاطع interrupt *



CPU ← بسويهم

Return address

Main program is running

GIE=0

return address

⓪ Disable all interrupt

① Stack ← RA

② PC ← ISR

ل 0x0004

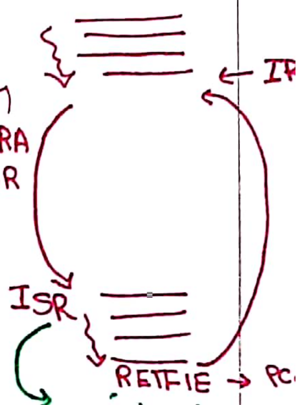
Vector address ل ما في مخصص لكل سبب لل interrupt فبغض النظر شو مصدره

ISR execution starts

هنا ال 4 مصادر بروح عال 0x0004 والمصدر بعرفه بالأمام بالسلايدات لما أشيك عال Flag

RETFIE

Main program continues



من وين أجيب هذا ال address

↓

0x0004 ← interrupt vector address

↓

محجوز لنستخدمه ال cpu لما يصير في interrupt عطول بروحه.

The 16F84A Interrupt Structure

- **How to use interrupts ?**

1. Start the interrupt service routine at 0x0004
2. Clear the flag of the used interrupt in the INTCON register (if it is not cleared on reset, e.g. RBIF) → لأنه كانت فيتكه X

بشكل عام ← BCF INTCON, RBIF

3. Enable the corresponding interrupt by setting its bit in INTCON register
4. Enable global interrupts by setting the GIE bit

5.a. clear The interrupt Flag

5.b. End the interrupt subroutine with **RETFIE** instruction to resume program execution

المهم اكتبه قبل الـ RETFIE مش مهم كان بالنص ولا بالنهاية ← ex: bcf INTCON, INTF ← (clear)

ISR { (clear) RETFIE

IF=1 ↓ لازم أخليها 0 عشان ما يضل يكرر نفس الـ interrupts ويدخل بـ infinit loop

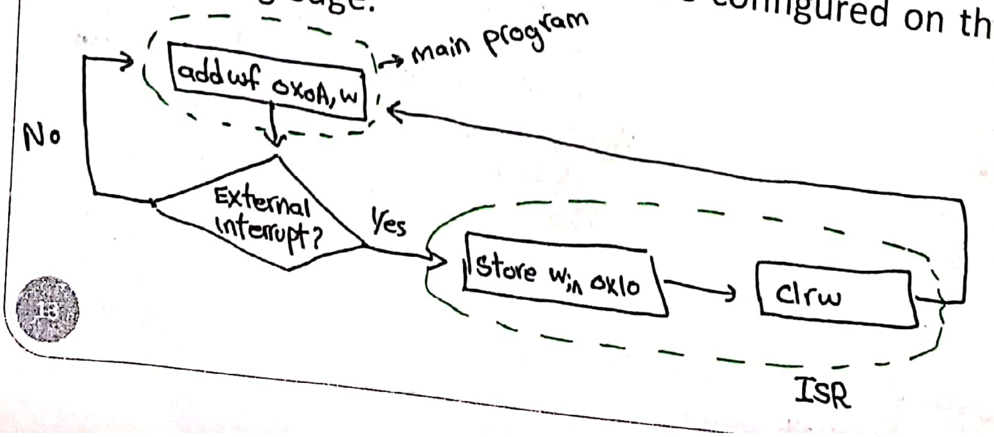
* 2/3/4 → عادة تكتب بال Main programm.

* IF (interrupt Flag) are not cleared by the retfie.

The 16F84A Interrupt Structure

- **Example 1** بضل يجمع محتويات 0x0A مع حالها لحد ما يقاطعه RB0

Write a PIC16F84 program that continuously adds the content of memory location 0x0A until an external interrupt is observed on RB0. In this case the result is stored in location 0x10 and the working register is cleared. The interrupt should be configured on the arrival of rising edge.



The 16F84A Interrupt Structure

```
         #include   p16F84A.inc            ; include the definition file for 16F84A
         org        0x0000                 ; reset vector
         goto       START
         org        0x0004                 ; define the ISR
         goto       ISR
         org        0x0006                 ; Program starts here
START    bsf        STATUS , RP0           ; select bank 1
         bsf        OPTION_REG , INTEDG ; select to interrupt on rising edge
         bsf        INTCON , INTE          ; enable external interrupt on RB0/INT
         bsf        INTCON , GIE           ; enable global interrupts
         bcf        STATUS , RP0           ; select bank 0
         molw       0x00                   ; clear W
ADD      addwf      0x0A , W               ; add the contents of 0x0A to W
         goto       ADD                    ; keep adding until an interrupt occurs

         org        0x00BC                 ; location of ISR
ISR      movwf      0x10                   ; on interrupt store the accumulated result
         clrw                              ; clear working register
         bcf        INTCON , INTF          ; clear the interrupt flag
         retfie                            ; return from the ISR
         end
```

الك حرية الكتابة ولكن هيك أرتب

لأنه موجود ؟ Bank1

لو شلت هاي مو مشكلة حيبلش من 5 بدل 6

ما الها داعي لأنه by default هي 1

* clear flag :
bcf INTCON,INTF
ما في داعي أحطها لإنه INTF ← by default هي 0

لو هون صار الـ int ثم بس يخلصه برجع هون

لو شغلنا ← عال RB0 بخلص الـ inst اللي بإيده وبجي Interrupt

اختياري زي ما بدك وممكن نشيلها

لما يصير هون int واخلصه برجع هون

---

main ≡

Context Saving

Ⓦ ← كان مخزن قيمة معينة وضرورية فلو صار interrupt احتمال كبير يتغير الـ w فكيف أحافظ عقيمة الـ w أثناء تنفيذ الـ interrupt

- What if the main program is to preserve the W register and the interrupt service routine uses it?
  - Save it temporarily in memory at the beginning of the ISR
    MOVWF TEMP ; push → بكتبها ببداية الـ ISR
  - Restore the value at the end of ISR
    MOVF TEMP, W ; pop → قبل ما ارجع (RETFIE)

* ممكن استخدم هاي الشغلة في الـ Subroutine أيضا *

- What if we want to preserve some memory location such as the STATUS register on interrupt? → هالمرة عال Status بدي أحافظ عليه
  - Save it temporarily in memory at the beginning of the ISR
    SWAPF STATUS,0 ; push
    MOVWF TEMP
  - Restore the value at the end of ISR
    SWAP TEMP, 0 ; pop
    MOVWF STATUS

ما بنستخدم MOVF مشان ما تتغير عال Z Flag لأنه بدي أحافظ عال Status

* ممكن أكون بدي أحافظ عال W وال Status بنفس الوقت. جرب اعمل code لهيك

* غالبا كل مصدر من مصادر ال interrupt بنفذ code معين ، كيف أعرف مصدر ال interrupt لو كان عندي أكثر من مصدر ؟ في بداية ال ISR بحتاج أعمل (check interrupt source)

بعمل check لل Flag لأنه أنا عندي address 0x0004 واحد فقط لل interrupt فلو كان عندي أكثر من مصدر لل interrupt لازم أعمل هيك.

Ex: in high level language:
↓
If source = INT...
If source = PBchg...
وهكذا

The 16F84A Interrupt Structure

Multiple Interrupts

- Note that there is only one interrupt vector for all types of interrupts
- In other words, regardless of the interrupt type, the microcontroller will start executing from location 0x0004 on any interrupt
- How to determine the source of interrupt ?
- Check the interrupt flag bits in the INTCON register at the beginning of the interrupt service routine to determine what is the source of the interrupt !

لو كان قيمته صفر حيعمل Skip لو قيمته 1 → حيكون فيه interrupt وحيكمل عادي

↓ بعمل check عالأربع مصادر

```
Interrupt_SR    btfsc intcon,0    ;test RBIF
                goto  portb_int   ;Port B Change routine
                btfsc intcon,1    ;test INTF
                goto  ext_int     ;external interrupt routine
                btfsc intcon,2    ;test T0IF
                goto  timer_int   ;timer overflow routine
                btfsc eecon1,4    ;test EEPROM write complete flag
                goto  eeprom_int  ;EEPROM write complete routine
```

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

* كل ال Source interrupt في ال PIC168F4A هم Maskable .

Slide 16 : ISR btfsc INTCON, RBIF

goto Portbchange → بكون label ل code هنا ال interrupt

btfsc INTCON, INT وهكذا... (Portbchange ═ retfie)

goto ext

btfsc INTCON, T0IF بدل سطرين ال check → (EEPROM ═ RETFIE)

goto Timer0

btfsc EECON1, EEIF → بالعادة ما بحتاج أكتبه لأنه لو ولا واحد من ال 3 مصادر اللي قبل كان هو سبب ال interrupt فأكيد هو السبب بكون.

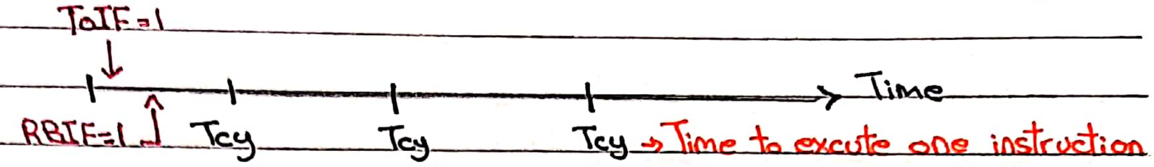
goto EEPROM

* لو عندي مصدرين مثلاً بعمل check لوحدة بس لأنه لو ما كانت تلقائي حتكون الثانية وهكذا....

Slide 16 :

* شرح فكرتين في الـ interrupt :

①

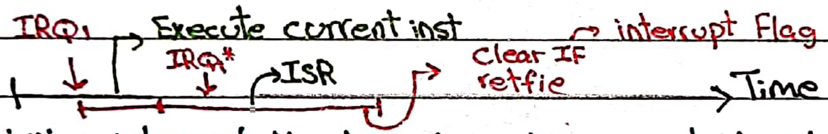


* نتخيل هنا الـ Time والبرنامج تبعي قاعد بيتنفذ وفي اللحظة اللي مؤشر عليها باللون الوردي صار في عندي interrupt request من Type T0IF وبنفس الـ inst صار عندي request من RBIF = 1 ، فلمّا الـ cpu تخلص الـ inst اللي صار عليها الـ interrupt requests رح تكتشف إنه في 2 interrupt requests فمين بدها تخدم بالأول ؟ ما بنقدر نحكي ياخدهم عالترتيب لإنه الـ cpu ما عندها Time stamp تعرف مين إجا أول . فعليا أنا اللي بحدد مين ينفذ أول بـ code الـ check في الصفحة السابقة ، فهو على ترتيب الـ code تبع الـ check بنفذ الـ interrupt .

* بالـ code كنا أول Flag بشيك عليه هو RBIF فهو رح ينفذ هو أول مع إنه إجا بعد الـ T0IF .

* الـ order اللي برتب فيه الـ check code هو بحدد الـ order of Service (priority).

②



* نتخيل إنه إحنا بنستخدم فقط مصدر واحد من مصادر الـ interrupt ، نتخيل صار عندي كما مؤشر عليه عالـ Time ← interrupt request من مصدر رقم واحد : IRQ_1 فالـ cpu حتكمل الـ current inst وبعدين تنفذ الـ ISR وترجع وهنا بالوضع الطبيعي.

لو افترضنا إنه IRQ_1 بغض النظر شو نوعها صم تطلب الخدمة بشكل متكرر وسريع شو رح يصير ؟ المشكلة إنه نفس المصدر بعت 2 request والتباعد الزمني بينهم قليل بالنسبة للوقت اللي بحتاجه لتنفيذ الـ ISR فهيك الـ IRQ_1^* مارح نشوف ورح نخسرها.

فكيف أتعامل مع هذا الموضوع ؟ رح أحاول أخفف من هاي المشكلة قدر ما بقدر عن طريق إنه الـ Clrea IF بدل ما أحطها بنهاية ISR code بحطها بالبداية فهيك بقدر نشوف الـ IRQ_1^* فلو أنا عارف إنه الـ interrupt تاعي هيك بطلب بشكل متكرر.

* ممكن كمان إني أحاول أكتب ISR code بزمن أقل من الزمن اللي بين الـ 2 request بقدر أشوف الـ IRQ_1^* لإنه حكون خلصت خدمة الـ request الأولى ، فبكتب بس بالـ ISR اللي بحتاجه . * فلازم أعرف التباعد الزمني بينهم عشان بناءً عليه بحدد هل عادي ولا أحل المشكلة.

① idea 1 → multiple request different Sources .

② idea 2 → multiple request from the Same Sources.

PAGE

DATE

* الـ interrupt هو الطريقة الأخرى البديلة لأعرف إذا حد بده يستفيد من الـ CPU.

* الطريقة اللي كانت هي الـ Palling.

* لو حكيت ما بدي استخدم الـ interrupt بدي استخدم الـ Palling فكيف أكتبها بالـ code؟

* عن طريق الـ INTF أيضًا، ونفس الكلام ينطبق على كل الـ interrupt Sources أيضًا.

* فالـ Interrupt Flag بقدر استخدمه بالـ interrupt وكذلك

بالـ Palling :

```
wait    BTESS INTCON, INTF
        goto wait
        ====
```

بشيكلي على الـ INTF لو كان 1 أعمل Skip

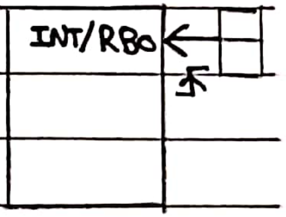
لو لا بنكمل عادي.

INTE=0
GIE=0

لا يوجد interrupt

زي كأنه هاد الـ app بحكيلي هنالك استنى لبين ما يصير في عندك rising edge على Port B قبل ما تكمل باقي البرنامج

* الـ Palling بستنى لحد ما يلاقي القيمة اللي بده ياها بس يلاقيها بكمل.



← ممكن يطلب يعد أو أشغله Timer كـ

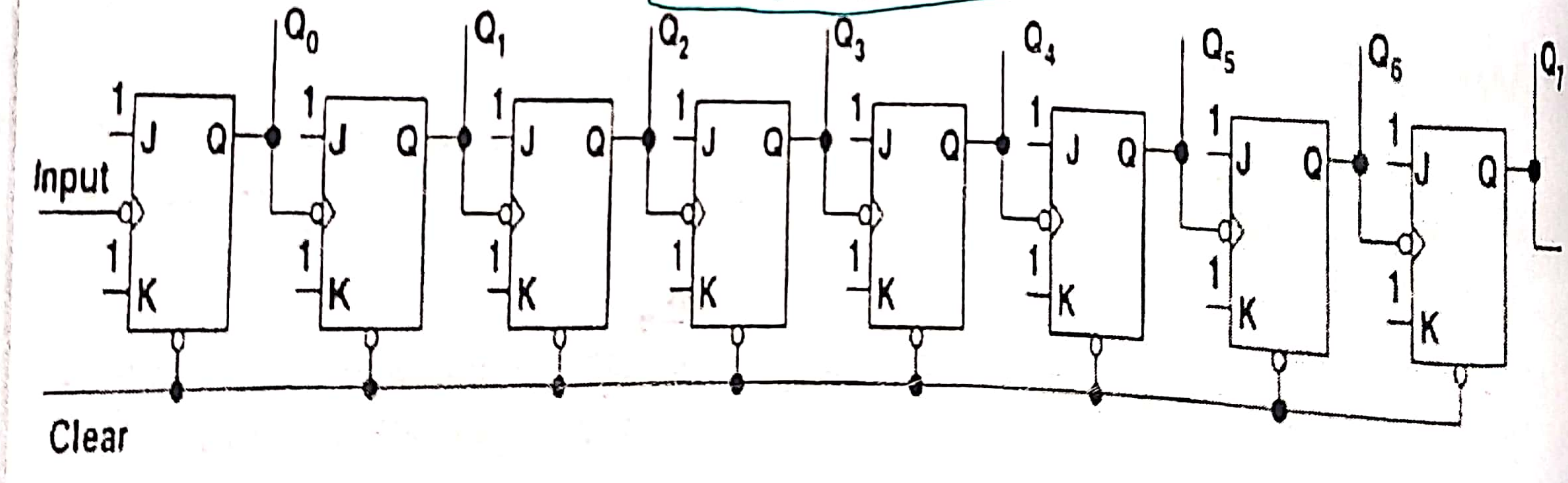
← حالة خاصة من ال Counter

Counters and Timers

- Digital counters can be built with flip-flops. They can count up or down, reset, or loaded with initial value
- When the most significant bit changes from 1 to 0 , this indicates an overflow. This signal can be used to interrupt the microcontroller
- *If the counter operates using a clock with known frequency we can use it as a timer*

→ 0 → 1 → 2 → . . . → 255

overflow



ل asyncronace ← كل bit هو ال clk للي بعده (يعني مختلفات عن بعض)

* sync ← يكون ال input مشترك بينهم كله زي الي كنا نصممه أول

Scanned with CamScanner

Counters and Timers

- **Timer applications**

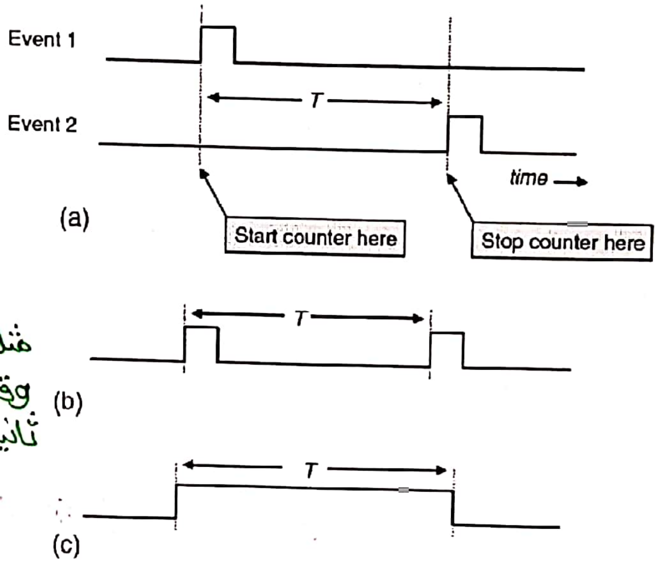
*بدي يكون عندي قدرة إنه الMicrocontroller تقدر تتعامل مع ال Time عن طريق احط هاردوير خاص يقدر يعد ال Time

(a) Measure the time between two events

(b) Measure the time between two pulses

(c) Measure a pulse duration → مثلا لو ضغط على كبسة وقت معين يعمل شغلة معينة مثلا لو ضغط ثانية يشغل ضو لو ثانيتين ضوين وهكذا .

Use **polling** or **interrupts**



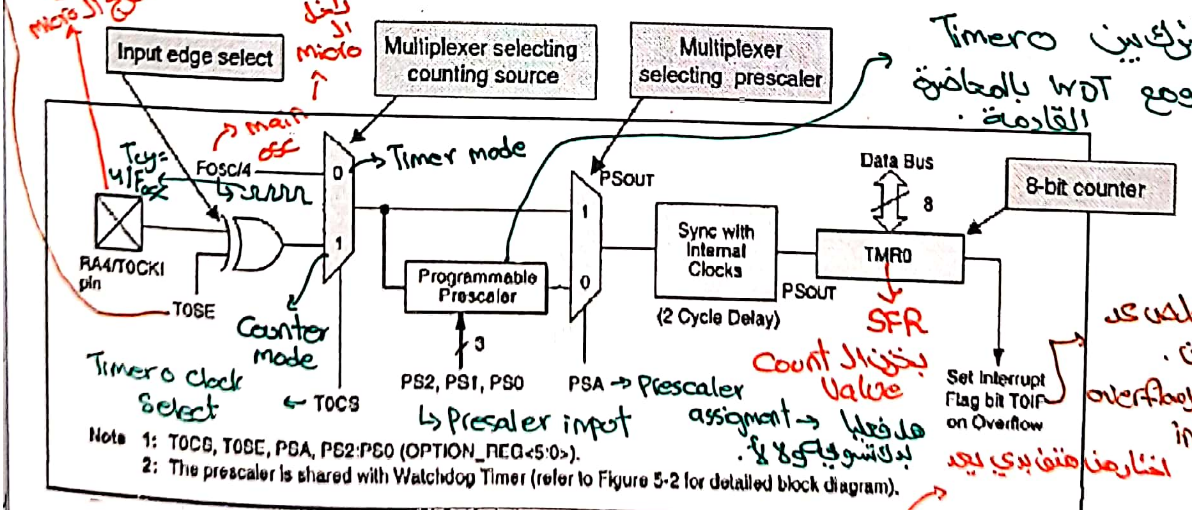
وظيفته يحدد لو استخدمت RA4/TOCKI ك input هل بدك الا increment يصير عال rising أو ال falling (Timer 0 Source Edge)

---

* ممكن تشتغل ك Counter hardware* مستقل بقدر استخدمه من خلال Special Funct reg

The 16F84A Timer 0 Module

مشترك بين Timer0 ومع WDT بالمحاضرة القادمة .



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

لو صار 1 خلص عد بكون . أخذنا حدث ال overflow أنه interrupt !

- 8-bit counter , memory address 0x01 (readable / writable) → جوه الرسمة السابقة اللي بالسلايدات
- Configurable counter using the OPTION register (0x81)
- Two sources for the timer clock : instruction cycle clock (Fosc/4) or RA4/T0CKI
- The programmable prescaler is shared with the Watchdog Timer WDT
- The value of frequency division is determined by PS2, PS1, and PS0 bits in the OPTION register

* ايا شي بدي اتحكم فيه بكون عن طريق SFR
hardware ← لم أو أحكي معاه

The 16F84A Timer 0 Module

- **The Option Register – Timer related bits**

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **RBPU:** PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG:** Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS:** TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

يعني لما نشغل البرنامج تلقائي
بكون counter
هو Timer
micro

bit 4 **T0SE:** TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

لو كان المحسس خارجي

bit 3 **PSA:** Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0:** Prescaler Rate Select bits

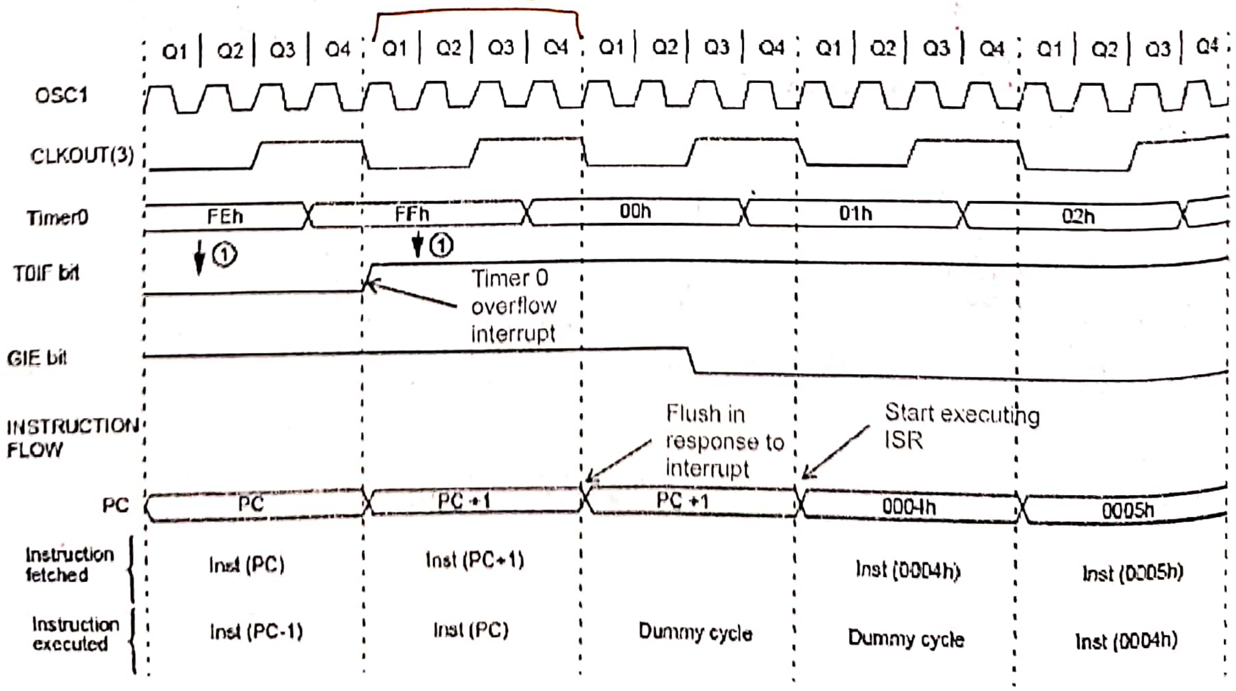
Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

$2^{PS_0;PS_1;PS_2+1}$
↳ concatenate

The 16F84A Timer 0 Module

- **Timer Timing**

Tcy



Note 1: Interrupt flag bit T0IF is sampled here (every Q1).
2: Interrupt latency = 4TCY where TCY = instruction cycle time.
3: CLKOUT is available only in RC oscillator mode.

للحظة اللي بينهي عد فيها يكون فعليا مر 5ms بدون استخدام ال interrupt

The 16F84A Timer 0 Module

بدي أسوي config لـ Timer0 واستخدمه بالبرنامج تعي بحيث انو من اللحظة اللي ببلش يعد فيها

- **Example 2: Write a program that generates a 5 ms delay using the TMR0 module without using interrupts. Assume the clock frequency is 800 KHz.**

  ↳ اسوي Palling

  - Fosc = 800 KHz → the timer internal clock = Fosc/4 = 200 KHz → instruction cycle = 5 us → timer increment every 5 us

    ↳ Tcy ↳ بس انا بدي كل 5ms
  - For these settings, the timer generates an interrupt after 256 * 5 us = 1280 us only ?!
  - How about changing the prescale factor ?
    - 256 x prescale x 5 us = 5 ms → prescale = 3.9 ~= 4
    - This will generate a delay of 4 x 256 x 5 us = 5.12 ms
  - What if we need more accurate delay !! We can play around with the count value (we don't have to start from 0 always) → لو بدي بالزبط 5 فلعب بعدد ال increment
    - N x prescale x 5 us = 5 ms --> N x prescale = 1000 → we can select the prescale 8 and the count N to be 125 P = 8 / TMR0 = 131
    - We have to load TMR0 with 256 – 125 = 131 as initial value

افتراض ما بدي استخدم P

$* \text{Time} = \#inc * T * P$

$5ms \overset{??}{=} 256 * 5 Msec * 1$

$= 1.2ms << 5ms$

↳ صغيرة عن القيمة المطلوبة

$5*10^{-3} = 256 * 5*10^{-6} * P$

$\rightarrow P = 3.9 \approx 4$

$\text{Times} = 256 * 5*10^{-6} * 4 = 5.12 \text{ msec}$

↳ وهاي كثير قريبة لل Value المطلوبة

more accurate delay ↳ $5ms = \#inc * 5*10^{-6} * p \rightarrow P * N = 1000$ بقدر أحلها بسهولة لأنه قيم P معروفة وقليلة

↳ بصير اجرب بكل قيم P ولازم N تطلع قيمة بقدر آخزنها بـ 8-bit

The 16F84A Timer 0 Module

$P = 2 \rightarrow N = 500$ X

ممكن اختار وحدة منهم

$P = 4 \rightarrow N = 250 \rightarrow N = 250 \rightarrow TMR0 = 6$ ✓

$P = 8 \rightarrow N = 125 \rightarrow TMR0 = 131$ ✓

$P = 16 \rightarrow N = 62.5 \rightarrow 63$ X → لأني قربتها

256 - 250

- **Example – cont'd**

$5*10^{-3}$ - overhead $= N*T*P$

```
        #include p16f84A.inc
        org     0x0000
        goto    start
        org     0x0010
start   . . . .
        call    delay5
        . . . .
delay5  movlw   D'131'   ;   preload T0, it overflows after 125 counts
        movwf   TMR0
        bsf     STATUS, RP0     ;select memory bank 1
        movlw   B'00000010'     ;set up T0 for internal input, prescale by 8
        movwf   OPTION_REG
        bcf     STATUS', RP0    ;select bank 0
del1    btfss   intcon,T0IF     ;test for Timer Overflow flag
        goto    del1            ;loop if not set (no timer overflow)
        bcf     intcon,T0IF     ;clear Timer Overflow flag
        return
```

→ Some code

هنا انزاد عال delay الأصلي احتجنهم لأوصل

$P\left(\begin{matrix}(2+5)*5*10^{-6}\\ +2\end{matrix}\right) \rightarrow \text{return}$

↳ ملاح يكون 5 بالزبط

حيكون أكبر

start ← فالمفروض احسب بكون كاملة 5 وع الز 5 بوصل..

delay5 ↓ hardware delay بطريقة Palling

		T0CS	T0SE	PSA	PS2	PS1	PS0
X	X	0	X	0	0	1	0

option reg

Timer ك تشتغل ما بهمني

* اي شي X افرضه 0 دائما

0x02 = OPTION_REG

ليكتب option Reg

حيبلش يعد بعد انتهاء ال inst

Palling

↳ حطينها هنا وهو مش interrupt لأنه ممكن بستخدم delay5 أكثر من مرة فلو ما عملنا clear لو ناديته مرة ثانية ما حيعمل

Watchdog Timer

reset لما يصير في overflow للـ microcont... عال بيعملو

- Special timer internal to the microcontroller that is continually counting up. → دائما بعد
- If enabled and it overflows, the microcontroller is reset
- Can be used to reset the Microcontroller if a program fails or gets stuck
- Properties
  - The WDT timer is enabled/disabled by the *WDTE* bit in the configuration word
  - It has its own internal *RC oscillator* →

  - The nominal time-out period is *18 ms* → Time أقل →
  - It can be extended through the prescaler bits in the *OPTION* register (up to 128x18 ms= 2.3 sec) → Clrwdt بجربو يسوي overflows كل ثانيتين لأقلل من ال
    أكبر قيمة Prescaler
  - The WDT timer can be cleared by software using the *CLRWDT* instruction
- **How does the watchdog timer know if the program is stuck ???!!! It does not!** ← اجابته بالرفض

Sleep Mode

← زي باللابتوب

لما أكون مو بحاجة ال sys بس بدي أحافظ عال current State

- An important way to save power!
- The microcontroller can be put in sleep mode by using the *SLEEP* instruction
- **Once in sleep mode, the microcontroller operation is almost suspended**
  - (The oscillator is switched off)
  - The WDT is cleared. If the WDT is enabled, it continues running
  - Program execution is suspended
  - All ports retain their current settings
  - $\overline{PD}$ and $\overline{TO}$ bits are cleared and set respectively
  - (Power consumption falls to a negligible amount)

بنطفي ال OSC circuit

فال TOINT مستحيل يصير لانه TMR0 ما بشتغل

هاد ماحيكون شغال لإنه لما ندخل ال Sleep ← TMR0

- **To exit the sleep mode** → TOINT, INT, EEP, PB
  - Interrupt occurs (even if GIE = 0) ← 4 interrupt
  - WDT wake-up
  - External reset the MCLR pin

Interrupt occurs and WDT wake-up: Program continues execution from PC+1

External reset the MCLR pin: MC is reset !

عشان الي ممكن يصحوا ال micro من ال Sleep

Summary

- Microcontrollers can deal with time by using timers and interrupts
- Interrupts saves the microcontrollers computational power as they require its attention when they occur only
- Most interrupts are configurable
- Hardware timer can be used as a counter or a timer and it is very useful in measuring time

---

Parallel Ports, Power Supply, and the Clock Oscillator

Chapter 3

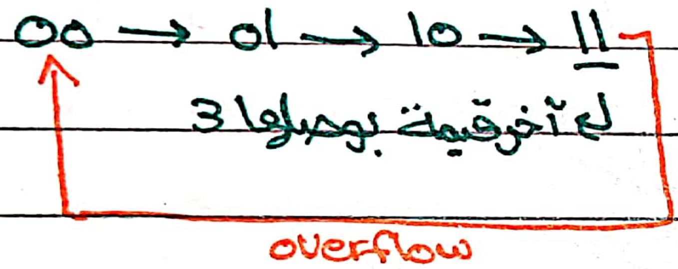
Dr. Iyad Jafar

* الـ falling بستنى لحد ما يلاقي القيمة اللي بده ياها بس يلاقيها بكمل.

Slide 17 :

* الـ Counter عبارة عن تطبيق للـ Sequential logic Circuit وهو عبارة عن مجموعة من الـ Flip-Flop مشبوكة مع بعض بطريقة معينة بحيث كل ما صار عال الـ Counter سواء rising / Falling edge بنتقل من count value لـ count value.

EX : 2-bit counter / modulo-3 counter :

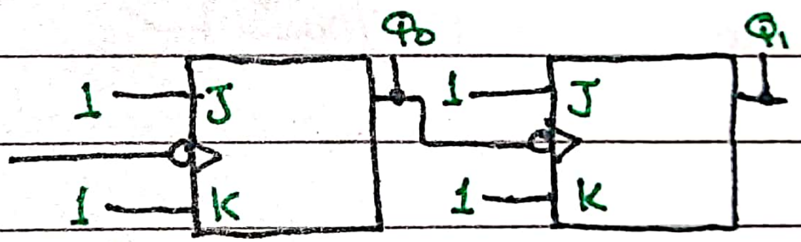


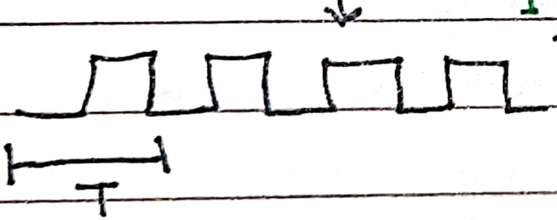
↳ لأنه آخر قيمة بوصلها هي 3

* معناه كل ما صار عندي falling edge على الـ Q0 الـ output لـ Q1 ممكن يتغير

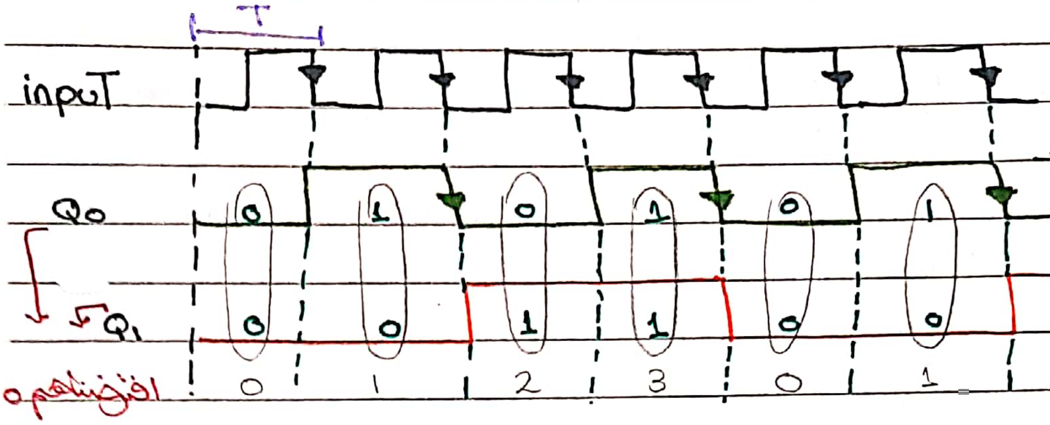
↳ Design :

input ← بجيب اتحكم فيه





$J = K = 1 \xrightarrow[\text{falling}]{\downarrow} Q_0(t+1) = \overline{Q_0(T)}$ Toggle



كلما يجيها falling حتسوي Toggle

افترضناهم 0
1 K و J

* Q_0 ← least sig bit / Q_1 ← most sig bit

ال Count حيدخل بعد هيك

* كم بعد كم من falling edge بوصلني

* لو كانت T معروفة فبيك أنا بعد ال Time

* لو دخلت ال input Period تبعته معروفة فبقدر أخلي ال Circuit هاي تحسب ال Time

* Time = #increments * T

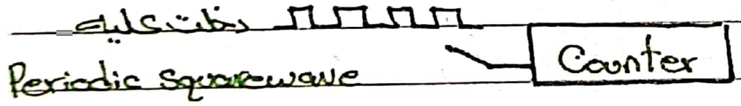
أكبر قيمة من صفر لصفر

* n-bit counter → $Time_{max} = 2^n * T$

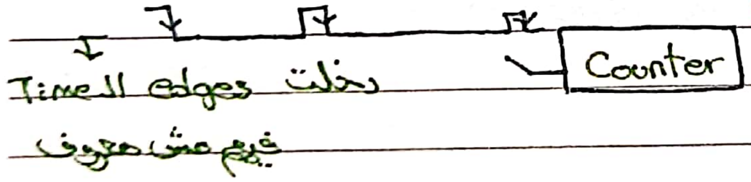
* Periodic ← متباعدة بشكل متساوي زي مثالنا فوق

* هل بالضرورة دائماً ال Circuit اللي موجودة عنا تكون ال input اللي داخل عليها Periodic ؟

ال Timer هو حالة خاصة من ال Counter ، ولو ما كان ال input ← periodic فأنا فعلياً هيك بعد events :



بعد edges / هيك بصير ال Timer بفترة زمنية متساوية (Timer/counter)



→ براقب ال falling edge فهيك هو Counter

* ال Timer هو Counter ولكن الاختلاف بكون بال input لو كان periodic فهو Timer لو كان ال input مش periodic فهو counter بعد ال edges بس

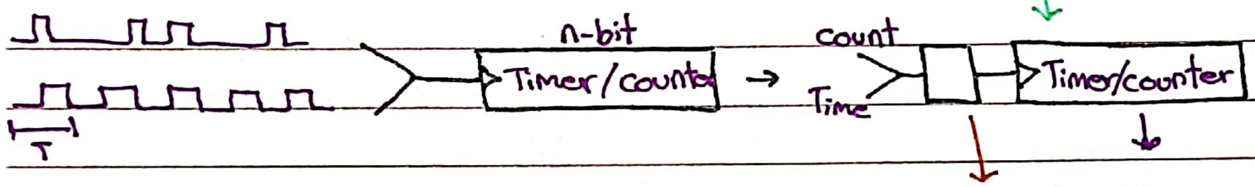
constant

* To increase Timemax = $2^n * T$ (n-bit counter) :

- increas T → T↑ F↓ → فهيك n ثابتة و T ثابتة فهل في طريقة أكبر فيها الـ Timemax ؟

العلاقة ب Fosc فغالبا بنلاقيها ثابتة لإنه غالبا جاية من الـ main Fosc

فعليا ما بنقدر. بس لقدام حنشوف في Flexibility لإنه غالبية الـ Counters والـ Timers اللي في PIC16F84A بتكون هيك



بجيبوه وبحطوا circuit صغيرة وتسمى prescaling hardware عبارة عن hardware component وهو عبارة عن Frequency devider ← بياخذ الـ Frequency الـ (Timer) وبقسمها فقيمة معينة فبقل F فيزيد T . وعادة بكون بنقدر نتحكم بالقيمة الي نقسمها حسب range الأرقام الي مسموحلك .

new T → أكبر من T الأصلية

لو كان Time بس

→ Timemax = # increments * T * P

2^n ↓ ثابتة ↓ ها اللي أضفته

* Time (بشكل عام) = # increments * T * P

* بالنسبة للـ counter فوجود الـ Prescalling hardware برضه بتقسم عدد الـ events :

⇒ ∴ p = 2 → count value = 3

6 / 2 ← عدد الـ events الأصلية

Slide 23 :

Note : writing to TMR0 register clears the prescaler. Thus, we should always write to TMR0 before specifying the prescaler

ملاحظة الها علاقة باستخدام الـ option reg و TMR0 ، إذا بدك تستخدم الـ Prescaler مع TMR0 وبدك تعطيه value مش صفر بلزم تتأكد انه تكتب ع TMR0 أول بعدين عالـ option reg لإنه الـ hardware مصمم بحيث كلما تكتب ع TMR0 بصير Clear للـ prescaler فإذا بعكسهم ما بعطيني غلط بس ما بيمشي زي ما بدك .

PAGE

DATE

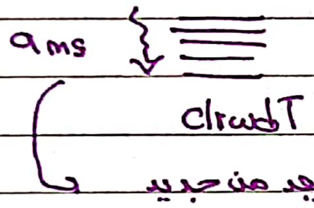
Slide 24 :

شو بستفيد من الـ Watchdog Timer : مثال انطلب منك تبني Embedded sys وظيفته يقيس درجة الحرارة والضغط الي موجودين داخل الأرض وحفرنا حفرة ورمينا الـ sys تحت وبعتلنا إشارات wireless هنا الـ sys بدرجة الحرارة والضغط، لسبب ما الـ sys علق فكيف بدي أرجع أشغله؟ يا بسحب الـ system وأعمله reset أو أفصل الـ Power أو بسوي Enable الـ watchdog Timer وبخليه سوي reset كل فترة زمنية معينة.

* هل بالضرورة يكون في شي مشكلة لما يوصل الـ WDT للـ overflow ؟ لا، مش بالضرورة بعد كل فترة بعمل فيها reset إنه تكون صارت مشكلة، فلازم أفهم الـ WDT إنه أنا شغال وأموري تمام وحتى لو سويت overflow ما تعملي اشي ولا تعملي set

* لازم استخدم clrwdT ←

كل ما أعمل seT للـ micro → 10ms



ما وصلت 10 فبصفر الـ WdT وبخليه يعد من جديد

* لازم أعرف قديه الـ Time الي بحتاجه لأنفذ جزء من الـ code وكل ما وجدت إنه هنا الـ Time قريب من الـ overflow Time لازم أحط clrwdt

Slide 25 :

* الـ WDT إذا كان enable وسوّى overflow بسوي reset عالـ micro في حالة الـ Normal mode، الـ WDT حتى لما تدخل الـ micro في حالة Sleep Mode بخليه شغال لإنه الـ clk تبعته مستقلة لحالها، فبهاي الحالة إذا كان الـ WDT شغال وسوى overflow والـ micro بـ Sleep mode ما رح يسوي reset حيسوي wake up.

Outline

- Why Do We Need Parallel Ports?
- Hardware Realization of Parallel Ports
- Interfacing to Parallel Ports
- The PIC 16F84A Parallel Ports
- The Power Supply
- The Clock Oscillator

interface) وغالبا هو
I/O Pins الـ
وهو مهم لنقل المعلومات

---

* حنركز على الـ Parallel digital port

Why Do We Need Parallel Ports?

لازم تاخذ مني وتعطيني (Data) م

- Almost any microcontroller needs to transfer digital data from/to external devices and for different purposes → لأهداف مختلفة

  من / الى

  - Direct user interface – switches, LEDs, keypads, displays
  - Input measurement - from sensors, possibly through ADC
  - Output control information – control motors and actuators
  - Bulk data transfer – to other systems/subsystems

← تبادل المعلومات

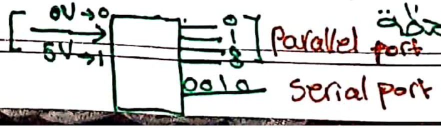
- Transfer could be serial or parallel ! Analog or digital !

بدخل الـ I/O & one pin

بنقل الـ bit بالتوازي بنفس ال اللحظة

Digital ← مثلاً ما بقدر يقرأ 3.5 بقربها لل 5 وبفهم انها logic1

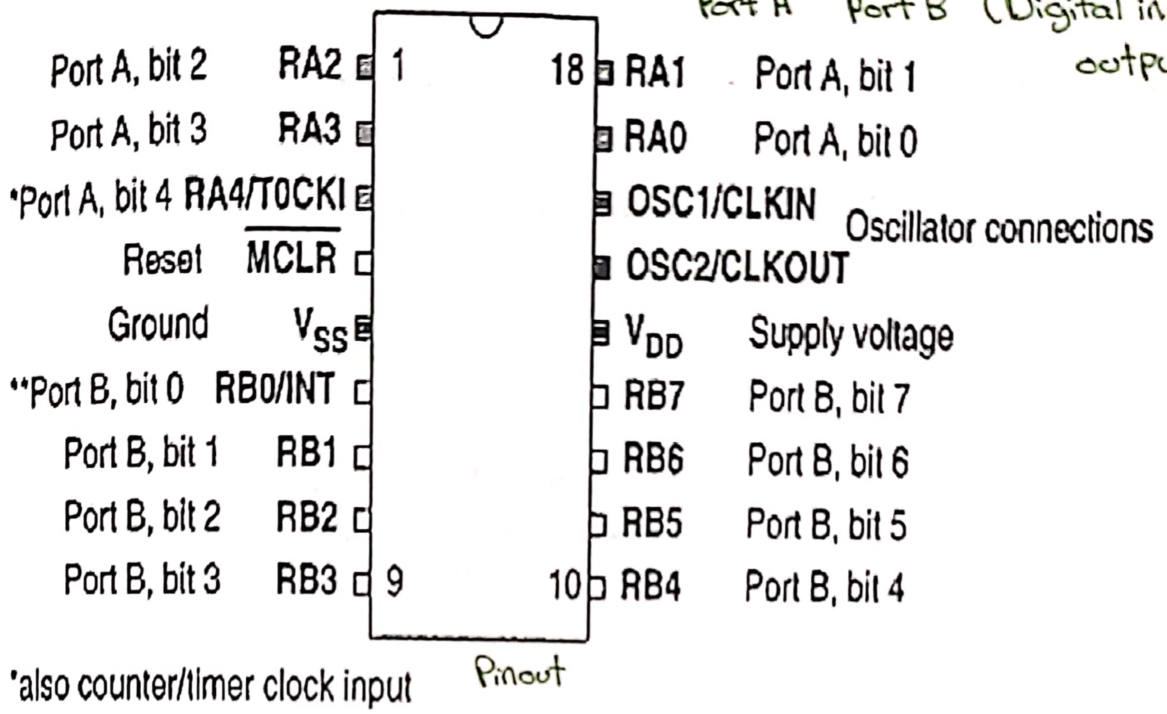
Analog ← عكس digita بفهم الأرقام وما بقربها ل 0 أو 5



Scanned with CamScanner

The PIC 16F84 Parallel Ports

Port A Port B (Digital input / output)



*also counter/timer clock input

**also external interrupt input

The PIC 16F84 Parallel Ports

PORT A

بقدر استخدمه ك input outputs

- *5-bit general-purpose* bidirectional digital port

RA0 → RA4

*انتبه تحدد ال bank قبل ما تشتغل مشان شغلك يمشي صح

EX: * movlw 0xA
movwf PORTA] write/output
* movf PORTA,w] input/read

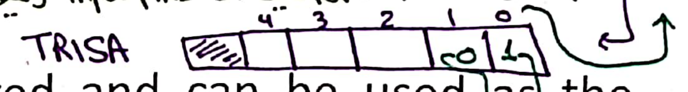
- **Related registers**
  - Data from/to this port is stored in *PORTA* register *(0x05)*

عالمستوى ال Pin مش ال Port

  - Pins can be configured for input or output by setting or clearing corresponding bits in the *TRISA* register *(0x85)*

بنقدر تحدد بال Port مين ال input pins ومين ال output pins حسب ما بدك عادي

حدد ال Pins



- Pin *RA4* is multiplexed and can be used as the clock for the TIMER0 module

معناها بدي اياها output

معناها input

The PIC 16F84 Parallel Ports

PORT B

↳ RB0 → RB7

- *8-bit general-purpose* bidirectional digital port
- **Related registers**

  Data register ←
  - Data from/to this port is stored in *PORTB* register *(0x06)*
  - Pins can be configured for input or output by setting or clearing, corresponding bits in the *TRISB* register *(0x86)*, respectively

  ↓ Direction reg بحدد ال
  يعني هلال Pin عبارة عن
  input ولا output
- **Other features**
  - Pin *RB0* is multiplexed with the external interrupt INT and has Schmitt trigger interface
  - Pins *RB4 – RB7* have a useful 'interrupt on change' facility

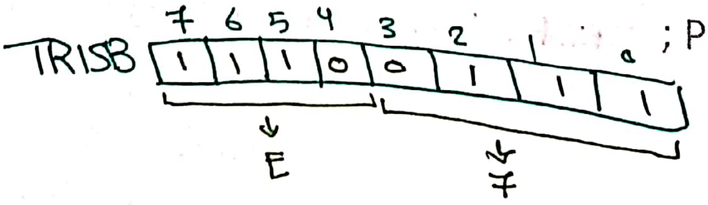
---

The PIC 16F84 Parallel Ports

- **Example 1 –** *configuring port B such that pins 0 to 2 are inputs, pins 3 to 4 outputs, and pins 5 to 7 are inputs*

→ لو ما بعرف أنا وين موجود لازم أحطها عشانه reg
موجود فقط

```
bsf  Bank1 ب  STATUS , RP0        ; select bank1
movlw         0xE7
movwf         TRISB
                                  ; PORTB<7:5> input,
                                  ; PORTB<4:3> output
                                  ; PORTB<2:0> input
```



The PIC 16F84 Parallel Ports

يعني TRISB كله أصفار

- **Example 2 –** *configuring PORTB as output and output value 0xAA*

```
bsf        STATUS , RP0        ; select bank1
clrf       TRISB               ; PORTB is output
movlw      0xAA
bcf        STATUS , RP0        ; select bank0
movwf      PORTB               ; output data
```

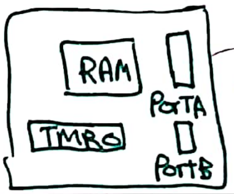
→ Port B Reg لإنو بدي احكي مع

- **Example 3 –** *configuring PORTA as input, read it and store the value in 0x0D*

يعني TRISA كله 1

```
bsf        STATUS , RP0        ; select bank1
movlw      0xFF
movwf      TRISA               ; PORTA is input
bcf        STATUS , RP0        ; select bank0
movf       PORTA, W            ; read data
movwf      0x0D                ; save data
```

يعني يكون أول 5 هم 2 لأنه PorTA بس 5 inpuT/output



مش موجودين داخل ال Mem بس لأسجل

التعامل معهم أخصصلهم addresses من ال address space الي معاي و اتعامل معهم ك Mem location

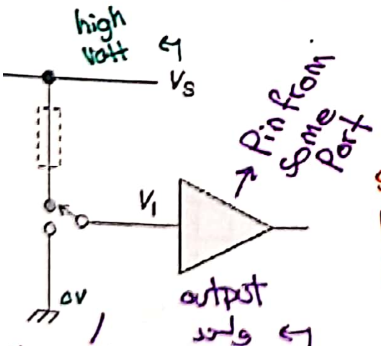
Interfacing to Parallel Ports

بسكر make ← → break بفتح

زي كبسات الابتوب ← **Switches** (Input devices)

(pushbuttons, toggle, slide, thumbwheel)

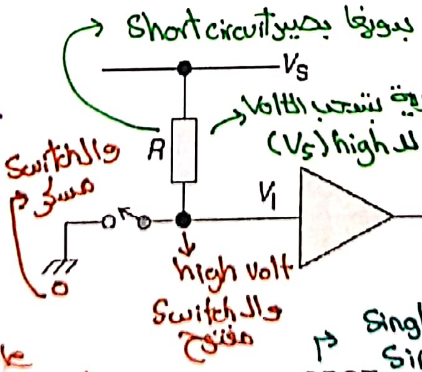
بنعطي ال micro معلومات.



واحد ← / نقطة ارتكاز وحدة
Single Pole
→ double through
↓
ممرين

Interfacing to SPDT switch. A current limiting resistor might be needed

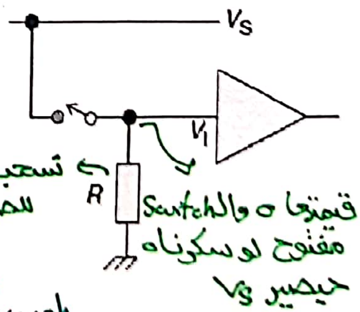
سهل التعامل معاه



→ Single pole Single through

Interfacing to SPST switch. To reduce wasted current, the pull-up resistor R should be high (10-100KOhms)

↓
أرخص وأبسط بس مشكلته انه بس بيشبك مع one value.
ومن طرق الحل الواسعة الي فوق



Interfacing to SPST switch using a pull-down resistor

Ex:

نشبكته لأشوف

الValue اللي بتطلع من الPort MC

Interfacing to Parallel Ports

Light Emitting Diodes (LEDs)

output ← لازم يكون الPort

- LEDs can be driven from a logic output as long as the current requirements are met. Interfacing of LEDs depending on the logic type and their capability to source and sink current

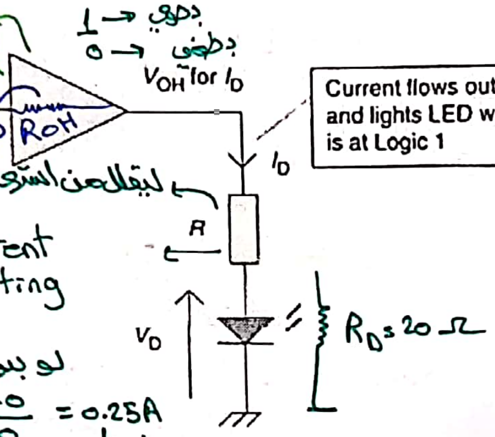
هون الPort output ← Sourcing current ← بطلع

1 → بضوي
0 → بطفي

في مقاومة داخلية

ليقلل من استهلاك الطاقة والتيار

Current limiting



Current flows out of the gate and lights LED when output is at Logic 1

لو بدونه

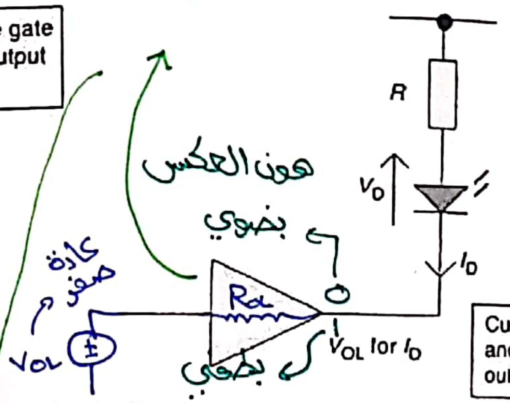
$I = \frac{5-0}{20} = 0.25A$ ↳ large

$I_V = \frac{V_{OH}}{R_D}$

V_{OH} Logic gate output high voltage

هون الPort ← output Sinking current → بدخل current

هون العكس



Current flows into gate and lights LED when output is at Logic 0

V_{OL} Logic gate output low voltage

For current source: $V_{OH} = RI_D + V_D$

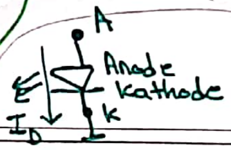
$$R = \frac{V_{OH} - V_D}{I_D}$$

For current sink: $V_S = V_{OL} + RI_D + V_D$

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

معروف

معروف



$V_D < V_\gamma$ ↳ cutoff

$V_D \geq V_\gamma$ → Conduct → بيضوي

* الI/O Ports ليس باتجاه الVoltage باتجاه الcurrent

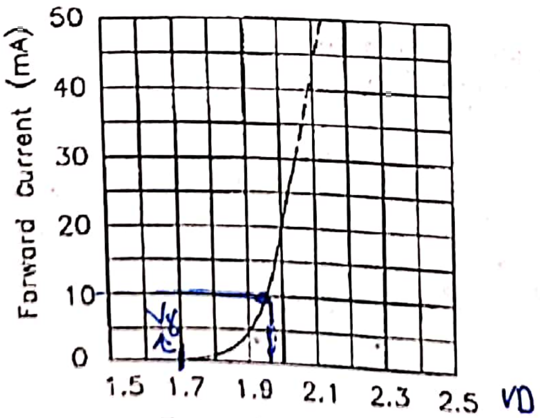
$R = \frac{V_{OH} - I_D R_{OH} - V_D}{I_D}$

وما أهملت مقاومة الPort أدق

Interfacing to Parallel Ports

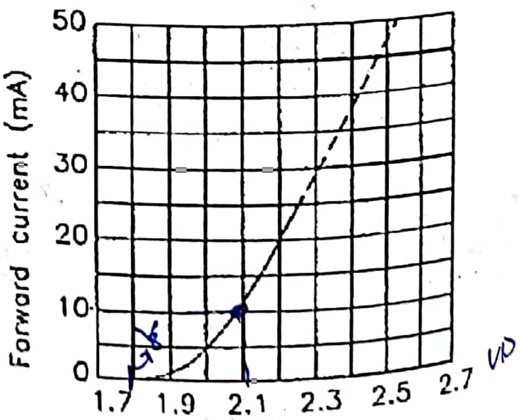
Light Emitting Diodes (LEDs)

- A special type of diodes made of semiconductor material that can emit light when forward biased



Forward Voltage (V)
FORWARD CURRENT Vs.
FORWARD VOLTAGE

Type number: L-441D
Wavelength = 627 nm
15mcd typ.@ 10 mA



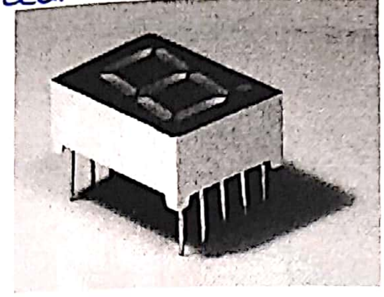
Forward Voltage (V)
FORWARD CURRENT Vs.
FORWARD VOLTAGE

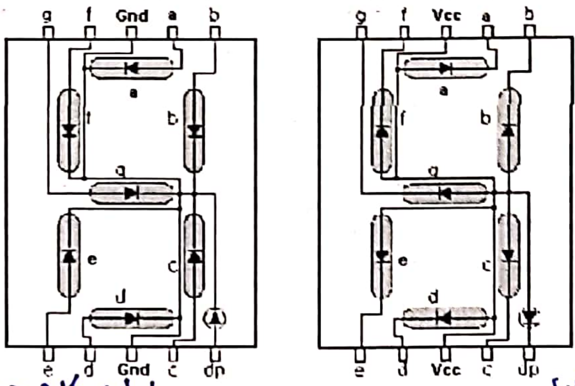
Type number: L-44GD
Wavelength = 565 nm
12mcd typ.@ 10 mA

Interfacing to Parallel Ports

7-Segment Display

له داخليًا عبارة عن كل وحدة منها diode
ومرات بكونوا 8 مع ال decimal Point



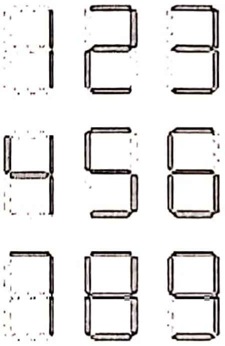


مشبوكين من ال input
a,b,c,d,e ,f,g

عكس ال common cathode

عكس الكلام

له كلهم مشبوكين عليه



Digit Shown	a	b	c	d	e	f	g
0	1	1	1	1	1	1	0
1	0	1	0	0	0	0	0
2	1	1	0	1	1	0	1
3	1	1	1	1	0	0	1
4	0	1	1	0	0	1	1
5	1	0	1	1	0	1	1
6	1	0	1	1	1	1	1
7	1	1	1	0	0	0	0
8	1	1	1	1	1	1	1
9	1	1	1	1	0	1	1

Illuminated Segment (1 = Illumination)

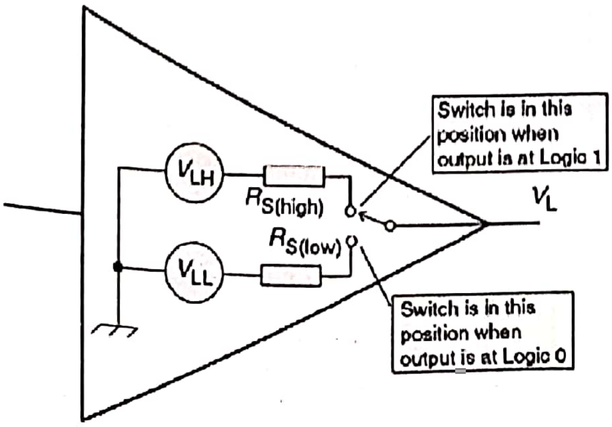
اذا بدي أشغل أي وحدة
لازم ال volt اللي
داخل عال input
يكون أعلى من اللي
هون بال Gnd

له لو كان common anod حيكون العكس

Interfacing to Parallel Ports

Port Electrical Characteristics

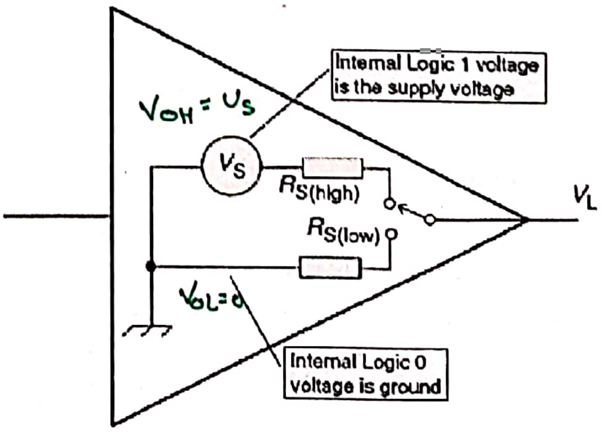
- Logic gates are designed to interface easily with each other, especially when connecting gates from the same family
- The concern arises when connecting logic gates to non-logic devices such as switches and LEDs



Switch is in this position when output is at Logic 1

Switch is in this position when output is at Logic 0

Generalized model



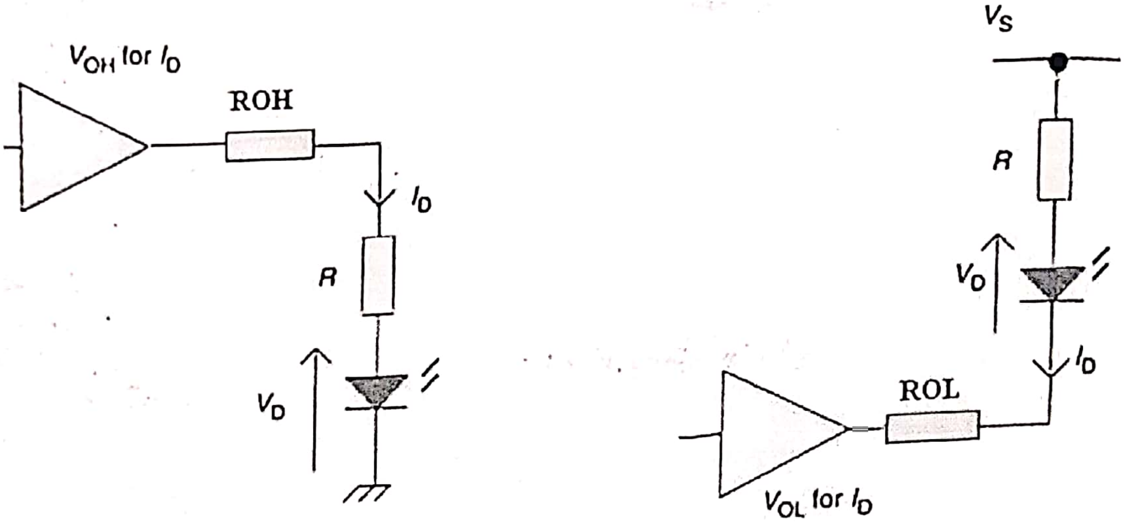
Internal Logic 1 voltage is the supply voltage

Internal Logic 0 voltage is ground

CMOS model ← هاد بيهمنا

Interfacing to Parallel Ports

Light Emitting Diodes (LED)



V_{OH} Logic gate output high voltage

V_{OL} Logic gate output low voltage

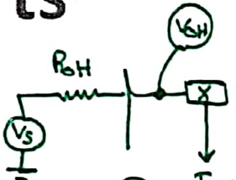
Computation of limiting resistors when internal resistance of the port pin is considered

---

The PIC 16F84 Parallel Ports

أنا بشوفه

وهو Vs ↓



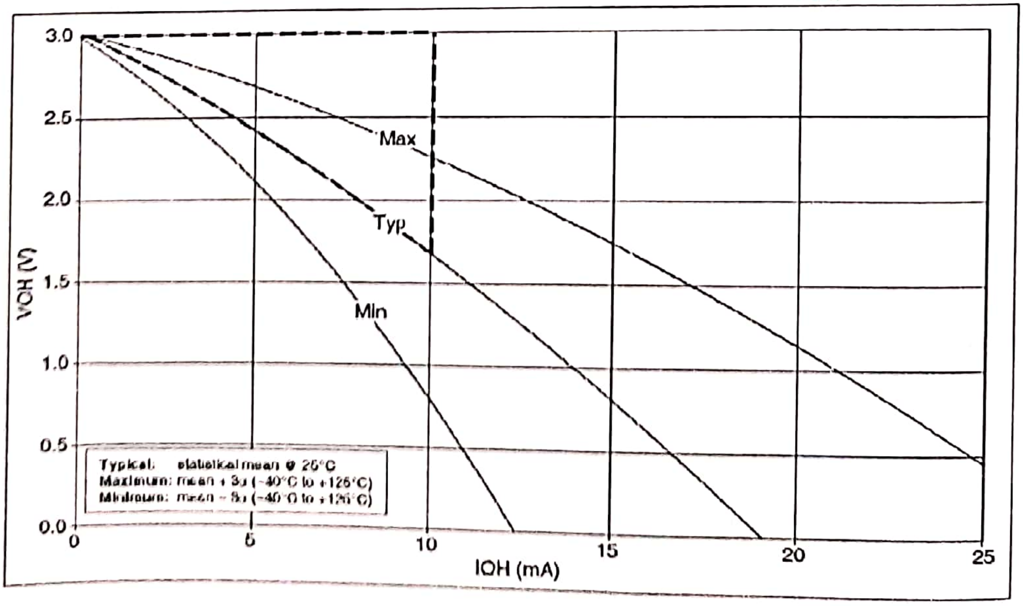
Port Output Characteristics

constant

$V_{OH} = V_s - IR$

الميل

V_{OH} vs. I_{OH} (VDD = 3V, −40 to 125°C)

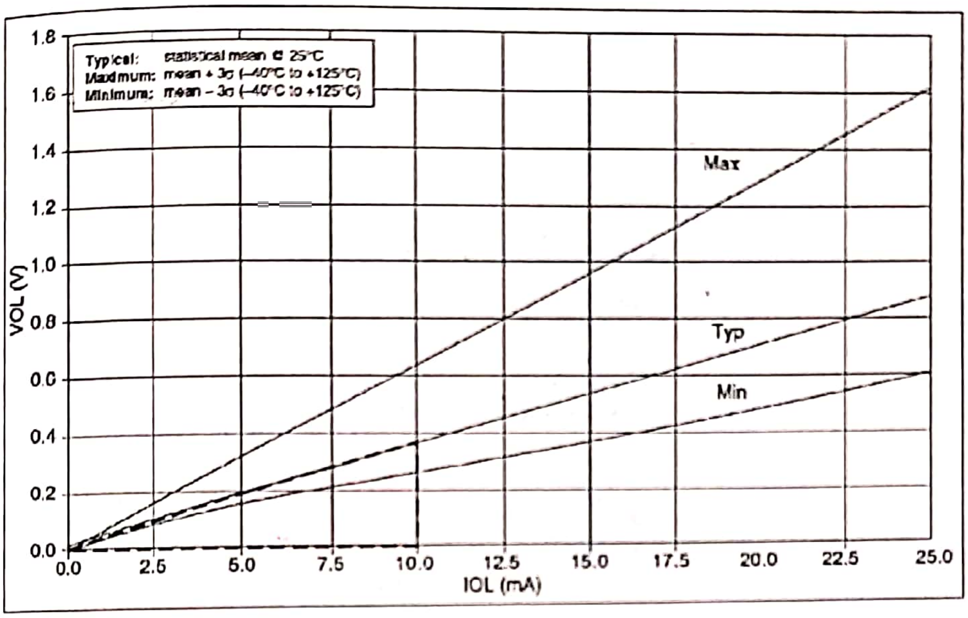


$R_{OH} = 130\ \Omega$

The PIC 16F84 Parallel Ports

Port Output Characteristics

V_{OL} vs. I_{OL}(VDD = 3V, −40 to 125°C)



$V_{OL} = IR$ ↓ الميل

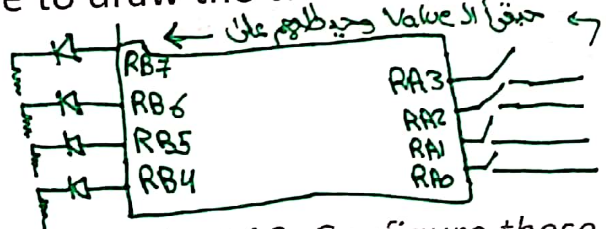
V_{OL} ↓

لو علي كثير
ممكن يصير
ينشاف V_{OH}
فبدي
انتبه

R_{OL} = 36 Ω

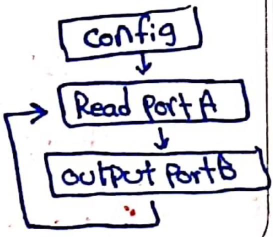
Example 3.1

- **Example** – Write a program that continuously reads an input value from 4 switches connected to PORTA (RA3-RA0) and display the value on 4 LEDs connected to PORTB (RB7-RB4). Make sure to draw the circuit and configure the ports properly.



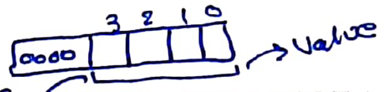
- ***Requirements:***

1) *Connect four switches to RA3-RA0. Configure these pins as input.*
2) *Connect four LEDs to RB7-RB4. Configure these pins as outputs.*

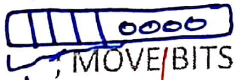


Example 3.1

```
        #include  "P16F84A.INC"
TEMP    EQU       0X20
        ORG       0X0000
; --------------------- MAIN PROGRAM ------------------------------------
MAIN    BSF       STATUS,RP0          ; SELECT BANK 1
        MOVLW     B'00001111'
        MOVWF     TRISA               ; CONFIGURE RA3-RA0 AS INPUT
        MOVLW     B'00000000'
        MOVWF     TRISB               ; CONFIGURE RB7-RB4 AS OUTPUT
        BCF       STATUS, RP0

REPEAT  MOVF      PORTA, W            ; READ FROM PORT A
        ANDLW     0X0F                ; MASK THE LOWER 4 BITS IN PORTA
        MOVWF     TEMP
        SWAPF     TEMP, F             ; MOVE BITS TO RB7-RB4
        MOVWF     PORTB

        GOTO      REPEAT
        END
```



هيك حتكون بال W



حتتبدل جنبوا

what if we don't want to affect the lower Pins of PORTB?

or operation

(فكري فيها) حنحط قبل ال MOVWF PORTB

Example 3.2

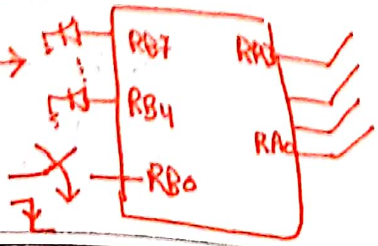
- **Example –** Modify the program and the circuit in Example 3.1 such that the switches are read and displayed when an external interrupt occurs (falling edge) only.

بس يصير falling edge

نفس المثال السابق بس عملية القراءة والكتابة مش حتتم إلا لما أكبس ال Switch

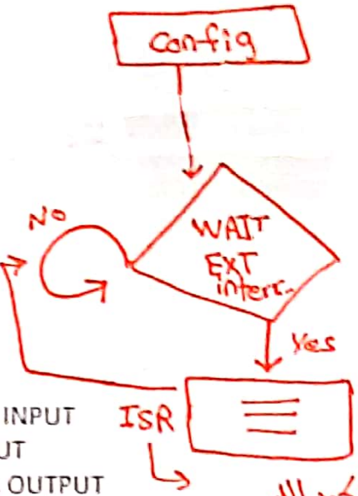
- ***Requirements:***

1) *Connect four switches to RA3-RA0. Configure these pins as input.*
2) *Connect four LEDs to RB7-RB4. Configure these pins as outputs.*
3) *Connect a switch to RB0 and configure it as input*

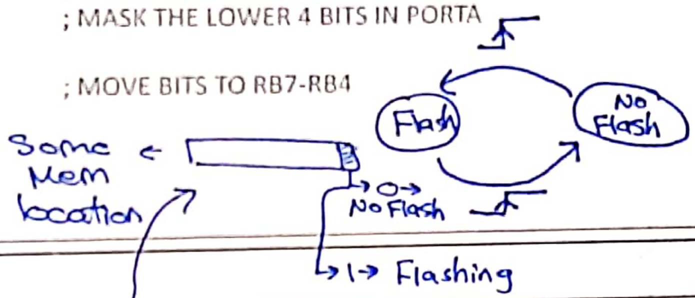


Example 3.2

```
            #include    "P16F84A.INC"
TEMP        EQU         0X20
            ORG         0X0000
            GOTO        MAIN
            ORG         0X0004
            GOTO        ISR
            ;------------------ MAIN PROGRAM ----------------------
MAIN        BSF         STATUS,RP0          ; SELECT BANK 1
            MOVLW       B'00001111'
            MOVWF       TRISA               ; CONFIGURE RA3-RA0 AS INPUT
            MOVLW       B'00000001'         ; CONFIGURE RB0 AS INPUT
            MOVWF       TRISB               ; CONFIGURE RB7-RB4 AS OUTPUT
            BCF         OPTION_REG, INTEDG  ; INTERRUPT ON FALLING EDGE
            BCF         STATUS, RP0
            BSF         INTCON, INTE        ; ENABLE INTERRUPT
            BSF         INTCON, GIE
WAIT        GOTO        WAIT                ; WAIT FOR INTERRUPT
            ;------------------- ISR ---------------------------------
ISR         MOVF        PORTA, W            ; READ FROM PORT A
            ANDLW       0X0F                ; MASK THE LOWER 4 BITS IN PORTA
            MOVWF       TEMP
            SWAPF       TEMP, F             ; MOVE BITS TO RB7-RB4
            MOVWF       PORTB
            BCF         INTCON, INTF
            RETFIE
            END
```



الكود اللي بالمثال السابق بس هلا صار جوا ال interrupt



* ال Flash بصير بال Main.

Example 3.3

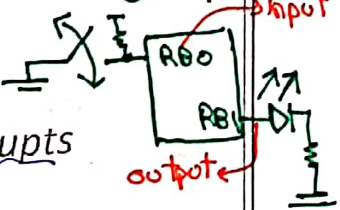
بضوي و بطفي بتباعد معين

- **Example –** Write a program to control the flashing of a LED that is connected to RB1 using a pushbutton that is connected to RB0. The LED starts flashing upon the arrival of the first rising edge on RB0. Afterwards, successive edges toggle the state of flashing (On, off, on, …). When the LED is flashing, this implies that it is 0.5 second ON and 0.5 second OFF. *Assume 4MHz clock.*

اذا كان شغال بطفي واذا كان طافي بشتغل

- ***Requirements:***

1) *Configure RB0 as input and RB1 as output*
2) *Enable external interrupt (INTE) and global interrupts (GIE)* No Flashing و Flashing من System لوظيفته بغير من حالة ال
3) *Write a 0.5 second delay routine*
4) *Keep track of the current status of flashing (on/off)* represent state as one bit



* بدي أتبع متى أسوي Flash ومتى لا

Example 3.3

```
        #include    "P16F84A.INC"
FLASH   EQU         0X20           ; STORE THE STATE OF FLASHING
COUNT1  EQU         0X21           ; COUNTER FOR DELAY LOOP
COUNT2  EQU         0X22           ; COUNTER FOR DELAY LOOP
        ORG         0X0000
        GOTO        START
        ORG         0X0004
        GOTO        ISR
; ------------------------------- MAIN PROGRAM ------------------------------------------
START   CLRF        FLASH          ; CLEAR FLASHING STATUS
        BSF         STATUS,RP0     ; SELECT BANK 1
        MOVLW       B'00000001'    ; CONFIGURE RB0 AS INPUT AND RB1 AS OUPUT
        MOVWF       TRISB
        BSF         OPTION_REG, INTEDG ; SELECT RISING EDGE FOR EXTERNAL INTERRUPT
        BSF         INTCON , INTE  ; ENABLE EXTERNAL INTERRUPT
        BSF         INTCON , GIE   ; ENABLE GLOBAL INTERRUPT
        BCF         STATUS,RP0     ; SELECT BANK 0
        CLRF        PORTB          ; CLEAR PORTB; TURN OFF LED
WAIT    BTFSS       FLASH , 0      ; IF BIT 0 OF FLASH IS CLEAR THEN NO FLASHING
        GOTO        WAIT           ; WAIT UNTIL BIT 0 IS SET
        MOVLW       B'00000010'
        XORWF       PORTB , 1      ; COMPLEMENT RB1 TO FLASH
        CALL        DEL_p5sec
        GOTO        WAIT
```

ممكن أشيلها

لهون بعمل config

No Flash

Flash

بيعمل complement

W → 0000 0010 ⊕
PB → XXXX XX?X
XXXX XX$\bar{?}$X

بنشتغل عهاي بس

$A \oplus 0 = A$
$A \oplus 1 = \bar{A}$ → عشان هيك $\bar{?}$

أو بدال هيك:

```
BSF PortB,1
Call Delay
BCF PortB,1
call Delay
goto wait
```

Example 3.3

```
--------------- INTERRUPT SERVICE ROUTINE ---------------
ISR     MOVLW       0x01
        XORWF       FLASH , F      ; COMPLEMENT THE STATUS
        BCF         INTCON , INTF  ; CLEAR THE INTF FLAG
        RETFIE
; --------------- DELAY ROUTINE -----------------------
DEL_p5sec
        MOVLW       D'0'
        MOVWF       COUNT1
        MOVLW       D'244'
        MOVWF       COUNT2
LOOP    NOP
        NOP
        NOP
        NOP
        NOP
        DECFSZ      COUNT1 , F
        GOTO        LOOP
        DECFSZ      COUNT2 , F
        GOTO        LOOP
        RETURN                     ; delay 0.500207 seconds

        END
```

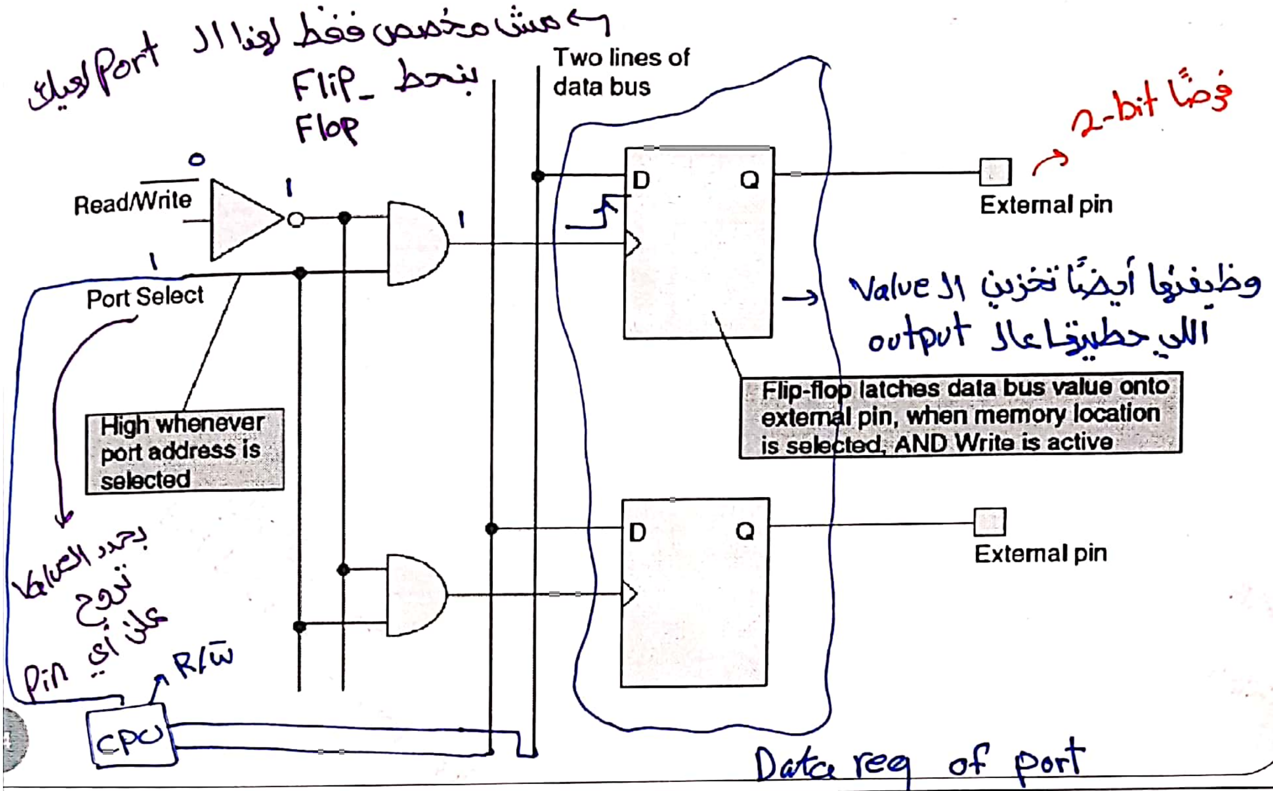
W 0000 0001 ⊕
Flash XXXX XXX ?
XXXX XXX $\bar{?}$

خده ك exercise الك توقف ال interrupt وتعمله بدل Polling.

Parallel Port طريقة بناء ال

Hardware Realization of Parallel Ports

Output Parallel Port



← مش مخصص فقط لهذا ال Port لعليك

بنحط Flip_Flop

High whenever port address is selected

بحدد الValue تروح على أي Pin

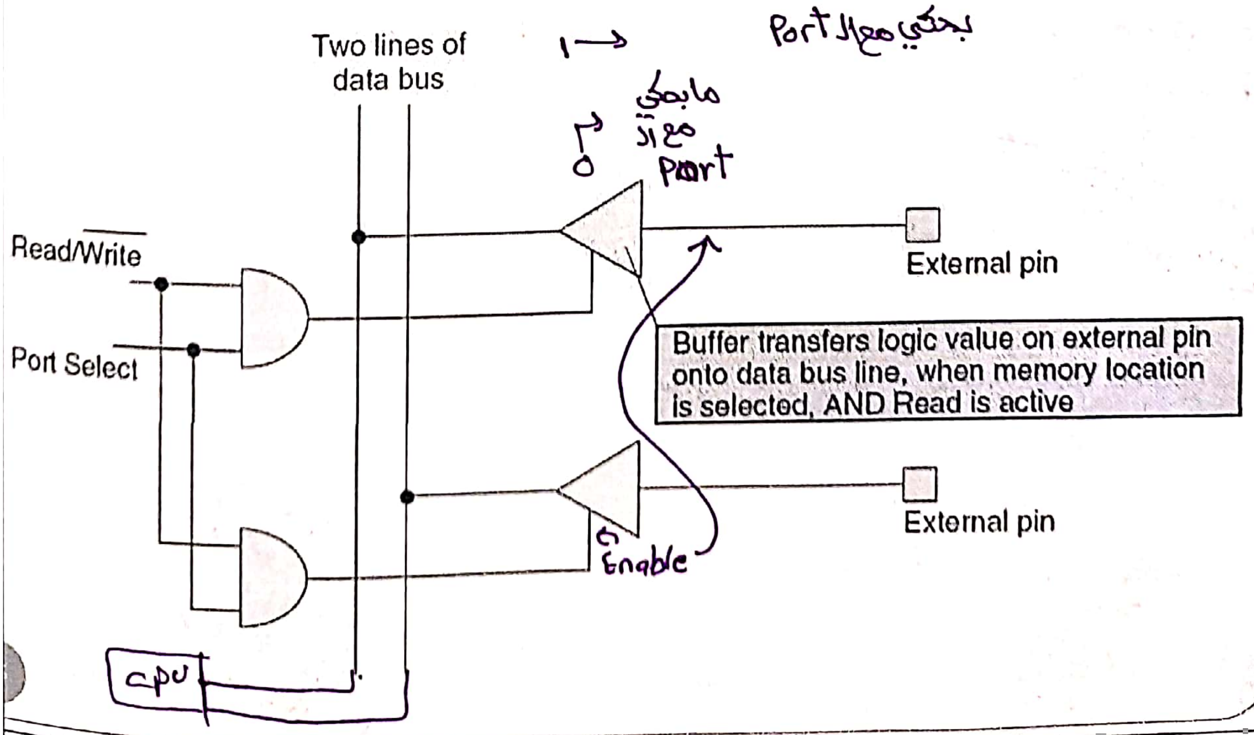
Flip-flop latches data bus value onto external pin, when memory location is selected, AND Write is active

وظيفتها أيضًا تخزين ال Value اللي حطيتها عال output

Data reg of port

Hardware Realization of Parallel Ports

Input Parallel Port



بحكي معال Port

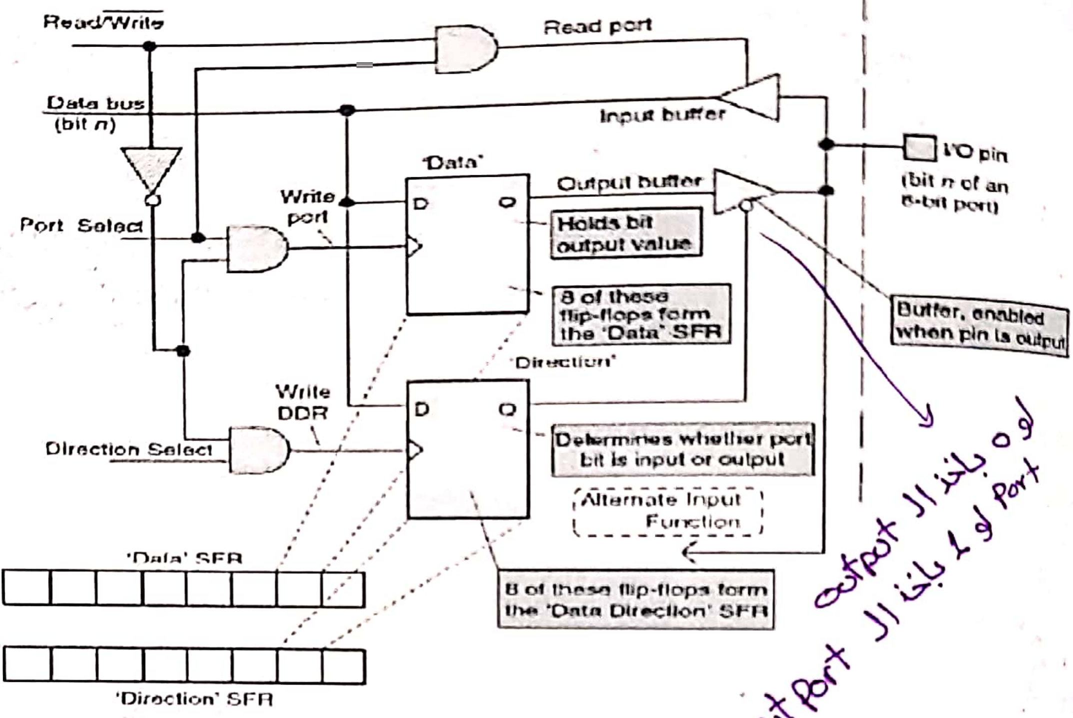
مابحكي مع ال Port

Buffer transfers logic value on external pin onto data bus line, when memory location is selected, AND Read is active

Hardware Realization of Parallel Ports

Bidirectional Parallel Port

جمع الصعق



PORTA

TRISA

في حالتنا

لو 0 باخذ ال output port

لو 1 باخذ ال input Port

Hardware Realization of Parallel Ports

PORT B

PINS RB3:RB0

0 → active

1 → disactive

سوا active low بشغل لو كان 0 =

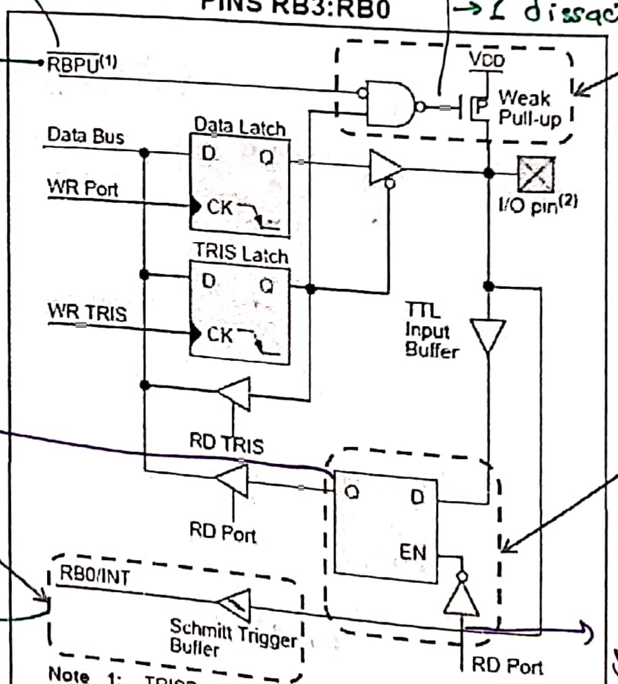
آخر bit بال option-reg

بقدر اسوي في Enable

ال pull-up reg

ب Port B

Configurable pull-up resistors using RBPU bit in the OPTION register



Note 1: TRISB = '1' enables weak pull-up (if RBPU = '0' in the OPTION_REG register).

2: I/O pins have diode protection to VDD and VSS.

بفيدني بتثبيت ال Value خلال عملية القراءة

Latches input data whenever the port is read

لو كان 0 ستو ما تتغير D بتضل زي ما هي Q

لو 1 اللي موجود ع D بظهر ع Q

Pins بتأدي أكثر من وظيفة ← Multiplexed input

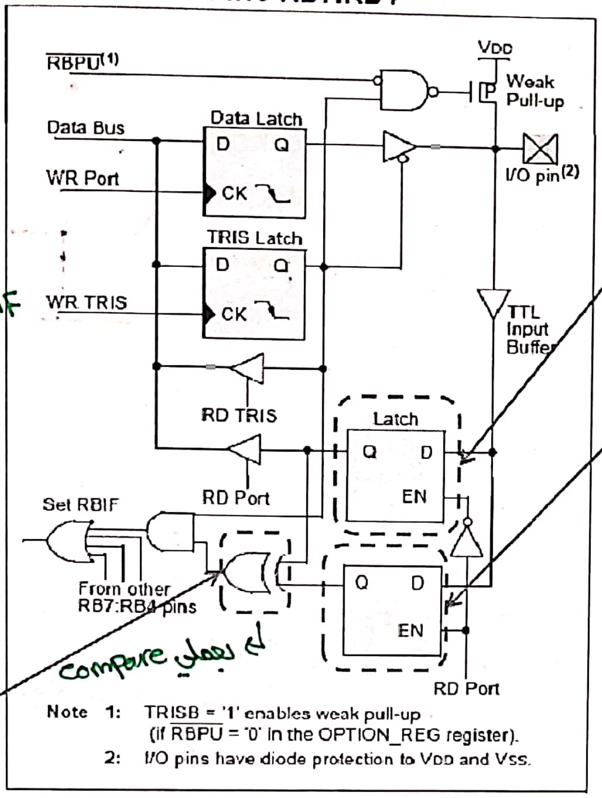
interrupt circuit ← خاص ب RB0

Scanned with CamScanner

Hardware Realization of Parallel Ports

PORT B

PINS RB7:RB4



Note 1: TRISB = '1' enables weak pull-up (if RBPU = '0' in the OPTION_REG register).

2: I/O pins have diode protection to VDD and VSS.

Lathes data on port read

Holds previous latched data

MovF PoRTB,w
BCF option_REG, RBIF

↑
Clearing the RBIF bit ?

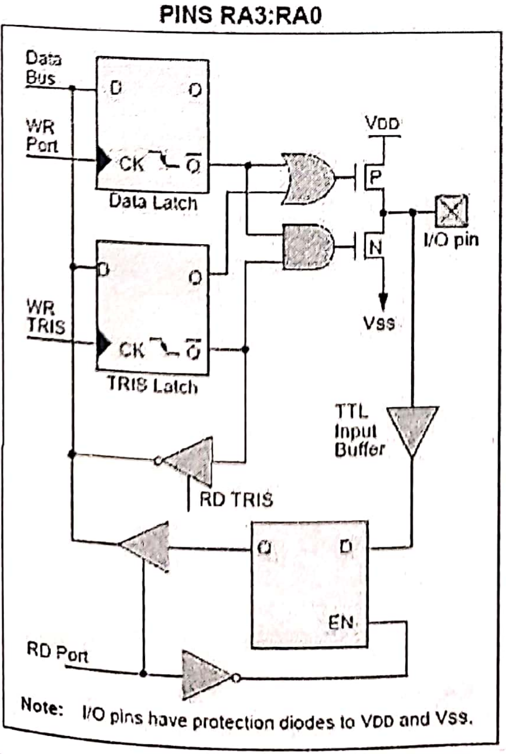
Compares previous and present port input values

Hardware Realization of Parallel Ports

نفس المبدأ

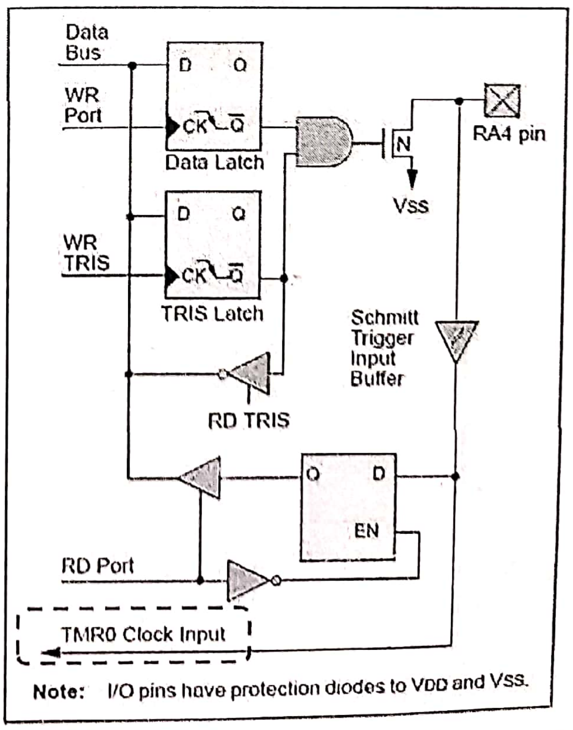
PORT A

PINS RA3:RA0



Note: I/O pins have protection diodes to VDD and VSS.

RA4

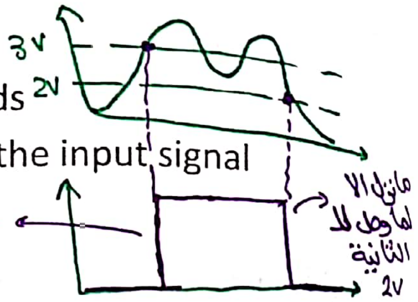


Note: I/O pins have protection diodes to VDD and VSS.

Hardware Realization of Parallel Ports

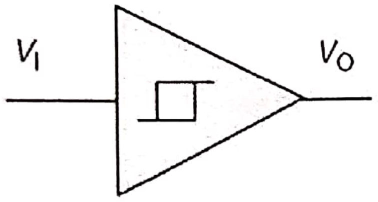
Electrical Characteristics

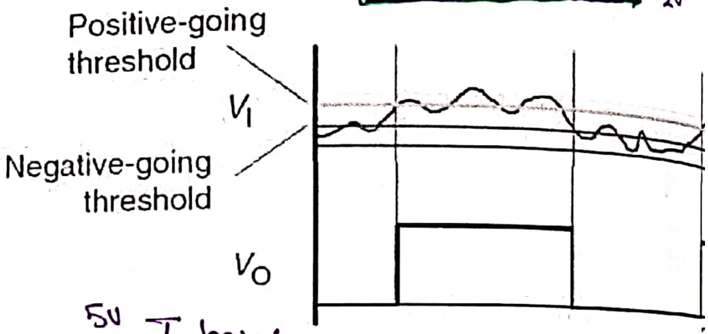
- **Schmitt Trigger Input**
  - A special type of gate with two thresholds
  - Remove fluctuations and corruptions in the input signal

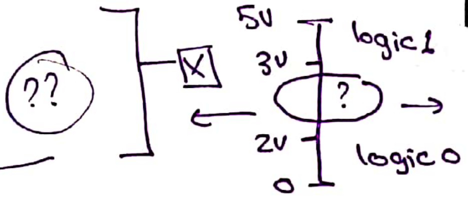


ماطلع إلا لما وصل 3V

مانزل إلا لما وصل للثانية







شو ال system حيشوفها

1 او 0

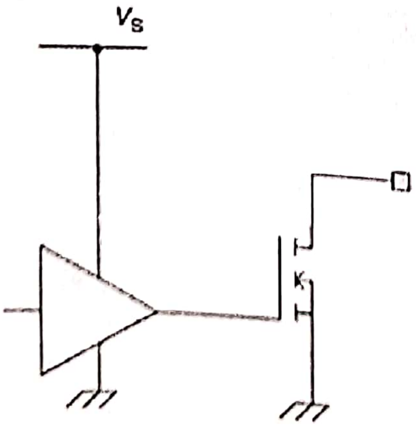
الحل

قراءة

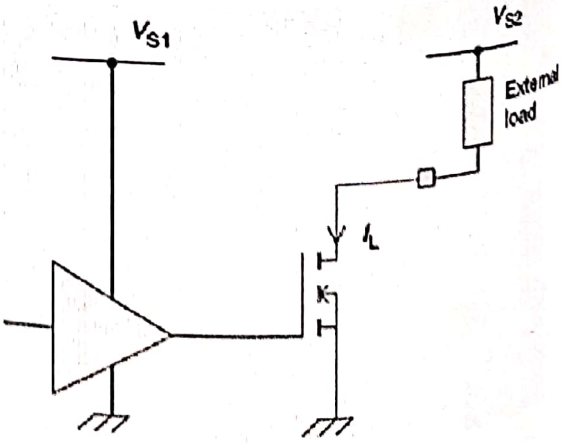
Hardware Realization of Parallel Ports

Electrical Characteristics

- **Open Drain Output**
  - Flexible style of output that can be adapted as a standard logic output or a direct drive for small loads



Open Drain Output

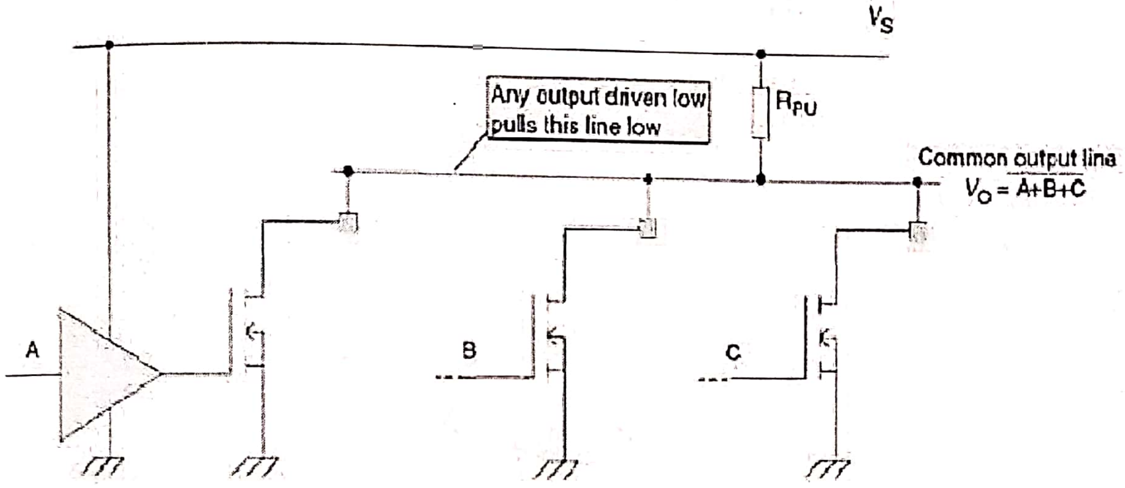


Open Drain Output Driving A Small Load

Hardware Realization of Parallel Ports

Electrical Characteristics

- **Open Drain Output**
  - Can be used as a wired-OR

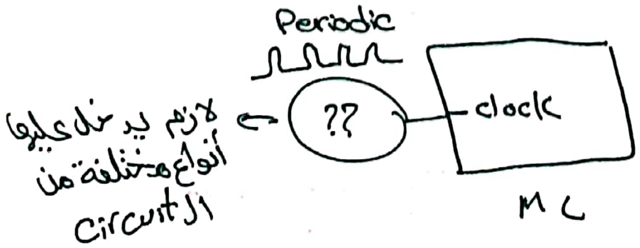


The Oscillator

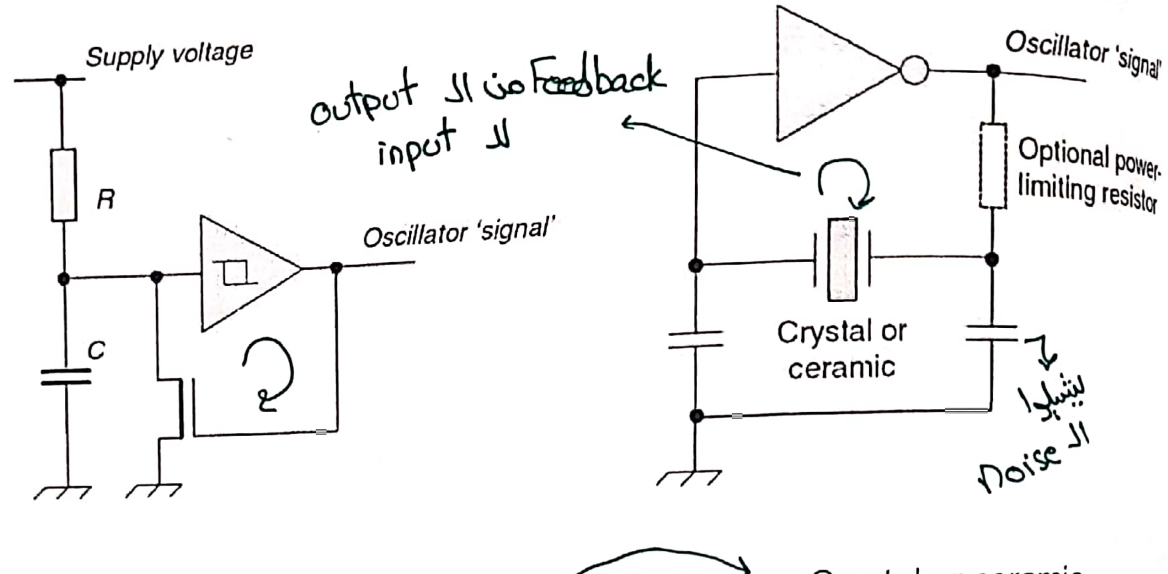
- The choice of clock determines the operating characteristics for the microcontroller

- Faster clock gives faster execution, but more power consumption

- Accurate and stable operation of the microcontroller requires accurate and stable clock



The Oscillator
Oscillator types



Resistor–capacitor (RC).
- *low cost*
- *not precise* ← مش دقيق

البديل →

Crystal or ceramic
- *expensive*
- *stable and precise* ← دقيقة
- *mechanically fragile* ← ممكن تخرب لو وقعت لأنه الكريستالة خربت بتكون

بجيبها مصدر الclock

The PIC 16F84A Oscillator

- The 16F84A can be configured to operate in four different oscillator modes using the F0SC1 and F0SC0 in the configuration word

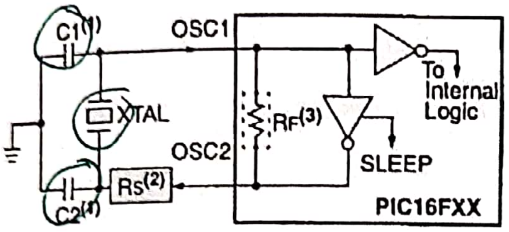
R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u	R/P-u
CP	CP	CP	CP	CP	CP	CP	CP	CP	CP	PWRTE	WDTE	FOSC1	FOSC0
bit13													bit0

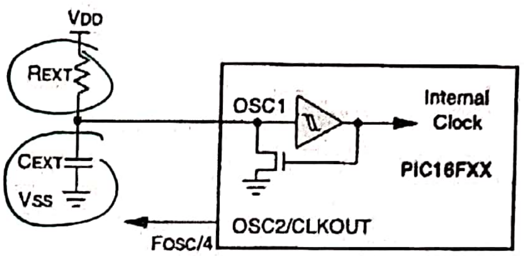
F0SC1	F0SC0	Mode
0	0	LP oscillator – *intended for low frequency (<200 KHz) crystal application to reduce power consumption*
0	1	XT oscillator – *standard crystal configuration (1-4 MHz)*
1	0	HS oscillator – *high speed (>= 4MHz)*
1	1	RC oscillator - *requires external resistor an capacitor*

The PIC 16F84A Oscillator

- The 16F84A has two oscillator pins ; OSC1 and OSC2.

الي هدوين بالأخضر انت بتجيبهم الباقي موجودين وجاهزين.



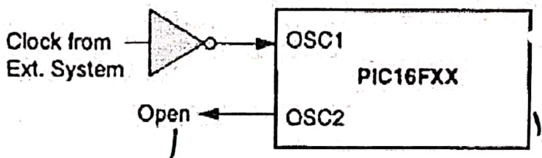


XT configuration

RC configuration

التحديد ٥
وظيفتها تخبرك أي
Circuit الي حتحكم فيها
شغلك RC ولا XT وهكذا

Clock دخلت ←
جاهزة ما استخدمت
ولا بيركت
من الي
فوق



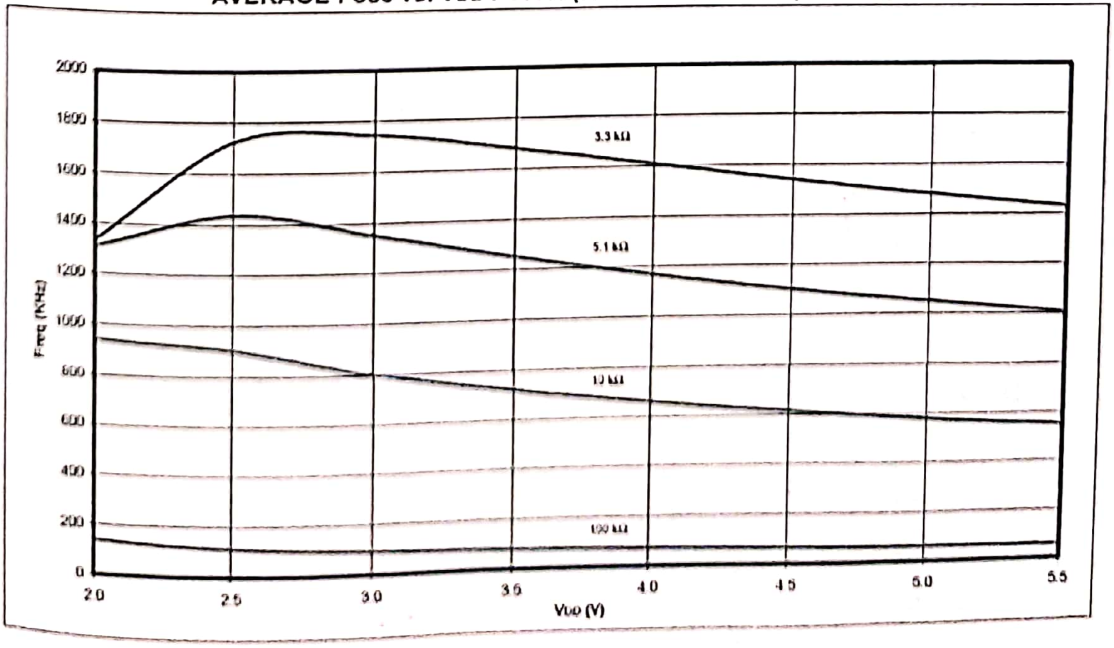
External Clock

عشان أفهمها إنه ال clock خارجية
لل MC

The PIC 16F84A Oscillator

- RC oscillator frequency dependence on power supply

AVERAGE FOSC vs. VDD FOR R (RC MODE, C = 100 pF, 25°C)



The Power Supply

PIC16F84A-04
(Commercial, Industrial, Extended)
PIC16F84A-20
(Commercial, Industrial, Extended)

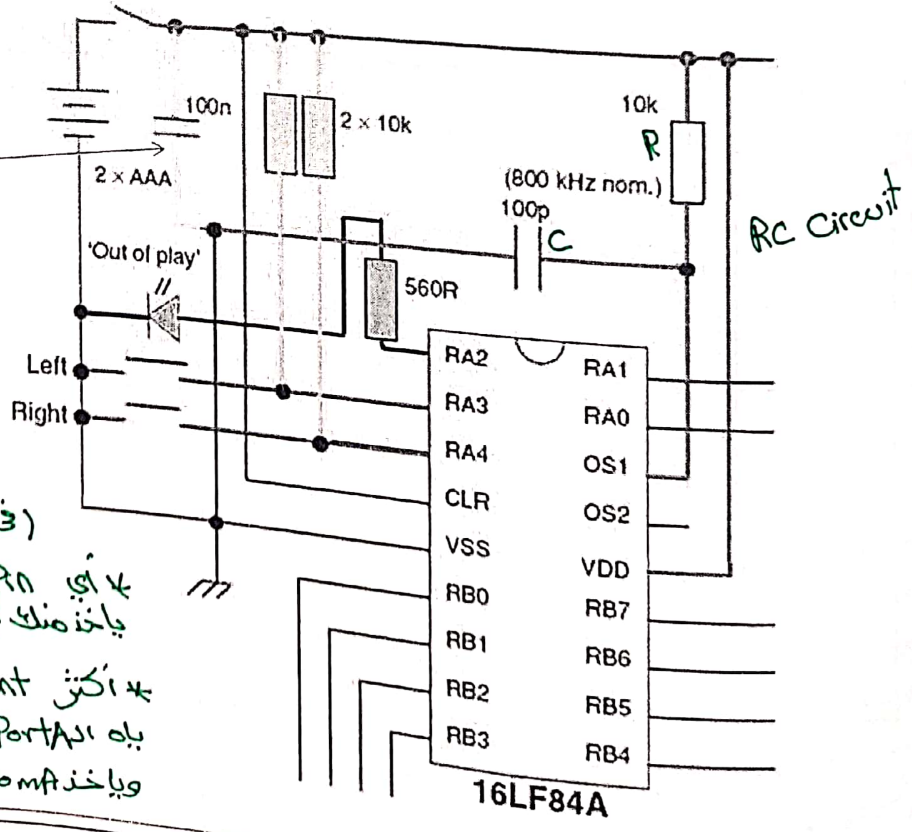
Standard Operating Conditions (unless otherwise stated)
Operating temperature 0°C ≤ TA ≤ +70°C (commercial)
-40°C ≤ TA ≤ +85°C (industrial)
-40°C ≤ TA ≤ +125°C (extended)

Param No.	Symbol	Characteristic	Min	Typ†	Max	Units	Conditions
	VDD	**Supply Voltage**					
D001		16LF84A	2.0	—	5.5	V	XT, RC, and LP osc configuration
D001 D001A		16F84A	4.0 4.5	— —	5.5 5.5	V V	XT, RC and LP osc configuration HS osc configuration
D002	VDR	**RAM Data Retention Voltage (Note 1)**	1.5	—	—	V	Device in SLEEP mode
D003	VPOR	**VDD Start Voltage** to ensure internal Power-on Reset signal	—	Vss	—	V	See section on Power-on Reset for details
D004	SVDD	**VDD Rise Rate** to ensure internal Power-on Reset signal	0.05	—	—	V/ms	
	IDD	**Supply Current (Note 2)**					
D010		16LF84A	—	1	4	mA	RC and XT osc configuration (Note 4) Fosc = 2.0 MHz, VDD = 5.5V
D010		16F84A	—	1.8	4.5	mA	RC and XT osc configuration (Note 4) FOSC = 4.0 MHz, VDD = 5.5V
D010A			—	3	10	mA	RC and XT osc configuration (Note 4) FOSC = 4.0 MHz, VDD = 5.5V (During FLASH programming)
D013			—	10	20	mA	HS osc configuration (PIC16F84A-20) FOSC = 20 MHz, VDD = 5.5V
D014		16LF84A	—	15	45	µA	LP osc configuration FOSC = 32 kHz, VDD = 2.0V, WDT disabled

---

The Power Supply

100 nF decoupling capacitor



(في سكرينشوت لهالحكي)

* أي Pin بال MC بيقدر ياخذ منك أو بعطيك 25 mA

* أكثر current بقدر يعطيك باه ال PortA هو 50 mA وياخذ 80 mA

Summary

- Parallel ports allow the exchange of data between the outside world and the CPU
- It is essential to understand the electrical characteristics and internal circuitry of ports
- All microcontrollers need a clock. The clock speed determine the power consumption
- Active elements of the oscillator are usually built inside the microcontroller and the designer selects the type and configure it
- It is a must to understand the power requirements of the microcontroller

---

بنقلهم واحد ورا الثاني مش مع بعض ←

Starting with Serial

Chapter 10
Sections 1,2,9,10

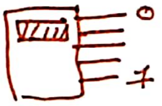
Dr. Iyad Jafar

Outline

- Introduction
- Synchronous Serial Communication
- Asynchronous Serial Communication
- Physical Limitations
- Overview of PIC 16 Series
- The 16F87xA USART
- Summary

Introduction

- Microcontrollers need to move data to and from external devices من و إلى
- In general, two approaches
  - ***Parallel*** مجموعة من الـ bits ← النقل بنفس اللحظة
    - Data word bits are transferred at the same time
    - A wire is dedicated for each bit
    - Simple and fast but expensive
    - Short distances → مشاكل



كلما زاد الطول بيغلى فبنقصره

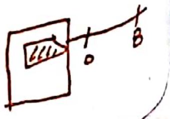
expensive

① بسبب إنه

② عشان التداخل بين الأسلاك الموجودة بزيد كل ما طولت

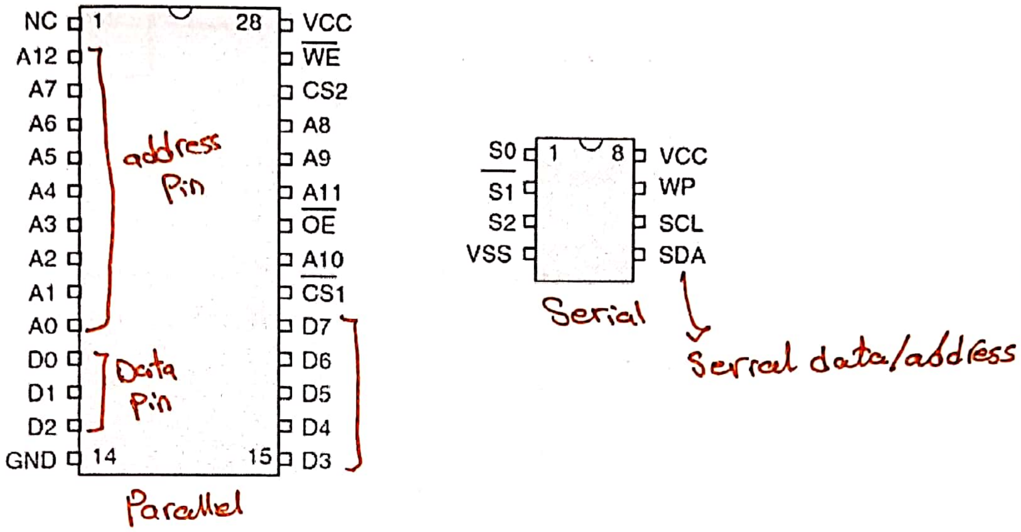
لانه بيتناسب مع طريقة تخزين بعض الكمبيوترات للمعلومات

  - ***Serial*** ← بعالج بعض مشاكل الـ Parallel
    - Bits are transferred one after another over the same link/wire /channel
    - Requires complex hardware to transmit and receive
    - Slow but cheap
    - Short and long distances



Introduction

- Two memories of the same size. However, one uses parallel transfer while the other uses serial

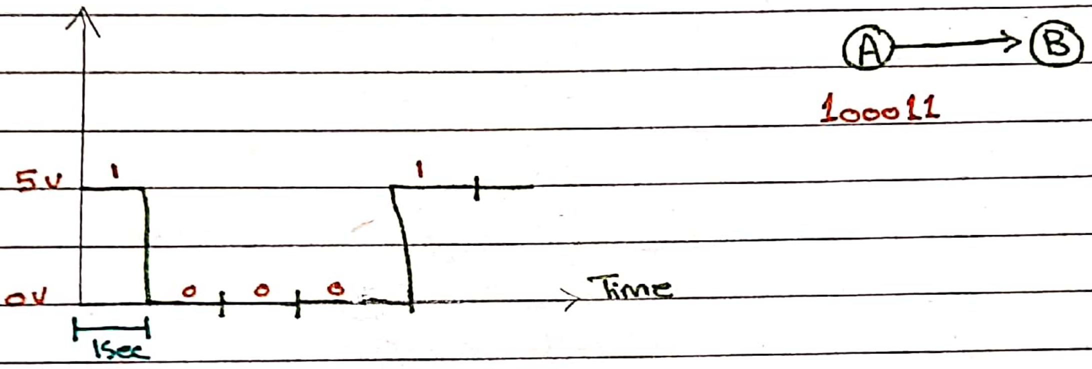


Serial Communication

- Bits are transferred one after another on the same wire !!!
- **Challenges**
  - How to distinguish the start and end of the bit ?
  - How to determine the start and end of a word ? ↳ bits مجموعة ال
- **Two approaches** طريقة: ال Time والتنسيق بينظم
  - Synchronous serial communication
    - A separate clock signal is sent in parallel with the data
    - Each clock cycle represents one bit duration
  - Asynchronous serial communication مافي تزامن لنقل ال clock مع الداتا
    - No clock signal !
    - Timing is derived from the data itself

Slide 5 :

بدي أرسم الـ data اليوم تنتقل من الـ SystemA لـ الـ System B بالنسبة للزمن :

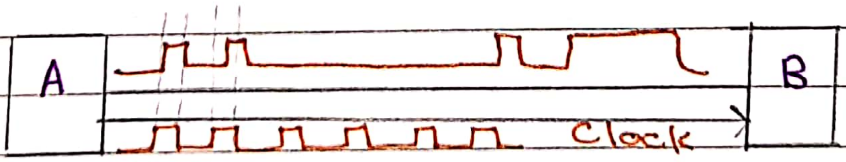


* الـ B كيف حيعرف إنه مثلاً الـ 0v أو الـ 5v اللي وصلته عبارة عن Single
ولا أكثر من bit. مثلاً لو وصله 5v كيف حيعرف هاي الـ 5v عبارة عن 1 بس ولا 11
ولا أكثر؟ لازم A و B يكونوا متفقين ع duration معينة. مثلاً الـ Time من A إلى B
بياخذ 1sec ، فمن اللحظة اللي يبعت فيها A بكون عند A و B ساعة بحيث إنه A كل
1sec بتطلع 1bit و B كل 1sec بقرأ 1 bit.

* من التحديات الأخرى إني أعرف بداية ونهاية الـ word (group of bits) اللي حنقلها
من A لـ B؟ حيكون متفق A مع B على حجم الـ word وحيكون عند B عداد (counter)
يبين كم bits استلم وهنا الـ Counter بكون بناء على الـ duration of bit وعلى
الـ word size.

*Two approaches of Timing :

1- Synchronous : إذا كان A بده يبعت لـ B مجموعة من الـ bits بسرعة معينة،
فممكن يبعت الـ clock ← in parallel مع الـ data



بصير واحد يعمل Sampling على الـ rising/falling والثاني ↲
يبعت الـ Value على الـ edge اللي عكس الأولى.

* مش شرط الـ Clock حد من A, B يبعتوا. ممكن تكون خارجية وتبعت لـ A و B

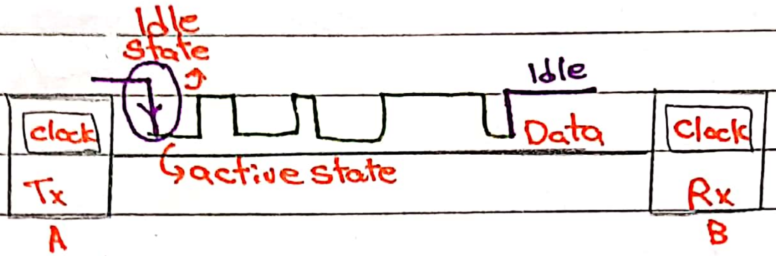
* الـ Clock بتنبعت تزامناً مع الـ Data.

* مشكلته صار بدي 2 wires و 2 pins عشان ابعت كل شيء؟ وحيصير في cost وتداخل
بين الأسلاك لو كانت طويلة فبرجع بستخدم أسلاك قصيرة (short distance)

PAGE

DATE

2- Asynchronous : لن أقوم بإرسال ال data بشكل متزامن مع ال clock

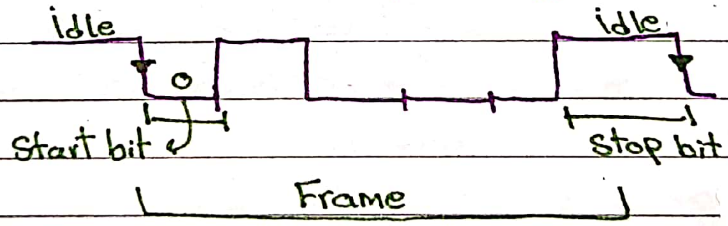


*حنفرض إنه داخل A في Clock وداخل B في Clock والشرط الرئيسي إنه Clock A = Clock B أو قريبة جداً منها . فالساعتين حيعدوا بنفس السرعة . بس هاد لحاله مو كافي ، لازم نعد بنفس اللحظة ، مشكلتنا في بداية العد : سلك ال Data بالبداية لما مايكون في Data يبعتها بكون 1 أول مايصير في Data بصير 0 ، هاي اللحظة الزمنية المؤشر عليها A حيشغل فيها الساعة تاعته وبنفس الوقت بسوي Triggerd ل B وبشغل الساعة تبعته فهيك بيلشو سوا.

* بكون في 0 بعد ال idle دايمًا بكون اسمه Start bit ومن هون بنشغل الساعة.

* لما يخلص ال A عملية ال Transmission لازم يرجع السلك لحالة ال idle وتسمى Stop bit بطفي عندها الساعة.

* ال Frame عبارة عن ال Data مع ال Start/stop bit :



* أفضل من ال Synch ولكن هناك شروط لتساوي Clock A / B .

* مشكلته بطيء : Synch of 8 bits → 8 sec / word

asynch of 8 bits → 10 sec / word

↳ 1 sec for start bit

↳ 1 sec for stop bit

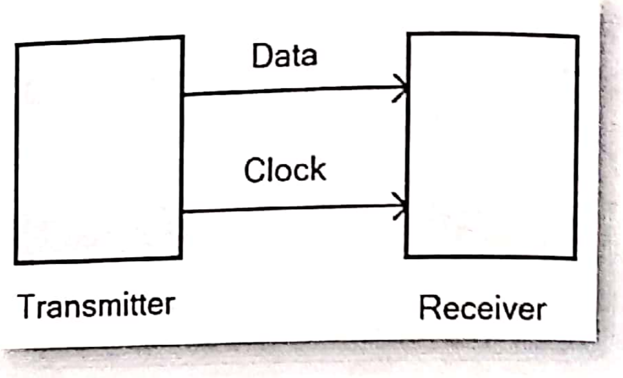
* على الرغم من ذلك بحل مشاكل ال Synch .

* حنركز على ال asynch .

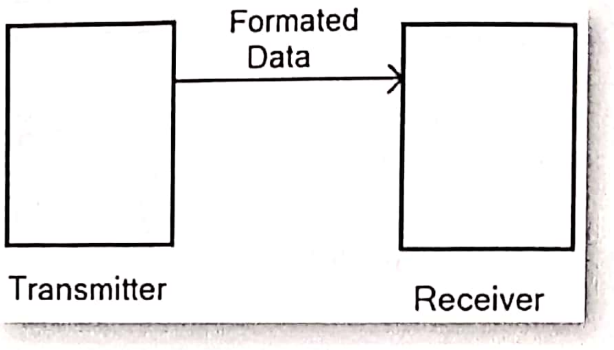
Scanned with CamScanner

Serial Communication

Synchronous



Asynchronous



Serial Communication

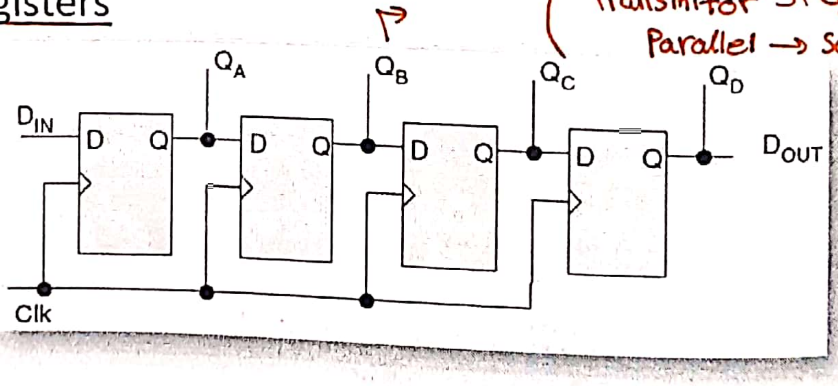
- Data inside the memory and microprocessor is formatted in parallel. How to transmit it serially?
- Shift registers

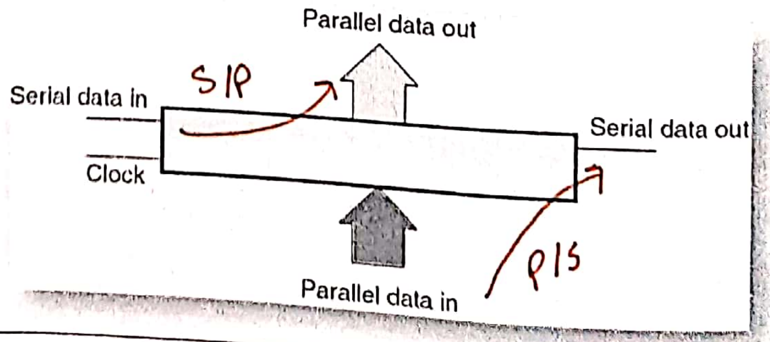
عشان أفصل ال word ل bits

بدخل عليه ال data parallel وهو بيجيلي ال bit ..

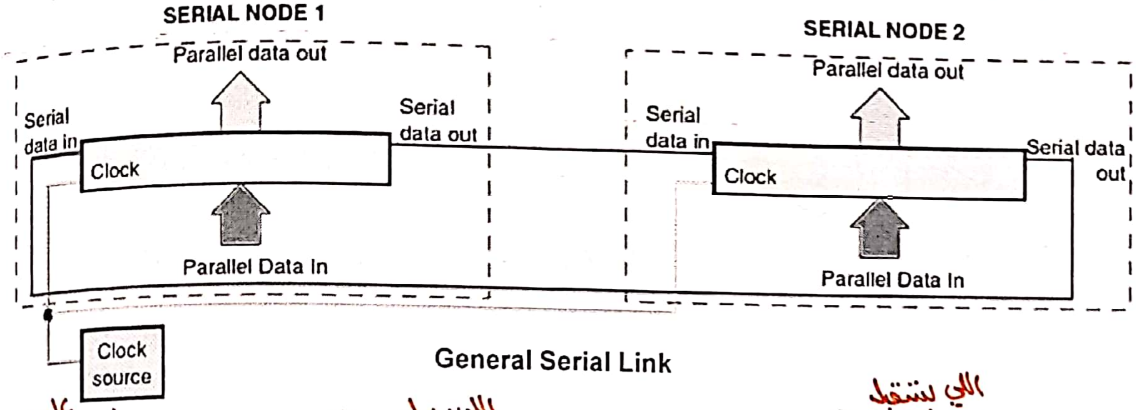
(بيحتاجها ال Transmitor
Parallel → Serial)

(المستقبل يحتاج
Serial → Paralle



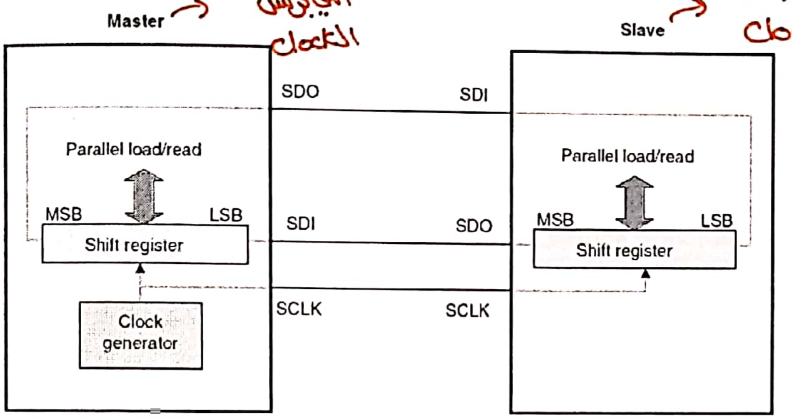


Synchronous Serial Communication



General Serial Link

clock ← خارجية

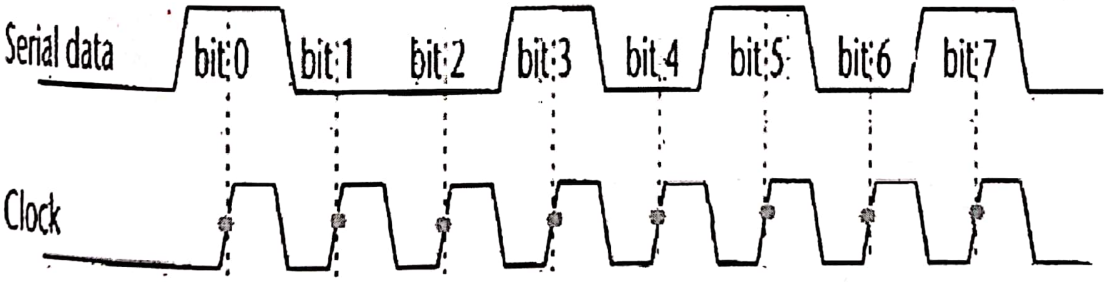


Master → الي بيرسل الclock

Slave → الي بستقبل الclock

Synchronous link implemented using a microcontroller

Synchronous Serial Communication



Advantages

- Simple hardware
- Efficient
- High speed

Disadvantages

- Extra line for the clock → frequency
- The bandwidth needed for the clock is twice the data bandwidth
- Data and clock may lose synchronization over long distance →

لأنه حيصير شي Time delay وتأخر للداتا أعلى

Asynchronous Serial Communication

- No clock signal !
- The transmitter and receiver should operate a clock at the same rate
- To synchronize the clocks of the transmitter and receiver, data is framed with a start and stop bits

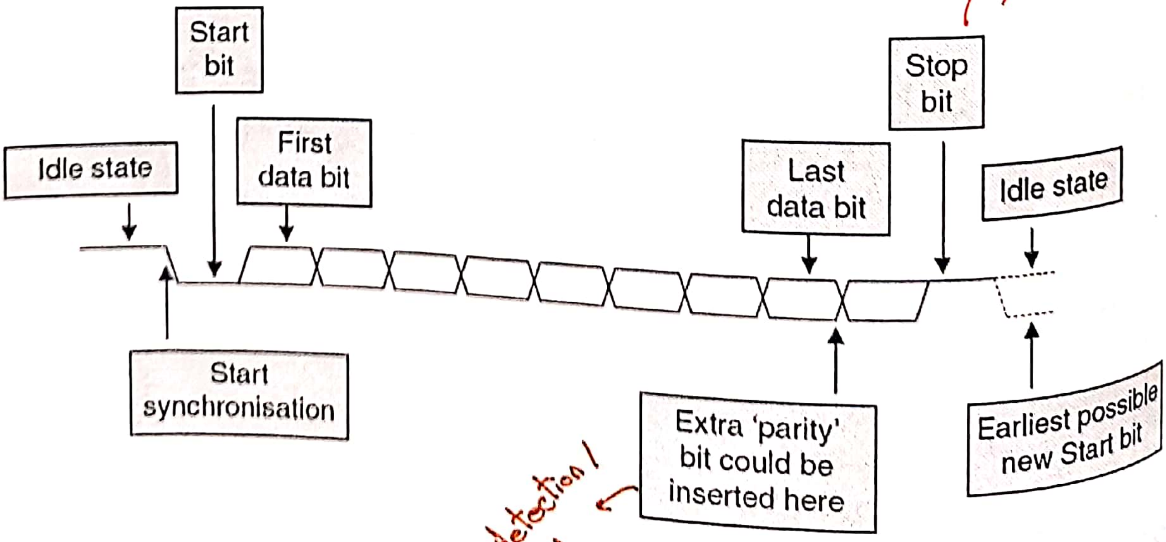
---

مشكلة حصير ابعت bit أكثر.

Asynchronous Serial Communication

- **Framing**

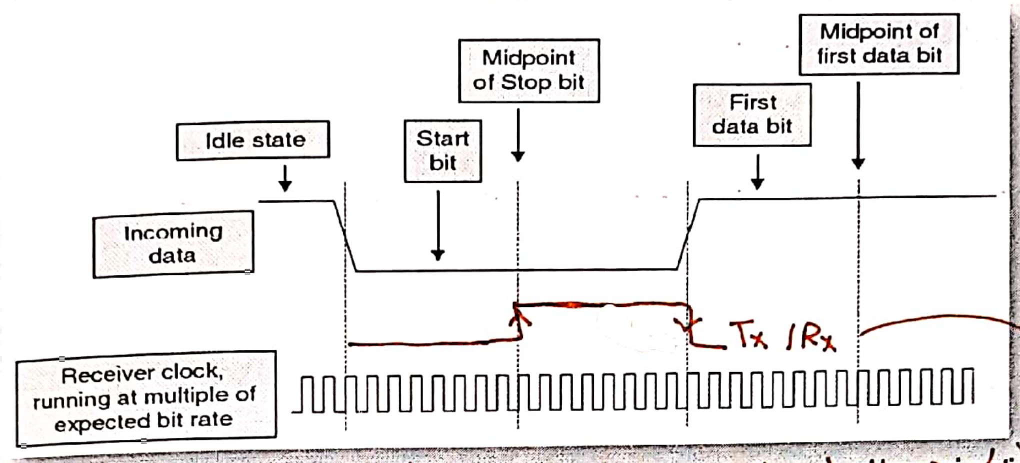
مسؤولية ال Transmitor



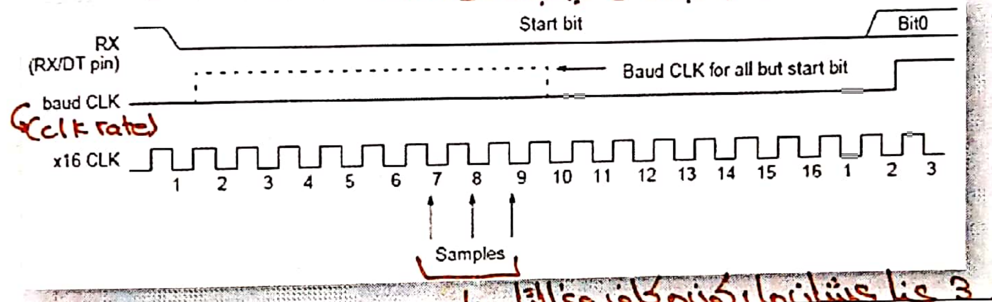
error detection / correction bit

Asynchronous Serial Communication

- Synchronization



فعليًا هيك بكون شغال وبشوف أكثر قيمة قرأها بتكون هي الجواب واللي بعدها واللي بعدها وهكذا. وهيك أفضل

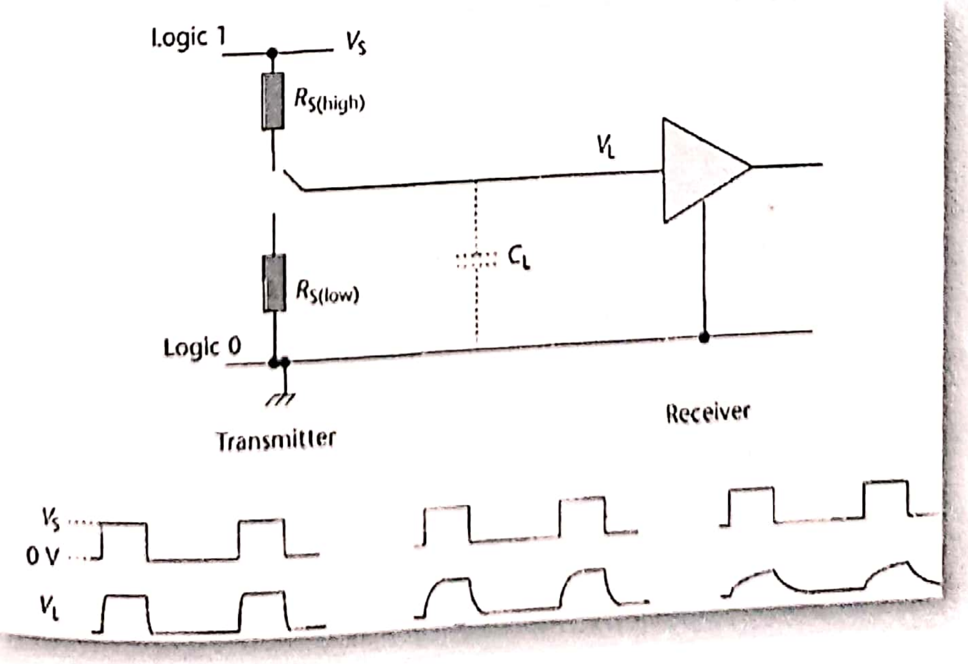


بياخذ 3 عنا عشان ما يكون مكلف وغالبًا دا بياخذ اللي بالنص

Physical Limitations (اقرأيهم)

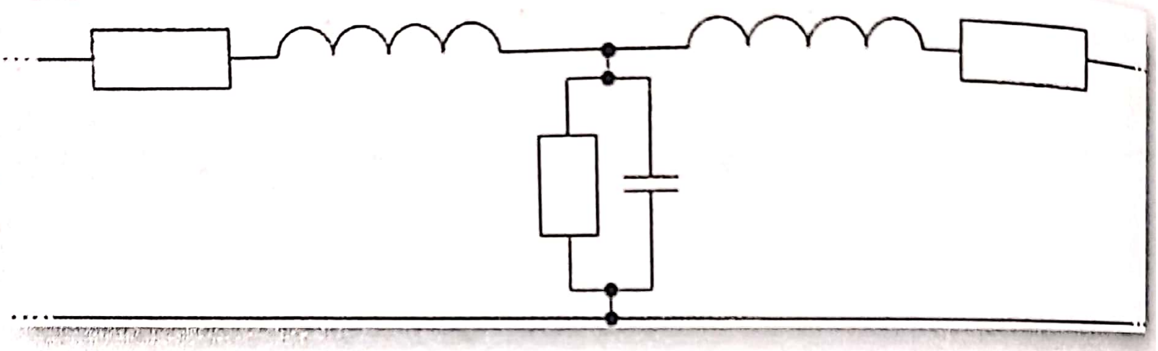
المشاكل اللي ممكن تواجهني

- Time Constant effect



Physical Limitations

- Transmission Line Effects
  - Characteristic impedance and reflections
  - Lines should be terminated properly



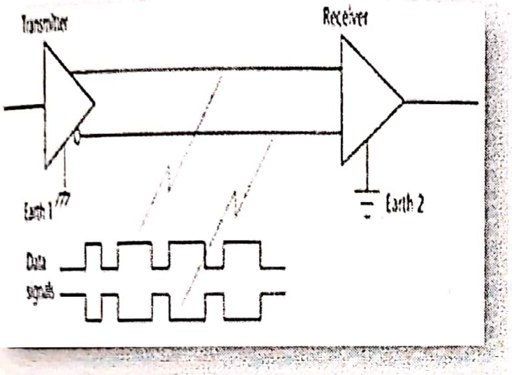
Physical Limitations

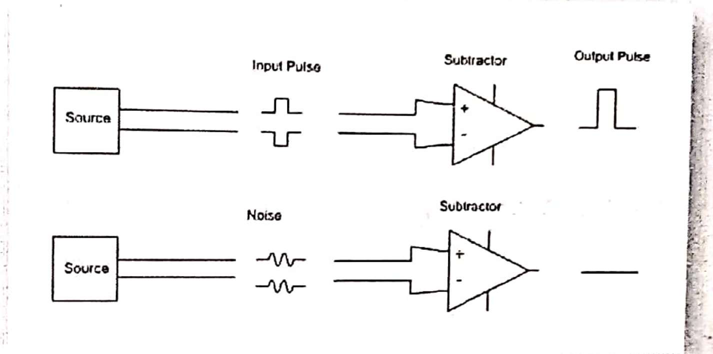
تيجيني مقاطعة على السلك تبعي زي مثلا ينفتح راديو ولا يرن تلفون.

- **Electromagnetic Interference**
  - Generated due to high voltage rates of change.
  - How to minimize:
    - At source:
      - reduce voltage rate of change.
    - In communication link:
      - large separation from source of interference.
      - Increase data voltage.
      - Screening
      - Use optical links
    - At receiver:
      - Use filtering techniques

Physical Limitations

- **Ground Differentials**
  - With longer wires, ground potential at one point might not be the same at another point.
  - Solutions:
    - Differential transmission.
    - Electrical isolation
    - Use optical communication links





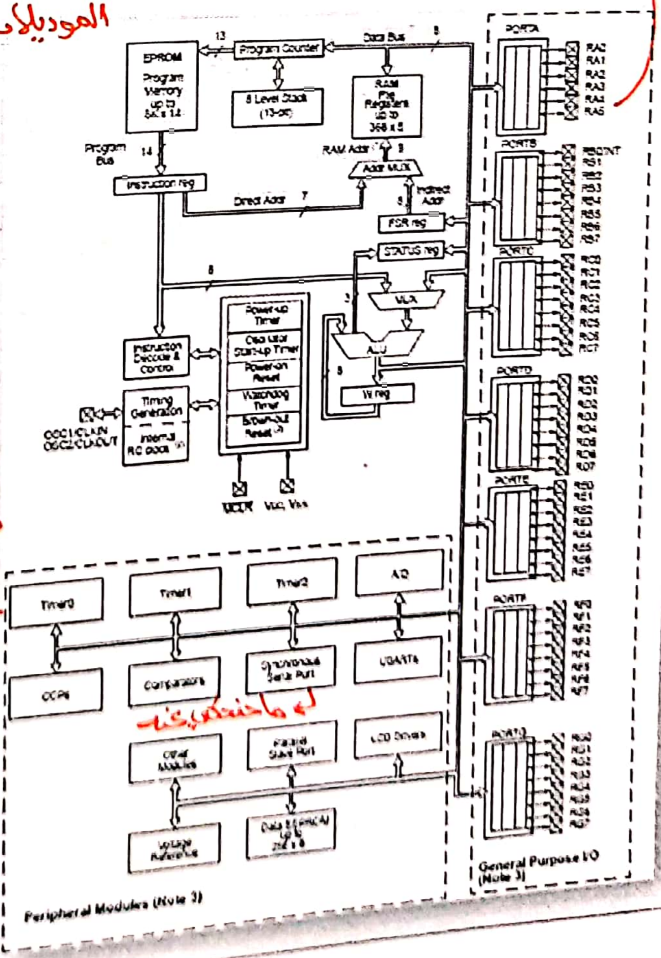
ومع انو استخدمناه قبل 5bit بس عادي في كتير أنواع وأعداد للـbits للـport بسبب الموديلات

Overview of the PIC 16 Series

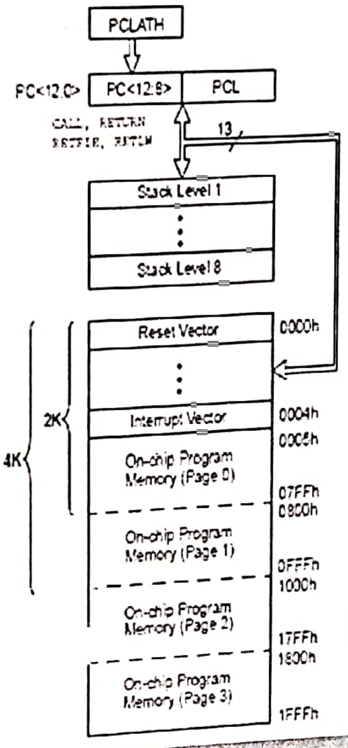
- We have already seen the PIC 16F84A
- Other members in the series have more features:
  - Additional I/O ports
  - More HW timers
  - A/D converters
  - LCD Drivers
  - USARTs → Asynch نفعل
  - Synchronous Serial
  - Comparators
  - ....

كانكاس هو هلا في أكثر من اثنين لسبب الموديلات

لو ماحنحكي عنه



Overview of the PIC 16 Series



	File Address		File Address		File Address		File Address
INDF	00h	INDF	80h	INDF	100h	INDF	180h
TMR0	01h	OPTION_REG	81h	TMR0	101h	OPTION_REG	181h
PCL	02h	PCL	82h	PCL	102h	PCL	182h
STATUS	03h	STATUS	83h	STATUS	103h	STATUS	183h
FSR	04h	FSR	84h	FSR	104h	FSR	184h
PORTA	05h	TRISA	85h		105h		185h
PORTB	06h	TRISB	86h	PORTB	106h	TRISB	186h
PORTC	07h	TRISC	87h	PORTF	107h	TRISF	187h
PORTD	08h	TRISD	88h	PORTG	108h	TRISG	188h
PORTE	09h	TRISE	89h		109h		189h
PCLATH	0Ah	PCLATH	8Ah	PCLATH	10Ah	PCLATH	18Ah
INTCON	0Bh	INTCON	8Bh	INTCON	10Bh	INTCON	18Bh
PIR1	0Ch	PIE1	8Ch		10Ch		18Ch
PIR2	0Dh	PIE2	8Dh		10Dh		18Dh
TMR1L	0Eh	PCON	8Eh		10Eh		18Eh
TMR1H	0Fh	OSCCAL	8Fh		10Fh		18Fh
T1CON	10h		90h		110h		190h
TMR2	11h		91h		111h		191h
T2CON	12h	PR2	92h		112h		192h
SSPBUF	13h	SSPADD	93h		113h		193h
SSPCON	14h	SSPSTAT	94h		114h		194h
CCPR1L	15h		95h		115h		195h
CCPR1H	16h		96h		116h		196h
CCP1CON	17h		97h		117h		197h
RCSTA	18h	TXSTA	98h		118h		198h
TXREG	19h	SPBRG	99h		119h		199h
RCREG	1Ah		9Ah		11Ah		19Ah
CCPR2L	1Bh		9Bh		11Bh		19Bh
CCPR2H	1Ch		9Ch		11Ch		19Ch
CCP2CON	1Dh		9Dh		11Dh		19Dh
ADRES	1Eh		9Eh		11Eh		19Eh
ADCON0	1Fh	ADCON1	9Fh		11Fh		19Fh
General Purpose Registers (2)	20h	General Purpose Registers (3)	A0h	General Purpose Registers (3)	120h	General Purpose Registers (3)	1A0h
	7Fh		EFh		16Fh		1EFh
		Mapped in Bank0 70h - 7Fh (4)	F0h FFh	Mapped in Bank0 70h - 7Fh (4)	170h 17Fh	Mapped in Bank0 70h - 7Fh (4)	1F0h 1FFh
Bank0		Bank1		Bank2 (3)		Bank3 (3)	

كانو مش يكون بموديلنا اللي كنا بنشتغل عليه

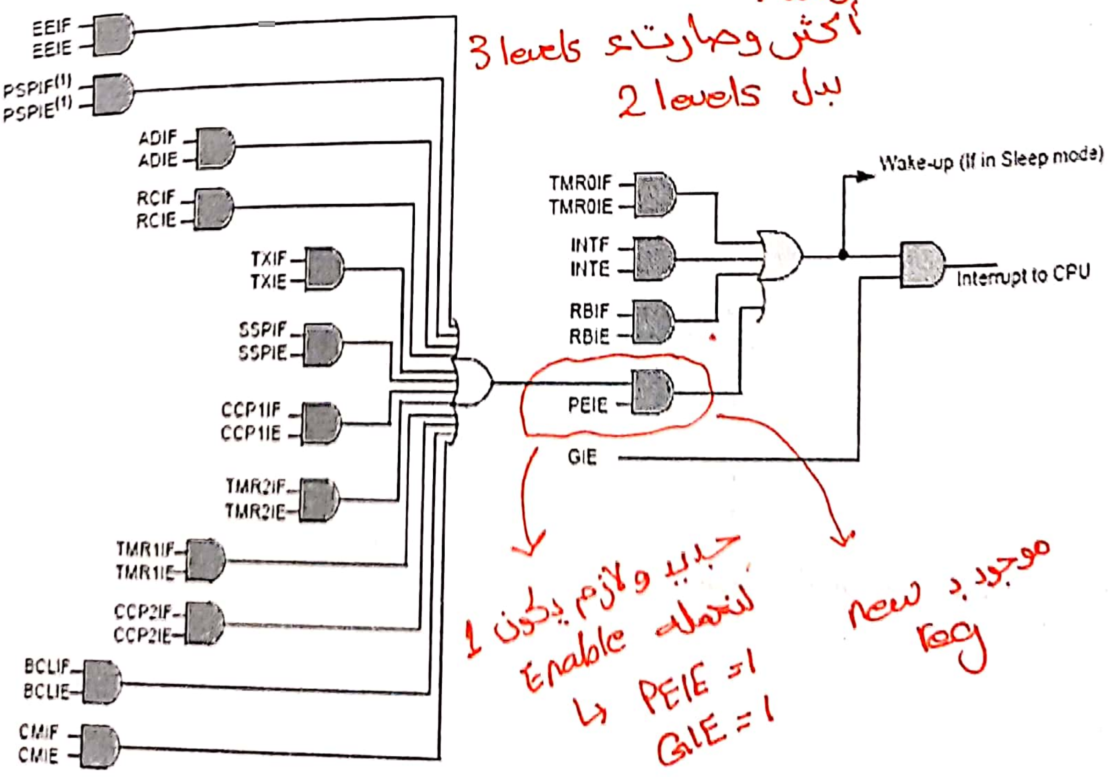
PE IE اللي تحت

PE IE اللي تحت

ال data Mem كمان بتتغير وبتكبر حسب الموديل

Overview of the PIC 16 Series
Interrupt Logic for 16F874A/16F877A

لـ توسعنا أكثر وصارت 3 levels بدل 2 levels



جديد ولازم يكون 1 لنفعله Enable لـ PEIE =1 GIE =1

موجود بـ new reg

Overview of the PIC 16 Series

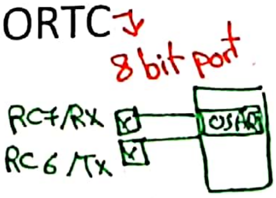
Device	Pins	Features
16F873A 16F876A	28	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM, **2 serial,** → USART + Synch اللي شفناها بالرسمة 5 10-bit ADC, 2 comparators
16F874A 16F877A	**40**	**5** parallel ports, 3 counter/timers, 2 capture/compare/PWM, **2 serial,** → USART + synch اللي شفناها بالرسمة **8** 10-bit ADC, 2 comparators

The 16F87xA USART

بشغلها Sync و async

- The 16F87XA family has a Universal Synchronous Asynchronous Receiver Transmitter (USART)
  - Configurable → بقدر أحدد شو طبيعة عملها → بشغلها Transmitter/ Reciver أو الاثنين سوا
  - Half duplex synchronous master or slave → استقبل بس أو ارسل بس
  - Full-duplex asynchronous transmitter and receiver → استقبل وارسل بنفس الوقت
- The USART → داخل ال PIC shares pins with PORTC → 8 bit port
  - pin 7 being the receive line
  - pin 6 being the transmit line
- Operation involves the following registers → اللي بحتاجها بالتعامل

استقبل باتجاه واحد أو أرسل باتجاه واحد مش سوا



TXSTA (0x98) TXREG (0x19) RCSTA (0x18)
RCREG (0x1A) SPBRG (0x99) PIE1 (0x8C)
PIR1 (0x0C) INTCON (0x0B, 0x8B,0x10B,0x18B)
TRISC (0x87)

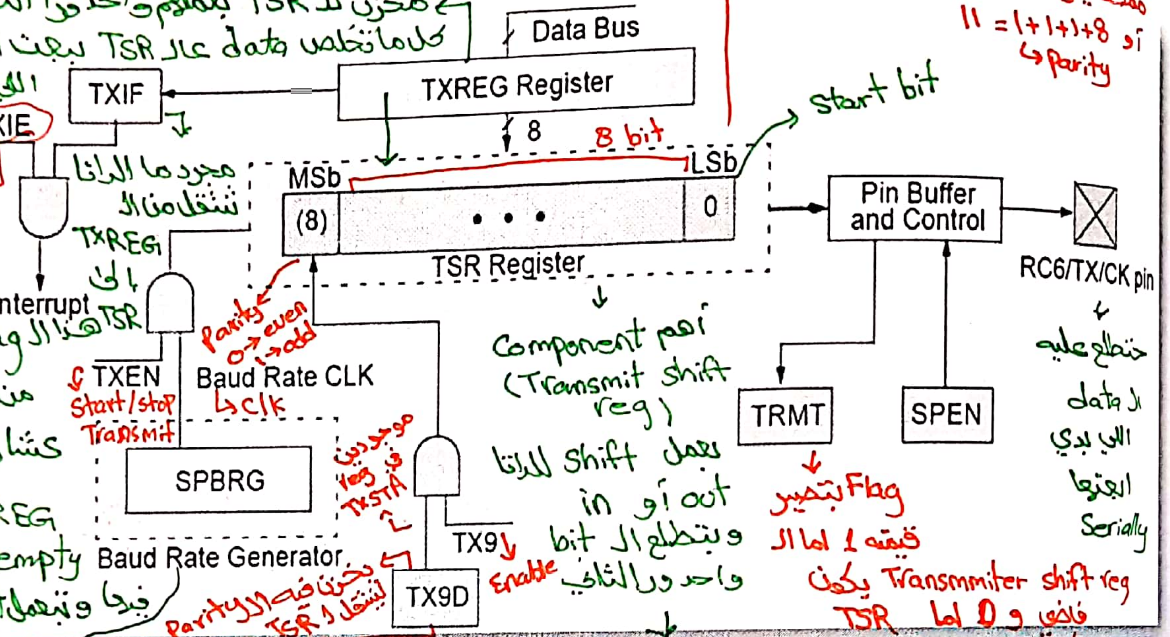
SPEN ← عن طريقه بفهم ال Micro إنه هذا ال pin بي استخدمه ك Serial port porpuses مش ك general Pin
↳ RC6/TX/CX
SPEN ↓ Serial port Enable bit

Transmiter ↓
The 16F87xA USART

- **Asynchronous USART Transmitter Block Diagram**

ال Frame ممكن يكون 1+1+8 = 10
أو 1+1+1+8 = 11
↳ Parity

← مخزن لل TSR بعملهم واحد ورا التاني
كلما تخلص data عال TSR بيبعت الي بعدها الي كنت مخزنها بال TXREG



Clear/ Set by the hardware

مجرد ما الداتا تنتقل من ال TXREG لل TSR هذا ال Flag حيصير 1 من قبل الهاردوير عشان يبلغك إنه TXREG صارت empty ونقدر نخزن فيها وتعمل interrupt لتسريع الوقت

بخزن فيه ال Parity لينتقل ل TSR

Component أهم (Transmit shift reg) بعمل Shift للداتا in أو out وبتطلع ال bit واحد ورا التاني

ماله address بقدرش أوصله مباشرة بوصله عن طريق TXREG فأنا ك Programmer بشوف TXREG فقط

TRMT ↓ بتصير Flag قيمته 1 لما ال Transmmiter shift reg يكون فاضي و 0 لما TSR يكون ببعت داتا.
بدل ما interrupt فنستخدم Polling

RC6/TX/CK pin ↓ حنطلع عليه ال data الي بدي ابعتها Serially

مسؤولة عن ال generation تبع ال clk قيمتها أنا بحددها

أنا ك Programer لازم أحسب ال Parity قبل ما أخزن ال Value الي حبعتها لل TSR عن طريق كود يعدلي عدد ال 1's اللي موجودين في ال 8 bit تاعوتك ويحددهم even ولا odd

بتحدد سرعة الداتا أثناء الارسال أو الاستقبال

The 16F87xA USART

- **Asynchronous USART Transmitter Operation Notes**

TXEN ← حتى لو كان 1 مابلش إلا لما تكون TSR مليانة مش فارغة

- Data is transmitted LSB first on RC6 pin
- The shift register TSR is buffered by the TXREG (19H) and is not accessible as a memory location
- Transmission is controlled by the TXEN bit which enables the clock to start the transmission
- To enable serial transmission on RC6, bit SPEN in RCSTA register has to be set
- To transmit data, it must be loaded in the TXREG. It is transferred to TSR immediately if no transmission or after the stop bit from previous transmission is sent out
- Transmission status is provided by two bits:
  - TXIF flag in PIR1 register indicates the status of TXREG. It is set when data is transferred to TSR. It is cleared on writing to TXREG. (TXIF is cleared by hardware and it is read-only).
  - TRMT flag in TXSTA it is set when the shift register is empty
- Parity bit can be sent out by using TXD9 bit and TX9 in TXSTA

The 16F87xA USART

TXSTA (98H)

R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0	R-1	R/W-0
CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D
bit 7			مهم				bit 0

bit 7 **CSRC:** Clock Source Select bit
<u>Asynchronous mode:</u>
Don't care.
<u>Synchronous mode:</u>
1 = Master mode (clock generated internally from BRG)
0 = Slave mode (clock from external source)

bit 6 **TX9:** 9-bit Transmit Enable bit
1 = Selects 9-bit transmission
0 = Selects 8-bit transmission

bit 5 **TXEN:** Transmit Enable bit
1 = Transmit enabled
0 = Transmit disabled
Note: SREN/CREN overrides TXEN in Sync mode.

bit 4 **SYNC:** USART Mode Select bit → by default
1 = Synchronous mode
0 = Asynchronous mode

bit 3 **Unimplemented:** Read as '0'

bit 2 **BRGH:** High Baud Rate Select bit
<u>Asynchronous mode:</u>
1 = High speed
0 = Low speed
<u>Synchronous mode:</u>
Unused in this mode.

bit 1 **TRMT:** Transmit Shift Register Status bit
1 = TSR empty
0 = TSR full

bit 0 **TX9D:** 9th bit of Transmit Data, can be Parity bit

The 16F87xA USART

- **Steps for Using the asynchronous transmitter**

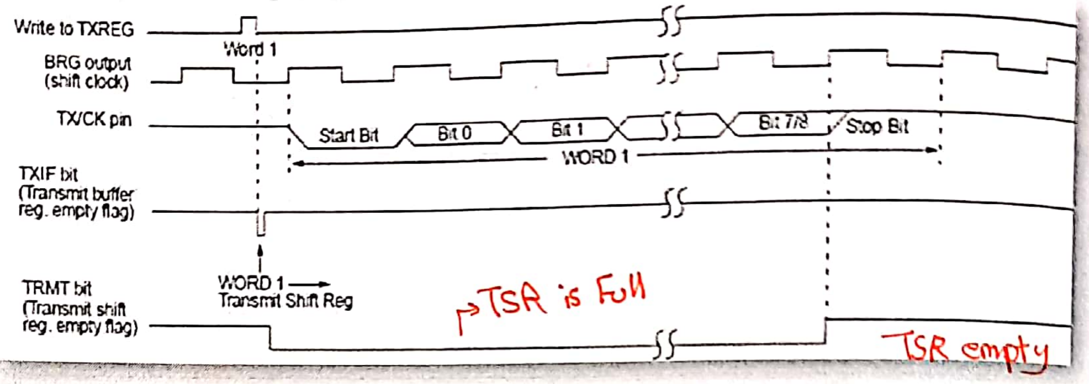
1. Clear TRISC<6> bit to configure RC6 as output
2. Set the SPBRG (0x99) register and BRGH (TXSTA<2>) bit to choose the appropriate baud rate (more on this later) → clk rate تحسيقية
3. Enable asynchronous serial port by clearing the SYNC (TXSTA<4>) → by default is cleared bit and setting the SPEN bit (RCTSA<7>) → To enable serial port
4. If interrupts are desired, set the TXIE (PIE1<4>), GIE (INTCON<7>), and PEIE (INTCON<6>) bits → if iwant use interrupt
5. If 9-bit transmission is desired, set the TX9 (TXSTA<6>) bit → لو بدي ابعث 9bit
6. Enable transmission by setting the TXEN (TXSTA<5>), which will set the TXIF (PIR1<4>) bit → To indicate Tx Reg is empty
7. If 9-bit transmission is selected, then the ninth bit should be loaded in TX9D (TXSTA<0>) → parity bit لو بدي ابعث ال
8. Load data in TXREG (0x19) to start the transmission

) → مهم الترتيب

The 16F87xA USART

كويس نخطط عليه

- **Timing of asynchronous transmission**



- **Registers involved in asynchronous transmission**

انحط بدل EEPROM وهادا داخل reg بجد

ملخص الـ reg والـ bits

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
19h	TXREG	USART Transmit Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

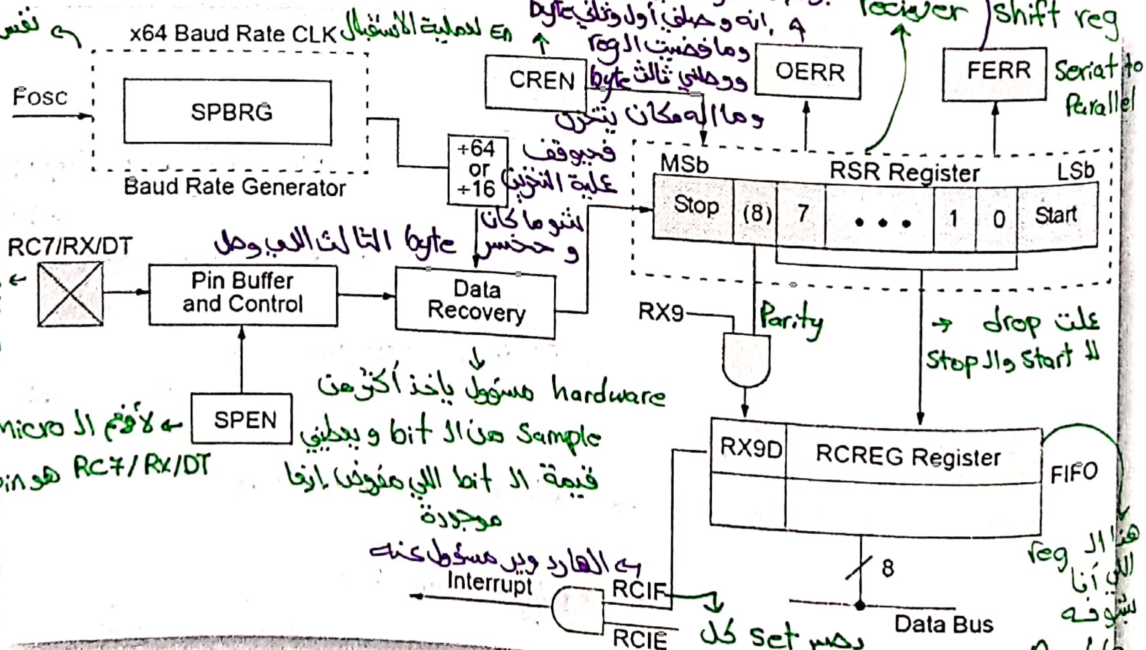
not shared اللي لونه أبيض

ولكنت بستقبل الـ bits والـ stop bit اللي وصلك كان 0 بدل 1 فهاي الحالة اسمها Framing error يعني في مشكلة عالخط

The 16F87xA USART

عكس شغل الـ Transmitter

- **Asynchronous Receiver**



نفس الـ Transmitter

En لعملية الاستقبال

over run عن طريقه بعرف إنه صار عندي مشكلة إنه وصلني أول وثاني byte وما فضيت الـ reg ووصلني ثالث byte وما إله مكان يتخزن فبيوقف عملية التخزين لشو ما كان وحسب byte الثالث اللي وصل

reciever shift reg

حطيتها أجزاء ثم أجمعها

Serial to Parallel

بستخدمه كـ input للـ receiver

لأفهم الـ Micro إنه Pin RC7/RX/DT هو reciever pin

hardware مسؤول ياخذ أكثر من Sample من الـ bit ويعطيني قيمة الـ bit اللي مفروض إنها موجودة

علشان drop الـ Start والـ Stop

الهارد وير مسؤول عنه

بصير set كل ما استقبلت one Frame

هذا الـ reg اللي أنا بشوفه Double buffer reg بخزن 2 values

زي ما شايفينه مقسوم فبخزن الـ bytes زي الطابور لحد ما حدا يقرأهم وقبل ما يخلص استقبال الـ 3 byte بكون خزنته (First in First out (FIFO))

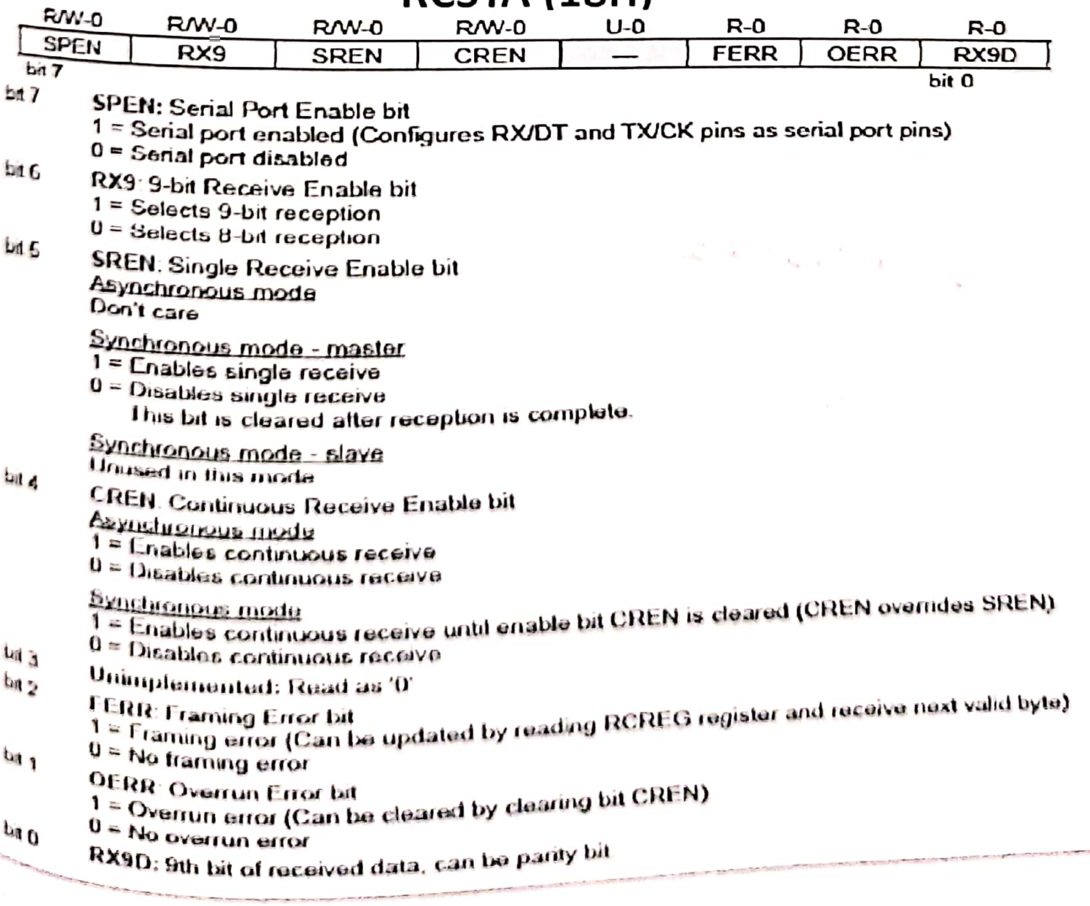
The 16F87xA USART

- Asynchronous USART Receiver Operation Notes
  - Data is received LSB first on RC7 pin
  - Reception is enabled by the CREN bit
  - At the heart of the block is the RSR register. Once a stop bit is detected, data is transferred to RCREG register, if it is empty, and the RCIF flag is set. (RCIF is cleared by hardware and it is read-only). On-receive interrupt can be enabled by RCIE bit
  - The RCREG is FIFO double buffered register
    - Can be used to receive bytes while reception continues in RSR
    - It can be read twice to read the received two bytes
    - If a stop bit is detected in RSR and the RCREG is still full, an overrun error occurs and it is indicated in OERR bit (The word is RSR is lost)
    - If OERR bit is set, shifting stops in RSR and transfers to the RCREG is inhibited !
    - To clear the overrun error, clear the CREN bit.
  - If the stop bit is received as clear in RSR a framing error occurs and it is indicated by the FERR bit.
  - The 9th bit of data RX9D → parity bit and FERR are also double buffered. It is essential to read the RCSTA register before the RCREG to avoid losing the corresponding values of RX9D and FERR →

لازم أقرأ ال Parity أول اشي عشان ما أخسره

The 16F87xA USART

RCSTA (18H)



R/W-0	R/W-0	R/W-0	R/W-0	U-0	R-0	R-0	R-0
SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D
bit 7							bit 0

bit 7 **SPEN:** Serial Port Enable bit
1 = Serial port enabled (Configures RX/DT and TX/CK pins as serial port pins)
0 = Serial port disabled

bit 6 **RX9:** 9-bit Receive Enable bit
1 = Selects 9-bit reception
0 = Selects 8-bit reception

bit 5 **SREN:** Single Receive Enable bit
Asynchronous mode
Don't care

Synchronous mode - master
1 = Enables single receive
0 = Disables single receive
This bit is cleared after reception is complete.

Synchronous mode - slave
Unused in this mode

bit 4 **CREN:** Continuous Receive Enable bit
Asynchronous mode
1 = Enables continuous receive
0 = Disables continuous receive

Synchronous mode
1 = Enables continuous receive until enable bit CREN is cleared (CREN overrides SREN)
0 = Disables continuous receive

bit 3 **Unimplemented:** Read as '0'

bit 2 **FERR:** Framing Error bit
1 = Framing error (Can be updated by reading RCREG register and receive next valid byte)
0 = No framing error

bit 1 **OERR:** Overrun Error bit
1 = Overrun error (Can be cleared by clearing bit CREN)
0 = No overrun error

bit 0 **RX9D:** 9th bit of received data, can be parity bit

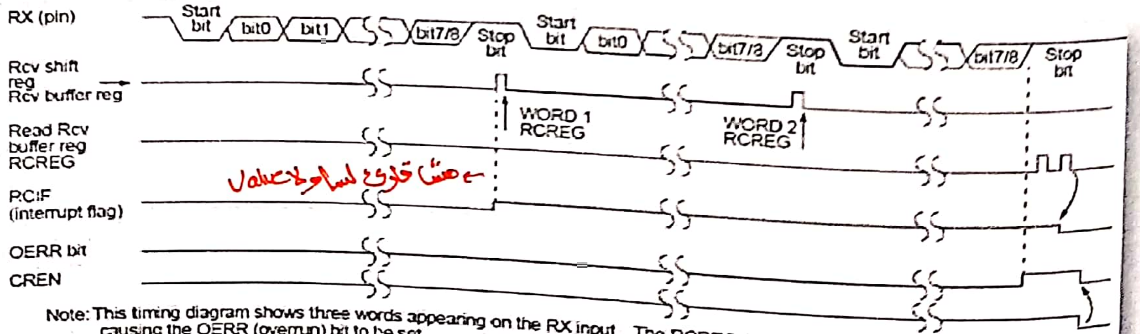
The 16F87xA USART

- **Steps for Using the asynchronous receiver**
  1. Set the SPBRG (0x99) register and BRGH (TXSTA<2>) bit to choose the appropriate baud rate
  2. Enable asynchronous serial port by clearing the SYNC (TXSTA<4>) bit and setting the SPEN bit (RCTSA<7>)
  3. If interrupts are desired, set the RCIE (PIE1<5>), GIE (INTCON<7>), and PEIE (INTCON<6>) bits
  4. If 9-bit reception is desired, set the RX9 (RCSTA<6>) bit
  5. Enable the reception by setting bit CREN (RCSTA<4>)
  6. The RCIF (PIR1<5>) will be set when reception of one word is complete and an interrupt will be generated if RCIE is set
  7. Read the RCSTA (0x18) to get the 9th bit and determine if any error occurred (OERR, FERR)
  8. Read the 8-bit received data by reading RCREG (0x1A)
  9. If any error occurred, clear the error by clearing the CREN

عكس اللي موجودين بالTransmitter

The 16F87xA USART

- **Timing of asynchronous reception**



Note: This timing diagram shows three words appearing on the RX input. The RCREG (receive buffer) is read after the third word, causing the OERR (overrun) bit to be set.

- **Registers involved in asynchronous reception**

→ read only

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh, 8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	R0IF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
18h	RCSTA	SPEN	RX9	SREN	CREN	—	FERR	OERR	RX9D	0000 -00x	0000 -00x
1Ah	RCREG	USART Receive Register								0000 0000	0000 0000
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
98h	TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
99h	SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

The 16F87xA USART

- **The BAUD Rate Generator**
  - The BAUD rate for USART is controlled by the value in the SPREG (99H), the SYNC and the BRGH bits in the TXSTA (19H)

مش هامتينا لأنه دايما شغالين async

→ Baud rate generate high.

SYNC	BRGH = 0	BRGH = 1
0 (asynchronous)	$$\frac{F_{osc}}{64(SPBRG+1)}$$	$$\frac{F_{osc}}{16(SPBRG+1)}$$
1 (synchronous)	$$\frac{F_{osc}}{4(SPBRG+1)}$$	

← بتعطي Baud rate أعلى

لازم نكون نوسع بـ 8 bit
لو ماوسعت بتجرب بالمعادلة الثانية وهكذا
بالنسبة للمعادلة الثانية

Example 1

→ serially

A program to transmit 3 bytes stored in locations 0x40, 0x41, and 0x42 serially with no parity at a rate of 9.6 Kbps. Assume PIC 16F877A with oscillator frequency of 20 MHz

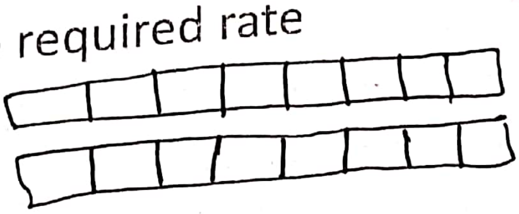
أجرب بالمعادلتين أو أرجع للـ Table
اللي فيه الحالات الشائعة

$\xrightarrow{\text{اخترنا}}$ SP BRG = 131
BRGH = 0

Requirements

1. setup the serial port for transmission
2. choose the appropriate value of SPBRG and BRGH to produce the required rate

* we ose : TxSTA
RCSTA



Example

```
                #include  p16F877A.inc    ; include the definition file for 16F77A
                org       0x0000         ; reset vector
                goto      START
                org       0x0004         ; define the ISR
ISR             goto      ISR
                org       0x0006          ; Program starts here
START           bsf       STATUS , RP0
                bcf       STATUS , RP1   ; select bank 1
                bcf       TRISC, 6       ; set RC6 as output
                movlw     D'31'
                movwf     SPBRG          ; set the SPBRG value
                bsf       TXSTA, TXEN
                bcf       STATUS, RP0    ; select bank0
                bsf       RCSTA, SPEN    ; enable serial transmission
                movlw     0x40
                mowf      FSR           ; FSR has the address of the first element
```

بقدر أشيله ما حستخدم interrupt

← لحد هاللحظة ما حيصير فيه Transmitter لأنه ما في داتا.

indirect addressing لـ

Example

* لو كنت حعمل interrupt :
١- كود ال WAIT ما حيكون
٢- مؤشر عالباقي بالأسفل

```
TX              movf      INDF, W       ; read byte to transmit
                movwf     TXREG         ; store in the transmission register
                incf      FSR, F        ; increment FSR to point to next address
WAIT            btfss     PIR1, TXIF    ; check if the TXREG is empty
                goto      WAIT
                movf      FSR,W
                sublw     0x43
                btfss     STATUS, Z     ; check if all values were transmitted
                goto      TX

DONE            goto      DONE
                end
```

→ polling

← لنشيك لو وصلت ل 43 لأنه ما بدي أبعث 43 فلو وصلت 43 معناها بعثت 42، 41، 40 فخلص خلصت

→ جوا ال Interrupt service routine رح يكون

وجرب كمل .

→ config
→ loop goto loop
→ ISR

Summary

- Serial communication transmits bits one after another in two modes: synchronous and asynchronous
- Stable and accurate clocking plays an important role in serial communication
- It is cheaper to use serial communication over long distances
- Some members of the 16 series are equipped with synchronous and asynchronous communication ports
- These ports can be configured to operated in different modes and rates

---

Analog to Digital Convertion

Data Acquisition and Manipulation

p80

Chapter 11
Sections 1 - 3

Dr. Iyad Jafar

Outline

- Analog and Digital Quantities
- The Analog to Digital Converter → هاردوير
- Features of Analog to Digital Converter
- The Data Acquisition System → الحصول على Data من نوع معين مثل الحرارة أو الرطوبة وهكذا!
- The 16F873 ADC
- Summary

* كل الـ data الي كنا نتعامل فيها كانت digital يعني بناخذ 0/1 فقط

*digital للخصائص الفيزيائية .

Analog and Digital Quantities

← التحويل للظواهر الفيزيائية للغة تفهمها الـ micro وهيك حلينا أول مشكلة.

الـ Analog

- Most signals that are produced by transducers are analog; continuously variable in time and can take infinite range of values
- Digital signals are *discrete representation* for the analog signals *in time and value* ← بتاخذ أي values وعنها قيمة عند كل لحظة زمنية
- Digital signals perform better and are easier to work with
- Analog signals have to be converted into digital form in order to be processed by the microcontroller
- The device that performs this conversion is called Analog to Digital Converter (ADC)

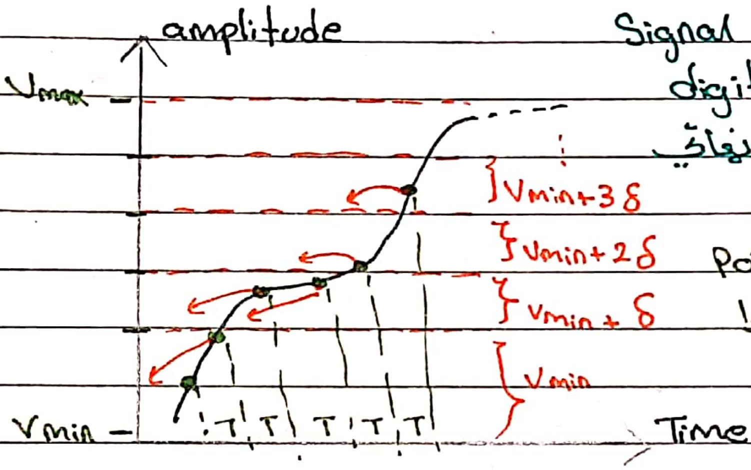
← ممكن يكون داخل الـ micro أو خارجها وبعمله interfacing

*حل المشكلة الثانية إني أقدر أحول الـ Analog value لقيمة digital بقدر يفهمها الـ micro .

Slide 3 :

* معظم ال physical quantities اللي بنتعامل معهم هم Analog quantities .

* Analog quantities : عند أي لحظة زمنية هاي ال Analog quantities نملك قيمة معينة ، وممكن تاخذ أي Value عادي من -∞ , ∞ .



Analog Signal

① * المشكلة لما أتعامل بهاي ال Signal مع ال micro/cpu لأنهم digital ما عندهم قدرة يتعاملوا مع عدد لانهائي من القيم .

* الحل : باخذ عدد من ال Points وكل فترة بشوفلهم ال Time وهذا ما يسمى ب Sampling فهيك حليت مشكلة إنها Continuous with time لأنه أخذنا عدد معين من ال Points مش كلها (Sampling every T: Sampling period !)

② * المشكلة الثانية إنه ال Value الوحدة بتاخذ عدد لانهائي من ال Values فأي نقطة بدها عدد لانهائي من ال digits حتى أمثلهم وال device ما بيقدر يعمل هيك .

* الحل : نفس ما عملنا على Time axes حنعمله على ال Value axes فبحط range للقيم اللي بدي أشوفها ($V_{min} \to V_{max}$) .

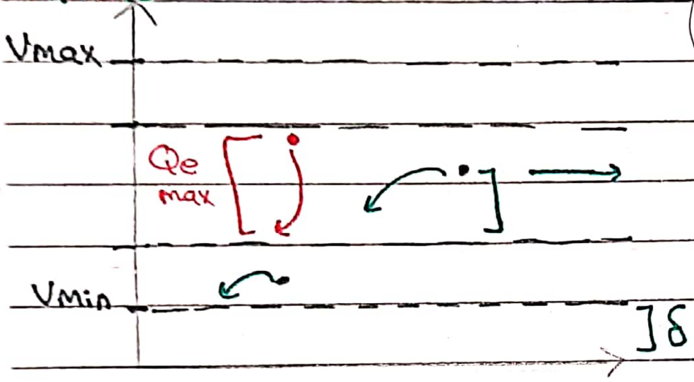
بعين خلال هذا ال range حاخذ نقاط محددة فهيك حددت عدد ال Values اللي بتعامل معها و الثمن اللي حدفعه نتيجة حلي للمشكلة وجود ال error ولكن رح أحاول أتعامل معه وأقلله.

اللي عملناه هون اسمه (Quantization) تحويل ال continuous values إلى discrete values .

③ * ثالث عملية بعد ال Sampling و ال Quantization هي ال Encoding وهي ال Values اللي اخترتها بال y-axes بدي أمثلهم بال binary ← e.g: $V_{min} \to 00$

$V_{min} + \delta \to 01$

* Note:



* وجود Vmax و Vmin مهم للحصر،
Input range = Vmax - Vmin

* أي نقطة أنا بوديها للـ Value الأقل
وهنا بصنع error بيسمى :
(quantization error) (Qe)

* Vmax تسمى أيضاً ($V_{ref}^{(+)}$) و Vmin تسمى ($V_{ref}^{(-)}$)

* الـ δ هي مقدار الخطوة اللي بمشيها لأنتقل من level لـ level بنقدر نحسبها من Vmin و Vmax وإذا بعرف كم عدد الـ bits اللي متوفرين عندي، وعدد الـ bits فعليا بحدد عدد الـ range

range = $2^{\text{number of bits}}$ *

Step Size (δ) = Input range / $2^{n \to bits}$
↳ اللي بقدر أمثلها بالـ Value

* كل ما زاد عدد الـ bits معناها عدد الـ levels فبزيد الدقة (resolution).
قديش قدرة الـ ADC على تمييز الـ Value ↲

* Qe هو قيمة متغيرة ولكن الـ maximum الها اللي هي أكون حفة الـ level اللي فوق وتنزل للـ level اللي تحدد (محدد بالرسمة) وهذا فعليا هو الـ resolution.
: max Qe = resolution.

Analog and Digital Quantities

(مقارنة)

→ Discrete and finite

Property	Analog	Digital
Representation	Continuous voltage or current	Binary Number
Precision	Infinite range of values	Only fixed number of digits combination are available
Resistance to Degradation	Suffers from drift, attenuation, distortion, interference. Recovery is hard	Tolerant to most forms of signal degradation. Error checking can be included for complete recovery → بقاوم التشويش بشكل أسهل وأفضل
Processing	Processing using op amps and other sophisticated circuits. Limited, complex, and suffers from distortion	Powerful computer-based techniques
Storage	Analog storage for any length of time is almost impossible	All semiconductor memory techniques are digital → بنخزن بنقدر بأكثر من نوع

* كله بحاول يشتغل بال Digital لأنه أفضل.

The Analog to Digital Converter

(بالدفتر)

- Conversion to digital form requires two steps
  - Sampling
  - Quantization

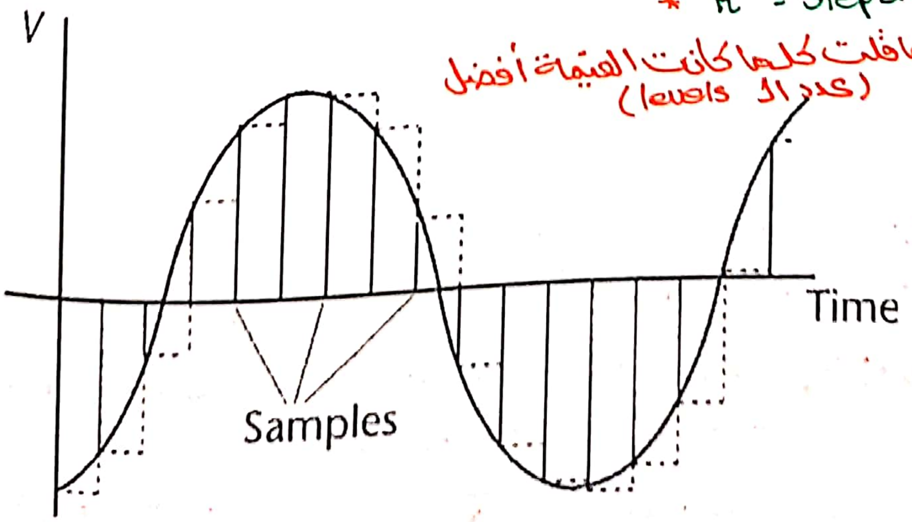
* $n = \log_2 \#\text{levels}$

* $R = \text{Stepsize} = \frac{\underbrace{V_{max} - V_{min}}_{\text{range}}}{2^n}$ volt/level → الوحدة

علة انت بتتحكم فيهم بس الهم range

2^n → معروفة

← كلما قلت كلما كانت القيمة أفضل (عدد ال levels)



Features of Analog to Digital Converter

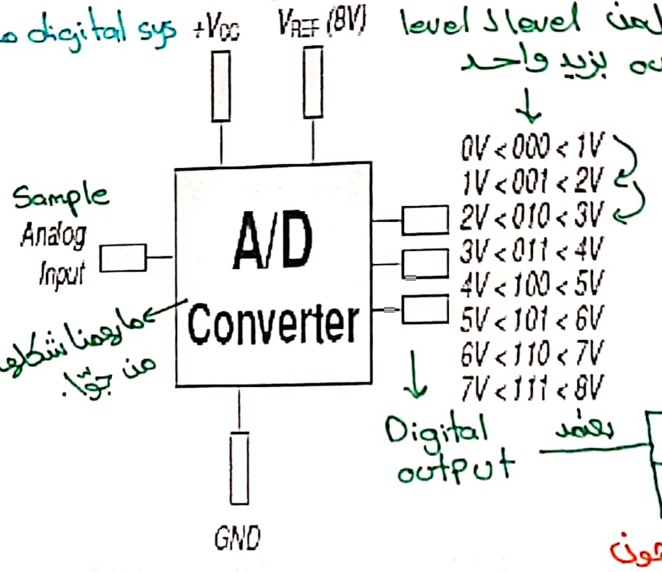
* $R = \frac{8-0}{2^3} = 1$ v/level ← كلما ال sample input تزيد بمقدار 1 volt ال digital output حيزيد بمقدار واحد

↳ LSB change

- **Conversion Characteristics**
  - The ADC accepts a voltage that is infinitely variable and converts it to one of a fixed number of output values

سوو ↓ لما يكون ناتجو 1

← لازم اكتب كود افهم ال digital sys اذا شفت 000 معناها 0V وهكذا لإنه ال digital sys مابفهم هاد الاشي.



لما انتقلت من level ل level ال output بزيد واحد ↓

0V < 000 < 1V
1V < 001 < 2V
2V < 010 < 3V
3V < 011 < 4V
4V < 100 < 5V
5V < 101 < 6V
6V < 110 < 7V
7V < 111 < 8V

ما يهمنا شكلها من جوّا.

Digital output بعتمد
- Vmax = 8v
- Vref(-) = 0
- عدد ال bits = مستخدم 3 bit هون

← مش راسماه هو افترضه صفر

Features of Analog to Digital Converter

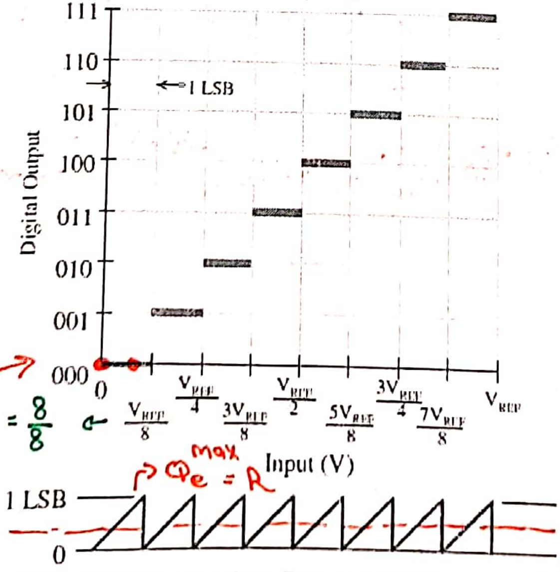
- **Conversion Characteristics**

التنين مستخدمين

Quantization Error

تمثيل الي فوق

← سوواه Shift left ب $\frac{1}{2}$ عشان يقللوا ال Q_e^{avg}

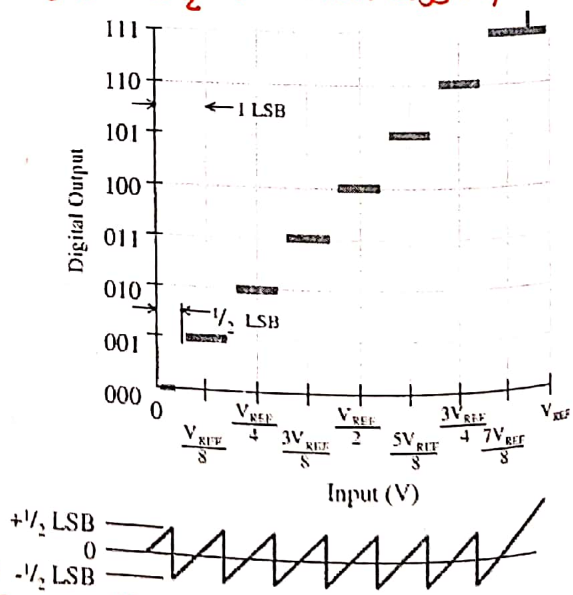


$1 = \frac{8}{8}$ ←

$\to Q_e^{max} = R$

The Magnitude of the Error Ranges from Zero to 1 LSB

avg Qe → أكبر من 0



$|Q_e^{max}| = \frac{1}{2}R$

avg Qe = 0 تقريبا فهيك أفضل

actual converted

Qe = 0V - 0V = 0 V

Qe = 0.7 V - 0V = 0.7 V

Features of Analog to Digital Converter

- **Reference voltages [V_{min},V_{max}]**
  - Determine the acceptable range of input analog voltage
  - Out of range input values are clipped → اللي برّا حنشوفها يا min يا max
  - *Unipolar or bipolar* → ممكن يكونوا negative → أيضًا Vmax أكبر من Vmin
    ↳ بس يكونوا موجب و Vmax أكبر من Vmin
  - Should be stable and accurate for proper operation
    stable ↳ مستقرة
    accurate ↓ مظبوط
  - *Input range $V_r = V_{max} - V_{min}$*
- **Resolution**
  - The amount by which the input voltage has to change to go from one output value to another
  - The more the output bits the more the output steps and finer is the conversion
  - *Resolution = $V_r / \boxed{2^n}$* → عدد ال levels
  - *Quantization error Q = resolution / 2* $\xrightarrow{\text{وممكن يكون}}$ Q = Resolution
    ↳ حسب شو ال curve اللي بستخدمه .

Features of Analog to Digital Converter

- **Conversion Characteristic**

Quantization error as a function of ADC bits

n	No. of quantisation levels 2^n	Max. quantisation error as % of range	Quantisation error for range of 5 V $V_r = 5V$
3	8	6.25	312.50 mV
4	16	3.13	156.25 mV
5	32	1.56	78.13 mV
6	64	0.781	39.06 mV
8	256	0.195	9.77 mV
10	1 024	0.0488	2.44 mV
12	4 096	0.0122	0.61 mV
16	65 536	0.00076	38.1 μV

$$\downarrow \frac{1}{2^{n+1}} * 100\%$$

$$\downarrow \frac{\frac{5}{2^{n+1}}}{5}$$

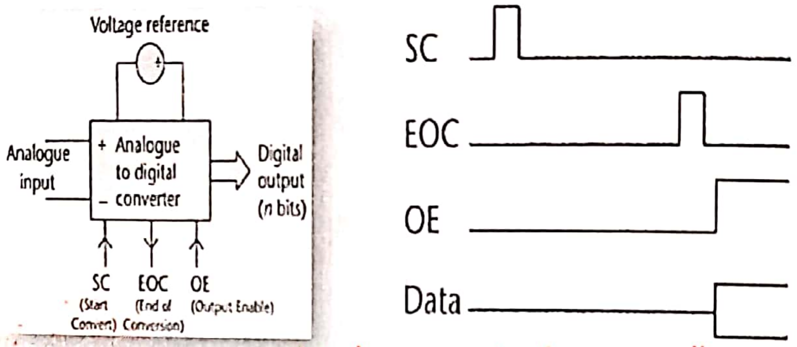
$$Q_e = \frac{5}{2^{n+1}} = \frac{\frac{5}{2^n}}{2}$$

Features of Analog to Digital Converter

- **Conversion Speed** → قديه سريع في عملية التحويل
  - Time for the ADC to do the conversion
  - Slow ADCs are used with low frequency signals
  - High accuracy ADCs take longer to complete conversion

← مابيكون ه أكبر في Time

- **Digital Interface**
  - Made up of control signals and data outputs
  - Data outputs – serial or parallel



لـ ظهور الـ data بس حط enable

The Analog to Digital Converter

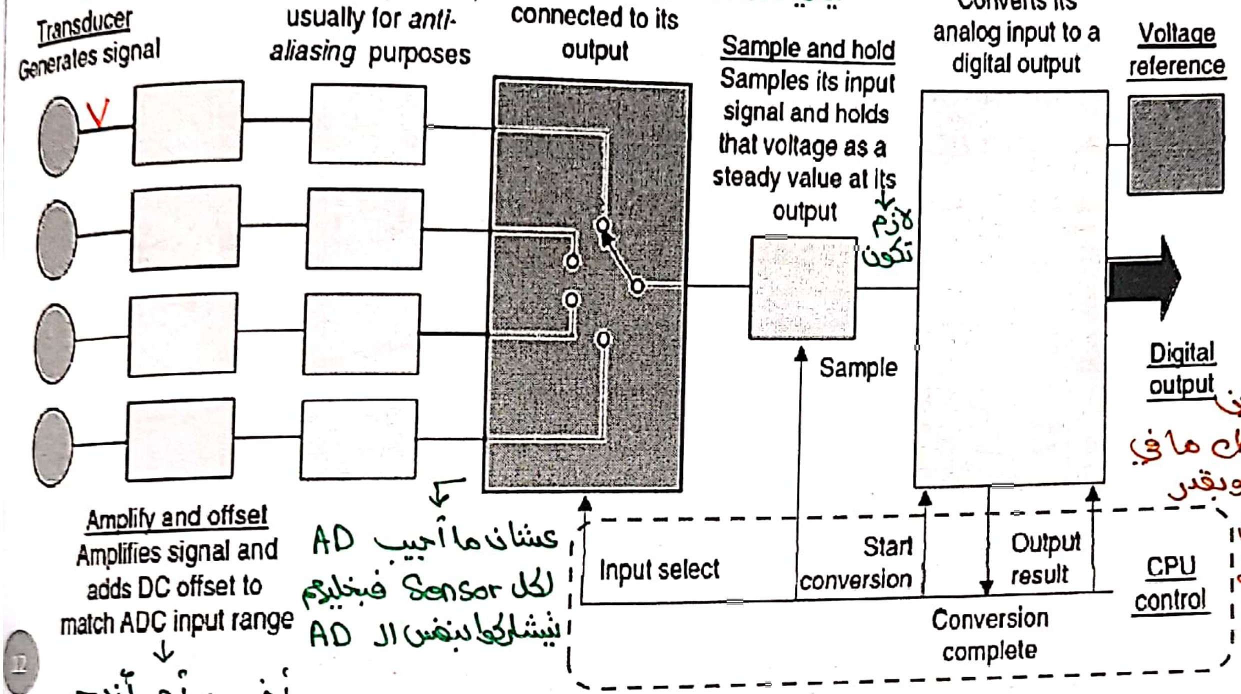
مش محتاج كلامها بس اقرأيه هو عبارة عن طرق لعملية التحويل

- **ADC Types**
  - *Dual Ramp ADC*
    - Slow but with high accuracy
  - *Flash Converter ADC*
    - Fast but less accuracy
    - Used with high speed signals such as video and radar
  - *Successive Approximation ADC*
    - Medium speed and accuracy
    - Used in general-purpose industrial applications
    - <u>*Commonly found in embedded systems*</u>

← أكثر انتشار

The Data Acquisition System

Elements of data acquisition system



Transducer
Generates signal

V

↑ يحول voltage لـ

Filter
Removes unwanted signal components, usually for *anti-aliasing* purposes

(noise) بيشيل الي ما بدي ياه

Multiplexer
Selects which input channel is connected to its output

→ Analog, electronic
يعني مش ١ و ٠

Sample and hold
Samples its input signal and holds that voltage as a steady value at its output

لازم تكون

ADC
Converts its analog input to a digital output

Voltage reference

Digital output

آخر 3 بكونوا موجودين بنفس الـ chip ما في داعي أبرمج وبقدر اتحكم فيهم عن طريق digital interface

Amplify and offset
Amplifies signal and adds DC offset to match ADC input range

↓ أضرب أو أزيح ليصير matching بين Vrefrence و Vsen لو احتجت

عشان ما أجيب AD لكل Sensor فبخليهم يشتركوا بنفس الـ AD

→ لما أمشي بالعكس حعمل عكس العمليات الي عملتها لأطلع الـ Vsen

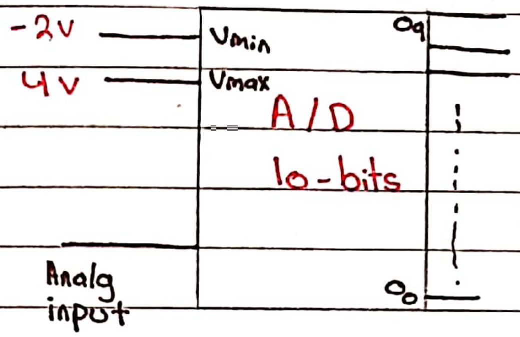
The Data Acquisition System

Elements of data acquisition system

- Amplification
  - Most sensors produce low voltages
  - Need to amplify to exploit the input range of the ADC
  - Voltage level shifting might be needed for bipolar signals
- Filtering
  - Pick the actual signal and restrict its frequency content to the sampling rate of the ADC to avoid aliasing
  - Remove unwanted signals
- Analog multiplexer
  - Used when working with multiple inputs instead of using multiple ADCs
  - Semiconductor switches

PAGE
DATE

Slide 11 : Example



a) if the digital output is $(00\ 00\ 01\ 11\ 10)_2$, then what is the analog value ?

Sol. $(00\ 00\ 01\ 11\ 10)_2 = (30)_{10}$

ل معناها أنا من Vmin مشيت

30 خطوة لأوصل لل level اللي فيه

ال Sample فهي مابتمثل ال analog value

↳ analog = Vmin + digital output * Resolution

$$= -2v + 30 * \frac{4-(-2)}{2^{10}}$$

$$= -2v + 30 * 5.86 * 10^{-3}$$

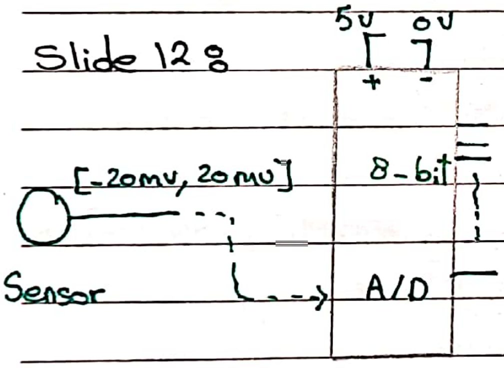
$$= -1.824\ V$$

b) if the analog input is 3.5 V, then what is the digital output ?

Sol. $3.5 = -2v + \text{digital output}\ (\#\text{ of steps}) * 5.86 * 10^{-3}$

digital output $= \lfloor (939.5)_{10} \rfloor = (939)_{10} = (1110101011)_2$

Slide 12 :



$R = \frac{5-0}{2^8} \cong 20mv \xrightarrow{\text{معناه}}$ * ال Values اللي حشوفها كـ output محكومة

بشغلتين : ال ref voltages و بال Resolution step

* أي Value أقل من 0 ماحشوفها فما حشوف أي شي negative

* فصار ال Sensor كانه من [0, 20mv] فصار ال output كله 0 لإنه قيمة

ال Voltage تبع ال Sensor اللي بعطيني ياها مش متناسق من ال refrence voltages

اللي أنا مختارهم. * الأفضل لهاد السيناريو يكون عندي القدرة أخلي ال refrence voltages

20 ، -20 mv بدل 0,5 mv فهيك بقدر أشوف كل ال Values بس ممكن ما أقدر أعمل هيك

فالحل أسوي preprocessing لل output voltage تاع ال Sensor بحيث إنه يتناغم مع

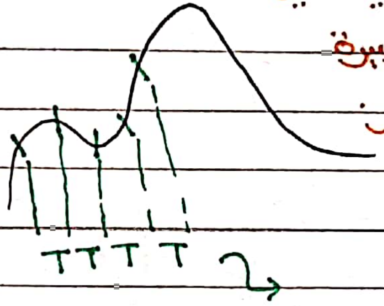
ال refrence voltage : ← ex : $(V_{sen} + 20mv) \times \frac{5}{40mv}$ عن طريق Amplifires

Scanned with CamScanner

Slide 12 :

*سبب آخر للحاجة إلى Filter غير انه يشيل ال noise :

Sampling



بدي أحتاج أقلل ال T لو في كثير تغيرات
يعني زمن ال T بين كل Sample والثانية قليل
عشان يلم كل التغيرات . وتكون كبيرة
لو كانت الرسمة ما فيها تغيرات كثير

كل قديه آخذ Sample ؟

فلما بدي آخذ ال Sample لازم آخذ بعين الاعتبار ال Signal قديه سريعة

* كل ال AD إلهم سرعة معينة فلو أعطيته Samples أسرع من سرعته المعينة بيطل يلحق علي فلازم أعطيه Sample period (T) أكبر من ال Convertion time

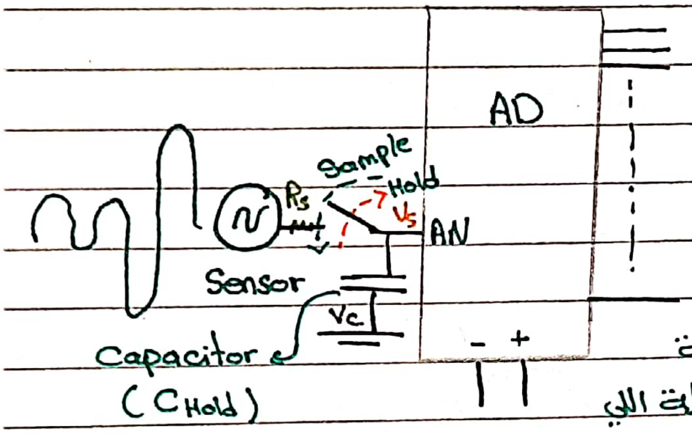
(Sampling Period (T) ≥ Conversion ، معناها أنا بدي minimum value لل T
معناها ما بقدر ألحق كل ال Signals يلحق عال Signals بمواصفات معينة بسرعات معينة فال Filter وظيفته الثانية يحدد السرعة تبعت ال Signal حتى آقدر أسوي matching لل Conversion Time of AD

* ال Filter بشيل كل ال Frequencies اللي أكبر من Values معينة ما بقدر إني اتعامل معها فبتصير ال Signal أبطأ.

* Sampling Theory :

$T_s = \frac{1}{F_s} \rightarrow F_s \geq 2 F_{max}$ ↷ أكبر Frequency بال Signal اللي بتعامل معها. (Band - limit) بنحكي عنه بالتفصيل بعدين.

Slide 12 : Sample and hold.



* كيف ممكن أسوي Sampling ؟

- Sampling كإني أكون مغمض عيوني وأفتح فجأة عند لحظة زمنية وأشوف كم قيمة الـ Signal بعدين أرجع أغمض وآخذ القيمة اللي شفتها وأعطيها الـ AD converter.

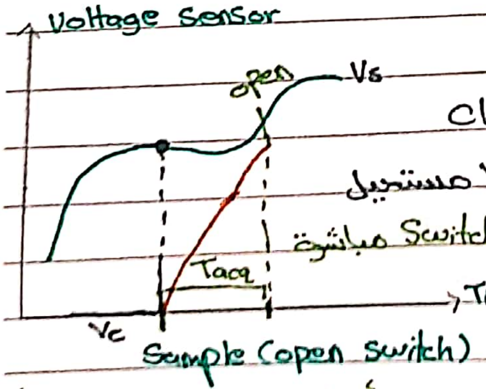
- بحط Switch كما هو موضح بالرسمة وطول ما هو مفتوح ما بشوف اشي ، باللحظة اللي بدي أطلع وآخذ الـ input sample بسكر الـ Switch لفترة زمنية بسيطة جداً مقدارها صفر بس هنا مش عملي لأنه في نقطتين مهمات : - لما أفتح الـ Switch قيمة Vs اللي قريتها حبطل أشوفها. - لو ضل Vs موجود فلازم أضل أشوفه لحد فترة زمنية معينة اللي هي Tconv. فكيف ممكن بعد ما أفتح هنا الـ Switch إني أحافظ على قيمة Vs (أخزنه) + كيف ممكن أخزن Analog voltage (أحط capacitor وأشبكه عالـ ground بحيث لما أسكر الـ Switch بنشحن بصير عليه Voltage ولما أفتحه حيكون العلاقة بالـ input voltage).

- الـ Sample and hold هي circuit ما بنبنيها بس أنا بتحكم فيها وبستخدمها كـ user.

- الـ Switch مستحيل يكون mecanical زي ما رسمناه فهو electronic Switch.

- Sample and hold circuit ضروري تكون.

- AD بعمل الـ Quantisation والـ Encoding أما الـ Sample and hold circuit بعمل الـ Sampling.



- في مشكلة بوجود الـ Sample and hold circuit : بدي أسوي عملية Sampling ، المفروض أول ما أسوي Close للـ Switch تكون Vc = Vs وهنا الكلام غير صحيح الـ Vc مستحيل تصير قيمته نفس Vs بـ 0 Time ، المفروض ما أفتح الـ Switch مباشرة لازم أستنى Time معين لأخذ Vsample والـ Time اللازم استناله لمالانهاية بس هذا غير منطقي فباجي عند لحظة زمنية معينة وبعمل open كما بالرسمة ، معناها أنا رضيت الـ Sample اللي أخذتها عند الـ open ما تكون تساوي Vs فحيكون عندي error صغير وكل ما كبرت Tacq كل ما صار أقرب للقيمة الحقيقية.

→

- كيف أحسب acq error : بدي أعرف ال Time constant تبع ال Circuit فعادة ال Sensor بكون إله مقاومة Rs فلازم أشوف ال Capacitor الوحيد اللي موجود عندي شو شايف مقاومة حتى أعرف ال Time const ومجرد ماعرفته بقدر أحسب كم لازم استنى حتى يكون عندي error بمقدار معين.

Ex : $V_c = 0.9 V_s$ → 10% error → $V_s - V_c = 0.1$

يعني برضى لما افتح ال Switch تكون قيمة Vc تساوي 0.9 Vs را

- من اللحظة اللي بسكر فيها ال Switch للحظة الي بفتح فيها ال Switch كم لازم استنى يعني كم ال eeq/Sampling Time

$$* \; V_c = V_s \left(1 - e^{-t/RC}\right)$$

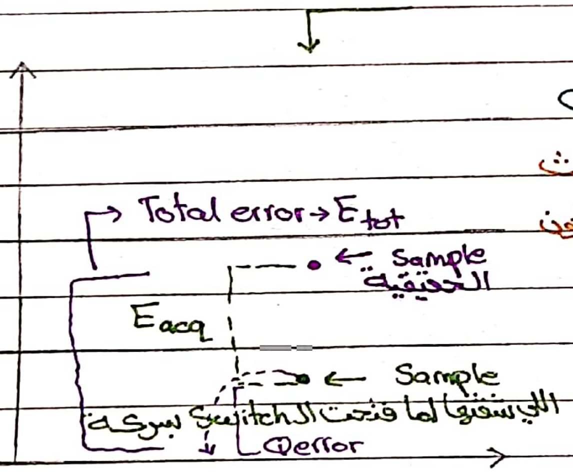
$$0.9 \cancel{V_s} = \cancel{V_s} \left(1 - e^{-t/\underset{\downarrow}{RC}}\right) \Rightarrow T_{acq} = 2.3 R_s C_{Hold}$$

هو المجهول ↲ معطاه

وممكن ماتكون بس R
S حسب معطيات السؤال.

- عملية ال Conv بتصير بعد ال Sample وخلالها مابقدر أعمل كمان Sample لإنه ماعندي إلا Capacitor واحد وهكذا!.

* من عملية التحويل من Analog to digital صار في عندنا مصدرين لل error :
① Qerror ② acq error



* فهل ممكن إنه نقلل من ال Total error ؟
حكو ليه مانصمم عملية ال Sampling بحيث إنه ال Sampling Time يكون كافي بحيث يكون ال $E_{acq} \le \max Q_{error}$
يعني مثلاً أكون حريصة إنه ال Sample بعد ال Eacq تكون في نفس ال level اللي كانت فيه ال Sample الحقيقية فهيك ببطل أشوف ال Eacq لإنه صار max هو ال Qerror وهنا بتطلب اني أخلي ال Switch مسكر لفترة زمنية مناسبة ← $V_s - V_c \le Q_{error}\ max$ ∽ $V_s - V_c = Q_{error}^{max}$

$$* \; V_s - V_c = Q_e^{max} \rightarrow V_s - V_c = \frac{V_r}{2^{n+1}} \rightarrow V_r - V_c = \frac{V_r}{2^{n+1}} \rightarrow V_c = \frac{V_{s\,or\,r}(2^{n+1}-1)}{2^{n+1}}$$

↲ لإنه Vs حيتغير من min value ل max value ↲ R/2 = أخذناه

هاي المعادلة بتعطيني العلاقة اللي لازم تكون بين Vs و Vc حتى يكون $E_{acq} \le Q_{error}^{max}$

Ex: 10-bit ADC

$$V_c = V_s \frac{(2^{11}-1)}{2^{11}} = 0.9995\, V_s \rightarrow$$ لازم هيك يكون

$$\Rightarrow V_c = V_s\left(1 - e^{-t/RC}\right) \rightarrow 0.9995 \cancel{V_s} = \cancel{V_s}\left(1 - e^{-t/RC}\right)$$

$$\Rightarrow t_{aq} = 7.6\, RC$$

لـ بعد ما يخلص الـ Tacq بفتح الـ Switch وببلش عملية الـ Conv وبتاخذ Time

ف: Total Time = Tacq + Tconv وخلال هاي الفترة ما بقدر أسوي أي شي ثاني.

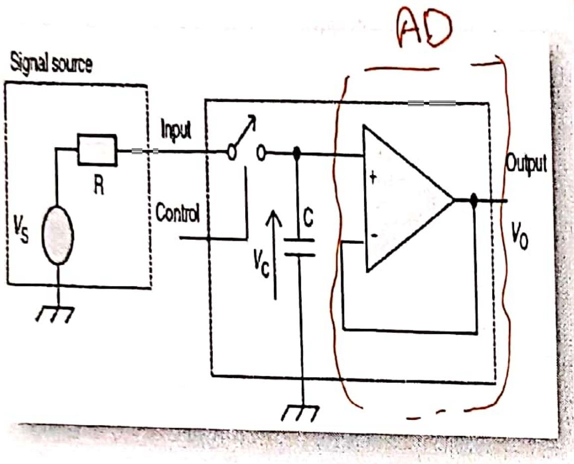
The Data Acquisition System

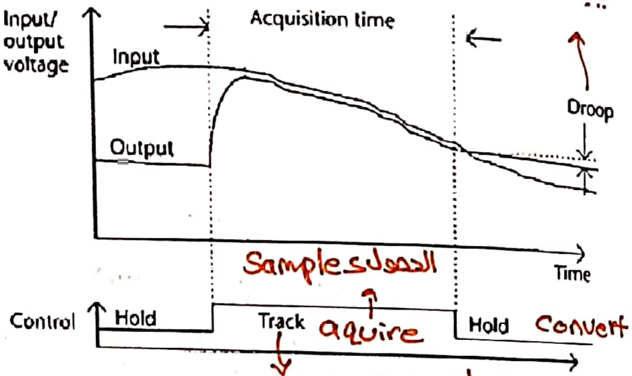
Elements of data acquisition system

- **Sample and Hold**
  - ADCs are unable to convert accurately a changing signal
  - We need to capture the sample value and hold it for the duration of the conversion process
  - Acquisition time !

مابقدر اتحكم فيه

بيلش بيشحن أو يفرغ ال Vc وهذا نوع error





الحصول Sample

aquire

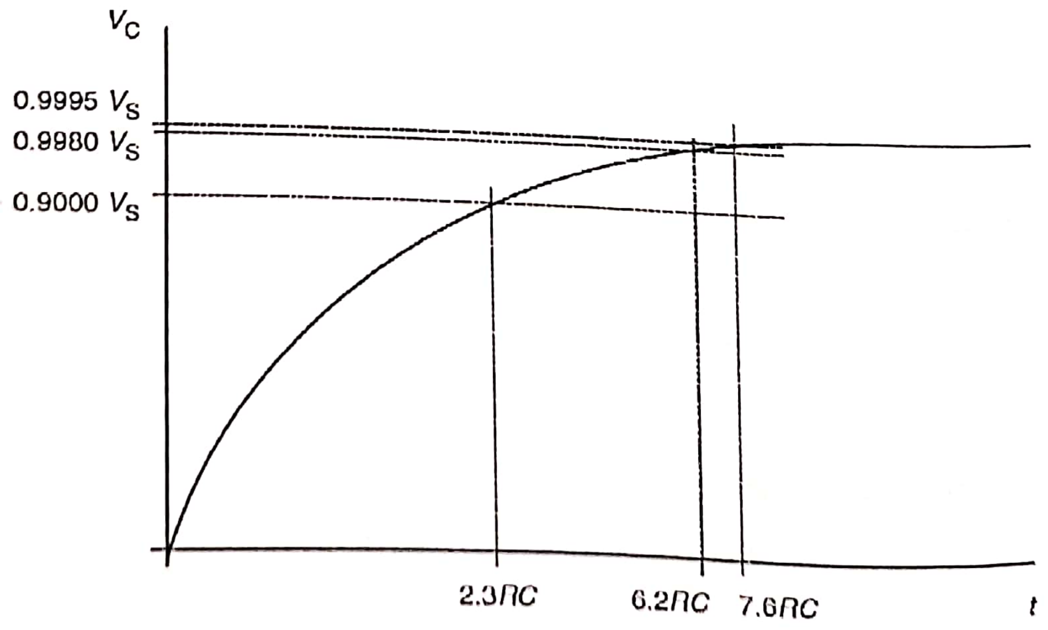
Convert

Capacitor voltage يحاول يلحق ال Input voltage

The Data Acquisition System

Elements of data acquisition system

- **Sample and Hold**

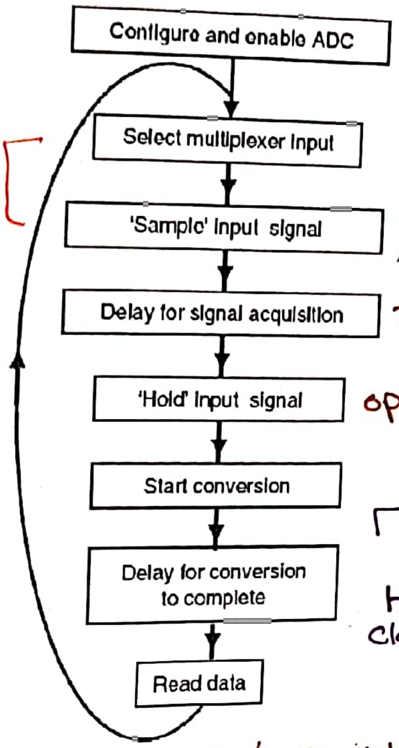


Acquisition time increase as we increase the resolution of the ADC

بشكل عام

The Data Acquisition System

Typical Timing Requirements for Analog to Digital Conversion



These stages merge if multiplexer forms part of S&H

هدول التنين عادة كخطوة وحدة .

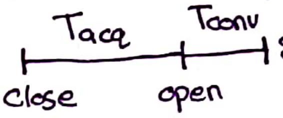
بيجوز ما اخترت من ال MUX بكون سكرت ال Sampling Switch

Taqq

open switch

Time بيفصل بين Sample و Sample

$T_s = \frac{1}{F_s} = T_{acq} + T_{conv}$



*خليه ببالك :

قبل أعرف أكبر Frequency قبل أدخلها عال System تبعي .

$F_{max} \leq \frac{F_s}{2}$ ← لو كان أكبر بستخدم الفلتر اللي حكينا عنه المرة الماضية .

Data Acquisition in Microcontroller Environment

لإنه بالعادة ال Embedded sys تتعامل مع physical quantities

- Embedded systems need ADCs ; usually they are integrated within the MC as 8 or 10 bit ADCs

ممكن تكون هذه العملية داخل ال MC أو خارج ال MC مثل ال pic684 لاتحتوي داخلها ADC .

- Integration is not easy !
  - Proper operation of ADCs demands clean power supply and ground and freedom of interference → ممكن تتأثر دقة عمل ال ADC
  - This is not easily available in digital devices

- Compromise accuracy of integrated ADCs !

تعايش مع وجود ال Noise

The PIC 16F87xA ADC Module

Device	Pins	Features
16F873A 16F876A	28	3 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, 5 10-bit ADC, → 1 ADC 2 comparators
16F874A 16F877A	40	5 parallel ports, 3 counter/timers, 2 capture/compare/PWM, 2 serial, 8 10-bit ADC, → 1 ADC 2 comparators

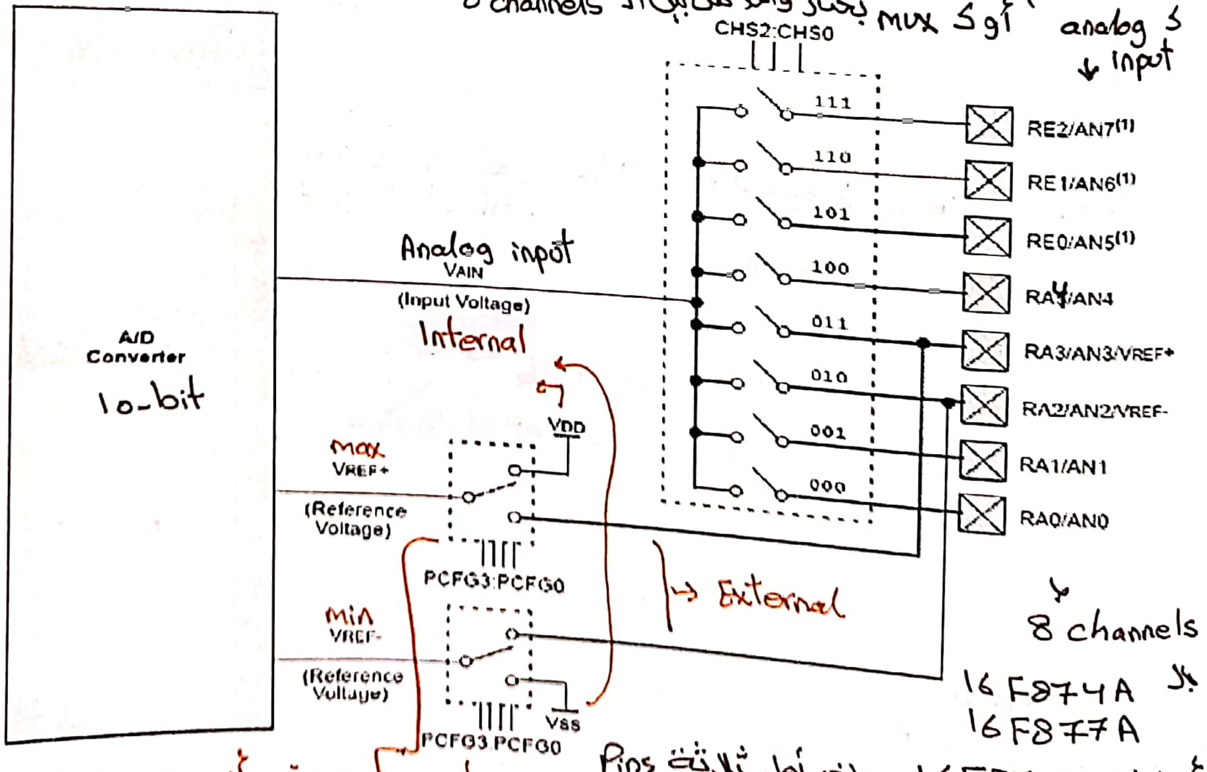
يعني ← 5channels يعني بعمل multiplixing بين 5 different input (5-to-1)

"(8-to-1) mux" 8channels يعني ← 8

The PIC 16F87xA ADC Module

ممكن استخدمهم ك analog input

أو ك mux بختار واحد من بين ال 8 channels



8 channels بال 16F874A 16F877A

أما في 16F873A 16F876A هاي أول ثلاثة Pins

← يقدر أبرمجهم واختار اللي بدي اياه

The PIC 16F87xA ADC Module

Related Registers (SFRs)

- Operation is controlled by two SFRs
  - **ADCON0 0x1F**
  - **ADCON1 0x9F**

→ ADC لل config عن طريقهم بحدد وبسوي

- Conversion result (10-bit) is placed in two SFRs
  - ADRESL 0x9E
  - ADRESH 0x1E

→ 2 reg عشان أخزن 10-bits بشوفهم كقطعة كاملة بس بالأصل هم منفصلين ل 2 reg كل واحد 8 bits

- ADC interrupt enable and flag are available in
  - ADIE ← PIE1 0x8C
  - ADIF ← PIR1 0x0C

→ هل خلصت conversion ولا ما خلصت

لو خلصت اعطي interrupt عشان اعمل شغل ثاني

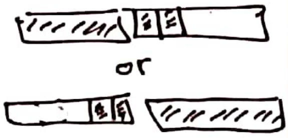
* أنا ما بحتاج 16-bit بس 10-bit فبقدر أختار بالتخزين أي reg اللي حياخذ ال bits الأقل

- Related registers
  - TRISA 0x85
  - TRISE 0x89 (in 40-pin devices)

→ Direction registers

16F874A
16F877A

أضافة الهم نتعامل مع
INTCON
GIE
PEIE



The PIC 16F87xA ADC Module

Controlling the ADC

(1) Switching on

- The ADC is switched on/off by setting/clearing ADON bit (ADCON0<0>) → by default = 0
- It is preferred to turn the ADC off when it is not needed as it offers some power saving → لو مو محتاجه يفضل تطفيه لحد ما يكون بدك ياه بتضويه .

(2) Setting Conversion Speed → فترة زمنية

- Operation of the ADC is governed (محكومة) by a clock with period T_{AD}
- For correct conversions, T_{AD} must be 1.6 us at least
- The ADC clock can be selected by software ($2T_{OSC}$, $4\ T_{OSC}$, $8\ T_{OSC}$, $16\ T_{OSC}$, $32\ T_{OSC}$, $64\ T_{OSC}$, or internal RC 2-4 us) بتختار انت
- Selection of ADC clock source is through ADCS2 (ADCON1<6>), ADCS1:ADCS0 (ADCON0<7:6>)
- If the system clock is fast (>500KHz), use it to derive the ADC clock. Otherwise, use the internal RC.

$T_{OSC} \rightarrow \frac{1}{F_{osc}}$

bit 6

كل ما طولتها كل ما بجيب القيمة بشكل أفضل

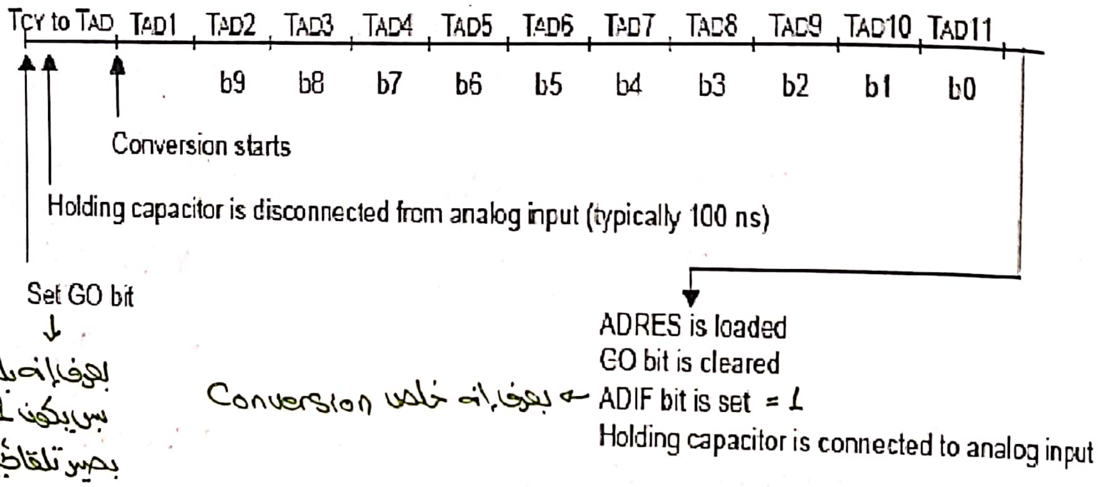
The PIC 16F87xA ADC Module

Controlling the ADC

Setting Conversion Speed

→ $T_{conv} = 12 T_{AD}$

- A full 10-bit conversion requires 12 T_{AD}



Conversion starts

Holding capacitor is disconnected from analog input (typically 100 ns)

Set GO bit

ADRES is loaded
GO bit is cleared
ADIF bit is set = 1
Holding capacitor is connected to analog input

بعرف إنه بلش Conversion
بس يكون 1 ولما نخلص
بصير تلقائي
0 وال ADIF
تلقائي بصير 1

بعرف إنه خلص Conversion ←

The PIC 16F87xA ADC Module

Controlling the ADC

(3) Configuring Inputs and Voltage Reference

داخلي أو خارج
و الأفضل أجيبكم من برة لأقلل
أي تشويش عقيمة ال
Power supply

- The ADCON1 and TRIS registers control the operation of the A/D port pins
- Inputs AN7 to AN0 can be configured as analog inputs or digital inputs.
- AN3 **(RA3)** and AN2 **(RA2)** can be used as the inputs for the external reference voltages separately
- Configuration is made through PCFG3:PCFG0 (ADCON1<3:0>)

5

لو اخترتهم
بطلوا يستخدموا
input analog
Values

(4) Channel Selection

- We can select one out of five (or eight channels) as the analog input using the bits CHS2:CHS0 (ADCON0<5:3>)
- Selection of the input channel closes the sampling switch.

← لما أسوي selection كأني بسكر ال Switch وبسوي ... data aqu

The PIC 16F87xA ADC Module

Controlling the ADC

قو cleared
ال hardware

set
قو ال user

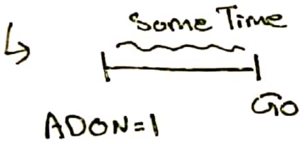
GO/$\overline{\text{DONE}}$

(5) Starting Conversion and Flagging its End

- Conversion can be started by setting the GO/DONE' (ADCON0<2>) bit. **This opens the sampling switch.**
- Once the conversion is complete, this bit is cleared to indicate the end of conversion

  by hardware
- *The GO/DONE' bit should not be set using the same instruction that turns on the A/D.*

لازم اشغل ال A/D بعدين استنى فترة لأفعل Go

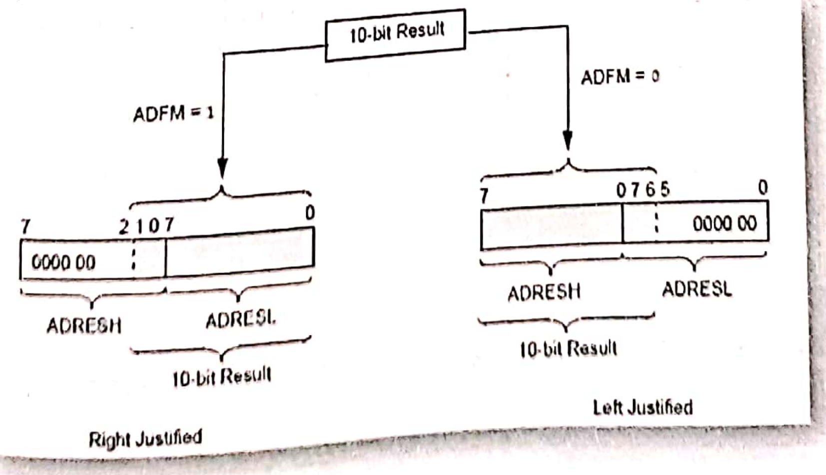


The PIC 16F87xA ADC Module

Controlling the ADC

(6) Formatting the result

- The ADC result is 10-bit data that is placed in ADRESH and ADCRESL (0x 1E and 0x9E respectively)
- The result can be left justified or right justified
- Selection of desired format is through the ADFM (ADCON1<7>) bit



The PIC 16F87xA ADC Module

ADCON0 Register 0x1F

2 من ال 3 المسؤولين عاختيار قيمة ال clk

by deafult

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON
bit 7							bit 0

له إله وظيفتين

bit 7-6 **ADCS1:ADCS0:** A/D Conversion Clock Select bits (ADCON0 bits in **bold**)

ADCON1 <ADCS2>	ADCON0 <ADCS1:ADCS0>	Clock Conversion
0	00	Fosc/2
0	01	Fosc/8
0	10	Fosc/32
0	11	FRC (clock derived from the internal A/D RC oscillator)
1	00	Fosc/4
1	01	Fosc/16
1	10	Fosc/64
1	11	FRC (clock derived from the internal A/D RC oscillator)

bit 5-3 **CHS2:CHS0:** Analog Channel Select bits

000 = Channel 0 (AN0) RA0
001 = Channel 1 (AN1) RA1
010 = Channel 2 (AN2) ,
011 = Channel 3 (AN3) ,
100 = Channel 4 (AN4) :
101 = Channel 5 (AN5) :
110 = Channel 6 (AN6)
111 = Channel 7 (AN7) RE2

bit 2 **GO/$\overline{\text{DONE}}$:** A/D Conversion Status bit

When ADON = 1:
1 = A/D conversion in progress (setting this bit starts the A/D conversion which is automatically cleared by hardware when the A/D conversion is complete)
0 = A/D conversion not in progress

bit 1 **Unimplemented:** Read as '0'

bit 0 **ADON:** A/D On bit
1 = A/D converter module is powered up
0 = A/D converter module is shut-off and consumes no operating current

---

The PIC 16F87xA ADC Module

ADCON1 Register 0x9F

R/W-0	R/W-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0
bit 7							bit 0

1 من ال 3 المسؤولين عاختيار ال clk

by default هي left

bit 7 **ADFM:** A/D Result Format Select bit
1 = Right justified. Six (6) Most Significant bits of ADRESH are read as '0'.
0 = Left justified. Six (6) Least Significant bits of ADRESL are read as '0'.

bit 6 **ADCS2:** A/D Conversion Clock Select bit (ADCON1 bits in shaded area and in **bold**)

bit 5-4 **Unimplemented:** Read as '0'

bit 3-0 **PCFG3:PCFG0:** A/D Port Configuration Control bits → بقدر أحدد كل وحدة من ال input شو طبيعتها

by deafult بكونوا Analog

لا مش كل الاحتمالات موجودة فإنت بتختار الأنسب اللي نستخدمه

PCFG <3:0>	RE2 AN7	RE1 AN6	RE0 AN5	RA4 AN4	RA3 AN3	AN2	RA1 AN1	RA0 AN0	VREF+	VREF-	C/R
0000	A	A	A	A	A	A	A	A	VDD	VSS	8/0
0001	A	A	A	A	VREF+	A	A	A	AN3	VSS	7/1
0010	D	D	D	A	A	A	A	A	VDD	VSS	5/0
0011	D	D	D	A	VREF+	A	A	A	AN3	VSS	4/1
0100	D	D	D	D	A	D	A	A	VDD	VSS	3/0
0101	D	D	D	D	VREF+	D	A	A	AN3	VSS	2/1
011x	D	D	D	D	D	D	D	D	—	—	0/0
1000	A	A	A	A	VREF+	VREF-	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	VDD	VSS	6/0
1010	D	D	A	A	VREF+	A	A	A	AN3	VSS	5/1
1011	D	D	A	A	VREF+	VREF-	A	A	AN3	AN2	4/2
1100	D	D	D	A	VREF+	VREF-	A	A	AN3	AN2	3/2
1101	D	D	D	D	VREF+	VREF-	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	VDD	VSS	1/0
1111	D	D	D	D	VREF+	VREF-	D	A	AN3	AN2	1/2

A = Analog input D = Digital I/O
C/R = # of analog input channels/# of A/D voltage references

The PIC 16F87xA ADC Module

Related Registers

" ملخص "

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on $\overline{\text{MCLR}}$, WDT
0Bh,8Bh, 10Bh,18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[(1)]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE[(1)]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
1Eh	ADRESH	A/D Result Register High Byte								xxxx xxxx	uuuu uuuu
9Eh	ADRESL	A/D Result Register Low Byte								xxxx xxxx	uuuu uuuu
1Fh	ADCON0	ADCS1	ADCS0	CHS2	CHS1	CHS0	GO/$\overline{\text{DONE}}$	—	ADON	0000 00-0	0000 00-0
9Fh	ADCON1	ADFM	ADCS2	—	—	PCFG3	PCFG2	PCFG1	PCFG0	00-- 0000	00-- 0000
85h	TRISA	—	—	PORTA Data Direction Register						--11 1111	--11 1111
05h	PORTA	—	—	PORTA Data Latch when written: PORTA pins when read						--0x 0000	--0u 0000
89h[(1)]	TRISE	IBF	OBF	IBOV	PSPMODE	—	PORTE Data Direction bits			0000 -111	0000 -111
09h[(1)]	PORTE	—	—	—	—	—	RE2	RE1	RE0	---- -xxx	---- -uuu

The PIC 16F87xA ADC Module

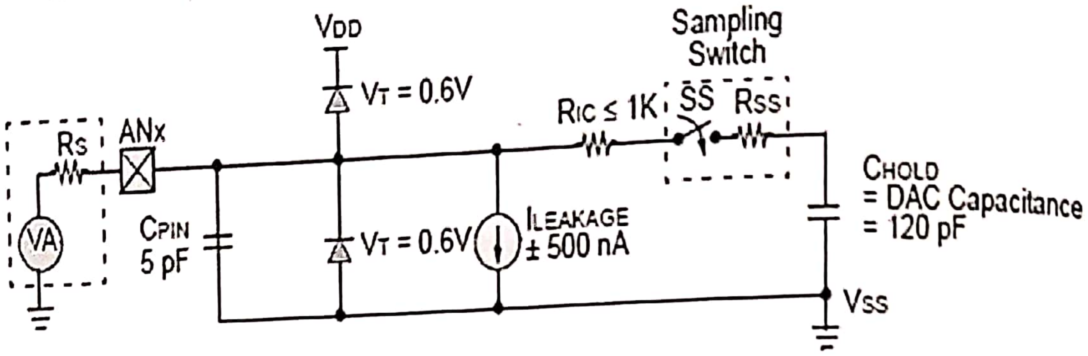
خطوات ال code

- **Steps for using the A/D module**
  1. Configure the A/D module
     a. Select analog pins/voltage reference and digital I/O (ADCON1)
     b. Select the A/D channel (ADCON0) CHS bits
     c. Select the conversion clock (ADCON0) ADCS bits
     d. Turn the A/D module on (ADCON0) ADON
  2. Configure interrupts (if desired)
     1. Clear ADIF (PIR1<6>) and set ADIE (PIE1<6>)
     2. Set PEIE (INTCON<6>) then set GIE (INTCON<7>)
  3. Wait the required acquisition time → بكون سويت الحسبة قبل
  4. Start conversion by setting the GO/DONE' bit → بس تخلص Conversion بصير 0
  5. Wait for conversion complete
  6. Read the A/D result register pair ADRESH:ADRESL

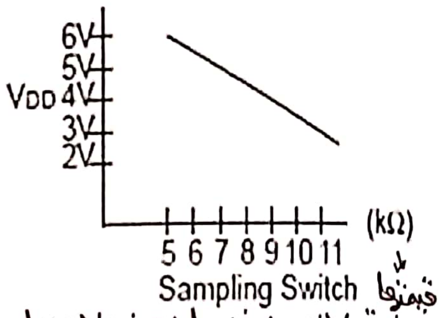
ال Polling بسوي عليه أو
عال ADIF أو بستخدم interrupt

The PIC 16F87xA ADC Module

- **The analog input model**



Legend:	CPIN	= input capacitance
	VT	= threshold voltage
	ILEAKAGE	= leakage current at the pin due to various junctions
	RIC	= interconnect resistance
	SS	= sampling switch
	CHOLD	= sample/hold capacitance (from DAC)



قيمتها محرزة لازم اخذها بعين الاعتبار

The PIC 16F87xA ADC Module

الدقة أكثر وأكثر عنما قبل

- **Calculating conversion speed (Qerror is ½ LSB)**

A/D Total Time = Acquisition Time + A/D Conversion time

= $T_{ACQ} + 12 * T_{AD}$

TACQ = Amplifier settling time

+ Hold capacitor charging time → $R_{eq} C_{Hold}$

+ Temperature coefficient → ممكن يتأثر بدرجة الحرارة

TACQ = $T_{AMP} + T_{HOLD} + T_{COFF}$

THOLD = $-(R_{IC}+R_{SS}+R_S) * C_{HOLD} * \ln(1/2\hat{}(n+1))$

= $-(R_{IC}+R_{SS}+R_S) * 120\text{ pF} * \ln(1/2048)$

= 7.6*R*C us

معادلة الفهم مش الحفظ

$$\text{A/D Total Time} = \overbrace{\overbrace{\underbrace{2\ \mu s}_{T_{AMP}} + 7.6RC + \underbrace{(\text{Temperature} - 25^\circ C)(0.05\ \mu s/^\circ C)}_{T_{COFF}}}^{T_{acq}} + \overbrace{12\ T_{AD}}^{T_{conv}}}^{T_s}$$

The PIC 16F87xA ADC Module

- **Calculating conversion speed example**

assuming:
* $A_c \leq Q_e$
* 10-bit

$R_{SS} = 7k\Omega$ ($V_{DD} = 5V$), $R_{IC} = 1k\Omega$, $R_S = 0$,
Temp = 35 °C, $T_{AD} = 1.6\ \mu s$

$$t_{ac} = 2\ \mu s + 7.6(7k\Omega + 1k\Omega + 0)(120pF) + (35 - 25)(0.05\ \mu s/°C) = 2 + 7.3 + 0.5 = 9.8\ \mu s$$

Total time = $t_{ac} + 12T_{AD} = 9.8 + 19.2\ \mu s = 29\ \mu s = T_s$

Maximum sampling rate ~= 34.5 KHz $= \frac{1}{29\ \mu s} = F_s$

ksample/sec

$F_{max} = \frac{F_s}{2} = 17.25$ kHz

The PIC 16F87xA ADC Module

- **Repeated Conversions**
  - When a conversion is complete, the converter waits a period of 2*TAD before it is available to start a new conversion
  - This time has to be added to the conversion time !

أول ما تخلص من Sample مباشرة تأخذ الثانية
وتبلش Conversion جديدة

- **Trading off conversion speed and resolution**
  - If resolution is not an issue, then we can start the conversion with correct clock then we switch it to higher clock
  - Consider only bits produced before switching the clock

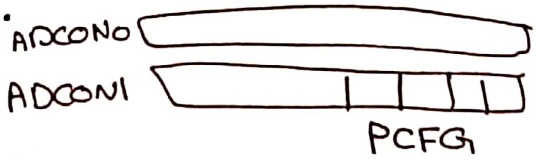
بكونوا دقيقين علشانهم عال clk
الي بدي ياها

لما سرعت ال clk
هدول ال bits ممكن مش
دقيقات وما بهمني.

بستفيد إني حصلت
ع 10 bits
بشكل أسرع

The PIC 16F87xA ADC Module

- **Example: use the ADC in PIC 16F877A to obtain one sample of an analog signal that is connected RA0.** ← AN0 **Assume the ADC clock to be Fosc/8 and reference voltage to be internal. The PIC is operating with Fosc = 4 MHz, VDD = 5 v, and temperature 25 C. The result should be right justified.**



Setup:

1) set RA0 as analog input
2) select the clock 8Tosc
3) generate appropriate delays (T_{acq} = 2+ 7.6 * (1K + 7K) * 120 pF = 9.3 us ~= 10 us)

جرب اكتب الكود باستخدام TIMR0 وسوي Polling

Example

```
                #include  p16F877A.inc      ; include the definition file for 16F77A
                org       0x0000            ; reset vector
                goto      START
                org       0x0004            ; define the ISR
ISR             goto      ISR
                org       0x0006            ; Program starts here
START           bsf       STATUS, RP0       ; select bank 1
                movlw     B'00000001'
                movwf     TRISA             ; set RA0 as input
                movlw     B'10001110'       ; select RA0 as analog input, result right
                            ↳ ACS2
                                            ; justified, and internal reference voltage
                movwf     ADCON1
                bcf       STATUS, RP0       ; select bank 0
                movlw     B'01000001'       ; turn on ADC, clock Fosc/8, select
                                            ; channel 0
                movwf     ADCON0
```

من هاللحظة صار Close ال Switch.

Example

نفس بعض

```
                ; start the conversion
                call        delay10us             ; acquisition time delay
                bsf         ADCON0, GO            ; start conversion
wait            btfsc       ADCON0, GO_DONE ; wait for conversion to complete
                goto        $-1

DONE            goto        DONE

delay10us       movlw       D'2'
                movwf       0x20                  ; counter for delay loop
more            nop
                decfsz      0x20,1
                goto        more
                return

                end
```

ارجع لورا one inst

أو اكتبي goto wait

≃ 10 Msc

فوت عالموقع اللي حكا عنه الدكتور وشوف الأمثلة المختصة بذلك

ucoz.net ما

Summary

- Most signals produced by transducers are analog in nature, while all processing done by a microcontroller is digital.
- Analog signals can be converted to digital form using an analog-to-digital converter (ADC).
- The 16F873A has a 10-bit configurable ADC module
- Data values, once acquired, are likely to need further processing, including offsetting, scaling and code conversion.

تفاعل الا microcontroller مع العالم الخارجي ك

The Human and Physical Interface

Chapter 8
Sections 1 - 9

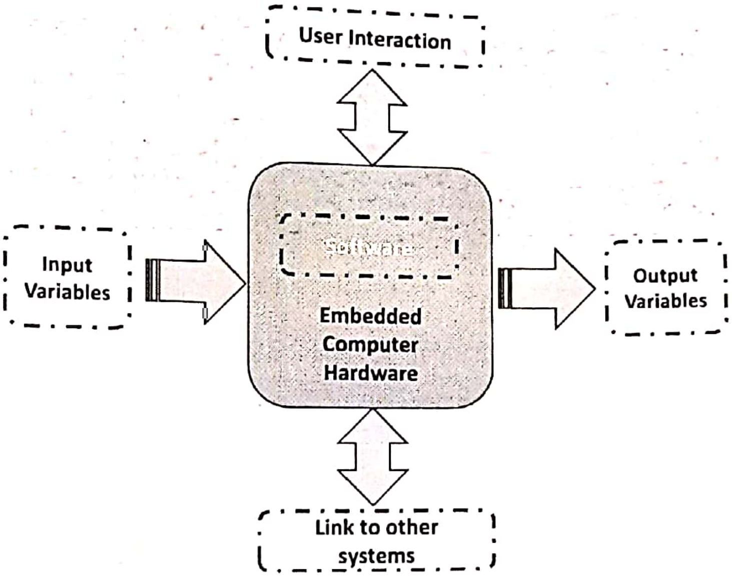
Dr. Iyad Jafar

Outline

- Introduction
- From Switches to Keypads
- LED Displays
- Simple Sensors
- Actuators
- Summary

Introduction

- Humans need to interface with embedded systems ; input data and see response
- Input devices: switches, pushbuttons, keypads, sensors
- Output devices: LEDs, seven-segment displays, liquid crystal displays, motors, actuators



User Interaction

Input Variables

Software

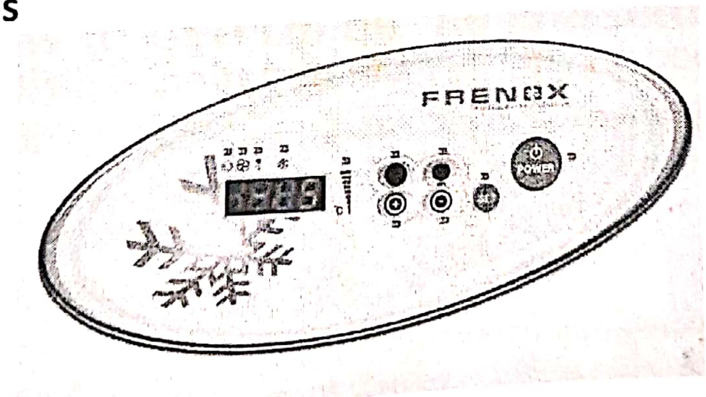
Embedded Computer Hardware

Output Variables

Link to other systems

Introduction

- **Examples**



Fridge Control Panel

آلة التصوير ← **Photocopier Control Panel**

Introduction

- **Examples**

Car Dashboard

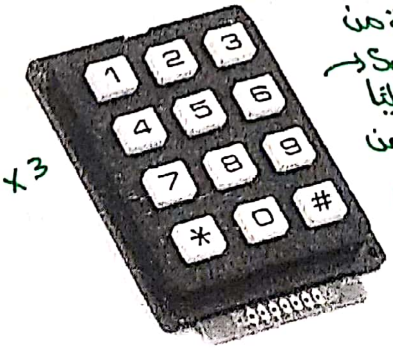
ولو حجزت 10 pins من الـ MC الموضوع مكلف فتحولنا للـ keypads

input component ؟

Moving From Switches to Keypads

- Switches are good for conveying information of digital nature
- They can be used in multiples; each connected to one port pin
- In complex systems, it might not be feasible to keep adding switches ?!
- Use keypads !
  - Can be used to convey alphanumeric values
  - A group of switches arranged in matrix form

4x3



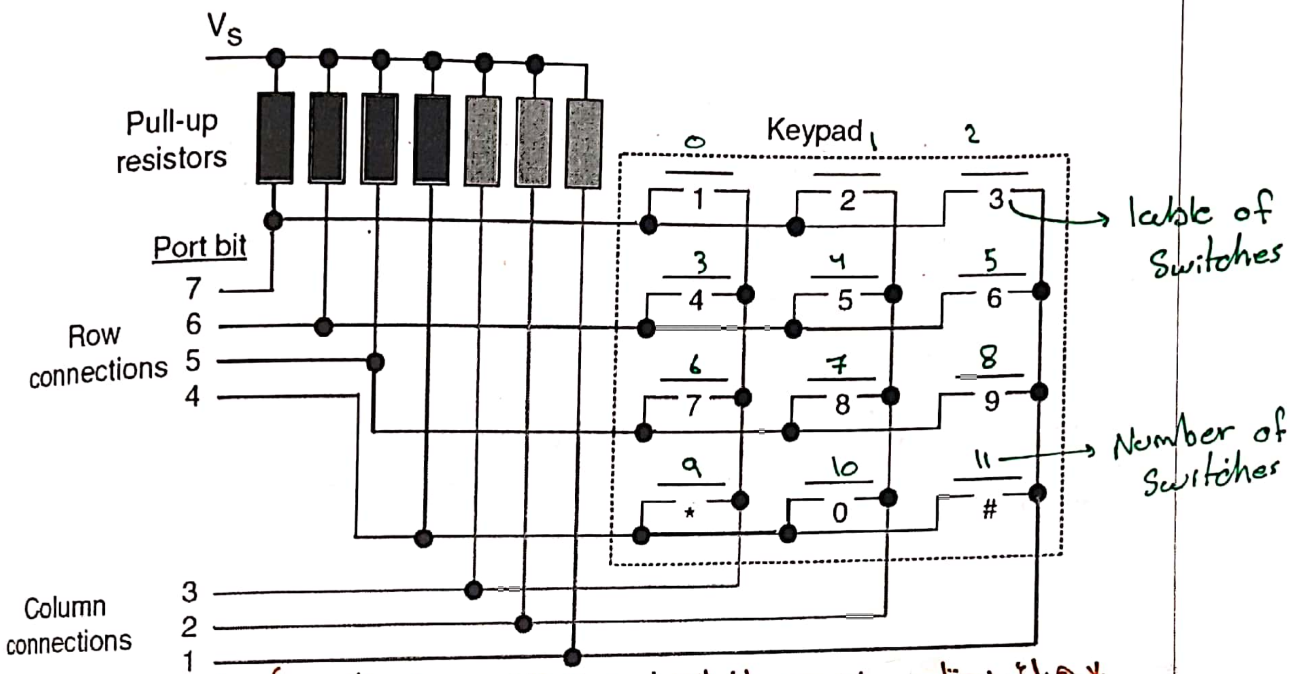
مجموعة من الـ Switches مشبكة داخليا بطريقة تقلل من عدد استخدام الـ Pins

4x4

"مصفوفة داخليا ك 2D"

Moving From Switches to Keypads

Internal Structure of Keypad



* هيك بحتاج 7 ← 4 للصف و 3 للعمود بدل ما كنت أحتاج 10 pins

* لأعرف ال Switch اللي انكبس بدي أعرف رقم السطر والعمود فمرة بخلي ال Row

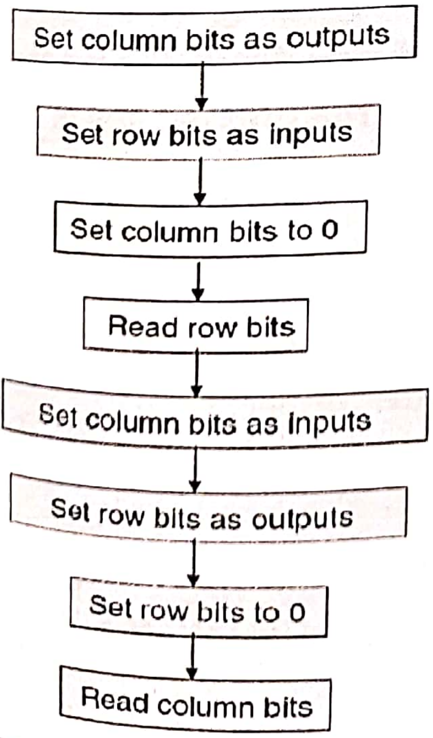
ك input وال column ك output ك 000 لمعرفة الصف ثم أجعل ال column

ك input وال Row ك output ك 000 لمعرفة العمود والعكس صحيح لو pull-down

وال input ال pins اللي حختاره حيكون مكانه 0 والباقي 1 والعكس صحيح لو pull-down،

Moving From Switches to Keypads

How to Determine the Pressed Key



1. Set column bits as outputs
2. Set row bits as inputs
3. Set column bits to 0
4. Read row bits
5. Set column bits as inputs
6. Set row bits as outputs
7. Set row bits to 0
8. Read column bits

غير مستخدم لأنه بحتاج 7 bits

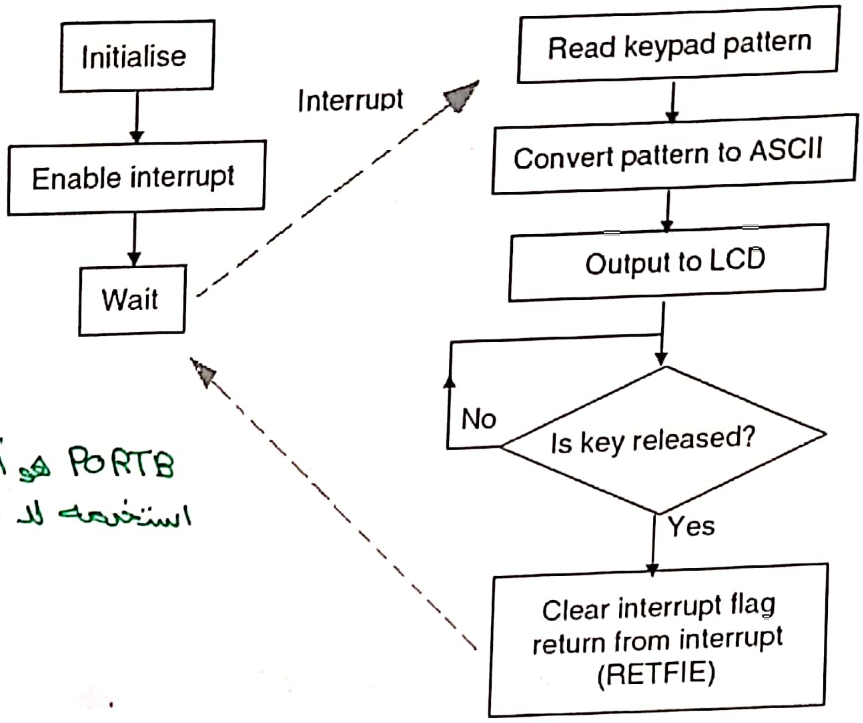
PORTB → RB7 → RB0

row bits ← 0111 011X → column bits

Key	Value Read
1	0111 011X
2	0111 101X
3	0111 110X
4	1011 011X
5	1011 101X
6	1011 110X
7	1101 011X
8	1101 101X
9	1101 110X
*	1110 011X
0	1110 101X
#	1110 110X

Moving From Switches to Keypads

Using Keypad in a Microcontroller



PORTB هو أفضل اشي
استخدمه للـ interrupt

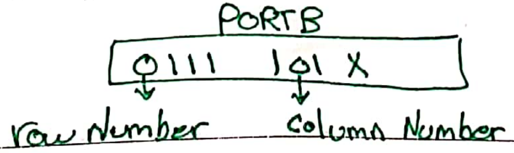
Moving From Switches to Keypads

Example 1

A program to read an input from a 4x3 keypad and display the equivalent decimal number on 4 LEDs. If the pressed key is not a number, then all LEDs are turned on.

مثلاً: كان * ولا #

- The keypad will be connected to MC as follows
  - Rows 0 to 3 connected to RB7 to RB4, respectively.
  - Columns 0 to 2 connected to RB3 to RB1, respectively.
- Use **PORTB on-change** interrupt
- Connect the LEDs to RA0-RA3
- Based on the pressed key, convert the row and column values to binary using a lookup table



0 ① 2
3 4 5
6 7 8
9 10 11

button number = row number ×3+Column number

row wise ترتيبهم

Keypad Interfacing Example

```
            #include        P16F84A.INC
ROW_INDEX   EQU             0X20
COL_INDEX   EQU             0X21
            ORG             0X0000
            GOTO            START
            ORG             0X0004
            GOTO            ISR
START       BSF             STATUS, RP0
            MOVLW           B'11110000'
            MOVWF           TRISB       ; SET RB1-RB3 AS OUTPUT AND
                                        ; RB4-RB7 AS INPUT
            MOVLW           B'00000000'
            MOVWF           TRISA       ; SET RA0-RA3 AS OUTPUT
            BCF             STATUS, RP0
            CLRF            PORTB       ; INITIALIZE PORTB TO ZERO
            MOVF            PORTB,W     ; CLEAR RBIF FLAG
            BCF             INTCON, RBIF
            BSF             INTCON, RBIE
            BSF             INTCON, GIE ; ENABLE PORT b CHANGE INTERRUPT
LOOP        GOTO            LOOP        ; WAIT FOR PRESSED KEY
```

column

↳ rows

Keypad Interfacing Example

```
ISR         MOVF            PORTB, W     ; READ ROW NUMBER
            MOVWF           ROW_INDEX
            BSF             STATUS, RP0  ; READ COLUMN NUMBER
            MOVLW           B'00001110'
            MOVWF           TRISB
            BCF             STATUS, RP0
            CLRF            PORTB
            MOVF            PORTB, W
            MOVWF           COL_INDEX
            CALL            CONVERT      ; CONVER THE ROW AND COLUMN
RST_PB_DIRC BSF             STATUS, RP0  ; PUT THE PORT BACK TO INITIAL SETTINGS
            MOVLW           B'11110000'
            MOVWF           TRISB        ; SET RB1-RB3 AS OUTPUT AND
            MOVLW           B'00000000'  ; RB4-RB7 AS INPUT
            MOVWF           TRISA        ; SET RA0-RA3 AS OUTPUT
            BCF             STATUS, RP0
            CLRF            PORTB
            MOVF            PORTB, W     ; REQUIRED TO CLEAR RBIF FLAG
            BCF             INTCON, RBIF
            RETFIE
```

← بنرجع PORTB للحالة اللي كان فيها حتى يقدر يقرأ كمان مرة من ال keypad

لازم أظل أقرأ بعدين أعمل clear

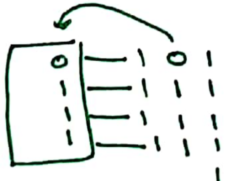
Keypad Interfacing Example

```
CONVERT        BTFSS      COL_INDEX,3 ;  IF  1ST  COLUMN, COL_INDEX=0
               MOVLW      0
               BTFSS      COL_INDEX,2 ;  IF 2ND  COLUMN, COL_INDEX=1
               MOVLW      1
               BTFSS      COL_INDEX,1 ;  IF 3RD  COLUMN, COL_INDEX=2
               MOVLW      2
               MOVWF      COL_INDEX  ;  STORE THE COLUMN INDEX

FIND_ROW       BTFSS      ROW_INDEX,7 ;  IF  1ST  ROW, ROW_INDEX=0
               MOVLW      0
               BTFSS      ROW_INDEX,6 ;  IF  2ND  ROW, ROW_INDEX=1
               MOVLW      1
               BTFSS      ROW_INDEX,5 ;  IF  3RD ROW, ROW_INDEX=2
               MOVLW      2
               BTFSS      ROW_INDEX,4 ;  IF  4TH ROW, ROW_INDEX=3
               MOVLW      3
               MOVWF      ROW_INDEX

               ;  CONTINUED ON NEXT PAGE
```

* طريقة موفقة الـ button مش بالضرورة هيك طبعا في طرق ثانية ممكن نفكر فيها
مثلا كتابة 12 ← if statement ويحول لرقام decimal ويكتب الـ if statement

Keypad Interfacing Example

```
COMPUTE_VALUE      MOVF       ROW_INDEX, W    ;  KEY # = ROW_INDEX*3 + COL_INDEX
                   ADDWF      ROW_INDEX, W
                   ADDWF      ROW_INDEX, W
                   ADDWF      COL_INDEX, W    ; THE VALUE IS IN W
         ; CHECK IF VALUE IS GREATER THAN 11. THIS HAPPENS WHEN THE BUTTON IS RELEASED
         ; LATER, AN INTERRUPT OCCURS WITH ALL SWITCHES OPEN, SO THE  MAPPED  VALUE IS ;
         ; ABOVE 11
                   MOVWF      0X30          ; COPY THE BUTTON NUMBER
                   MOVLW      0X0C
                   SUBWF      0X30,W
                   BTFSC      STATUS, C   ; WILL NOT WORK CORRECTLY, OVERFLOW OCCURS
                   GOTO       LL
                   MOVF       0X30, W
                   CALL       TABLE
                   MOVWF      PORTA         ; DISPLAY THE NUMBER ON PORTA
                   RETURN
LL
```



لما شلنا ايدنا عاد الـ Switch رجعت هيك ولما قارنّا الداخلي بالخارجي صار في interrupt ولكن هذا الـ interrupt خاطئ فالحل هو هذا

Keypad Interfacing Example

```
TABLE           ADDWF           PCL, F
                RETLW           0X01
                RETLW           0X02
                RETLW           0X03
                RETLW           0X04
                RETLW           0X05
                RETLW           0X06
                RETLW           0X07
                RETLW           0X08
                RETLW           0X09
                RETLW           0X0F        ; ERROR CODE
                RETLW           0X00
                RETLW           0X0F        ; ERROR CODE

                END
```

→ (*) رقم ٩

→ (#) رقم ١١

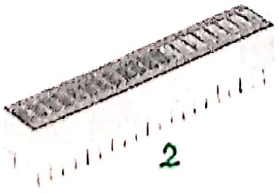
* انت بتفسر ال button بس ينكبس شو حتكون وظيفته وبتفهم ال MC

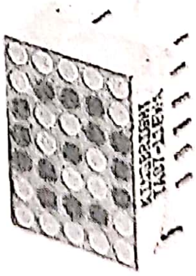
ممكن تشوف هذا المثال عالموقع (..... ucoz.net/index/cpe333_embedded)

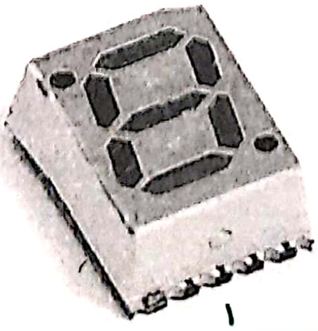
→ output component

LED Displays

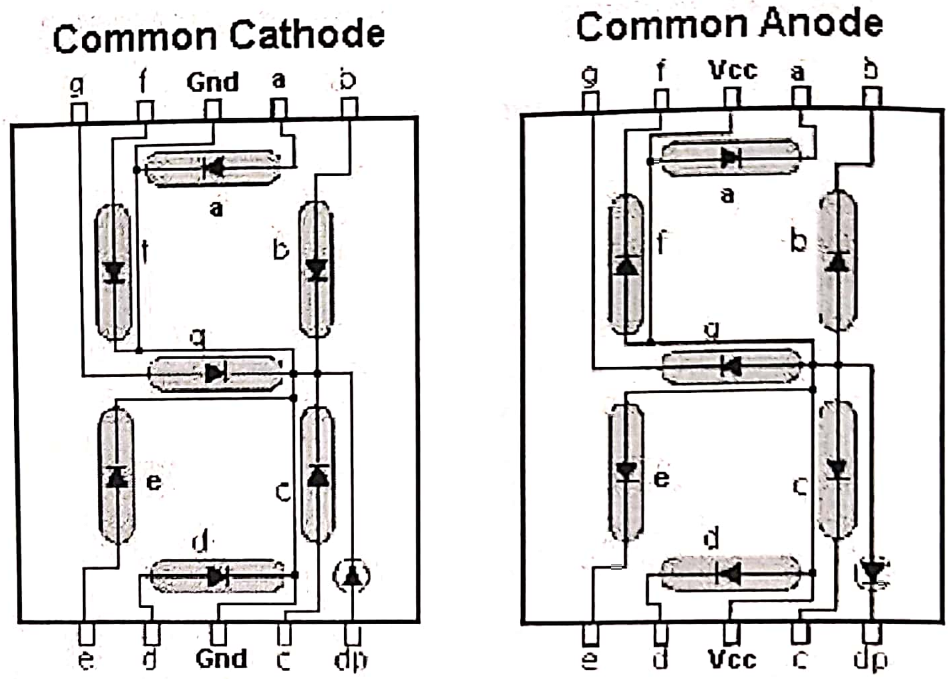
- Light emitting diodes are simple and effective in conveying information
- However, in complex systems it becomes hard to deal with individual LEDs
- Alternatives
  - 1 Seven segment displays
  - 2 Bargraph
  - 3 Dot matrix
  - 4 Star-burst







Seven Segment Display

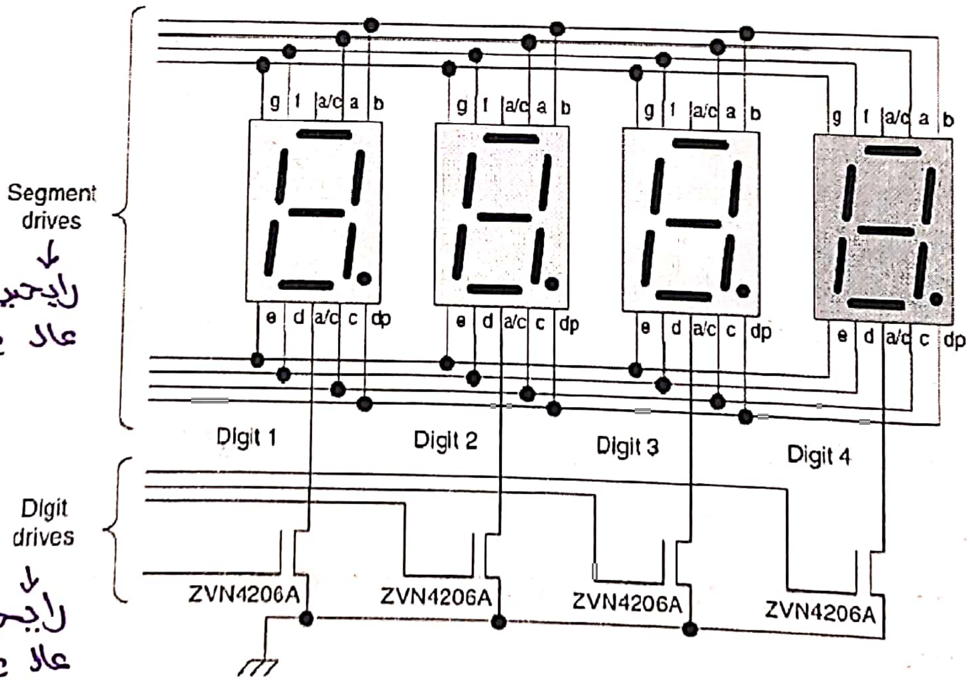


---

*لازم أضل أكرر الـ displays
عشان أضل شايف الرقم
ما بجملها مرة وحدة.

Seven Segment Display

Multiplexing of seven segment digits



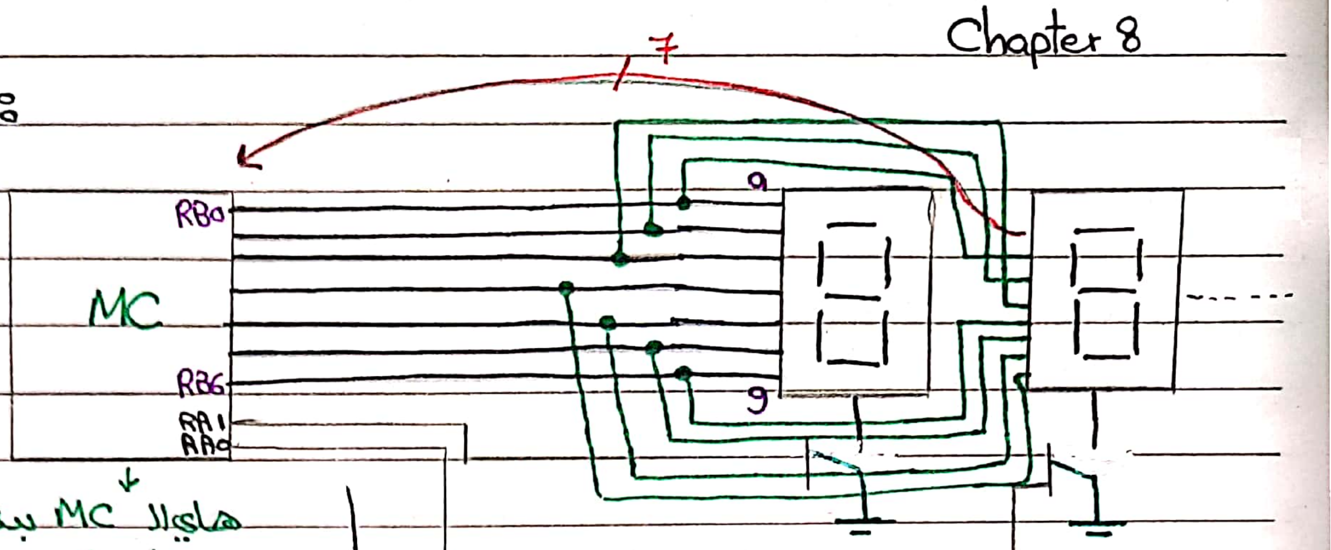
Segment drives → رايحين عال MC

Digit drives → رايحين عال MC

Connection	Port Bit
Segment a	RB7
Segment b	RB6
Segment c	RB5
Segment d	RB4
Segment e	RB3
Segment f	RB2
Segment g	RB1
Segment dp	RB0
Digit 1 drive	RA0
Digit 2 drive	RA1
Digit 3 drive	RA2
Digit 4 drive	RA3

Chapter 8

Slide 17 :



هاي الـ MC بدي استخدمها حتى أتحكم بالـ 7 segment display.

Transistors (electronic Switches)

* المفترض كما هو موضح بالأحمر : إني آخذ الـ Ports تاعت الـ Seven Segment display التاني وأشبكها عالـ MC بس هيك رح يخلصوا الـ Pins اللي بالـ MC عندي فقط لغرض إني أسوي display عالـ 7 segment display . (# pins = 7 * N → displays عدد الـ).

* الحل هو الـ Seven Segment display multiplexing : الفكرة فيه بدال ما أخصص مجموعة من الـ pins لكل وحدة من الـ 7 segment display بحكي خليني أسوي multiplexing (sharing) إني أسمح لـ 7 segment displays إنها تشترك بنفس الـ pins كما هو موضح باللون الأخضر.

* بهاي المشاركة أي output عالـ Pins رح يظهر نفسه عالـ 2 displays ومش هادا اللي بدي ياه فالحل إني أحط Switch على كل display يوضحلي وين بدي أظهر الرقم كما هو موضح بالأخضر. وحسكّر الـ Switch عالـ display اللي بدي أظهر عليها الناتج وهيك بزمن معين العين ما حتميز وحتشوف الشيئين خاويين لو كان الزمن مناسب. الـ Switches المستخدمة مش mechanical لإنه بدي أتحكم فيهم عن طريق الـ MC فبدي ياهم electronic يعني Transistors

(# Pins = 7 + N)

data ↲ ↳ displays عدد الـ

→

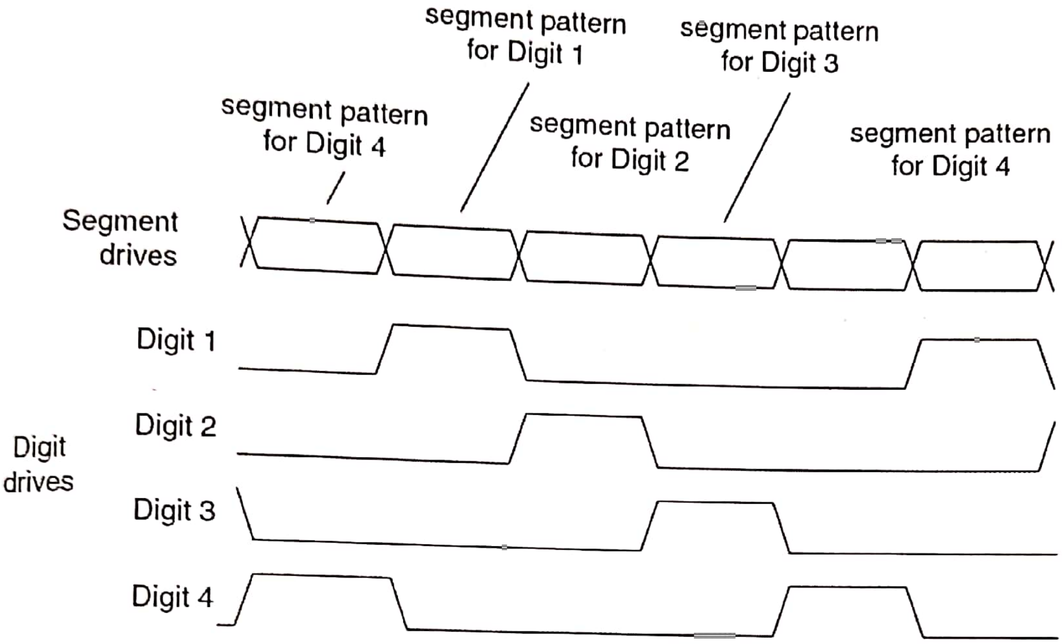
PAGE

DATE

*ممكن أرجع أقلل ال pins باستخدام External hardware مثل إني أجيب
1-to-2 decoder وأشبكه مع ال MC وهو بحتاج فقط 1 pin وبشبك ال output
الخارج مع ال 2 Switches وهيك قللت عدد ال pins لـ : $\#pins = 7 + \log_2 N$

Seven Segment Display

Multiplexing of seven segment digits

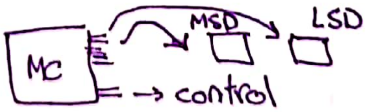


↲ ويكرر هاي العملية over time

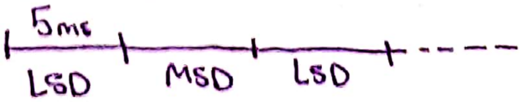
Seven Segment Display

Example 2

A program to count continuously the numbers 0 through 99 and display them on two seven segment displays. The count should be incremented every 1 sec. Oscillator frequency is 3 MHz.



- Connect the seven segment inputs a through g to RB0 through RB6, respectively
- Connect the gates of the controlling transistors to RA0 (LSD) and RA1 (MSD)
- The main program will be responsible for display and multiplexing every 5 ms



$F = \frac{1}{5ms} = 200 HZ$

↳ وهذا رائع

Seven Segment Display Example

```
                #INCLUDE        PICF84A.INC
LOW_DIGIT       EQU             0X20
HIGH_DIGIT      EQU             0X21
COUNT           EQU             0X22
                ORG             0X0000
                GOTO            START
                ORG             0X0004
ISR             GOTO            ISR
START           BSF             STATUS, RP0
                MOVLW           B'00000000'  ; set port B as output
                MOVWF           TRISB
                MOVWF           TRISA        ; SET RA0-RA1 AS OUTPUT
                BCF             STATUS, RP0
                CLRF            PORTB
                CLRF            PORTA
                CLRF            LOW_DIGIT   ; CLEAR THE COUNT VALUE
                CLRF            HIGH_DIGIT
                CLRF            COUNT
```

مالها داعي

Seven Segment Display Example

```
DISPLAY         BSF             PORTA , 0
                BCF             PORTA,  1
                MOVF            LOW_DIGIT, W   ; DISPLAY LOWER DIGIT
                CALL            TABLE          ; GET THE SEVEN SEGMENT CODE
                MOVWF           PORTB
                CALL            DELAY_5MS    ; KEEP IT ON FOR 5 MS
                BCF             PORTA, 0
                BSF             PORTA, 1
                MOVF            HIGH_DIGIT, W   ; DISPLAY  HIGH DIGIT
                CALL            TABLE           ; GET THE SEVEN SEGMENT CODE\
                MOVWF           PORTB
                CALL            DELAY_5MS    ; KEEP IT ON FOR 5 MS
                ; CHECK IF  1 SEC ELAPSED
                INCF            COUNT,F      ; INCREMENT THE COUNT VALUE IF TRUE
                MOVF            COUNT, W
                SUBLW           D'100'
                BTFSS           STATUS, Z
                GOTO            DISPLAY     ; DISPLAY THE SAME COUNT
```

≈ 10 ms

→ $100 * 10 * 10^{-3} = 1$ Sec

Scanned with CamScanner

Seven Segment Display Example

```
; TIME TO INCREMENT THE COUNT
CLRF        COUNT
INCF        LOW_DIGIT, F   ; INCREMENT LOW DIGIT AND CHECK IF > 9
MOVF        LOW_DIGIT, W
SUBLW       0X0A
BTFSS       STATUS, Z
GOTO        DISPLAY
CLRF        LOW_DIGIT

INCF        HIGH_DIGIT, F ; INCREMENT HIGH DIGIT AND CHECK IF > 9
MOVF        HIGH_DIGIT, W
SUBLW       0X0A
BTFSS       STATUS, Z
GOTO        DISPLAY
CLRF        HIGH_DIGIT
GOTO        DISPLAY
```

Seven Segment Display Example

```
DELAY_5MS   MOVLW       D'250'
            MOVWF       0X40
REPEAT      NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            NOP
            DECFSZ      0X40,1
            GOTO        REPEAT
            RETURN
```

5 ms
when 3 MHZ

Seven Segment Display Example

```
TABLE              ADDWF  PCL,        1
                   RETLW      B'00111111'              ;'0'
                   RETLW      B'00000110'              ;'1'
                   RETLW      B'01011011'              ;'2'
                   RETLW      B'01001111'              ;'3'
                   RETLW      B'01100110'              ;'4'
                   RETLW      B'01101101'              ;'5'
                   RETLW      B'01111101'              ;'6'
                   RETLW      B'00000111'              ;'7'
                   RETLW      B'01111111'              ;'8'
                   RETLW      B'01101111'              ;'9'


                   END
```

common cathode ↙

روح عموقع ucoz.net لرؤية المثال.

← الها علاقة بال physical qauntity

Sensors

← مصنوعين من مادة حساسة للظاهرة الفيزيائية الي أنا موجودة فيها

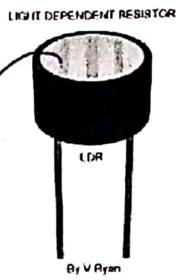
- Embedded systems need to interface with the physical world and must be able to detect the state of the physical variables and control them
- <u>Input transducers</u> or sensors are used to convert physical variables into electrical variables. Examples are the light, temperature and pressure sensors
- <u>Output transducers</u> convert electrical variables to physical variables.

Sensors

Light-dependent Resistors

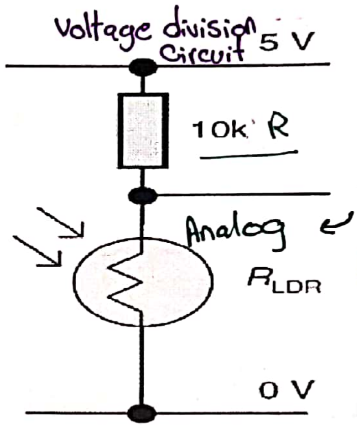
لـ قيمتها متغيرة تعتمد على شدة الإضاءة

- A light-dependent resistor (LDR) is made from a piece of exposed semiconductor material مكشوفة لـ
- When light falls on it, it creates hole–electron pairs in the material, which improves the conductivity.



$V_o = \frac{R_s}{R_s + R} * 5v$ ← متغيرة

شدة الإضاءة Illumination (lux)	المقاومة R_{LDR} (Ohms)	Vo
Dark	2M	5
10	9000	2.36
1000	400	0.19



الـ Voltage الموجود على مقاومة الـ Sensor وأدخله ع ADC وأصير أحسب كم الـ Voltage وأحسب شدة الإضاءة

لـ

كل ما زادت شدة الإضاءة كل ما المقاومة بتنزل.

لـ من الـ V بقدر أطلع الـ R

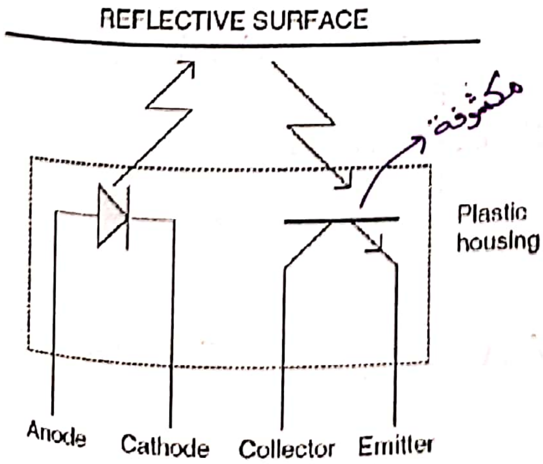
Sensors

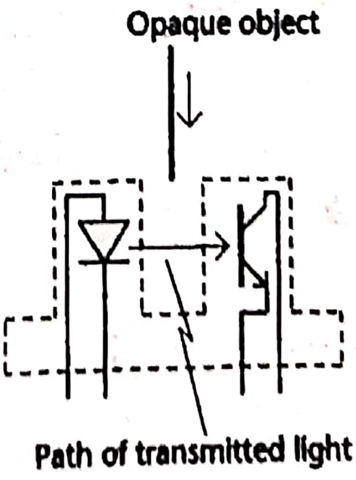
Optical Object Sensing

→ أعرف في object أمامي لمسافة معينة.

- Useful in sensing the presence or closeness of objects
- The presence of object can be detected
  - If it breaks the light beam
  - If it reflects the light beam

* بناءً عمقدار الـ current والـ Voltage بقدر أعرف قديه بعيد الـ object.

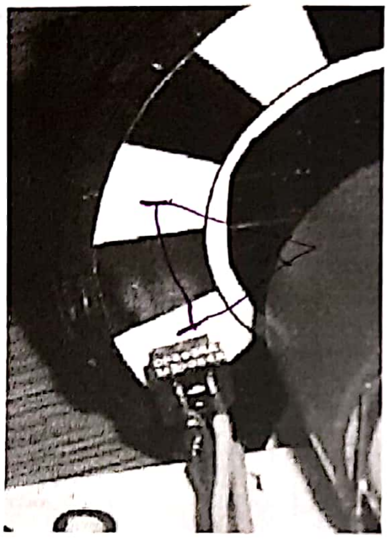


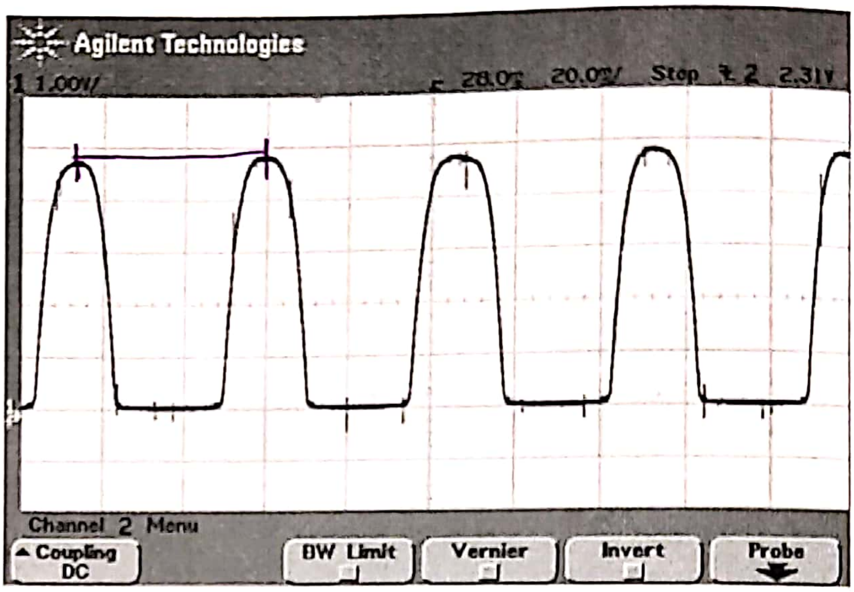


Sensors

Opto-sensor as a Shaft Encoder

- Useful in measuring distance and speed





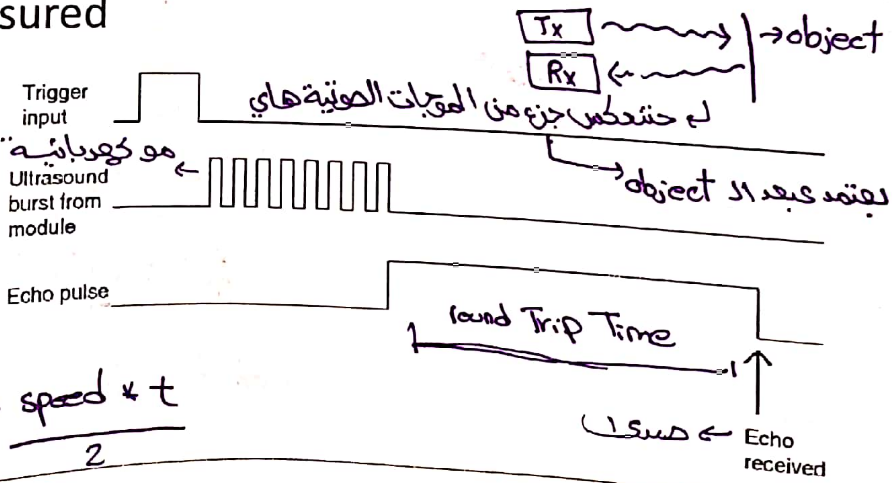
لحتى يقدر يتحكم بسرعة الروبوت.

Sensors

فوق صوتي

Ultrasonic Object Sensor

- Based on reflective principle of ultrasonic waves
- An ultrasonic transmitter sends out a burst of ultrasonic pulses and then the receiver detects the echo
- If the time-to-echo is measured, distance can be measured



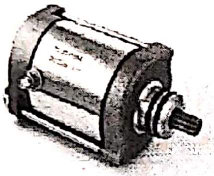
بعد الـ object ← $d = \frac{speed * t}{2}$

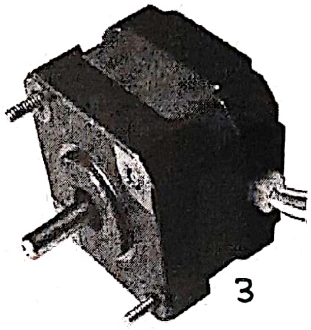
physical quantities التحكم بال

Actuators: motors and servos

- Embedded systems need to cause physical movement
- Linear or rotary motion
- Most actuators are electrical in nature
  - 1 • Solenoids (linear motion)
  - 2 • DC Motors
  - 3 • Stepper motors
  - 4 • Servo motors


1


2


3


4

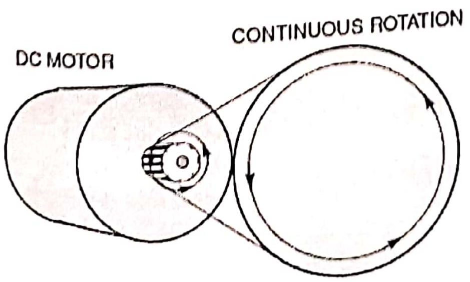
معظمهم ما بقدر أشبكهم مباشرة
عال MC

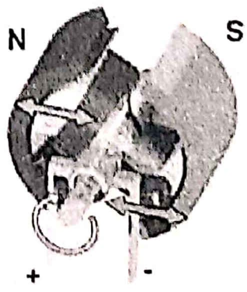
DC Motors

الحركة في DC
متواصلة بوجود supply

- Range from the extremely powerful to the very small
- Wide speed range
- Controllable speed
- Good efficiency
- Can provide accurate angular positioning with angular shafts
- Only the armature winding needs to be driven

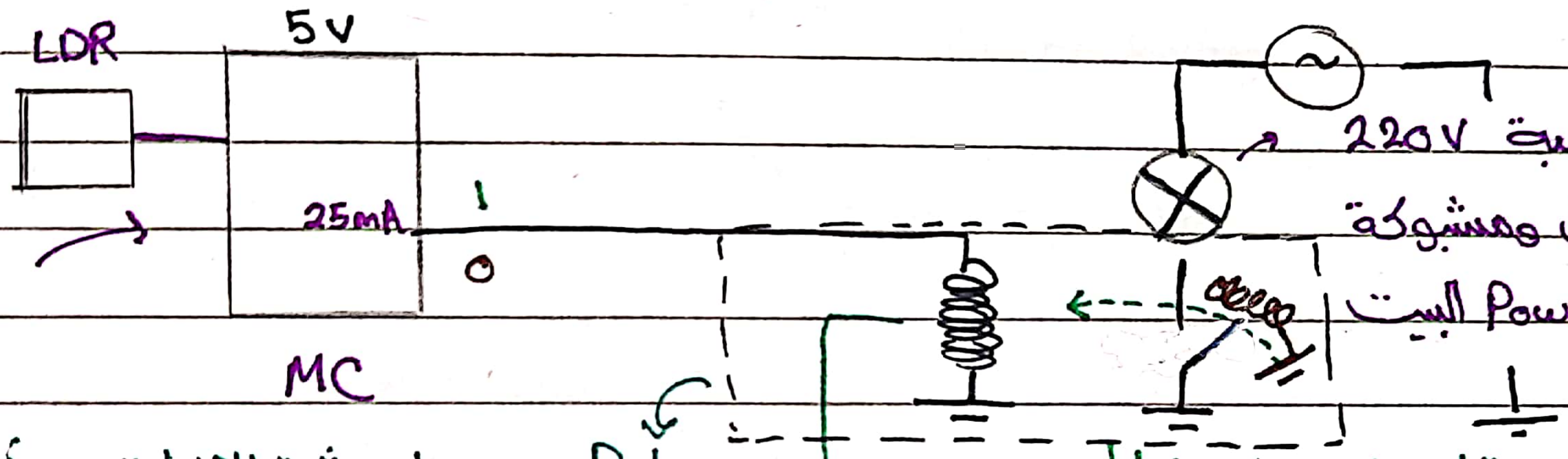
* لو بدي أغير السرعة بغير ال Power supplay
* لو بدي أغير اتجاه الدوران بغير اتجاه ال Polarity
اللي موجودة عالملف عال DC Motor .





* المغناطيس ثابت والسلك بلف.

Slide 318



هاي اللمبة 220V
بتشتغل ومشبوكة
ع Power البيت

* بتلك كبسة الإضاءة يلي أتحكم بلمبة موجودة بالغرفة هل أضويها ولا أطفيها.

لما يمر فيه تيار بولد مجال مغناطيسي وبصير ما
عنده قدرة جذب فممكن يجذب ال Switch
وهذا حيصير لما يكون ال output = 1
بمجرد ما يرجع ال Value ل 0 بكون عندي زمبرك
يرجع يفتح ال Switch .

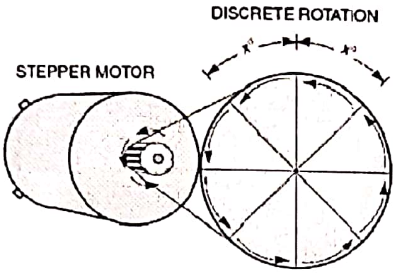
Relay
عن طريقه بتحكم بفتح وإغلاق
ال Mechanical Switch

* عادة كمية التيار اللي بحتاجها حتى أتحكم بال Switch مش قليلة فعادة حتى مع وجود
ال Relay بحتاج أجيب Transistor للتحكم بوضع ال Relay .

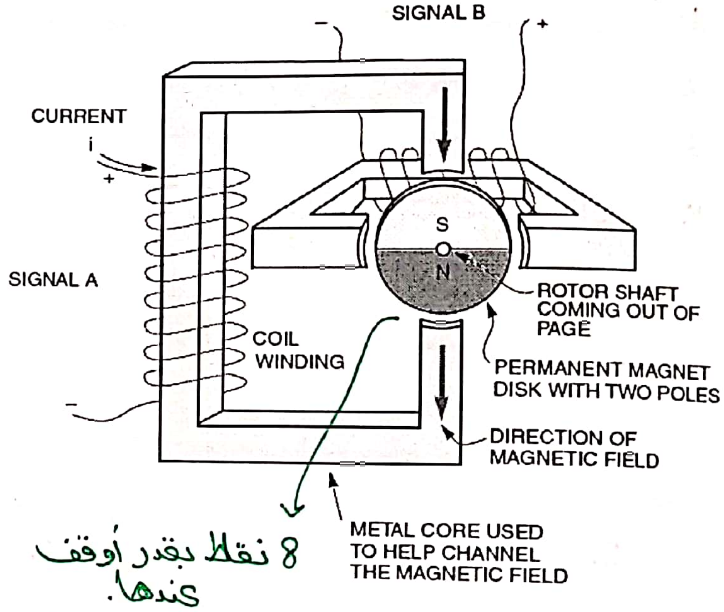
← بقدر استخدمه لأتحرك ب Steps .
مش ألف الدورة كاملة .

Stepper Motors

- A **stepper motor** (or **step motor**) is a synchronous electric motor that can divide a full rotation into a large number of steps.

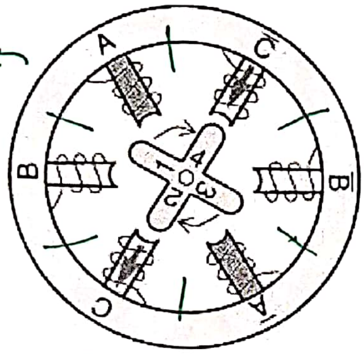


عكس ال DC المغناطيس هو الي بتحرك



8 نقاط بقدر أوقف عندها.

← زدت ال Cores صار يوقف بأماكن أكثر .



*لكل core حينشبك ع 2 pins واحد output وواحد input فبتطلب عدد pins كبير.

Stepper Motors

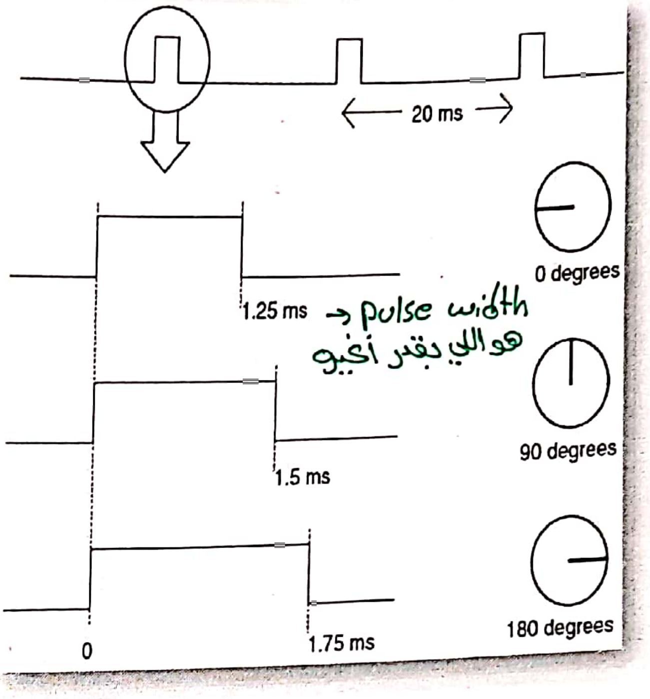
- **Features**
  - Simple interface with digital systems
  - Can control speed and position
  - More complex to drive
  - Awkward start-up characteristics
  - Lose torque at high speed
  - Limited top speed
  - Less efficient

ومستخدم بالروبوتات الصغيرة

بستخدمه لأنه بخطوات لكن ب Pins أقل مو بحاجة لعدد Pins كبير بحتاج سلك واحد.

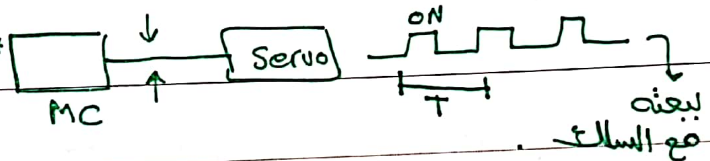
Servo Motors

- Allows precise angular motion
- The output is a shaft that can take an angular position over a range of 180°
- The input to the servo is a pulse stream whose width determines the angular position of the shaft



→ pulse width هو اللي بقدر أغيرو

pulse width modulation



---

*ال MC دائمًا مش وظيفتها Power the compo- وظيفتها control the component

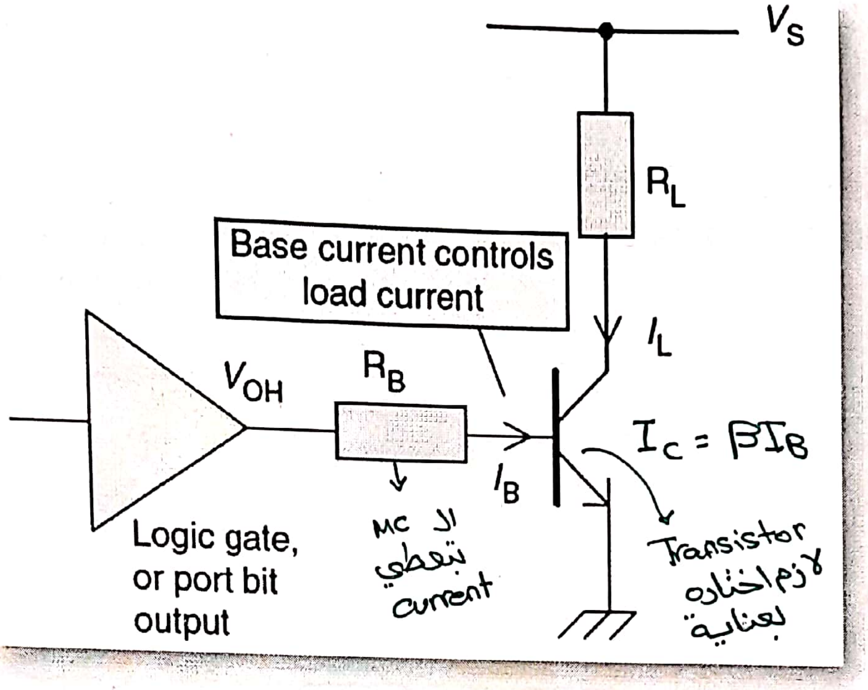
Interfacing to Actuators

*معظم ال devices اللي بشبكها عال MC ال rating تاعها عالي فما بقدر أشبكها مباشرة عال MC

- Microcontrollers can drive loads with small electrical requirements
- Some devices, like actuators, require high currents or supply voltages
- Use switching devices → Transistor عبارة عن
  - Simple DC switching using BJTs or MOSFETs
  - Reversible DC switching using H-bridge

Interfacing to Actuators

Simple DC interfacing



Base current controls load current

Logic gate, or port bit output

ال MC بتعطي current

$I_C = \beta I_B$

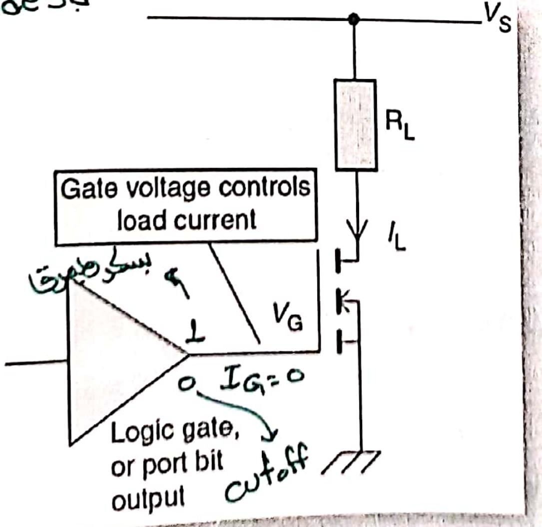
Transistor لازم اختاره بعناية

* محبذ دائما ما نشبك ال device مع ال MC مباشرة ، نحط Switch أفضل .

Interfacing to Actuators

Simple DC interfacing

مهم ، وظيفته لما أطفي ال load ال inductor حيفرغ ال charge تاعته بال diode



Gate voltage controls load current

Logic gate, or port bit output

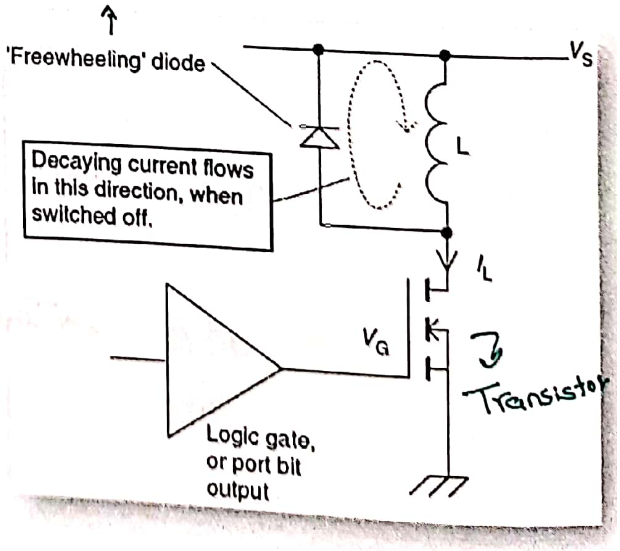
بسكر طريق

1

0 $I_G=0$

cutoff

Resistive load



'Freewheeling' diode

Decaying current flows in this direction, when switched off.

Logic gate, or port bit output

Transistor

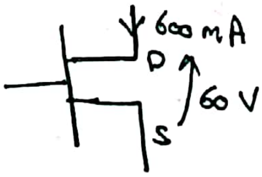
Inductive load ← زي ال motors و ال Relay

Interfacing to Actuators

Simple DC interfacing

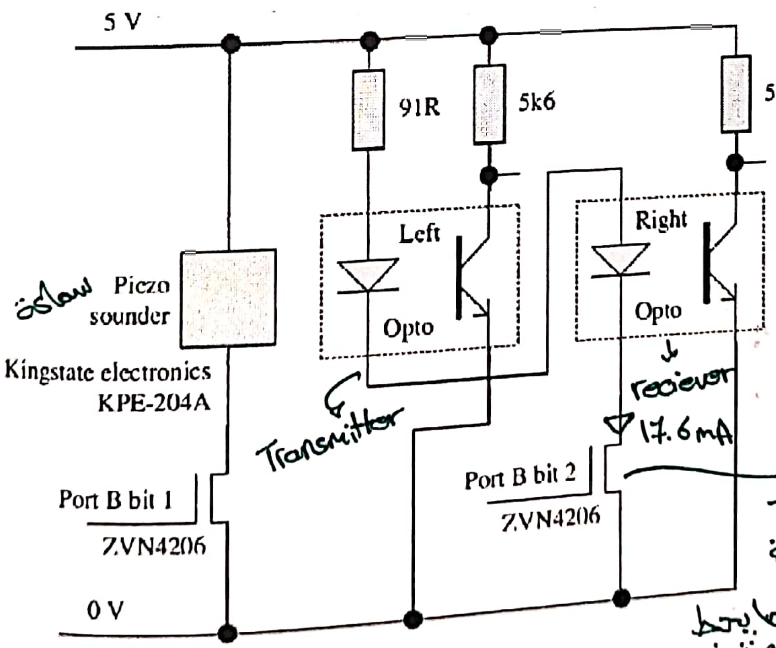
Characteristics of two popular logic-compatible MOSFETs

Characteristic	ZVN4206A	ZVN4306A
Maximum drain-to-source voltage, V_{DS} (V)	60	60
Maximum gate-to-source threshold, $V_{GS(th)}$ (V)	3	3
Maximum drain-to-source resistance when 'on', $R_{DS(on)}$ (Ω)	1.5	0.33
Maximum continuous drain current, I_D	600 mA	1.1 A
Maximum power dissipation (W)	0.7	1.1
Input capacitance (pF)	100	350



Interfacing to Actuators

Driving Piezo Sounder and Opto-sensors



تاع الروبوت
اللي أخذناه
بالبداية

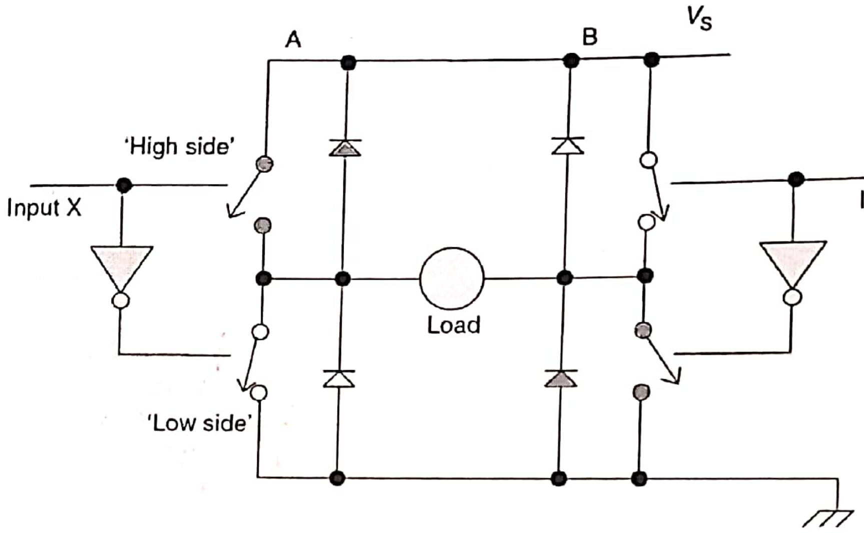
فضل
يحط Transistor
ما شبكوا مباشرة
عال MC عشان ما يحط
عليها load كبير وتخرب

- Piezo sounder ratings: 9mA, 3-20 V
- The opto-sensor found to operate well with 91 Ohm resistor. The diode forward voltage is 1.7V. The required current is about 17.6 mA

Interfacing to Actuators

Reversible DC Switching

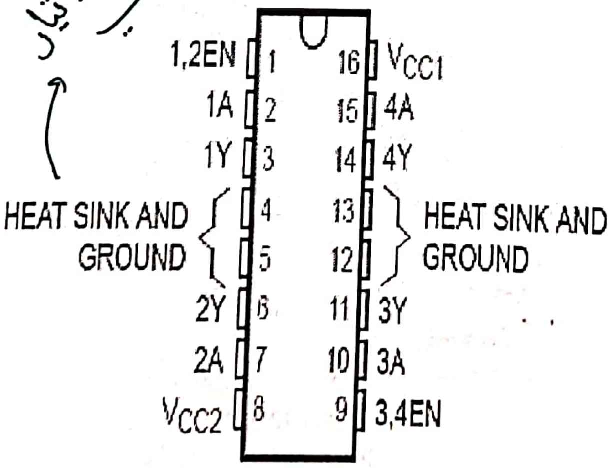
- DC switching allows driving loads with current flowing in one direction
- Some loads requires the applied voltage to be reversible; DC motors rotation depends on direction of current
- Use H-bridge !

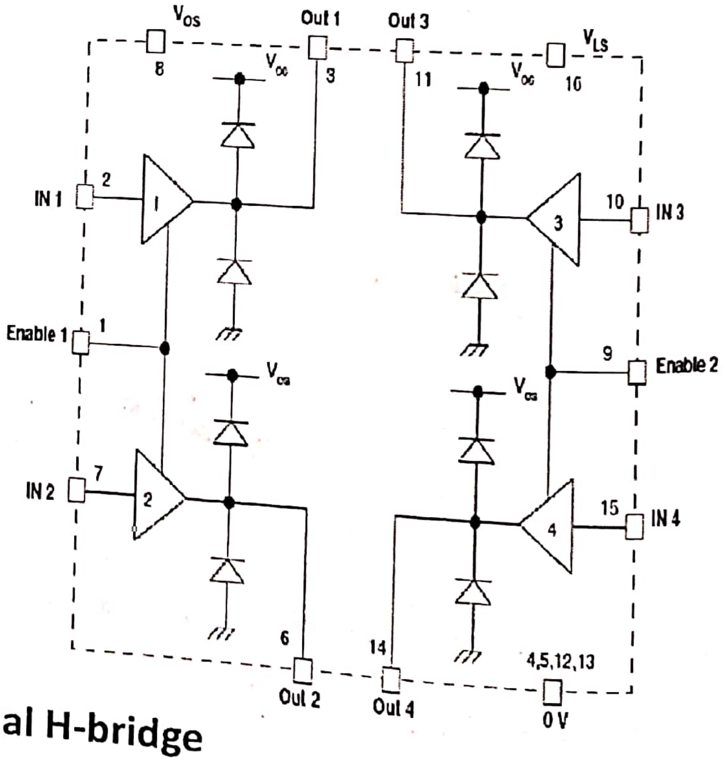


Interfacing to Actuators

Reversible DC Switching

يحمو بتمرير تيار عالي

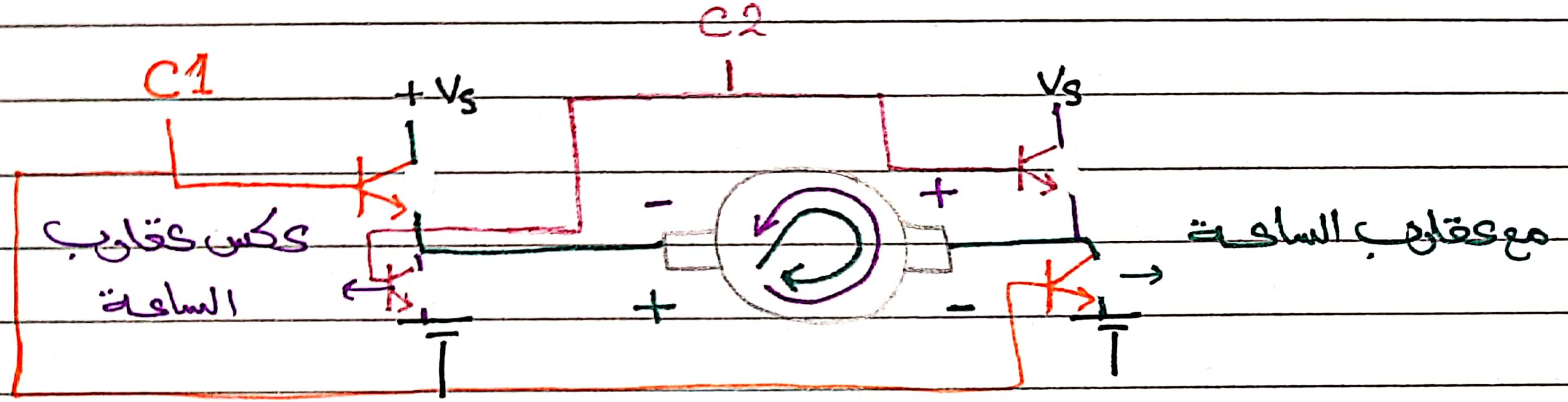




L293D Dual H-bridge

Peak output current 1.2 A per channel

Slide 41 :



*يعني DC Motor بلف مع أو عكس اتجاه عقارب الساعة ، حسب ال Polarity تبع ال Applay voltage.

*قدرتي بإني أتحكم بإتجاه الدوران تبع ال Motor مهمة جداً وهذا الكلام بتطلب مني إني أتحكم باتجاه ال Polarity اللي داخلة عهذا ال Motor والتحكم هذا بدي اياه يصير بطريقة الكترونية فلازم يكون عندي القدرة على فصل و توصيل أي من ال 2 paths المرسومين لاختار اتجاه اللف. زي كأنه بدي نوع من ال Switching كما موضح بالرسم.

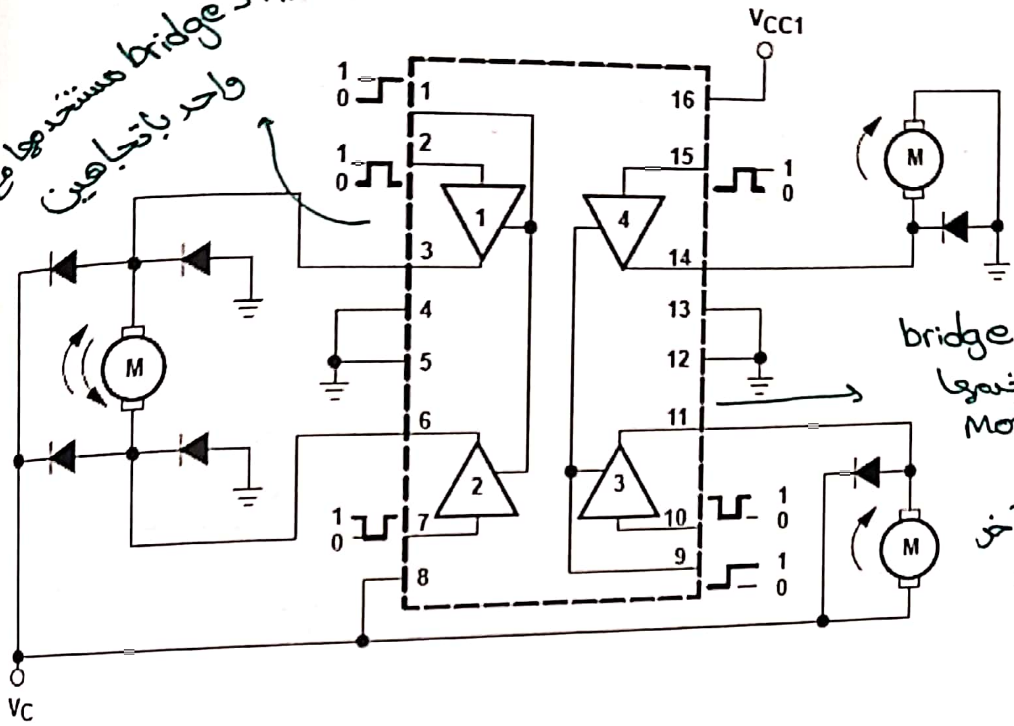
*هاي السيركت موجودة جاهزة بالسوق اسمها H-bridge (full-bridge).

Interfacing to Actuators

Reversible DC Switching

Driving three motors using L293D

هاي ال bridge مستخدمها مع Motor واحد باتجاهين



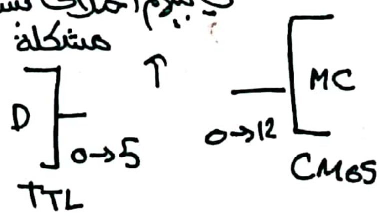
هاي ال bridge نحنا مستخدمها مع Motor ونحنا مع Moto آخر باتجاه واحد

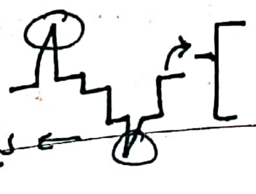
(مهم)

More on Digital Input

- When acquiring digital inputs into the microcontroller, it is essential that the input voltage is within the permissible and recognizable range of the MC
- Voltage range depends on the logic family; TTL, CMOS, …
- Interfacing within the same family is safe
- What for the case →
  من وين ممكن تيجي المشاكل :-
  - Interfacing to digital sensors →
  - Signal corruption
  - Interference →

في بينهم اختلاف بسبب مشكلة

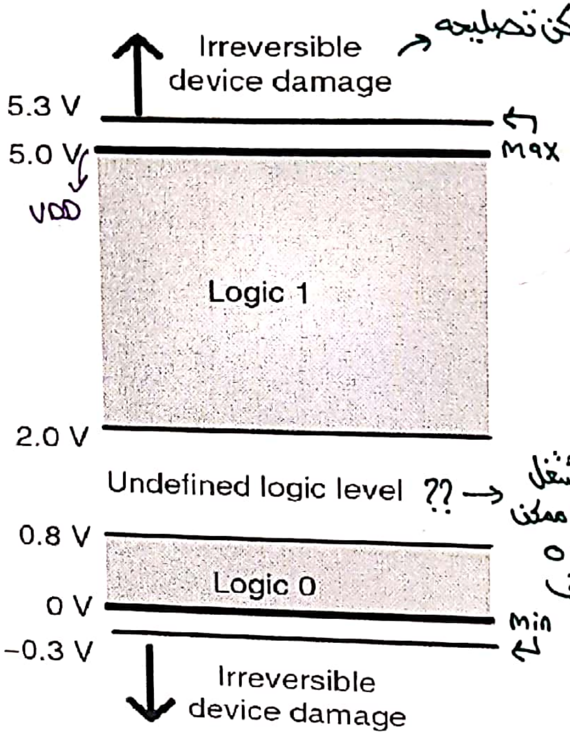




← بسببوا مشكلة

More on Digital Input

PIC16F873A Port Characteristics



خراب لا يمكن تصليحه

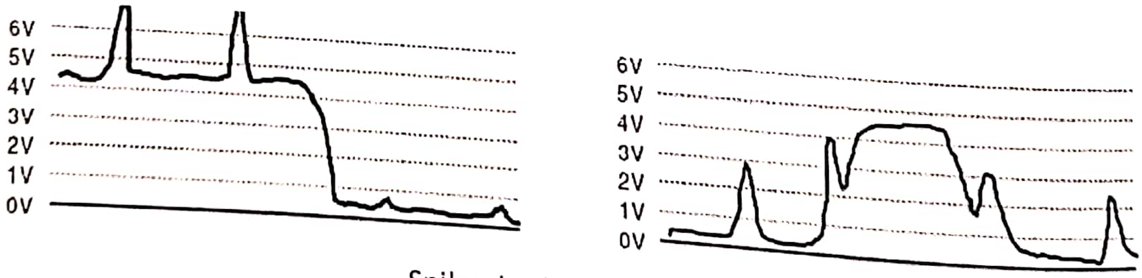
لا يفضل تشتغل فيها لإنه ممكن تكون 0/1 ما بنعرف

* لازم أحاول أضمن إنه ال Voltage input ضمن ال range اليكتني (0-5 V) عشان ما يصير غير مستخدم.

أمثلة :

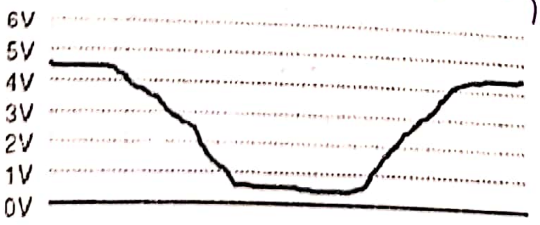
More on Digital Input

Forms of Signal Corruption

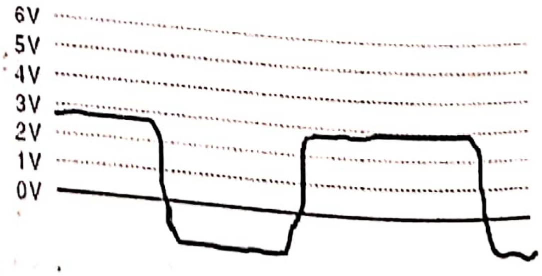


Spikes in the signal

ال edge بتنقل من state لأخرى بفترة زمنية أطول وكل ما زادت هاي الفترة كل ما صار عندي مشكلة وجود undefined logic level



Slow edge



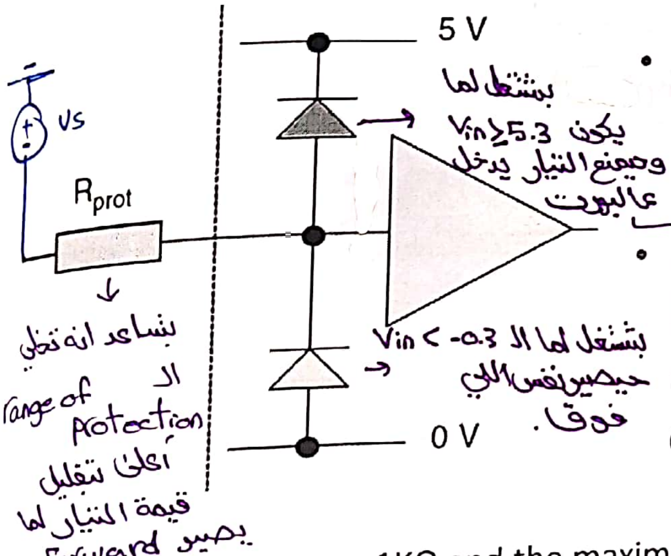
DC Offset in the signal

من الشغلات اللي ممكن أسويها حتى أتعامل مع المشاكل السابقة :

More on Digital Input

to deal with out of range Voltages.

① Clamping Voltage Spikes



- All ports are usually protected by a pair of diodes
  → short circuit
- An optional current limiting resistor can be added if high spikes are expected

ال diode اللي فوق ↑ max

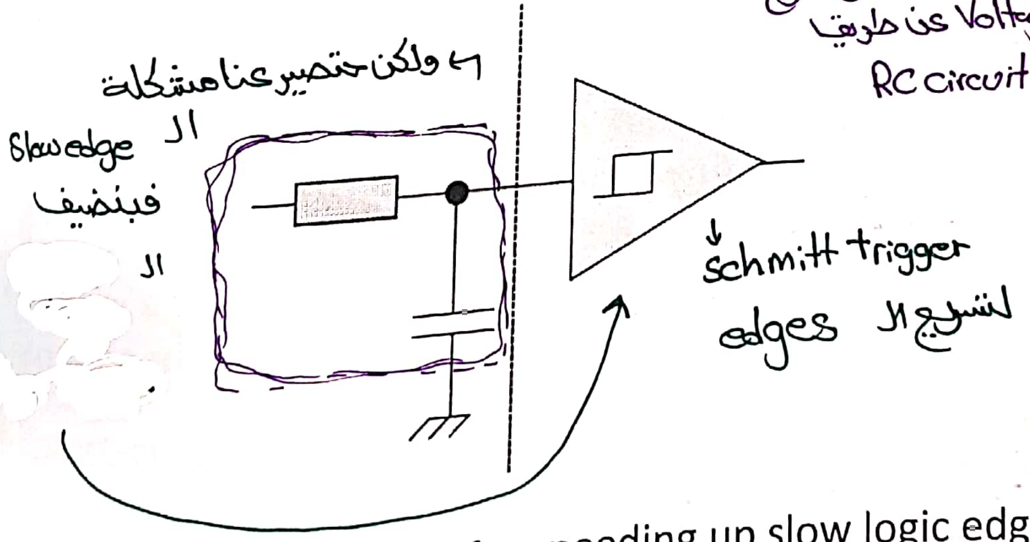
- **Question?** Let R_{prot} = 1KΩ and the maximum diode current is 20 mA when V_d = 0.3v, then what is the maximum positive voltage spike that can be suppressed? $-V_S + I_D(R_{prot}) - V_D + 5 = 0 \rightarrow V_S = 5 - 0.3 + 20 = 24.7\ V$ ↑ max

More on Digital Input

للتخفيف من ال Spikes

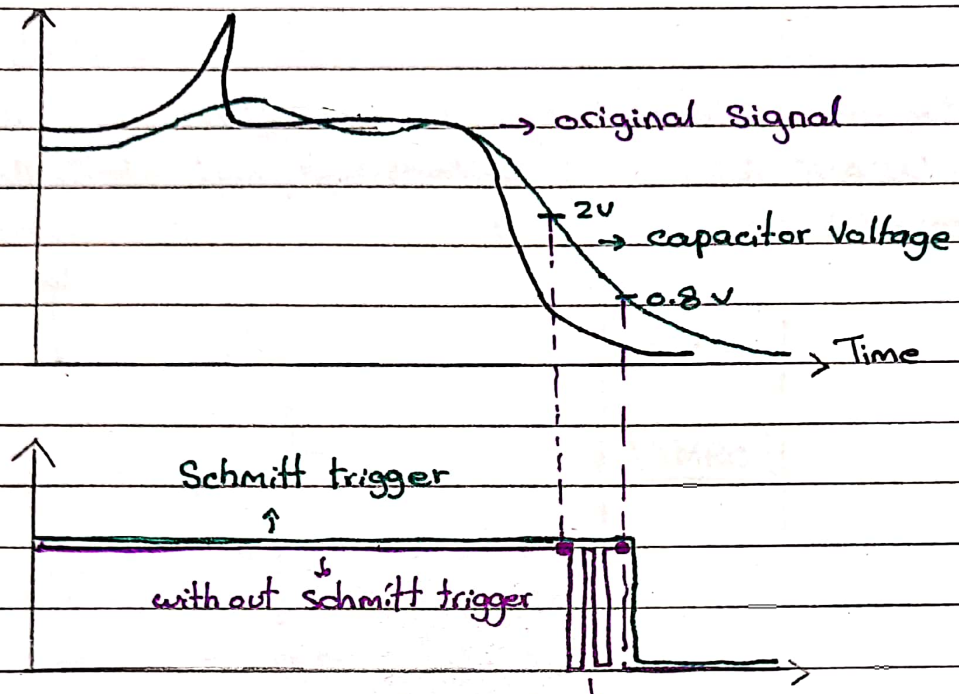
Analog Input Filtering

بحلها عن طريق مقاومة التغير السريع في ال Voltage عن طريق ال RC circuit



- Can use Schmitt trigger for speeding up slow logic edges.
- Schmitt trigger with RC filter can be used to filter voltage spikes.

Slide 48 : Schmitt trigger.



unknown مابعرف كيف بتغير

فالحل أدخل الـ Schmitt triggers Voltage أول الي باللون
الأخضر ، وميزته بتذكر إنه ما بصير يتغير إلا بين النقطتين
تاعته فما بصير في undefined كثير

الحل لل Switch bouncing

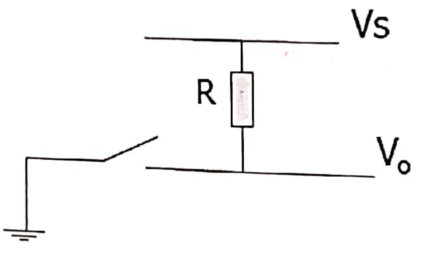
More on Digital Input

Switch Debouncing

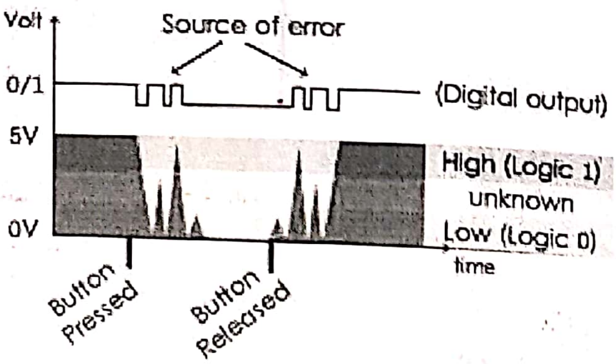
الحركة اللاإرادية للزمبرك الموصول بال Switch

وجودها بأثر علي من ناحية عمل ال Sys

- Mechanical switches exhibit bouncing behavior
- The switch contact bounces between open and closed
- A serious problem for digital devices ?!



Simple switch interface



- Switch debouncing!! hardware and/or software techniques

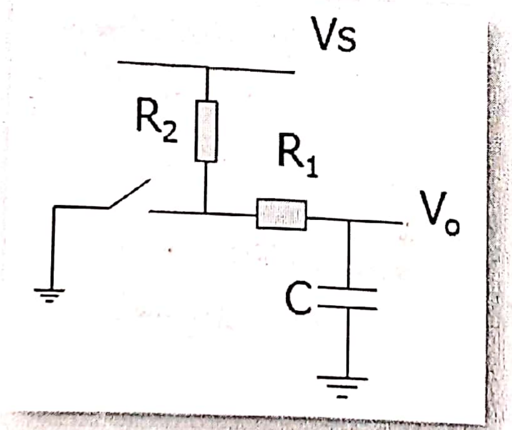
يحله باستخدام ال hardware أو ال Software

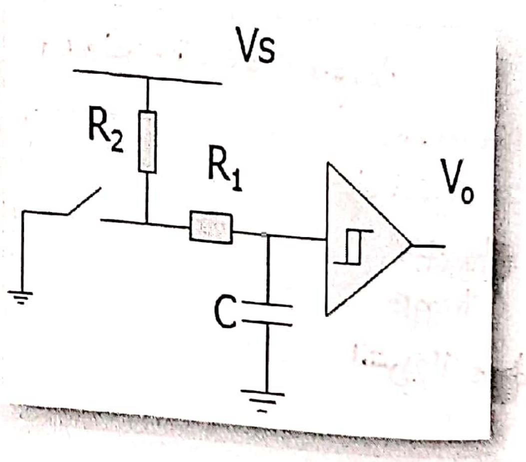
---

hardware

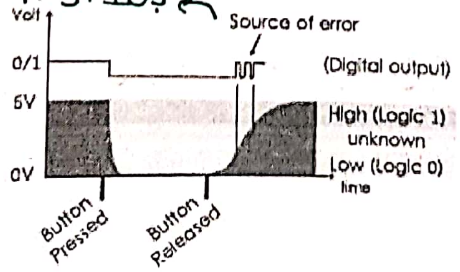
More on Digital Input

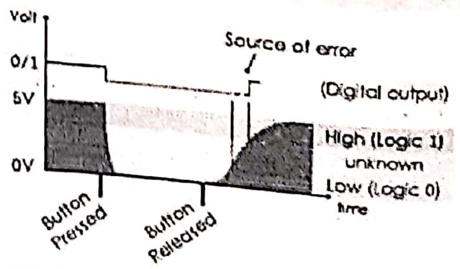
Switch Debouncing





بحط ال Schmitt مشان ماأشوفهم

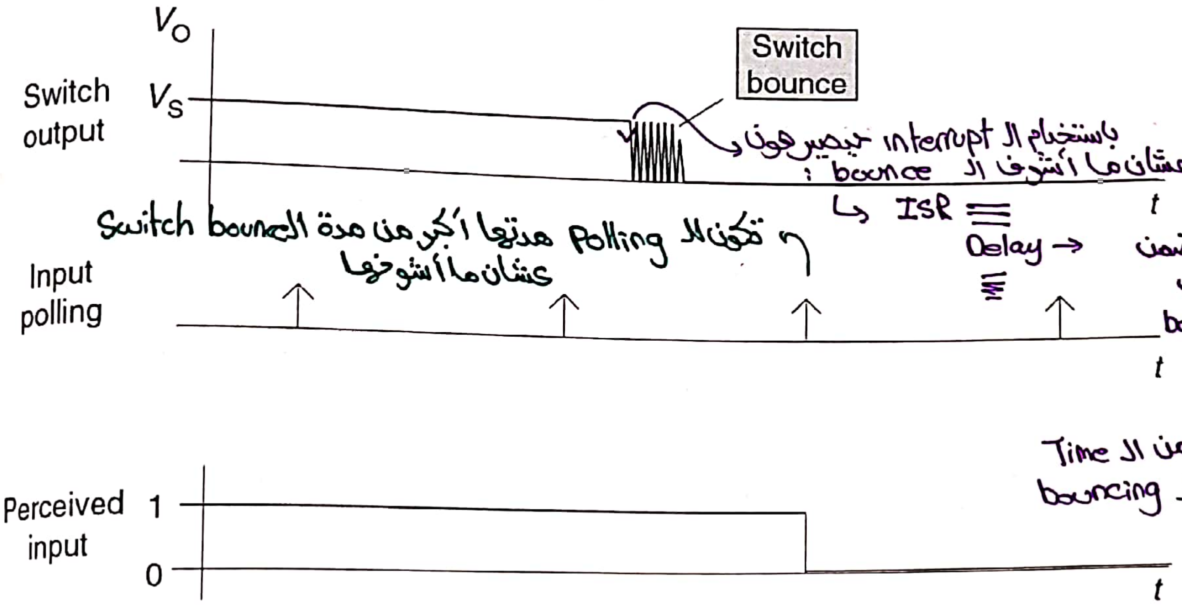




More on Digital Input

Switch Debouncing

→ Software



باستخدام ال interrupt حيصير هون
فعشان ما أشوف ال bounce :
↳ ISR ≡ Delay → عشان أضمن ما أشوف ال bounce وال Time تاع ال ISR بكون أكبر من ال Time تاع ال bouncing

تكون ال Polling مدتها أكبر من مدة ال Switch bounce عشان ما أشوفها

Summary

- Microcontrollers must be able to interface with the physical world and possibly the human world
- Switches, keypads and displays represent typical examples for interfacing embedded systems with the humans
- Microcontrollers must be able to interface with a range of input and output transducers.
- Interfacing with sensors requires a reasonable knowledge of signal conditioning techniques
- Interfacing with actuators requires a reasonable knowledge of power switching techniques

← بشكل عام بركز عموضوع ال Time وتطبيقات ال Timing.

Taking Timing Further

Chapter 9

Dr. Iyad Jafar

Outline

- Introduction
- Review of Timer 0 Module
- Timer 1 Module
- Timer 2 Module

فيهم ميزات اضافية

- Capture/Compare/PWM (CCP) → hardware Module
- Digital-to-Analog Conversion X
- Frequency Measurement X
- Summary

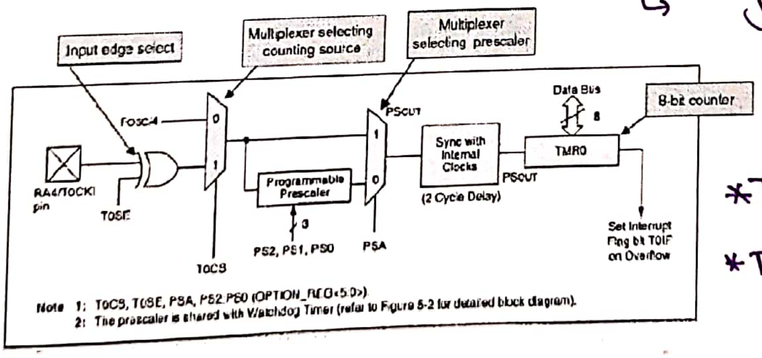
Introduction

- **Why do we need timers ?**
  - Maintaining continuous counting functions
  - Recording ('capturing') in timer hardware the time an event occurs
  - Triggering events at particular times
  - Generating repetitive time-based events
  - Measuring frequency, e.g., motor speed

* اذا بدي أريح ال cpu وما أحل المشكلة بال software بروح لل hardware.

Review of Timer 0 Module

شرحناه فيما قبل ب Slide 19 chapter 6



Note 1: T0CS, T0SE, PSA, PS2:PS0 (OPTION_REG<5:0>).
2: The prescaler is shared with Watchdog Timer (refer to Figure 5-2 for detailed block diagram).

$*Time = \#inc * \frac{4}{F_{osc}} * pre$

$*Time_{max} = 2^8 * \frac{4}{F_{osc}} * 256$

للتحكم بال Timer0

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
$\overline{RBPU}$	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

Address			File Address
00h	indirect addr.[1]	indirect addr.[1]	80h
01h	TMR0	OPTION_REG	81h
02h	PCL	PCL	82h
03h	STATUS	STATUS	83h
04h	FSR	FSR	84h
05h	PORTA	TRISA	85h
06h	PORTB	TRISB	86h

bit 7 $\overline{RBPU}$: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 INTEDG: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 T0CS: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 T0SE: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 PSA: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 PS2:PS0: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

More Timer Modules

Device	Pins	Features
16F84A	18	**1 8-bit timer** 1 5-bit port 1 8-bit port
16F873A 16F876A	28	3 parallel ports, **3 counter/timers,** **2** capture/compare/PWM, 2 serial, 5 10-bit ADC, 2 comparators
16F874A 16F877A	40	5 parallel ports, **3 counter/timers,** **2** capture/compare/PWM, 2 serial, 8 10-bit ADC, 2 comparators

يعتمد عوجود Timer1, Timer2

Timer 1 Module

Timer0 نفس مبدأ ← باختلافين :

- **Features**
  - 16-bit timer/counter (0000H – FFFFH)
  - Count value in TMR1H (0x0F) and TMR1L (0x0E)
  - TMR1 operation controlled by T1CON (0x10)
  - Three clock sources

① الاختلاف إنه 16-bit مش 8-bit

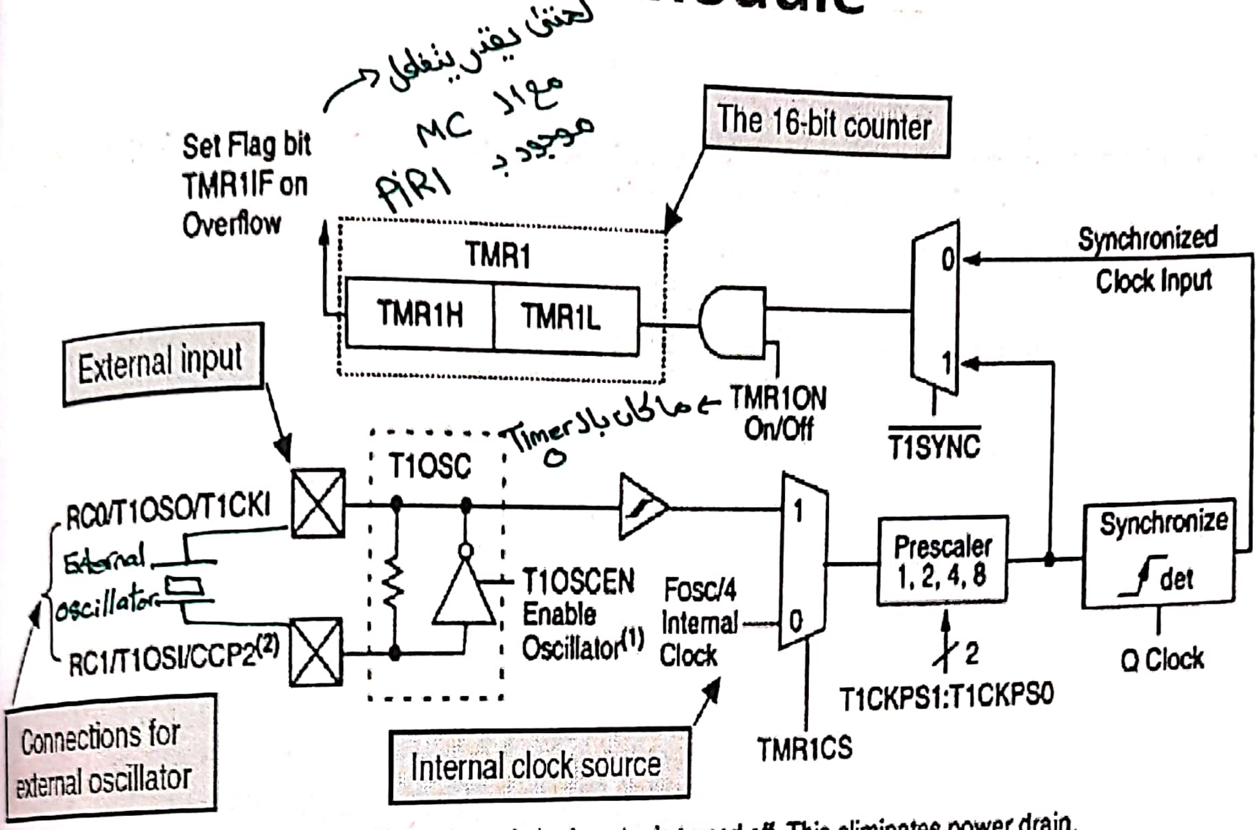
2 reg ← لأنه ال count value عبارة عن 16-bit

① Internal clock Fosc/4

② External input (RC0/T1OSO/T1CKI) for counting purposes
  - Count on rising edge (after the first falling edge)

③ External oscillator (RC1/T1OSI/CCP2) → Timer0 ماكان موجود ب ②
  - Removes the dependency on the main oscillator
  - Intended for low frequency oscillation up to 200KHz (typically 32.768 KHz)
  - Counting continue in sleep mode

Timer 1 Module



Note 1: When the T1OSCEN bit is cleared, the inverter is turned off. This eliminates power drain.

$* \ Time = \#inc * \frac{4}{Fosc} * pre \rightarrow Time_{max} = 2^{16} * \frac{4}{Fosc} * 8$

Timer 1 Module

T1CON Register (0x10)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{T1SYNC}$	TMR1CS	TMR1ON
bit 7							bit 0

بختاروا واحد من ال 3 خيارات لل Clock Sources

bit 7-6 **Unimplemented:** Read as '0'

bit 5-4 **T1CKPS1:T1CKPS0:** Timer1 Input Clock Prescale Select bits

11 = 1:8 Prescale value
10 = 1:4 Prescale value
01 = 1:2 Prescale value
00 = 1:1 Prescale value

bit 3 **T1OSCEN:** Timer1 Oscillator Enable bit

1 = Oscillator is enabled
0 = Oscillator is shut off. The oscillator inverter and feedback resistor are turned off to eliminate power drain

bit 2 **$\overline{T1SYNC}$:** Timer1 External Clock Input Synchronization Select bit

When TMR1CS = 1:
1 = Do not synchronize external clock input
0 = Synchronize external clock input

When TMR1CS = 0:
This bit is ignored. Timer1 uses the internal clock when TMR1CS = 0.

bit 1 **TMR1CS:** Timer1 Clock Source Select bit

1 = External clock from pin T1OSO/T1CKI (on the rising edge)
0 = Internal clock (Fosc/4)

bit 0 **TMR1ON:** Timer1 On bit

1 = Enables Timer1
0 = Stops Timer1

Timer 2 Module

- **Features**
  - 8-bit ~~counter~~/timer

    فعليا هو Timer فقط

  - Count value in TMR2 register (0x11)
  - TMR2 operation controlled by T2CON (0x12)
  - No external clock input
  - Has Capture and Compare register PR2 (0x92) and pulse width modulation capability

    هذا اللي يميزه عن باقي ال Timers

Timer 2 Module

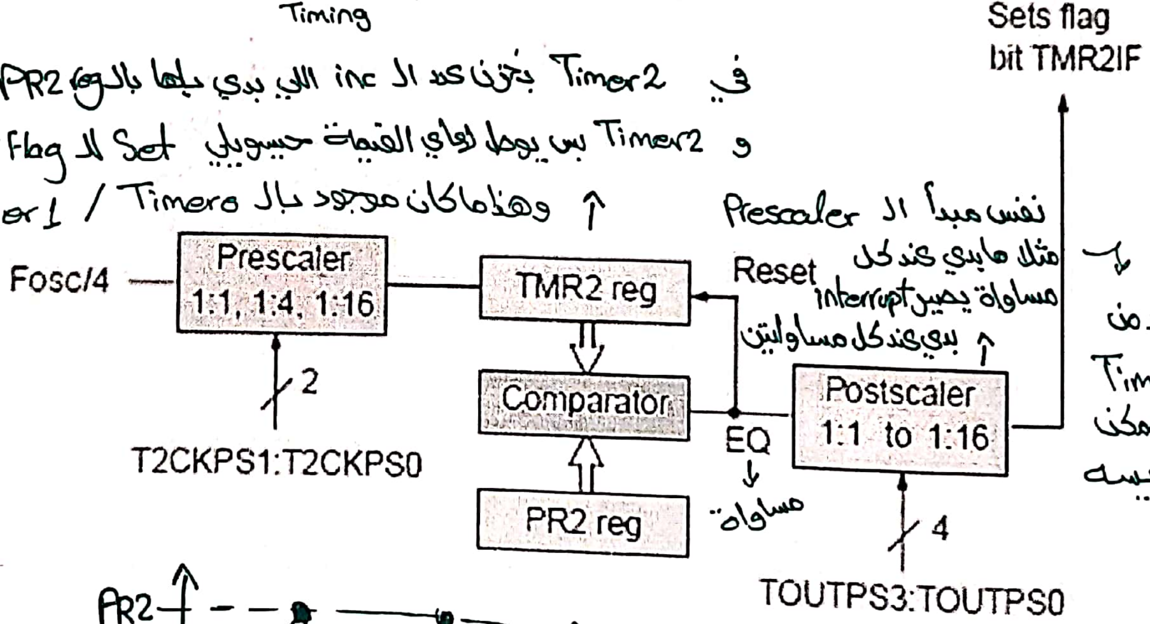
* Time = $(PR2 + 1) * \frac{4}{Fosc} * pre * post$

To reset Timing

في Timer2 بخزن عدد ال inc اللي بدي ياها بالـ PR2

و Timer2 بس يوصل لهاي القيمة حيسويلي Set لل Interrupt Flag

وهذا ما كان موجود بال Timer0 / Timer1



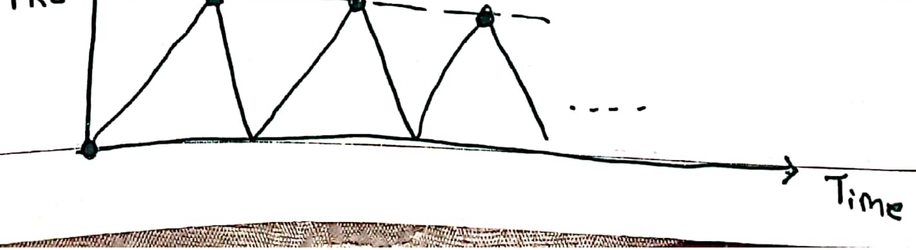
EQ → مساواة

نفس مبدأ ال Prescaler

مثلا ما بدي عند كل مساواة يصير interrupt

بدي عند كل مساواتين

بزيد من ال Time اللي ممكن اقيسه



Timer 2 Module

T2CON Register (0x12)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
bit 7							bit 0

bit 7 **Unimplemented:** Read as '0'

bit 6:3 **TOUTPS3:TOUTPS0:** Timer2 Output Postscale Select bits

0000 = 1:1 Postscale
0001 = 1:2 Postscale
•
•
•
1111 = 1:16 Postscale

bit 2 **TMR2ON:** Timer2 On bit

1 = Timer2 is on
0 = Timer2 is off

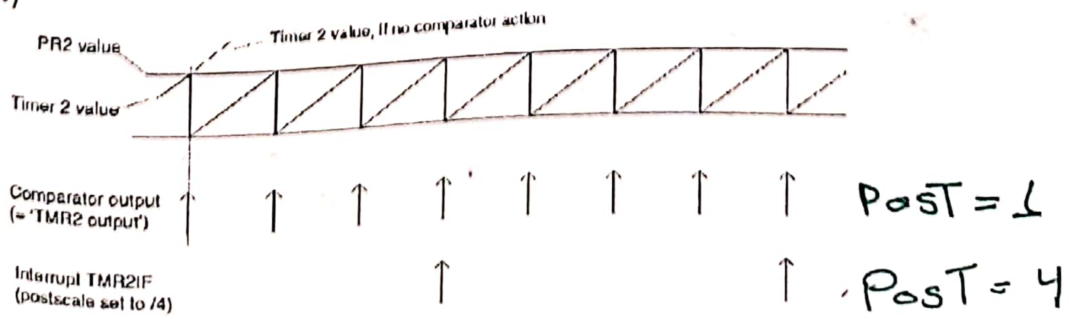
bit 1:0 **T2CKPS1:T2CKPS0:** Timer2 Clock Prescale Select bits

00 = Prescaler is 1
01 = Prescaler is 4
1x = Prescaler is 16

Timer 2 Module

The PR2 register, comparator and prescaler

- Timer2 has a period register PR2 (0x92) that can be preset by the programmer
- The content of this register is continuously compared with the Timer2 when it is running
- When TMR2 equals PR2,
  - TMR2 is cleared
  - The comparator output (same as TMR2IF in PIR) is high which can be used as interrupt if TMR2IE (PIE) is set
  - The comparator output can be post-scaled by T2OUTPS3:T2OUTPS0 bits (T2CON)



hardware component بيقدملي Timing operations إلها علاقة بـ Timer1 / Timer2 ←

Capture/Compare/PWM Modules

- Embedded systems need to deal with time events such as setting an **alarm** or **recording** the time of an event
- This can be easily achieved by adding one or more registers to the timer/counter registers
  - A register that records the time. It is called the Capture register
  - A register that triggers an alarm. It is called the Compare register
- The PIC 16 series combine these functionalities in the Capture/Compare/PWM (CCP) modules which interact with Timer1 and Timer2 modules

CCP Mode	Timer Resource
Capture	Timer1
Compare	Timer1
PWM	Timer2

Pin RC2 ← CCP1]
Pin RC1 ← CCP2] → طريقة عملهم نفس الاشي

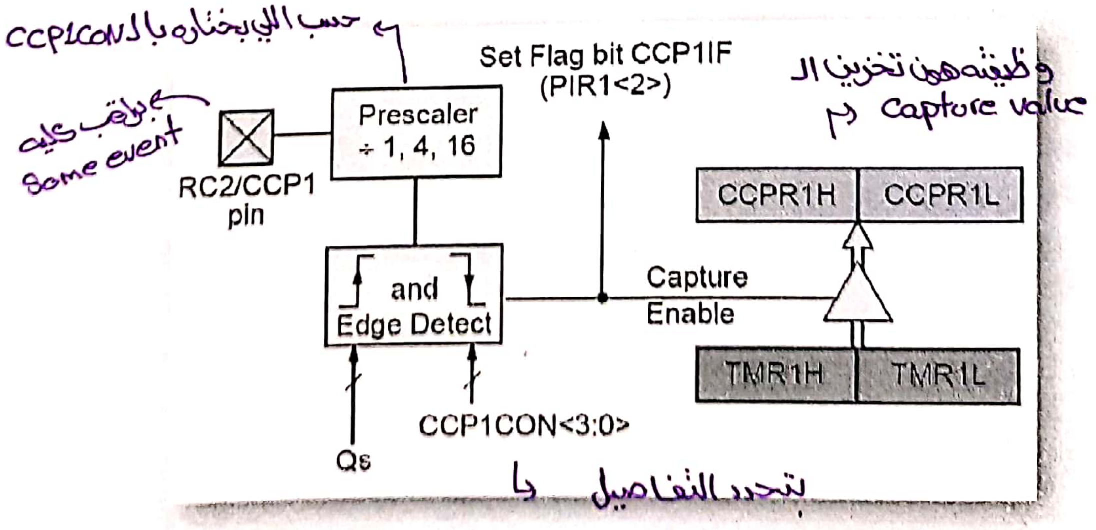
- **The PIC16F873A has two such modules**
  - Each has two 8-bit registers CCP1H (0x16) and CCP1L (0x15) for module CCP1 and CCP2H (0x1C) and CCP2L (0x1B) for module CCP2 → ccp وظيفتهم معرفة الوظيفة لل
  - These registers can be used to capture a value from the timer, store the value to compare with, or store the duty cycle of PWM stream
  - Mode of operation is controlled by CCP1CON (0x17) and CCP2CON (0x1D) registers

---

Capture/Compare/PWM Modules

Capture Mode

- The compare register operates like a stopwatch!
- Can record the value of the timer when an event occurs

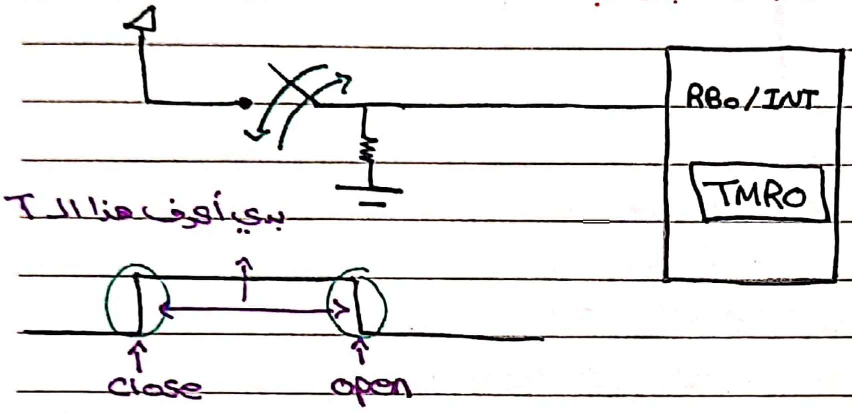


Block diagram of CCP1 module in capture mode

Chapter 9

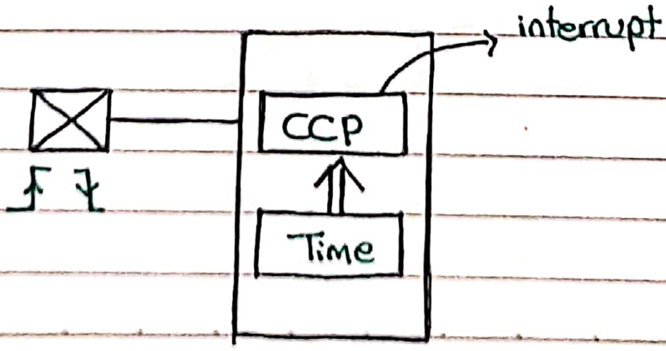
Slide 13 8

* لوحكينا إنه بطور Embedded system بدي أعرف لما واحد بيسكر ال Switch ويفتحه قديه مدة إغلاق ال Switch (من اللحظة اللي بسوي فيها Close للحظة اللي بفتح فيها ال Switch) مثل في بعض الأجهزة بقلك اضغط لمدة كذا ثانية عشان يعمل هيك، هكذا.

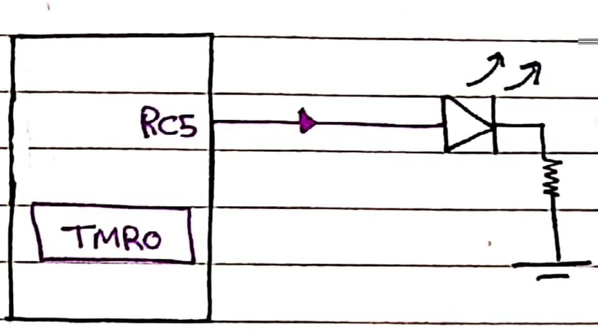


* الحل 8 ممكن أكتب برنامج يحكي إنه خذ ال Switch ودخله على RB0 وفعّل ال External interrupt وسويله config بحيث يسوي interrupt على ال rising edge وال Falling edge ويجيب Timer0 وكل ما صار عندي interrupt سواء rising أو falling edge أروح أسوي capture لل Time وأقرأ ال T عند ال rising وال falling ثم أطرحهم من بعض لمعرفة ال Time اللي بدي إياه . $T = T_1 - T_2$
وهذا العمل بيسمى T capturing

* لكن لما يكون ال System كبير وفيه كثير operations ، إني أحجز ال MC أو أخصص جزء من ال MC لعمل هاي العمليات ممكن يكون مش مفيد وبنفس الوقت كتابة الكود لهاي العملية ممكن يصعب علي أنا ك Programmer ، عشان هيك بحطوا hardware مميز بيسموه CCP فعملية ال capturing بتصير منفصلة تماماً عن ال cpu وبهيك بريح ال cpu . والمطلوب من هذا ال hardware كلما يعمل capture ينبه ال cpu فأكيد في interrupt معاه :



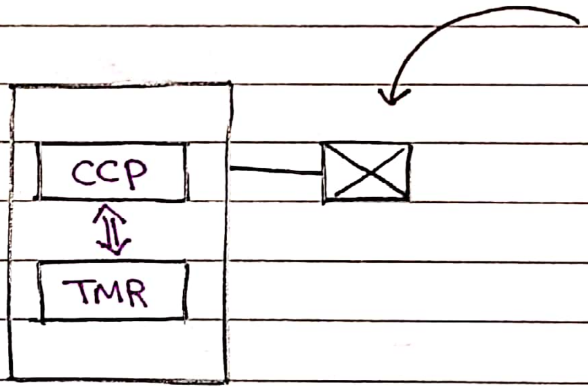
* العملية الثانية اللي بتقدموا هاي الـhardware هي الـ Compare والـ Compare بشبه وظيفة الـ Alarm.



بحكي مثلاً من اللحظة الحالية لغاية بعد 3 ثواني بدي أشغل الـ LED قبل هيك الـ LED طافي.

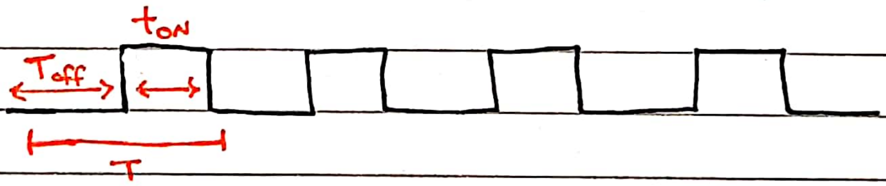
*بوصل ك output عالـ RC5 وبحط TMR0 و بحكي من اللحظة اللي بشغل فيها TMR0 كم لازم يعد الـ TMR0 لحتى أعرف إنو صار مارق 3 ثواني وبظل أقارن لحد ما يصير 3sec بعدين بطلع الـ output = 1 لحتى أضوي الـ LED.

* هنا الكلام: باستخدام الـ ccp hardware بقدر استخدمها ك Compare بإني أشغل الـ Timer و بسوي config لهاي الـ module، إنها تضلها تسوي مقارنة بين الـ Timer و Value أنا مخزنها جواها وبالوقت اللي بتساووا الثنين مع بعض بعمل شغلة معينة:



* العملية الثالثة اللي بنفذوها هاي الـ hardware هي: PWM (pulse-width-modulation)

* أنا بحتاج إني أطلع periodic squarwave بـ Frequency معين وبنفس الوقت بحتاج إني أتحكم بالـ pulse width (ton):

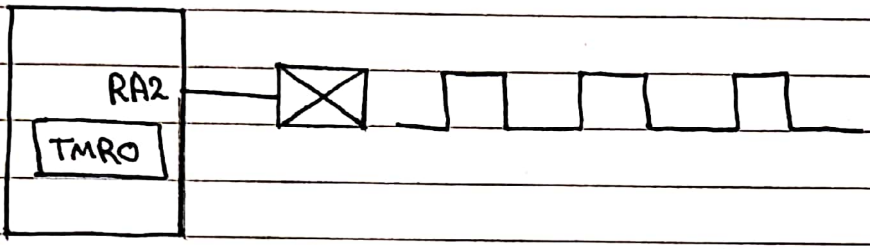


* بالـ ton نسبتها ومقدارها بحدد الزاوية اللي الـ Motor حيروح عليها.

* وهنا النوع لما أشبكه ك motor لازم يضل مستمر وأظل أعطيه periodic squarewave

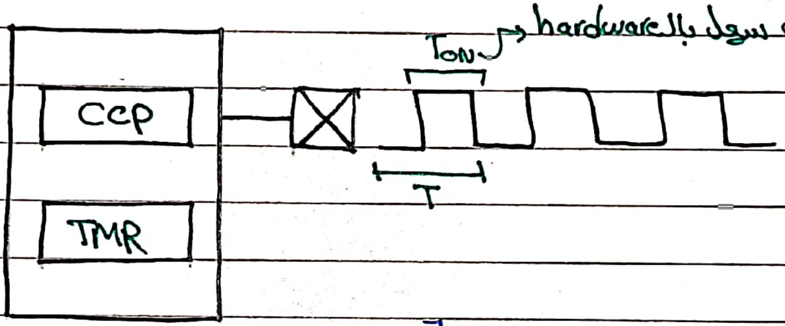
PAGE
DATE

هذا الكلام بقدر أسويه باستخدام ال cpu بإني أكتب كود بيسوي config على TMR0 بحيث إنه يعد ال off Time ولما يخلص بعمله config يعد ال ON Time وبسوي output وأكرر هالعملية :



*ولكن عشان ما أسوي load على ال cpu بقدر أعمل هاي العملية باستخدام hardware ccp
بحيث بسويلها config عشان تطلعلي squarewave Periodic ب Frequency و T معينة :
ON ↓ T

بقدر أغيره بشكل سهل بال hardware ←

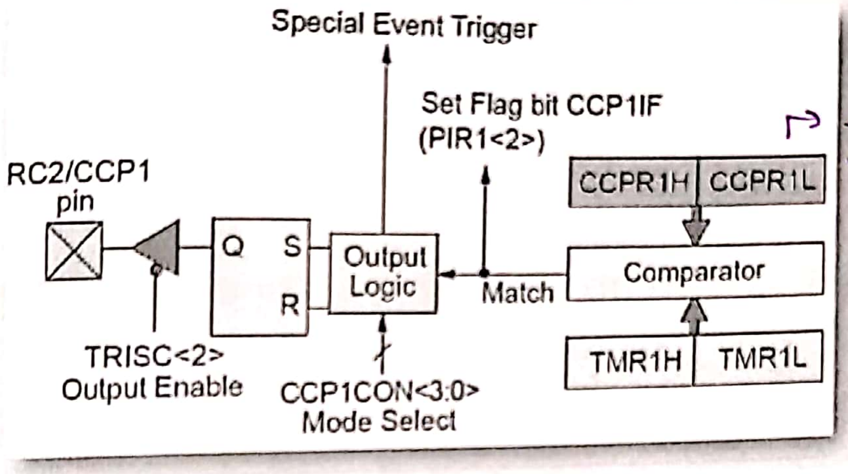


* duty cycle = $\frac{t_{ON}}{T} * 100\%$] → نسبة ال tON

Capture/Compare/PWM Modules

Compare Mode

- The value stored in CCPR1H and CCPR1L is continuously compared to Timer1 registers
- The associated output pin can be set or cleared



← هنا وظيفته يخزن ال Value اللي بدي أقارن فيها

Block diagram of CCP1 module in compare mode

Capture/Compare/PWM Modules

CCP Control Registers: CCP1CON and CCP2CON

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
—	—	DCxB1	DCxB0	CCPxM3	CCPxM2	CCPxM1	CCPxM0
bit 7							bit 0

↓ يعني 1 أو 2

← بحددوا ال mode اللي حستخدمه

bit 7:6 **Unimplemented:** Read as '0'

bit 5:4 **DCxB1:DCxB0:** PWM Duty Cycle bit1 and bit0

<u>Capture Mode:</u>
Unused

<u>Compare Mode:</u>
Unused

<u>PWM Mode:</u>
These bits are the two LSbs (bit1 and bit0) of the 10-bit PWM duty cycle. The upper eight bits (DCx9:DCx2) of the duty cycle are found in CCPRxL.

bit 3:0 **CCPxM3:CCPxM0:** CCPx Mode Select bits

0000 = Capture/Compare/PWM off (resets CCPx module) → default
0100 = Capture mode, every falling edge
0101 = Capture mode, every rising edge
0110 = Capture mode, every 4th rising edge
0111 = Capture mode, every 16th rising edge
1000 = Compare mode, Initialize CCP pin Low, on compare match force CCP pin High (CCPIF bit is set)

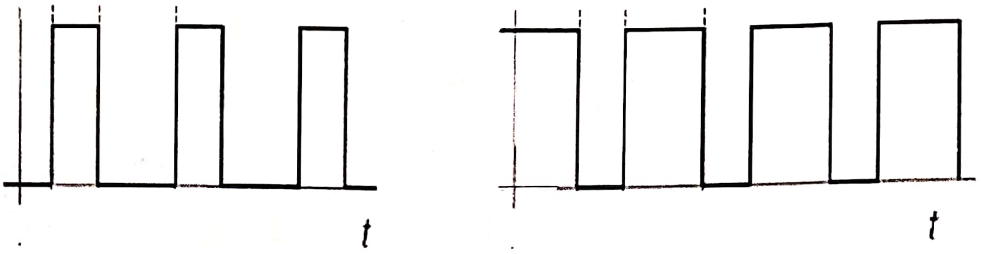
↑

1001 = Compare mode, Initialize CCP pin High, on compare match force CCP pin Low (CCPIF bit is set) → عكس هاي
1010 = Compare mode, Generate software interrupt on compare match (CCPIF bit is set, CCP pin is unaffected) → بس بتعمل Set
1011 = Compare mode, Trigger special event (CCPIF bit is set)
11xx = PWM mode

Capture/Compare/PWM Modules

Pulse Width Modulation

- In many applications, it is required to have a stream of pulses with controllable width/duration

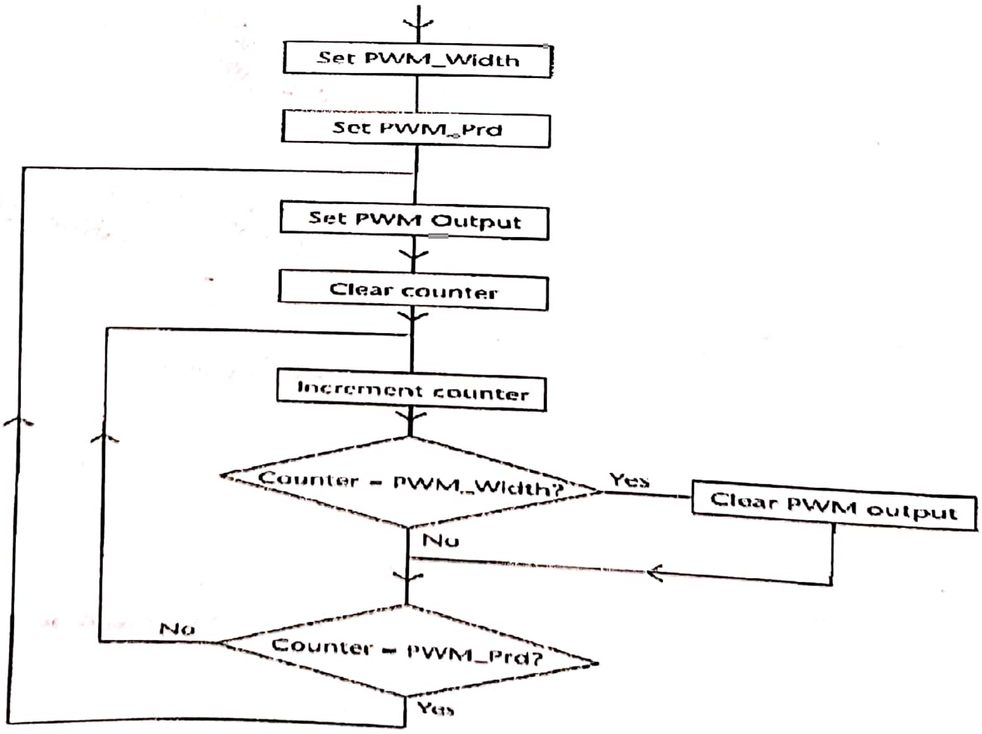


- In embedded systems this can be done in software or hardware

---

Capture/Compare/PWM Modules

Pulse Width Modulation (software)



Set PWM_Width

Set PWM_Prd

Set PWM Output

Clear counter

Increment counter

Counter = PWM_Width?

Yes

Clear PWM output

No

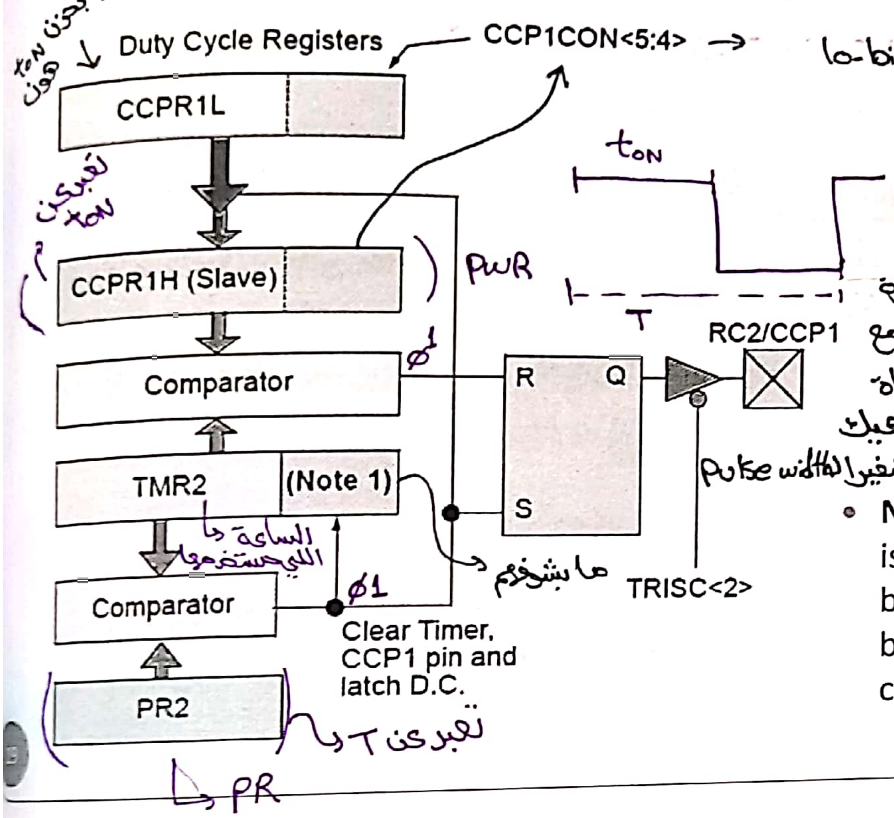
Counter = PWM_Prd?

No

Yes

Capture/Compare/PWM Modules

Pulse Width Modulation (using CCP module)

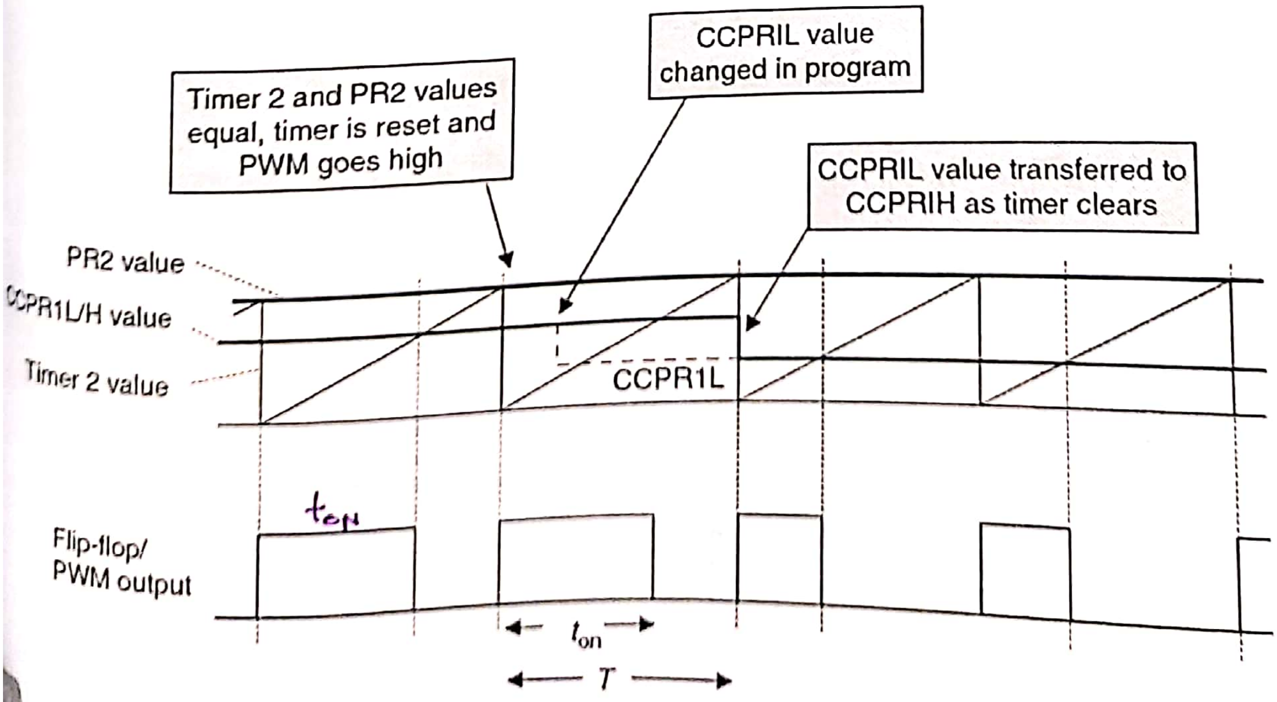


* لو بدي أخزن ton عال reg CCPR1H ما بقدر أحكي معاه مباشرة واحنا شغالين بال PWM mode بقدر أحكي مع ال CCPR1L ولما يصير في مساواة بتنتقل القيمة لل CCPR1H عشان هيك ما حينقلوا قيمة ال ton وما حيتغير ال Pulse width

- **Note 1:** The 8-bit timer is concatenated with 2-bit internal Q clock, or 2 bits of the prescaler, to create 10-bit time base.

Capture/Compare/PWM Modules

Pulse Width Modulation (using CCP module)



Capture/Compare/PWM Modules

Pulse Width Modulation (using CCP module)

Calculations

Period ← $T = (\mathbf{PR2} + 1) \times (Timer2\ input\ clock\ period)$

$$= (\mathbf{PR2} + 1) \times \{\underbrace{T_{osc} \times 4}_{\text{or } \frac{4}{F_{osc}}} \times (Timer2\ prescale\ value)\} * \cancel{Post}$$

لما يشتغل مع TMR2 و PWM مش موجود فبمحيه .

t_{on} = (pulse width register) × (PWM timer input clock period)

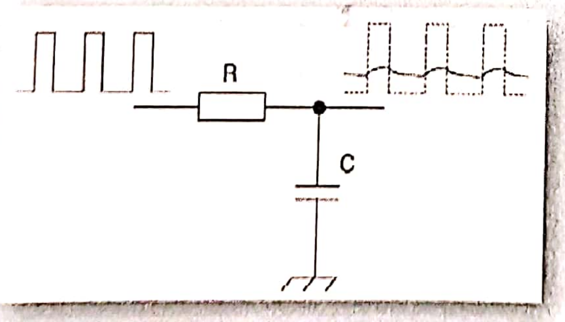
= (pulse width register) × {T_{osc} × (Timer2 prescale value)}

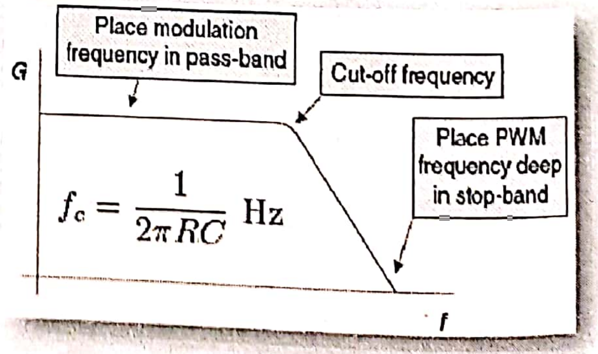
pulse width register = **CCPR1L :: CCP1CON<5:4>**

:: → Concatenate

PWM and Digital To Analog Conversion

- PWM is perhaps primarily used for load control
- Can be used for simple and effective digital-to-analog conversion
  - Space-ratio is fixed
    - Low pass filter the PWM stream to obtain a DC signal with some ripple





- Space-ratio is modulated
  - Varying output voltage is produced

PWM and Digital To Analog Conversion

X

- Generating a Sine Wave - change the on-time for the PWM signal so the output of the LPF will be different

```
        clrf    pointer
sin_loop
        movf    pointer,w
        call    sin_table    ;get most significant byte
        movwf   ccpr1l       ;move it to the PWM output
        incf    pointer,f    ;increment the pointer
        movf    pointer,w
        call    sin_table     ;get the MS byte
        andlw   B'11000000'  ;we only use ms 2 bits
```

PWM and Digital To Analog Conversion

X

- Generating a Sine Wave

```
        movwf   temp
        bcf     status,c      ;adjust for CCP1CON
        rrf     temp,f
        rrf     temp,w
        iorlw   B'00001100'  ;set some CCP1CON bits
        movwf   ccp1con
        incf    pointer,f
        movf    pointer,w
        …
        call    delay1
        goto    sin_loop
```

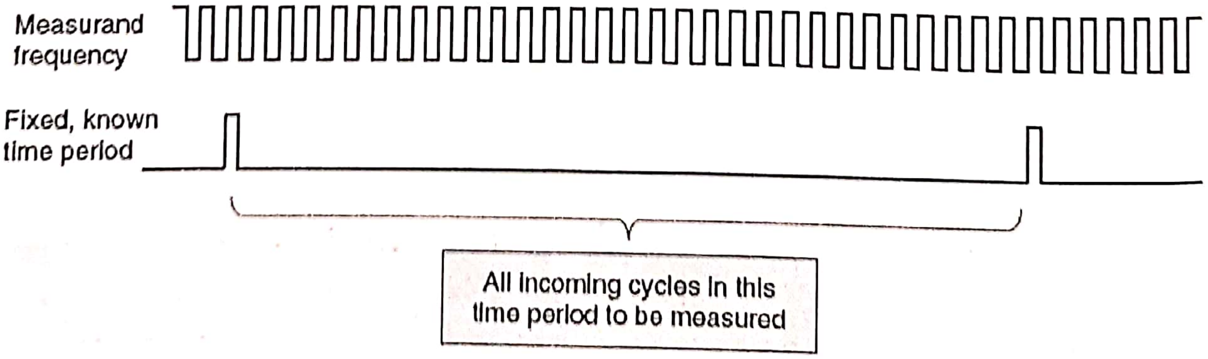
PWM and Digital To Analog Conversion

- Generating a Sine Wave

```
Sin_Table
        addwf pcl,1
        retlw 00      ;0 degrees, higher byte
        retlw 00      ;0 degrees, lower byte
        retlw 03      ;2 degrees, higher byte
        retlw 5A      ;2 degrees, lower byte
        retlw 06      ;4 degrees, higher byte
        retlw 0B2     ;4 degrees, lower byte
        ......
        ......
```

Frequency Measurement

- Frequency measurement is a very important application of both counting and timing
- Both a counter and a timer are needed
  - The timer to measure the reference period of time
  - The counter to count the number of events within that time.



Example 1

Write a program to flash a LED that is connected to RA0 continuously such that it is ON for 3 seconds and OFF for 3 seconds. Use TIMER1 module to generate the delay and assume Fosc = 4MHz.

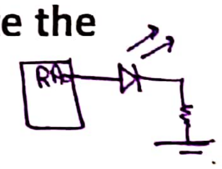


TABLE 6-2: REGISTERS ASSOCIATED WITH TIMER1 AS A TIMER/COUNTER

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF[1]	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
8Ch	PIE1	PSPIE[1]	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
0Eh	TMR1L	Holding Register for the Least Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
0Fh	TMR1H	Holding Register for the Most Significant Byte of the 16-bit TMR1 Register								xxxx xxxx	uuuu uuuu
10h	T1CON	—	—	T1CKPS1	T1CKPS0	T1OSCEN	$\overline{\text{T1SYNC}}$	TMR1CS	TMR1ON	--00 0000	--uu uuuu

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by the Timer1 module.

Example 1

- Maximum time that can be measured by TMR1 is

  Time = 2^16 * 4/Fosc * Prescaler * Post

  ↳ بال software
  مش بال hardware

  = 65536*1usec*8 = 0.5243 s

- How about we configure TMR1 to measure 0.5 sec and use a software counter (post-scaler) to count six times

  0.5 = N * 1 usec * P → N = 62500, P = 8

TMR1H:TMR1L = 65536 – 62500 = 3036 = 0x0B|DC

→ TMR1H = 0x0B ; TMR1L = 0xDC

- T1CON = 0x30

--	--	T1CKPS1	T1CKPS0	T1OSCEN	T1SYNC	TMR1CS	T1ON
0	0	1	1	0	0	0	0

↳ pre = 8

↳ Fosc/4 ←

Example 1

```
COUNTER      EQU        0x20
             # include  "PIC16F877.INC"
             org        0x0000
             goto       START
             org        0x0004
ISR          goto       ISR
START        bcf        STATUS, RP1        ; select bank 1
             bsf        STATUS, RP0
             clrf       TRISA              ; set RA0 as output
             movlw      B'00000110'        ; configure RA0 as digital
             movwf      ADCON1
             bcf        STATUS, RP0
FLASH        movlw      0x06               ;  initialize counter to 6
             movwf      COUNTER
WAIT_3sec    movlw      0x0B
             movwf      TMR1H              ; initialize TMR1H
```

← أحول ال Pin ل digital

Example 1

```
             movlw      0xDC
             movwf      TMR1L              ; initialize TMR1L
             movlw      0x30
             movwf      T1CON              ; initialize T1CON
             bsf        T1CON, TMR1ON      ; enable timer 1
WAIT_p5sec   btfss      PIR1, TMR1IF       ; wait for overflow
             goto       WAIT_p5sec
             bcf        T1CON, TMR1ON      ; stop timer
             bcf        PIR1, TMR1IF       ; clear interrupt flag
             decfsz     COUNTER, F
             goto       WAIT_3sec
             movlw      0xFF               ; change the state of RA0
             xorwf      PORTA, F
             goto       FLASH
             end
```

0.5 sec

Example 2

Consider the contents of the following registers

TMR2 = D'44'

PR2 = D'100'

T2CON = 0x39

If the instruction *bsf T2CON, T2ON* is executed, then how long does it take to set the TMR2IF in the PIR1 register ? Assume Fosc = 8 MHz.

$$\text{Time} = (PR2+1) * \frac{4}{Fosc} * \underbrace{pre * Post}_{T2CON}$$

بس بدي أنبه إنو ما حكالي انه Timer2 بيبلش كد من ال 44 دايمًا

Example 2

8

T2CON

--	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	T2ON	T2CKPS1	T2CKPS0
0	0	1	1	1	0	0	1

4

- Initially, the timer is off
- Executing the instruction enables the timer
- The time required to set the TMR2IF is

  Time = (PR2+1) * prescaler * postscaler * 4 /Fosc

  <u>if TMR2 is initialized to zero</u>.
- However, TMR2 = 44. So the time is

  ع أول مرة

  Time = (PR2-TMR2+1) * 4 * 4/8MHz + (PR2+1) * 4 * 4/8MHz * 7 = 1528 usec

  7 مش 8 لأنه أول مرة كانت كيركذم

Example 3

Write a program that configures and uses the CCP1 module in PIC16F873A to generate a periodic square wave of frequency 50 Hz and 25% duty cycle. Assume that Fosc = 800 KHz.

Requirements

لأنه بستخدم
مع CCP1

1) Configure RC2 as output
2) Configure TIMER2 module and compute the values to be placed in CCPR1L and PR2 registers which determine the duty cycle and the cycle time, respectively
3) Turn on the timer

Example 3

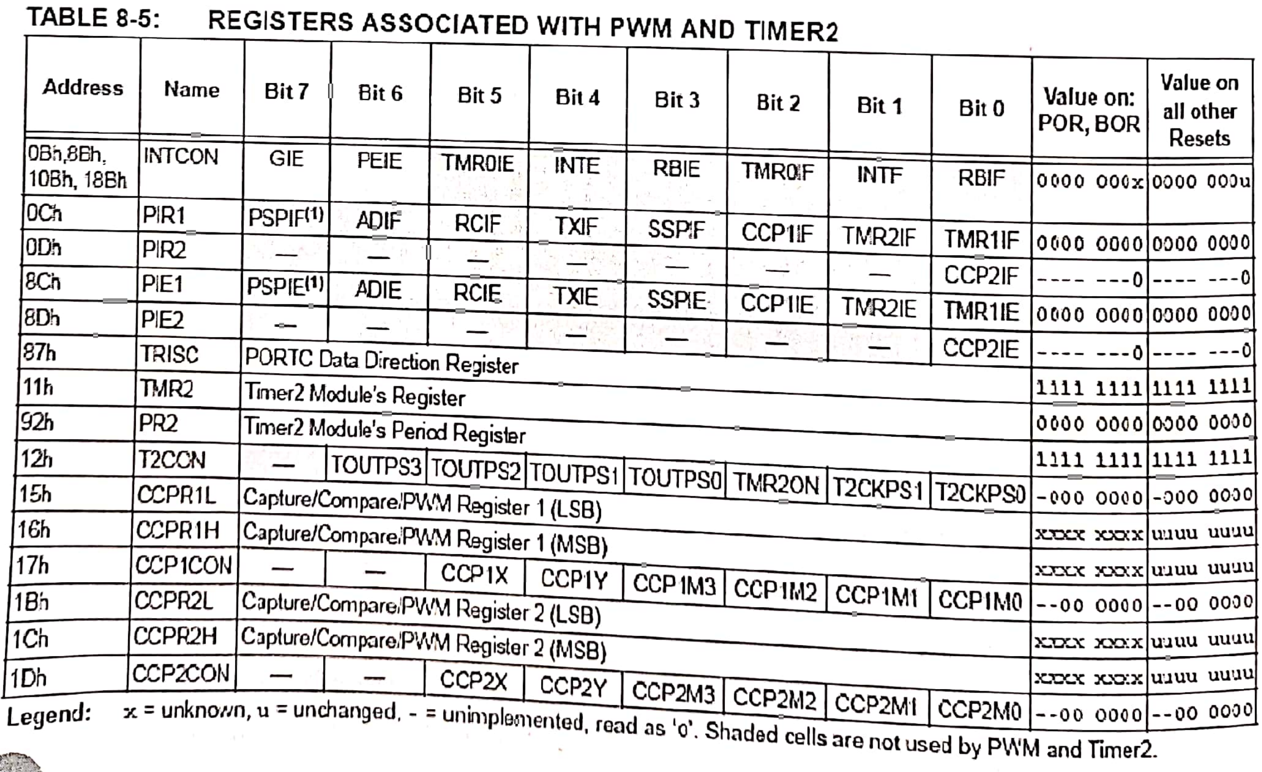


TABLE 8-5: REGISTERS ASSOCIATED WITH PWM AND TIMER2

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on: POR, BOR	Value on all other Resets
0Bh,8Bh, 10Bh, 18Bh	INTCON	GIE	PEIE	TMR0IE	INTE	RBIE	TMR0IF	INTF	RBIF	0000 000x	0000 000u
0Ch	PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
0Dh	PIR2	—	—	—	—	—	—	—	CCP2IF	---- ---0	---- ---0
8Ch	PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
8Dh	PIE2	—	—	—	—	—	—	—	CCP2IE	---- ---0	---- ---0
87h	TRISC	PORTC Data Direction Register								1111 1111	1111 1111
11h	TMR2	Timer2 Module's Register								0000 0000	0000 0000
92h	PR2	Timer2 Module's Period Register								1111 1111	1111 1111
12h	T2CON	—	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0	-000 0000	-000 0000
15h	CCPR1L	Capture/Compare/PWM Register 1 (LSB)								xxxx xxxx	uuuu uuuu
16h	CCPR1H	Capture/Compare/PWM Register 1 (MSB)								xxxx xxxx	uuuu uuuu
17h	CCP1CON	—	—	CCP1X	CCP1Y	CCP1M3	CCP1M2	CCP1M1	CCP1M0	--00 0000	--00 0000
1Bh	CCPR2L	Capture/Compare/PWM Register 2 (LSB)								xxxx xxxx	uuuu uuuu
1Ch	CCPR2H	Capture/Compare/PWM Register 2 (MSB)								xxxx xxxx	uuuu uuuu
1Dh	CCP2CON	—	—	CCP2X	CCP2Y	CCP2M3	CCP2M2	CCP2M1	CCP2M0	--00 0000	--00 0000

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0'. Shaded cells are not used by PWM and Timer2.

Example 3

- PWM signal specs
  - T = 1 /50 = 0.02 sec
  - Ton = 0.25 * T = 0.005 sec
- Need to configure the CCP1 and TIMER2
  - PR2 register

    T = (PR2+1) * 4 * Tosc * prescaler

    if we assume prescaler = 16, then PR2 = 249
  - Pulse-width register CCPR1L:CCP1CON<5:4>

    نعوض .Ton = PWR * Tosc * prescaler

    بعوضوا ب

    already the prescaler is chosen to be 16 → PWR = 250 =0xFA

    → CCPR1L = B'00111110' and CCP1CON<5:4> = B'10'
  - T2CON = 0x06
  - CCP1CON = B'00101100'

↓

00 1111 10 10

CCPR1L CCP1CON <5:4>

Example 3

```
        # include    "PIC16F877.INC"
        org       0x0000
        goto      START
        org       0x0004
SR      goto      ISR
TART    bcf       STATUS, RP1       ; select bank 1
        bsf       STATUS, RP0
        bcf       TRISC, 2          ; set RC2 as output
        movlw     D'249'
        movwf     PR2               ; set the cycle time in PR2
        bcf       STATUS, RP0
        movlw     0x3E
        movwf     CCPR1L            ; set the ON time in CCPR1L
        bcf       CCP1CON, 4        ; specify the LSBs of the ON time
        bsf       CCP1CON, 5
```

Example 3

```
        bsf     CCP1CON, 3
        bsf     CCP1CON, 2  ; configure  CCP1 in PWM and
        movlw   0x06
        movwf   T2CON       ; configure timer 2 and enable it


DONE    goto    DONE
        end
```

لإنه مافي اشي ثاني أسويه.

Summary

- Timing is essential element of embedded systems design
- Wide range of timers is available in PIC microcontrollers with clever add-on features such as capture, compare, and pulse width modulation
- It is very occasional to have several timers running simultaneously in an embedded system