

# The PIC Microcontroller

- PIC16f84A 8-bit enhanced with EEPROM



بمدينة المتكبرات

الدقيقة بلغة التجميع

Programming PIC Microcontroller with Assembly language

Include

- Introduction to Digital System



معهد التدريب النفطي / بيجي

المهندس

بسام أحمد صالح

Email: [CET.ENG2012@yahoo.com](mailto:CET.ENG2012@yahoo.com)

﴿ وَاللَّهُ أَخْرَجَكُمْ مِنْ بُطُونِ أُمَّهَاتِكُمْ لَا تَعْلَمُونَ شَيْئًا وَجَعَلَ لَكُمُ السَّمْعَ وَالْأَبْصَارَ وَالْأَفْئِدَةَ لَعَلَّكُمْ تَشْكُرُونَ ﴾

الحمدُ لله رب العالمين ، وأشهد أن لا إله إلا الله وحده لا شريك له ، وأشهد أن محمداً عبده ورسوله وصفيته وخليفه ، صلوات ربي وسلامه عليه وعلى آله وأصحابه أجمعين ، والتابعين ومن تبعهم بإحسان إلى يوم الدين . .

أما بعد :

نعيشُ اليوم في عصر تكنولوجيا المعلومات ، حيثُ يشهد عَصْرنا هَذَا تطوراً سريعاً لاسيما فيما يخصُ مجال تقنيات معالجة البيانات وكثرة الأجهزة التي تم اختراعها في هذا المجال مثل **Microprocessor , PLC , Microcontroller** ، فالقرن العشرين هو بلا شكَّ قرن الإلكتروني والصناعات الحاسوبية وبخاصة في العقود الأربعة الأخيرة ، فاختراع الترانزيستور شكّل قفزة نوعية في الصناعة الإلكترونية ، وغدا هذا العنصر فيما بعد عنصراً أساسياً في صناعة الدوائر الإلكترونية. نقطة التحول الثانية في الصناعة الإلكترونية بعد الترانزيستور تمثلت بنجاح العلماء في صناعة دوائر متكاملة (IC) متناهية الصغر ، وخصوصاً الاختراع المتحكم الدقيق **Microcontroller** والتي هي عبارة عن دائرة متكاملة وما يميز هذه الدائرة المتكاملة هي الوثوقية في الأداء إضافة إلى تطور تقنية تصنيعها و سعرها المنخفض الذي جعلها في متناول الجميع والمتحكم الدقيق يختلف عن بقية الدارات المتكاملة في أنه دائرة متكاملة قابلة للبرمجة ، أي أن عمله يتحدد وفق البرنامج المكتوب داخله ، وهو ذو وثوقية عالية ودقة متناهية في معالجة البيانات مما جعله العقل المدبّر في دارات التحكم الإلكترونية.

هذا الكتاب سيكون مدخلك نحو الفهم العميق لمبدء عمل المتحكم الدقيق وطريقة برمجة ، حيثُ يتكون هذا الكتاب من سبعة وحدات ، في الوحدة الاولى تناولنا موضوع الأنظمة العددية وكيفية تمثيلها ، اما الوحدة الثانية فكانت مقدمة عن بنية الانظمة القابلة للبرمجة ، وفي الوحدة الثالثة تناولنا البنية الصلبة للـ **Hardware** للمتحكم الصغري **Microcontroller** ، اما الوحدة الرابعة تناولت موضوع مسجل البيانات **File Register** ، وفي الوحدة الخامسة تم شرح طقم التعليمات **instruction set** الخاصة بالمتحكم الدقيق **Microcontroller** ، وفي الوحدة السادسة سوف نتعلم مبادئ البرمجة وطرق برمجة المتحكم الدقيق **Microcontroller** ، اما الوحدة الاخيرة فخصصتها عن الأدوات **Toolkit** والبرامج **Softwares** المستخدمة في عملية البرمجة .

المهندس

بسام أحمد صالح

البنية العلمية لمعهد نفط بيجي

# الفهرس

## المحتويات

٩	Numerical System الأنظمة العددية
٩	Decimal System النظام العشري
١٠	Binary system النظام الثنائي
١٠	System Conversion التحويل بين الأنظمة
١١	Binary to Decimal التحويل من النظام الثنائي إلى النظام العشري
١١	Decimal to Binary التحويل من النظام العشري إلى النظام الثنائي
١٣	Units of Measurement وحدات القياس
١٥	Singed and Unsigned Value الأعداد الموجبة والسالبة
١٥	Unsigned Value الأعداد بدون الإشارة
١٥	Signed value الأعداد بإشارة
١٦	Signe and Magnitude الإشارة والمقدار
١٧	1's Compliment المتمم الأول
١٩	2's Compliment المتمم الثاني
٢٠	Arithmetic Operations العمليات الحسابية على النظام الثنائي
٢٠	Addition Arithmetic Operation عملية الجمع
٢٢	Subtracting Using 2's Compliment الطرح باستخدام المتمم الثاني
٢٤	Carry State حالة المُحمل
٢٥	Number in Computer's Memories الأرقام داخل ذاكرة الحاسوب
٢٦	Overflow Condition حالة الطَّفْحان
٢٦	Examples أمثلة على الطَّفْحان
٢٨	Hex. Decimal النظام السادس عشر
٢٨	Hex.Decimal to Binary التحويل من النظام السادس عشر إلى النظام الثنائي وبالعكس
٣٠	Programmable Devices Archetector بنية الأنظمة القابلة للبرمجة
٣٠	Hard wire connection أنظمة الربط الصلب
٣٠	Programmable System الأنظمة القابلة للبرمجة
٣١	CPU Function وظيفية وحدة المعالجة المركزية
٣١	Fetching cycle دورة الجلب

٣١	.....	Execution cycle	دورة التنفيذ
٣٢	.....	Arithmetic and logic unit	وحدة الحساب والمنطق
٣٣	.....	Control unit	وحدة السيطرة
٣٦	.....	Memory	وحدة الذاكرة
٣٦	.....	Internal Structure of Memory	البنية الداخلية للذاكرة
٣٧	.....	Von-Neumann vs Harvard structure	بنية فون نيومان وبنية هارفارد
٣٨	.....	Input Unit	وحدة الإدخال
٣٨	.....	Output Unit	وحدة الإخراج
٤٠	.....	PIC16F84A	بنية المُسيطر الدقيق PIC16F84A
٤١	.....	Microcontroller ؟	ما هو المُسيطر الصغري ؟
٤٢	.....	Architecture of PIC16F84A	بنية المُسيطر الصغري PIC16F84A
٤٢	.....	Hardware	البنية الصلبة
٤٣	.....	Central Processing Unit (CPU)	وحدة المعالجة المركزية CPU
٤٣	.....	Direct Addressing	العنوانه المباشرة
٤٤	.....	Direct Addressing	العنوانه الغير المباشر
٤٥	.....	ALU (Arithmetic and logic unit)	وحدة الحساب والمنطق
٤٦	.....	Status Register	مسجل الحالة
٤٧	.....	Control Unit	وحدة السيطرة
٤٨	.....	Memory Organization	تنظيم الذاكرة
٤٨	.....	Data memory	ذاكرة البيانات
٤٩	.....	General Purpose Register(GPRs)	مسجلات الأغراض العامة
٤٩	.....	Special Purpose Register(SPRs)	مسجلات الأغراض الخاصة
٤٩	.....	Program Memory	ذاكرة البرنامج
٥٠	.....	Power Up Timer (PWRT)	مؤقت بداية اقلاع الطاقة
٥٠	.....	Oscillator Start Up Timer(OST)	مؤقت بداية اقلاع المذبذب
٥١	.....	Power On Reset (POR)	أعادة الأغلاق عند بداية تشغيل الطاقة
٥١	.....	Watchdog timer	مؤقت الحراسه
٥١	.....	Stack Memory	المكدس
٥١	.....	EEPROM Memory	الذاكرة EEPROM
٥١	.....	Timer	المؤقت
٥١	.....	Input/Output Unit	أطراف الإدخال والإخراج
٥٢	.....	PIC16F84A	الوصف الدقيق لأطراف المسيطر
٥٣	.....	Memory Clear	طرف تصفير الشريحة
٥٣	.....	Oscillator	المذبذب
٥٥	.....	File Register	السجل
٥٦	.....	General Purpose Register(GPRs)	مُسجلات الأغراض العامة

٥٦	.....	Special Purpose Register(SPRs)	مُسجلات الأغراض الخاصة
٥٧	.....	TRISA(Tri-State Buffer)	المُسجل
٥٧	.....	TRISB(Tri-State Buffer)	المُسجل
٥٧	.....	PORTA	المُسجل
٥٧	.....	PORTB	المُسجل
٥٨	.....	Status Register	مُسجل الحالة
٥٩	.....	Option Register	المُسجل
٦٢	.....	FSR Register (File Select Register)	مُسجل FSR
٦٢	.....	INDF Register (Indirect Addressing Register)	مُسجل INDF
٦٣	.....	TMRO Register (Timer Register)	مُسجل TMRO
٦٣	.....	Program Counter Latch low & Latch High	مُسجل PCL & PCLATH
٦٤	.....	INTCON(Interrupt Controller Register)	مُسجل INTCON
٦٥	.....	EEADR(EEPROM Address Register)	مُسجل EEADR
٦٥	.....	EEDATA(EEPROM Data Register)	مُسجل EEDATA
٦٥	.....	EECON(EEPROM Controller Register)	مُسجل EECON1
٦٦	.....	EECON(EEPROM Controller Register)	مُسجل EECON 2
٦٧	.....	Instruction Set	طَقَم التَّعليمات
٦٨	.....	Bit Orientation Operations	التَّعليمات على مُستوى البت
٦٨	.....	Bit Clear Flag	التَّعليمية BCF
٦٩	.....	Bit Set Flag	التَّعليمية BSF
٧٠	.....	Bit Test Flag Skip If Clear	التَّعليمية BTFSF
٧١	.....	Lateral and Control Instruction	تعليمات السيطرة والثوابت
٧١	.....	MOV Lateral to W Reg.	التَّعليمية MOVLW
٧١	.....	ADD Lateral to W Reg.	التَّعليمية ADDLW
٧٢	.....	AND Lateral and W Reg.	التَّعليمية ANDLW
٧٢	.....	CALL Statement	التَّعليمية CALL
٧٢	.....	Clear Watchdog Timer	التَّعليمية CLRWDT
٧٣	.....	Goto Statement	التَّعليمية Goto
٧٣	.....	Exclusive OR Lateral With W Reg.	التَّعليمية XORLW
٧٣	.....	Inclusive OR Lateral With W Reg.	التَّعليمية IORLW
٧٤	.....	Subtract Lateral from W Reg.	التَّعليمية SUBLW
٧٤	.....	SLEEP Instruction	التَّعليمية SLEEP
٧٤	.....	Return flag interrupt enable instruction	التَّعليمية RETFIE
٧٥	.....	Return Instruction	التَّعليمية RETURN
٧٥	.....	Return with lateral to W Reg.	التَّعليمية RETLW
٧٥	.....	Byte Orientation Operations	التَّعليمات على مستوى البايت

٧٦	ADD W Reg. to flag	ADDWF	التعليمة
٧٨	Swap W Reg. With Flag	SWAPF	التعليمة
٧٨	AND W Reg. with flag	ANDWF	التعليمة
٧٩	EX-OR W with flag	XORWF	التعليمة
٧٩	Inclusive OR W with flag	IORWF	التعليمة
٨٠	Clear Flag	CLRF	التعليمة
٨٠	No Operation Instruction	NOP	التعليمة
٨٠	Clear Working Register	CLRW	التعليمة
٨١	Compliment Flag Instruction	COMF	التعليمة
٨١	Decrement Flag	DECF	التعليمة
٨٢	Increment Flag	INCF	تعليمة
٨٢	Rotate Right Through Carry	RRF	تعليمة
٨٣	Rotate Left Through Carry	RLF	تعليمة
٨٣	Move Flag	MOVF	تعليمة
٨٥	Increment Flag Skip if Zero	INCFSZ	تعليمة
٨٦	Decrement Flag Skip if Zero	DECFSZ	تعليمة
٨٧	<b>Programming Concept</b>		<b>مبادئ البرمجة</b>
٨٩	Assembly language		لغة التجميع
٩٠	Program Structure		هيكل البرنامج
٩٠	PORTA & PORTB Configuration		تهيئة المنافذ
٩١	Writing Operations		عملية الكتابة
٩١	Reading Operations		عملية القراءة
٩٢	Programming Concept		مبادئ البرمجة
٩٢	Variable		المتغيرات
٩٣	Constant		الثوابت
٩٣	Comment		التعليقات
٩٤	#DEFINE		التعليمة
٩٥	ORG		التعليمة
٩٥	CBLOCK		التعليمة
٩٦	Macro Instruction		الماكرو
٩٦	Subroutine		الاجراءات
٩٨	branching Instruction		تعليمات التفرع
٩٨	if statement		تعليمة اذا الشرطية
٩٩	While Loop		تعليمة التكرار
١٠٢	Direct Addressing		العنونة المباشرة
١٠٢	UnDirect Addressing		العنونة الغير المباشرة

١٠٤	.....	Introduction to interrupt	مقدمة في المقاطعات
١٠٤	.....	interrupt philosophy	فلسفة المقاطعة
١٠٥	.....	Mechanism of interrupt	الآلية عمل المقاطعات
١٠٨	.....	Timers	المؤقت
١٠٩	.....	Timer modules	وحدة المؤقت
١٠٩	.....	Timer Enable	تفعيل خدمة المؤقت
١١١	.....	Timers interrupt	مقاطعة المؤقت
١١٢	.....	WATCHDOG TIMERS (WDT)	مؤقت الحراسة
١١٣	.....	watchdog timer preparation	خطوات تهيئة مؤقت الحراسة
١١٤	.....	watchdog timer operation	الآلية عمل مؤقت الحراسة
١١٥	.....	<b>Toolkit &amp; Simulation</b>	<b>العدد والمحاكاة</b>
١١٦	.....	easypic7	عملية توصيل المبرمجة بجهاز الحاسوب
١١٧	.....	hex. File	كتابة الشفرة Code وتوليد الملف
١١٧	.....	hex. File	خطوات توليد ملف
١٢٣	.....	Simulation using proteus	المحاكاة باستخدام برنامج proteus
١٢٣	.....	proteus 7	واجهة البرنامج 7

اللهم اكتب هذا العمل خالصاً لي ولوالدي



## Dedication

محمد (عليه الصلاة والسلام)

العراق المحيبي

الوالد (رحمة الله) والوالدة

زوجتي العزيزة

لكل مسلم يُريد عزة الأسلام وديننا

لكل شخص يُريد رفع رأس الوطن وكرامته

لكل أخ شخصي الذين أوصلاني إلى ما أنا عليه

لكل الشخص الذي كان السبب في إكمال هذا الكتاب

لكل أختوتي وأصدقائي....



## Numerical System

## الأنظمة العددية

مقدمة:

هذه الوحدة هي مقدمة للنظام العشري Decimal ، الثنائي Binary ، السادس عشر Hex. Decimal ، كيفية تمثيلها وكيفية التحويل بين هذه الأنظمة، في الحقيقة أن الغاية الأساسية لتقديم هذه الوحدة هي التعرف على بعض المفاهيم الأساسية مثل كيفية تمثيل الأعداد الموجبة والسالبة في النظام الثنائي، والتعرف على حالة الطفحان Overflow ، حالة المُحمل Carry ، وغيرها من المفاهيم الواجب التعرف عليها قبل الدخول في عالم المُسيطر الدقيق Microcontroller ..

## Decimal System

## النظام العشري

إن النظام الأكثر استخداماً في حياتنا اليومية في عمليات العد و الحساب هو النظام العشري Decimal system ، هذا النظام أساسه Radix هو 10 ، الأساس عشرة يعني أن هذا النظام مكون من عشر أعداد Digit لتمثيل القيم Values هي 0,1,2,3,4,5,6,7,8,9 تسمى هذه الأرقام معاملات Coefficient النظام العشري، فمثلاً العدد 123 يمكن تفسيره بشكل،

$$1 \times 10^2 + 2 \times 10^1 + 3 \times 10^0 = 100 + 20 + 3 = 123$$

نجد إن هذا العدد مكون من ثلاث مراتب Digit هي مرتبة الأحاد الذي هو الرقم "3" ، مرتبة العشرات الذي هو "20" ، مرتبة المئات الذي هو "100" ومجموع هذه المراتب يتكون العدد الذي هو "123" ، بصورة عامة يمكن تمثيل أي عدد بالنظام العشري بهذه الصيغة،

$$A_K \times 10^{n-1} + A_{K-1} \times 10^{n-2} + A_{K-2} \times 10^{n-3} \dots$$

حيثُ أن ،

- ❖  $A_k$  : تمثل الرقم Digit وموقعة من العدد Number ،
- ❖ 10 : تُمثل أساس Radix النظام العشري ،
- ❖ N : هو الأس الذي يُمثل مرتبة العدد

## Binary system

## النظام الثنائي

وهو النظام الذي يُستخدم في الأجهزة الرقمية Digital Devices لتمثيل الأعداد Numbers و تمثيل الإشارات Signals ، هذا النظام أساسه Radix هو "2" ، الأساس اثنان يعني أن هذا النظام مكون من رقمين Two Digit لتمثيل الأعداد هما "1" ، "0" يُسمى هذان الرقمان معاملات Coefficient النظام الثنائي، فمثلاً العدد "100"b تمثل العدد "4" بالنظام العشري ، والعدد "1101"b تمثل العدد "13" بالنظام العشري وهكذا ، أي أن هذا النظام مكون من سلسلة من الأصفار والوحدات فقط ..

ملاحظة:



- ❖ أصغر خانة في النظام الثنائي تُسمى (binary digit) bit ،
- ❖ أقصى خانة تقع على جهة اليسار تُسمى الخانة الأكثر أهمية (MSB (Most significant bit) ،
- ❖ أقصى خانة تقع على جهة اليمين تُسمى الخانة الأقل أهمية (LSB (Least significant bit) ،

الشكل 1.1 Figure يوضح عدد بالنظام الثنائي مكون من ثمان مراتب 8-bit ،



Figure 1.1

## System Conversion

## التحويل بين الأنظمة

أذا كنت تود برمجة مُسيطر دَقِيق Microcontroller ، يَجِب أن تُعرف كيفية التعامل مع النظام الثنائي Binary والنظام العشري Decimal وكيفية التحويل بين هذه الأنظمة ، لأنك ستحتاج عملية التحويل بكثرة أثناء عملية البرمجة programming .

## Binary to Decimal

## التحويل من النظام الثنائي إلى النظام العشري

للتحويل من النظام الثنائي إلى النظام العشري نضرب كل bit في أساس النظام Radix الذي هو '2' مرفوعة للأس رقم المرتبة أي ،

$$\text{Binary number} := (B_2 B_1 B_0)^2 \quad \text{Where } b_n := B_n \times 2^{\text{position}}$$

ثم نجمع كل  $b_n$  لاستخراج الناتج أي  $b_2+b_1+b_0$  وهكذا ، مع ملاحظة أن مرتبة الأحاد LSB تبدأ بالقيمة صفر أي أن position := 0

### ❖ مثال 1.1

حول العدد  $(1001)^2$  بالنظام الثنائي إلى ما يكافئه بالنظام العشري ؟  
الحل:

$$1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 8+0+0+1 = (9)^{10}$$

### ❖ مثال 1.2

حول العدد  $(0110)^2$  بالنظام الثنائي إلى ما يكافئه بالنظام العشري ؟  
الحل:

$$0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 0+4+2+0 = (6)^{10}$$

## Decimal to Binary

## التحويل من النظام العشري إلى النظام الثنائي

عند التحويل من النظام العشري إلى النظام الثنائي نتبع الصيغة التالية ، نقسم العدد الذي نريد تحويله إلى النظام الثنائي على أثنان "2" ، مثلاً العدد "10" ،

$$10 \div 2 = \quad \text{quotient is } 5 \quad \text{remainder} = 0 \quad \text{LSB}$$

ناتج قسمة quotient العدد  $10 \div 2$  هو "5" ، والمتبقي هو "0" ، الذي يعتبر هو الخانة الأقل أهمية LSB ، ثم نقسم العدد  $5 \div 2$  ونستخرج باقي القسمة وهكذا ... وتستمر العملية إلى أن يصبح العدد المقسوم أصغر من المقسوم عليه لتتوقف ،

$$5 \div 2 = \quad \text{quotient is } 2 \quad \text{remainder} = 1 \quad \text{bit.1}$$

$$2 \div 2 = \quad \text{quotient is } 1 \quad \text{remainder} = 0 \quad \text{bit.2}$$

$$1 \div 2 = \quad \text{invalid operation} \quad \text{remainder} = 1 \quad \text{MSB}$$

أذن مكافئ العدد  $(10)^{10}$  بالنظام الثنائي هو  $(1010)^2$

---

❖ مثال 1.3

حول العدد  $(14)^2$  بالنظام العشري الى ما يكافئ بالنظام الثنائي ؟

الحل:

$14 \div 2 =$	quotient is 7	remainder = 0	LSB
$7 \div 2 =$	quotient is 3	remainder = 1	bit.1
$3 \div 2 =$	quotient is 1	remainder = 1	bit.2
$1 \div 2 =$	invalid operation	remainder = 1	MSB

أذن مكافئ العدد  $(14)^{10}$  بالنظام الثنائي هو  $(1110)^2$

---

❖ مثال 1.4

حول العدد  $(15)^2$  بالنظام العشري الى ما يكافئ بالنظام الثنائي ؟

الحل:

$15 \div 2 =$	quotient is 7	remainder = 1	LSB
$7 \div 2 =$	quotient is 3	remainder = 1	bit.1
$3 \div 2 =$	quotient is 1	remainder = 1	bit.2
$1 \div 2 =$	invalid operation	remainder = 1	MSB

أذن مكافئ العدد  $(15)^{10}$  بالنظام الثنائي هو  $(1111)^2$

---

العدد العشري	العدد الثنائي
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

Table 1.1

## Units of Measurement

## وحدات القياس

كثير ما نسمع مصطلحات Giga byte , Mega byte ,Kilo byte التي تُشير بها الى سعات خزنية ، مثلاً نقول أن لدي قرص صلب Hard disk ذو سعة 120 Giga byte ، نحن بذلك نذكر وحدة القياس الخاصة بالأنظمة الرقمية ، حيث أن أصغر وحدة قياس في النظام الثنائي تسمى bit والتي تحمل قيمتان أما صفر أو واحد ، مجموع ثمان خانات 8-bit تسمى byte ، أي أن :

$$1 \text{ byte} = 8\text{-bit}$$

$$2 \text{ byte} = 2 \times 8\text{-bit} = 16\text{-bit}$$

الشكل Figure 1.2 يمثل الـ byte ،



Figure 1.2

أقصى عدد يُمكن أن يَستوعب البايت الواحد 1 byte هو  $(1111\ 1111)^2$  أي ما يكافئها بالإنظام العشري  $255^{10}$  ، هناك علاقة تستطع من خلالها معرفة أقصى عدد يمكن أن تستوعب تشكيلة من البتات Bits ، اذا عُلم عدد الخانات الثنائية Bit(binary digit) ،

$$\text{Maximum Value} = 2^N - 1$$

where N: is the number of bit

$$\text{Maximum Value} = 2^8 - 1 = 256 - 1 = 255$$

كما أن هناك قانون تستطيع من خلاله معرفة عدد البتات bit numbers اذا عُلم أعظم رقم Maximum Value وذلك باستخدام اللوغارتمات حيثُ،

$$\text{Number of bit}(N) = \log_2 X$$

where X is the maximum value

$$\text{Number of bit } (N) = \frac{\ln(X)}{\ln(2)}$$

❖ مثال 1.5

أذا كان لديك مستوعب Register (المستوعب او المسجل عبارة عن دائرة خزن رقمية) مكون من 16-bit فما هو اقصى عدد يمكن أن يتحملة المستوعب

الحل:

$$\text{Maximum Value} = 2^{16} - 1 = 65535$$

أذن أقصى عدد يُمكن تحمله هو 65535

❖ مثال 1.6

أذا كان لديك مستوعب Register أعظم رقم يستوعبه هو العدد  $32^{10}$  فما هو عدد البتات Bit Number الذي يتكون منه المستوعب ؟

الحل:

$$\text{Number of bit} = \frac{\ln(32)}{\ln(2)} = 5\text{-bit}$$

أذن عدد البتات هي 5-bit.

الوحدة	عدد الخانات الثنائية
1-byte	8-bit
1-word	16-bit
1-double word	32-bit
1-kilobyte	1024 byte
1-Megabyte	1024 kilobyte
1-Gigabyte	1024 Megabyte
1-Tera byte	1024 Gigabyte

Table 1.2

## Signed and Unsigned Value

## الأعداد الموجبة والسالبة

فلنا أن النظام الثنائي مكون من بسلسلة من الأصفار والواحدات Ones and Zeros ، و أن هذه السلسلة من الأصفار والواحدات عبارة عن إشارات كهربائية Electrical Signals ، السؤال الذي يتبادر الى الذهن كيف يتم تمثيل الاعداد السالبة في النظام الثنائي !!؟

يجب أن تعرف انه هناك نوعان من البيانات حسب طبيعة تمثيلها في الحاسبات الالكترونية،

Unsigned value

❖ الأعداد بدون الإشارة

Signed value

❖ الاعداد بأشارة

## Unsigned Value

## الاعداد بدون الإشارة

وهي الأعداد التي لا تحتوي على إشارة أي انها كلها أعداد موجبة ، لنتخيل لو كان لدينا مستوعب خزني Register مكون من 8-bit ، فأن مدى الأعداد التي يمكن تمثيلها هي

Unsigned Range of 8-bit

$$2^8 = 256$$

أي ان هناك 256 عدد موجب فقط .

## Signed value

## الأعداد بأشارة

وهي الأعداد التي تحتوي على إشارة سالبة مثل العدد -22 ، -10 ، ، بما أن الأجهزة الرقمية تتعامل مع الإشارات الكهربائية المكونة من بسلسلة من الأصفار والواحدات ، كيف سيتم تمثيل الرمز السالب ( - ) !!؟

هناك ثلاث طرق لتمثيل الأعداد السالبة والموجبة ،

Sign and Magnitude method  
1's Compliment method  
2's Compliment method

١- طريقة الإشارة والمقدار  
٢- طريقة المتمم الأول  
٣- طريقة المتمم الثاني

## Signe and Magnitude

## الأشارة والمقدار

تفترض هذه الطريقة حَجَزَ خانة bit لتمثيل إشارة العدد السالب أو الموجب ، عادة تكون خانة المحجوزة للإشارة العدد Sign هي خانة MSB ، إذا كانت هذه الخانة صفر Zero فذلك يدل على أن العدد موجب أما إذا كانت الخانة واحد One فيدل ذلك على أن العدد سالب ، مثلاً لمعرفة ما يقابل العدد  $(0000\ 0011)^2$  بالنظام العشري ، ننظر الى آخر خانة على جهة اليسار التي تحمل القيمة صفر فهذا يدل على ان العدد موجب ، هذا الجزء من العدد يسمى جزء الإشارة Sign ، ثم ننظر الى بقية العدد الذي يمثل جزء المقدار Magnitude ، الذي يمثل العدد '3' ، اذن العدد هو '3+' ، لنأخذ مثال آخر، لمعرفة ما يقابل العدد  $(1000\ 0110)^2$  بالنظام العشري ، بما أن آخر خانة على جهة اليسار قيمتها واحد فهذا يدل على ان العدد سالب ، ثم ننظر الى بقية العدد الذي يمثل العدد '6' ، اذن العدد هو '6-' ،

نستنتج من ذلك أن أي قيمة Values مكونة من جزئين جزء الإشارة Sign وجزء المقدار Magnitude

Sign (+,-)

Magnitude (value)

ملاحظة:



عيوب Disadvantage استخدام الأعداد السالبة Signed value هو خسارة المدى Range ، وذلك بتخصيصنا خانة للإشارة حيث،

Range of 8-bit

$$2^{8-1} = 2^7 = 128$$

الجدول Table 1.3 يوضح عدد مكون من 3-bit ممثل بصيغة عدد غير بدون إشارة Unsigned وصيغة عدد بأشارة Signe-Magnitude

عدد ثنائي	عدد بدون إشارة	صيغة إشارة - والمقدار
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-0 قيمة خطأ
101	5	-1
110	6	-2
111	7	-3

Table 1.3

لاحظ انه في حالة استخدام عدد بأشارة Signed value لعدد مكون من 3-bit سنجد انه المدى Range اصبح 4 وليس 8 ، والسبب في ذلك هو تخصيصنا الخانة MSB للأشارة ، أي أصبح المدى range هو  $2^2 = 4$  ، فأصبح لدينا اعداد موجبة من 0 الى 3 التي هي 0, 1, 2, 3 وأعداد سالبة من 4 الى 7 التي هي -3, -2, -1, 0- مع ملاحظة ان لا يوجد صفر أشارة سالبة فالصفر اشارة موجبة لذلك تجد في الجدول مقابل القيمة (0-) قيمة خاطئة،

### ❖ مثال 1.7

ما هو مدى Range لعدد بأشارة Signed Number مكون من 8-bit ؟

#### الحل:

$$2^8 = 256$$

$$2^{8-1} = 2^7 = 128$$

0 to 127

128 to 256

المدى Range لعدد مكون من 8-bit بدون إشارة هو بعد تخصيص خانة للأشارة يصبح المدى أي أن مدى الاعداد الموجبة التي يكون فيها MSB هو 0 هو ومدى الاعداد السالبة التي يكون فيها MSB هو 1 هو



ملاحظة:

في الحقيقة الحاسبات الألكترونية والأجهزة الرقمية Digital Device لا تستخدم أسلوب الأشارة والمقدار Sign-Magnitude لتمثيل الأعداد السالبة وذلك بسبب؟

- ❖ صعوبة معالجة Processing الأعداد السالبة باستخدام طريقة الاشارة والمقدار.
- ❖ تكرر حالة الصفر ، أو وجود عدد صفري سالب (0-).

## 1's Compliment

## المتمم الأول

تستخدم هذه الطريقة في بعض الاجهزة الرقمية لتمثيل الاعداد السالبة ، لايجاد سالب العدد  $(6)^{10}$  باستخدام المتمم الاول نقوم باتباع الخطوات التالية،

- ❖ نحول الرقم  $(6)^{10}$  الى النظام الثنائي  $(110)^2$
- ❖ نعكس كل خانة bit من العدد  $(110)^2$  ليصبح العدد  $(001)^2$

أذن سالب العدد  $(6)^{10}$  هو  $(1)^{10}$  ، الجدول Table 1.4 يبين الاختلاف بين طريقة المتمم الاول وطريقة الأشارة والمقدار وعدد بدون إشارة Unsigned لعدد مكون من 3-bit،

العدد الثنائي	عدد بدون إشارة	الأشارة- والمقدار	المتمم الأول
000	0	0	قيمة خطأ $0 = 7 = 111$
001	1	1	$110 = 6 = -1$
010	2	2	$101 = 5 = -2$
011	3	3	$100 = 4 = -3$
100	4	قيمة خطأ -0	$011 = 3 = -4$
101	5	-1	$010 = 2 = -5$
110	6	-2	$001 = 1 = -6$
111	7	-3	$000 = 0 = -7$

Table 1.4



ملاحظة:

عيوب Disadvantage استخدام طريقة المتمم الاول 1's Complement.

❖ وجود حالة السالب الصفري (0-).

أيجابيات Advantage استخدام طريقة المتمم الاول 1's Complement.

❖ سهولة معالجة الأرقام السالبة.

❖ مثال 1.8

جد سالب العدد  $(0010\ 0011)^2$  أي ما يكافئة بالنظام العشري  $(35)^{10}$  ؟

الحل:

❖ نَقْلِب كل خانة من العدد  $(0010\ 0011)^2$  ليكون لدينا  $(1101\ 1100)^2$

❖ نحول العدد  $(1101\ 1100)^{10}$  إلى النظام العشري  $(220)^{10}$  أي أن سالب العدد  $(35)^{10}$  هو  $(220)^{10}$ .

❖ مثال 1.9

جد سالب العدد  $(0010)^2$  أي ما يكافئة بالنظام العشري  $(2)^{10}$  ؟

الحل:

❖ نَقْلِب كل خانة من العدد ليكون  $(0010)^2$  لدينا  $(1101)^2$

❖ نحول العدد  $(1101)^{10}$  إلى النظام العشري  $(13)^{10}$  أي أن سالب العدد  $(2)^{10}$  هو  $(220)^{10}$ .

## 2's Compliment

وهي الطريقة الأكثر استخداماً في معالجة وتمثيل الأعداد السالبة في الحاسبات والجهزة الرقمية، وذلك كونها تتغلب على حالة الصفر المُكرر وكذلك عدم وجود عدد صفري سالب، مثلاً لإيجاد سالب العدد  $10^3$  بطريقة المتمم الثاني نتبع الخطوات التالية،

❖ نحول العدد  $10^3$  الى النظام الثنائي  $(0000\ 0011)^2$

❖ نقلب كل خانة bit ليكون لدينا  $(1111\ 1100)^2$

❖ نظيف واحد الى الناتج ليكون  $(1111\ 1101)^2$

أذن سالب العدد  $10^3$  بالنظام الثنائي هو  $(1111\ 1101)^2$  أي ما يكافئة بالنظام العشري  $10^3(253)$ .

الجدول Table 1.5 يبين الأختلاف بين طريقة المُتمم الأول وطريقة المُتمم الثاني وطريقة الإشارة والمقدار وعدد بدون إشارة Unsigned لعدد ثنائي مكون من 3-bit،

عدد ثنائي	عدد بدون إشارة	إشارة- ومقدار	المتمم الأول	المتمم الثاني
000	0	0	قيمة خطأ $0 = 7 = 111$	$000 = 0$
001	1	1	$110 = 6 = -1$	$111 = 7 = -1$
010	2	2	$101 = 5 = -2$	$110 = 6 = -2$
011	3	3	$100 = 4 = -3$	$101 = 5 = -3$
100	4	قيمة خطأ $-0$	$011 = 3 = -4$	$100 = 4 = -4$
101	5	-1	$010 = 2 = -5$	$011 = 3 = -5$
110	6	-2	$001 = 1 = -6$	$010 = 2 = -6$
111	7	-3	$000 = 0 = -7$	$001 = 1 = -7$

Table 1.5

ملاحظة:



أيجابيات Advantage استخدام طريقة المتمم الثاني 2's Compliment.

❖ سهولة معالجة الأرقام السالبة ،

❖ لا وجود لحالة الصفر السالب (0-)

❖ مثال 1.10

جد سالب العدد  $10^3$  (11) باستخدام المتمم الثاني ؟

الحل:

❖ نحول العدد  $10^3$  الى النظام الثنائي  $(1011)^2$

❖ نقلب كل خانة bit ليكون لدينا  $(0100)^2$

❖ نظيف واحد الى الناتج ليكون  $(0101)^2$

أذن سالب العدد  $10^3$  بالنظام الثنائي هو  $(0101)^2$  أي ما يكافئة بالنظام العشري  $10^3(5)$

جد سالب العدد  $(1001)^2$  أي ما يكافئة بالنظام العشري  $9^{10}$  ؟

الحل:

❖ نقلب كل خانة bit ليكون لدينا  $(0110)^2$

❖ نظيف واحد الى الناتج ليكون  $(0111)^2$

أذن سالب العدد  $9^{10}$  بالنظام الثنائي هو  $(0111)^2$  أي ما يكافئة بالنظام العشري  $7^{10}$

## Arithmetic Operations

## العمليات الحسابية على النظام الثنائي

تجري على النظام الثنائي العمليات الحسابية نفسها التي تجري على النظام العشري ، مثل عملية الجمع Adding ، الطرح Subtracting ، وغيرها من العمليات بالإضافة الى عمليات أخرى تُدعى العمليات المنطقية Logical Operation ، مثل عملية AND,OR,NOT وغيرها....

## Addition Arithmetic Operation

## عملية الجمع

إذا كان لدينا عدنان كل عدد مكون من 4-bit ، لنرمز للعدد الأول A أي ان هناك  $A_3 A_2 A_1 A_0$  ، ونرمز للعدد الثاني B أي أن هناك  $B_3 B_2 B_1 B_0$  ، ستجري عملية الجمع بين العددين  $B + A$  كما تجري عملية الجمع في النظام العشري كما هو موضح في الشكل Figure 1.2

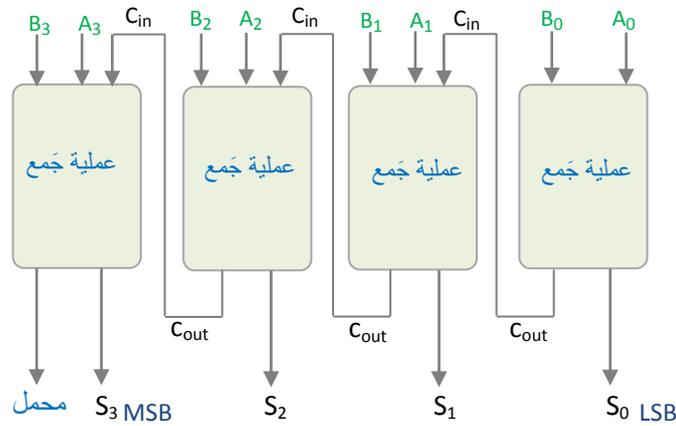


Figure 1.2

عند إجراء عملية الجمع ، سنجمع الخانة  $B_0$  مع الخانة  $A_0$  ، ناتج الجمع الذي هو  $S_0$  سوف نضعه في خانة LSB وإذا وجد محمل Carry( $C_{out}$ ) سينقل الى المرحلة التي تليها كما هو الحال في النظام العشري ، ثم نجمع الخانة  $A_1$  مع الخانة  $B_1$  مع قيمة المحمل  $C_{in}$  الذي هو نفسة المحمل القادم من المرحلة السابقة  $C_{out}$  أي  $C_{in} = C_{out}$  وهكذا ، عملية الجمع تتم بالطريقة التالية ،

$$\begin{array}{r}
 C_{in} \quad C_{in} \quad C_{in} \quad C_{in} \\
 A_3 \quad A_2 \quad A_1 \quad A_0 \\
 + \\
 B_3 \quad B_2 \quad B_1 \quad B_0 \\
 \hline
 Carry \quad S_3 \quad S_2 \quad S_1 \quad S_0
 \end{array}$$

لاحظ الجدول Table 1.6 الذي يبين حالات الإدخال  $C_{in}$ ,  $B_n$ ,  $A_n$  وحالات الأخرجات، التي هي ناتج عملية الجمع Sum والمحمل الخارج  $C_{out}$ ،

$A_n$	$B_n$	$C_{in}$	Sum	$C_{out}$
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1.6

❖ مثال 1.12

أجمع العدد  $(0110)^2$  مع العدد  $(0111)^2$  ؟  
الحل:

$$\begin{array}{r}
 0 \quad 1 \quad 1 \quad 0 \\
 1 \quad 1 \quad 1 \quad 0 \\
 + \\
 0 \quad 1 \quad 1 \quad 1 \\
 \hline
 Carry = 0 \quad 1 \quad 1 \quad 0 \quad 1
 \end{array}$$

أذن ناتج جمع العدد  $(6)^{10}$  مع العدد  $(7)^{10}$  هو العدد  $(13)^{10}$  الذي يكافئه بالنظام الثنائي  $(01101)^2$

أجمع العدد  $(1100)^2$  مع العدد  $(0011)^2$  ؟  
الحل:

$$\begin{array}{r}
 \textcircled{0} \quad \textcircled{0} \quad \textcircled{0} \quad \textcircled{0} \quad 0 \\
 1 \quad 1 \quad 0 \quad 0 \\
 \hline
 0 \quad 0 \quad 1 \quad 1 \quad + \\
 \hline
 \text{Carry} = 0 \quad 1 \quad 1 \quad 1 \quad 1
 \end{array}$$

أذن ناتج جمع العدد  $(12)^{10}$  مع العدد  $(3)^{10}$  هو العدد  $(15)^{10}$  الذي يكافئه بالنظام الثنائي  $(01111)^2$

## Subtracting Using 2's Compliment

## الطرح باستخدام المتمم الثاني

إذا اشتريت حاسبة علمية Scientific Calculator أو استخدمت الحاسبة العلمية المرفقة مع نظام التشغيل Windows ، لتقوم بعملية طرح عددين ، وذلك كي توفر الوقت والعناء بجعل الاله Machine تقوم بعملية الطرح عوضاً عنك، الية الطرح في هذه الأجهزة تتم باستخدام المتمم الثاني كما سنلاحظه في السطور التالية ، مثلاً إذا أردت طرح العدد  $(13)^{10}$  من العدد  $(5)^{10}$  ، باستخدام الحاسبة العلمية ، كما قلنا سابقاً ان النظام الذي تتعامل معه اللآلات الرقمية Digital Devices مثل الحاسبات العلمية والحاسبات الالكترونية personal computer والمسيطرات الدقيقة Microcontroller ، هو النظام الثنائي binary System ، وذلك بوجود دوائر خاصة تقوم بعملية الطرح ، لنرى كيف تتم عملية الطرح نقوم أولاً بتحول العدد  $(5)^{10}$  الى النظام الثنائي سنجد أن مكافئ العدد  $(5)^{10}$  هو  $(0101)^2$  ، ومكافئ العدد  $(13)^{10}$  هو  $(1101)^2$  ، أي أن العملية التي نريد أن نجريها هي  $(13 - 5)$  ، أو  $(1101 - 0101)$  ، الطريقة المستخدمة في عمليات الطرح هو بإيجاد المتمم الثاني 2's compliment للعدد المطروح ، بما أننا سنجد المتمم الثاني للعدد  $(5)^{10}$  أو  $(0101)^2$  والذي نقصد فيه اننا نجد سالب العدد  $(-5)^{10}$  أو  $(1011)^2$  لتصبح العملية  $(13 + (-5))$  ، أو  $(1101 + 1011)$  ، أنظر كيف أن عملية الطرح الآن أصبحت عملية جمع ، أي أن الدائرة التي تقوم بعملية الطرح هي نفسها التي تقوم بعملية جمع وهذه هي الغاية من استخدام المتمم الثاني ، الشكل Figure 1.3 يوضح مخطط صندوقي Block Diagram لدائرة جمع و طرح باستخدام المتمم الثان

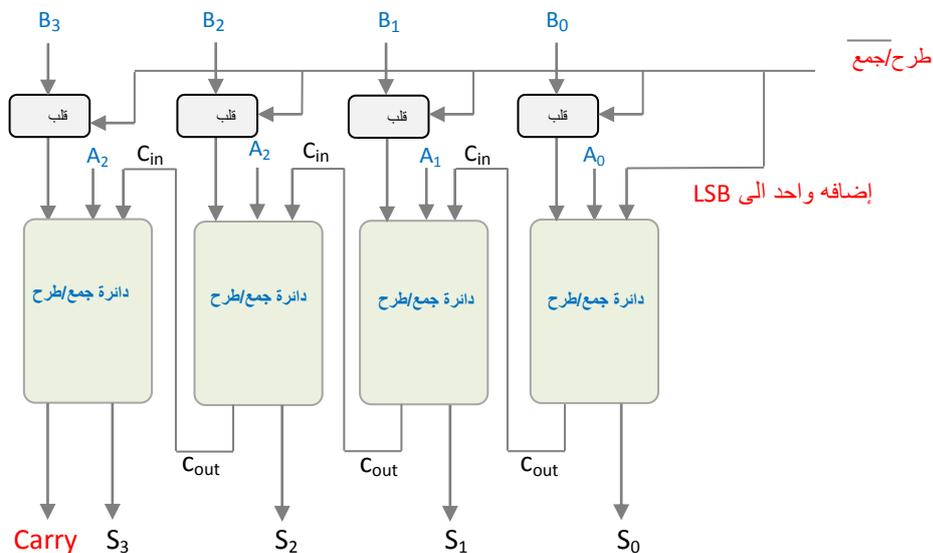


Figure 1.3

تتم عملية طرح  $10^{(5-13)}$  أو  $(1101 - 0101)^2$  كما هو معتاد كألآتي ،

	○	○	○	○	
	1	1	0	1	
	0	1	0	1	-
Carry	$X_n$	$X_n$	$X_n$	$X_n$	

خطوات ايجاد المتمم الثاني للعدد المطروح للتهيئة لأجراء عملية الطرح :

نجد اولاً المتمم الثاني للعدد  $10^{(5)}$  أو  $(0101)^2$  الذي هو  $(1011)^2$

	○	○	○	○	
	1	1	0	1	
	1	0	1	1	+
Carry	$X_n$	$X_n$	$X_n$	$X_n$	

بعد ذلك نجري عملية الجمع التي هي في الأصل عملية طرح!!؟

	○	○	○	○	
	1	1	0	1	
	1	0	1	1	+
Carry Omitted	$X_n$	$X_n$	$X_n$	$X_n$	
	1	0	0	0	

ملاحظة:

عند ظهور محمل carry في المتمم الثاني 2's Compliment يهمل،

النتائج النهائي هو  $(1000)^2$  أي ما يقابله بالنظام العشري  $10^8$  ، وهو فعلاً ناتج عملية الطرح  $10^{(5-13)}$  ،

❖ مثال 1.13

أطرح العدد  $(0100)^2$  من العدد  $(0101)^2$  ؟

الحل:

العملية التي نريد ان نجريها هي  $(5-4)$  أي ما يقابلها بالنظام الثنائي  $(0101)^2 - (0100)^2$

❖ نجد المتمم الثاني للعدد  $(4)^{10}$  ليكون لدينا  $(1100)^2$

❖ نجمع متمم العدد  $(4)^{10}$  مع العدد  $(5)^{10}$

1	1	0	0	
	0	1	0	1
		1	1	0
			0	0
<hr/>				
	0	0	0	1

Carry Omitted

❖ مثال 1.14

أطرح العدد  $(01110)^2$  من العدد  $(00111)^2$  ؟

الحل:

العملية التي نريد ان نجريها هي (3-6) أي ما يقابلها بالنظام الثنائي  $(01110)^2 - (00111)^2$

❖ نجد المتمم الثاني للعدد  $(6)^{10}$  ليكون لدينا  $(1010)^2$

❖ نجمع متمم العدد  $(6)^{10}$  مع العدد  $(3)^{10}$

0	0	1	0	
	0	0	1	1
		1	0	1
			1	0
<hr/>				
	1	1	0	1

No carry

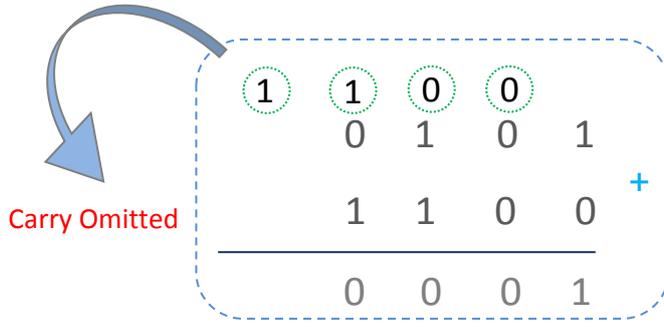
## Carry State

## حالة المُحمل

المُحمل هو الخانة الثنائية Bit التي تخرج من الخانة الأكثر أهمية MSB ، هذا المُحمل له فوائد كثيرة منها على سبيل المثال ، عند مُقارنة عددين ونريد معرفة أي العدد هو الأكبر ، لناخذ مثلاً العدد  $(4)^{10}$  والعدد  $(5)^{10}$  عند إجراء عملية الطرح التالية ،

$$Y = 5 - 4$$

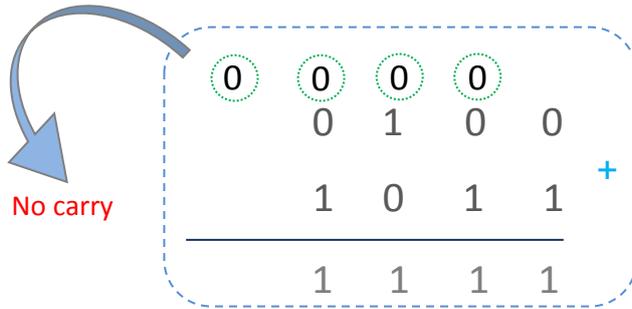
نقوم أولاً بأيجاد المكافئ الثنائي لكن من العدد  $(4)^{10}$  الذي هو  $(0100)^2$  والعدد  $(5)^{10}$  الذي هو  $(0101)^{10}$  ، كما تعلمنا سابقاً أن عملية الطرح تتم باستخدام المتمم الثاني ، أي اننا سنجد المتمم الثاني للعدد  $(4)^{10}$  الذي هو  $(1100)^2$  لتصبح العملية كالاتي



لاحظ أن ناتج العملية هو  $(1)^{10}$  وهو فعلاً ناتج عملية طرح  $(5 - 4)^{10}$  ، المهم لدينا هو ظهور محمل وهذا يدل على أن العدد الاول أكبر من العدد الثاني ، لنجري عملية الطرح التالية

$Y = 4 - 5$

نقوم أولاً بأيجاد المكافئ الثنائي لكن من العدد  $(5)^{10}$  الذي هو  $(0101)^2$  والعدد  $(4)^{10}$  الذي هو  $(0100)^{10}$  ، ثم نجد المُتمم الثاني للعدد  $(5)^{10}$  الذي هو  $(1011)^2$  لتصبح العملية كالآت



لاحظ أن ناتج العملية هو  $(1111)^{10}$  ، هذا الناتج هو عدد سالب ، سنناقشة لاحقاً ، المهم لدينا هو عدم ظهور محمل وهذا يدل على أن العدد الاول أصغر من العدد الثاني ،



- ❖ عند إجراء عملية طرح بين عددين ، فإن المحمل Carry يظهر عندما يكون العدد الاول أكبر من العدد الثاني،
- ❖ عند إجراء عملية طرح بين عددين ، فإن المحمل Carry لا يظهر عندما يكون العدد الاول أكبر من العدد الثاني،

## Number in Computer's Memories

## الأرقام داخل ذاكرة الحاسوب

لقد تعلمنا مما سبق أن أيجاد المُتمم الثاني ، معناه أيجاد سالب العدد ، مثلاً العدد  $(0110)^2$  المُتمم الثاني له أو سالب العدد هو  $(1010)^2$  ، السؤال الذي يتبادر الى الذهن كيف يُميز الحاسوب بين العدد  $(1010)^2$  الذي هو  $(-6)^{10}$  والعدد  $(1010)^2$  الذي هو العدد  $(10)^{10}$  ؟؟؟

لكي يستطيع الحاسوب التمييز بين العددين ، يستخدم أسلوب الإشارة والمقدار بحيث يُخصص الخانة MSB للإشارة ، أي اذا كان MSB واحد فهذا يدل على ان العدد سالب , أما اذا كان MSB صفر فهذا يدل على أن العدد موجب وحسب الجدول Table 1.7،

عدد ثنائي	الحالة	المكافئ بالمتعم الثاني فقط للعدد السالب
000	عدد موجب	0
001	عدد موجب	1
010	عدد موجب	2
011	عدد موجب	3
100	عدد موجب	-4
101	عدد سالب	-3
110	عدد سالب	-2
111	عدد سالب	-1

Table 1.7

لاحظ العَدَد  $(0110)^2$  هو عَدَد موجب والسَّبب في ذلك هو إن أخر بت bit على جهة اليسار أي البت الأكثر أهمية MSB يحمل القيمة صفر ، عندما نجد المُتعم الثاني للعدد  $(0110)^2$  والذي هو  $(1010)^2$  ، أنظر إلى الخانة Bit الأكثر أهمية MSB أصبحت تَحمل القيمة واحد مشيرة بذلك إلى إن العدد هو سالب Negative ..

## Overflow Condition

## حالة الطَّفْحان

أن الوحدات المسؤولة عن عملية تخزين الأرقام داخل الأجهزة الرقمية تسمى المسجلات Registers ، تخيل أن لدينا حاسبة علمية وحدات الخزن الداخلية لها عبارة عن مسجل Register ذات سعة بايت واحد 1-byte ، أي أن المسجل يتقبل اعلى قيمة له التي هي  $(1111 1111)^2$  أي ما يقبلها بالنظام العشري  $10^{255}$  ، نستنتج من ذلك أن إمكانية التخزين داخل الاجهزة الرقمية محدودة ، تخيل لو أردنا جمع العدد 255 مع العدد 1 ماذا يحدث؟؟؟

عملية الجمع ستكون ناتجها 256 أي ما يقابلها بالنظام الثنائي  $(1 0000 0000)^2$  ، لقد تجاوز الناتج العدد المسموح ، الذي هو 255 ، فتظهر حالة الطَّفْحان التي تسبب لنا تصفير المسجل Reset Register .

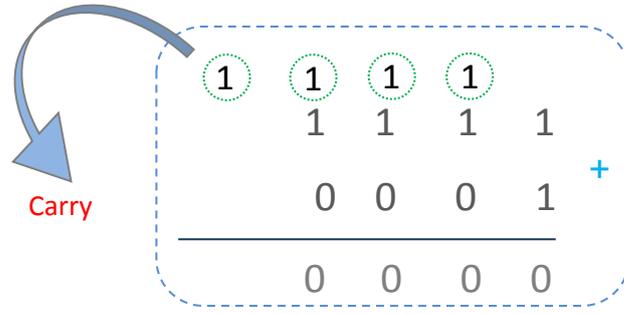
أن حالة الطَّفْحان ايضاً تظهر في الحالة التالية ، عند تغير اشارة ناتج عملية حسابية مثلاً عند جمع عددين موجبين وأصبح الناتج عدد سالب تظهر حالة الطَّفْحان ، أو عند جمع عددين سالبين واصبح الناتج عدد موجب فتظهر حالة الطَّفْحان ، هذا النوع من الطَّفْحان يسمى الطَّفْحان بالأشارة ،

## Examples

## أمثلة على الطَّفْحان

يختلف الطَّفْحان عند التحدث عن الأرقام الموجبة فقط ( الأرقام بدون إشارة ) Unsigned Numbers عنه في الأرقام بإشارة Signed Numbers .

لنأخذ مثال لنوضح فية عملية الفيضان لنجمع العددين التاليين ،



عملية الجمع التالية تحتمل تفسيرين،

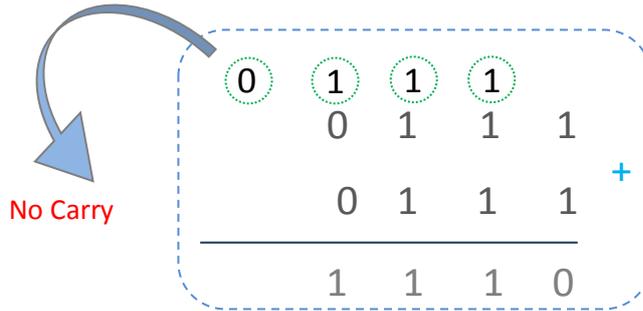
### ○ التفسير الاول :

لنفرض أن العددين  $(1111)^2$  ،  $(0001)^2$  هما من نوع الأعداد بدون الإشارة Unsigned Numbers ، أي أننا نريد جمع العدد  $(15)^{10}$  مع العدد  $(1)^{10}$  ليكون الناتج عدد موجب الذي هو  $(16)^{10}$  وهو فعلاً ما حدث حيث ان الناتج هو  $(1\ 0000)^2$  ، في هذه الحالة يظهر فيضان Overflow من النوع الاول الذي فية يتصفر الناتج Reset ، أما فيضان Overflow بالإشارة ليم يظهر في هذه الحالة،

### ○ التفسير الثاني :

لنفرض أن العددين  $(1111)^2$  ،  $(0001)^2$  هما من نوع الأعداد بأشارة Signed Numbers ، أي أننا نريد جمع العدد  $(1111)^{10}$  وهو عدد سالب ، اذا وجدنا المتمم الثاني لة نجد انه  $(-1)^{10}$  ، مع العدد  $(1)^{10}$  ليكون الناتج عدد موجب الذي هو  $(0)^{10}$  وهو فعلاً ما حدث حيث ان الناتج هو  $(0000)^2$  ، والمحمل يهمل كما تعلمنا عند إجراء عملية الطرح باستخدام المتمم الثاني، في هذه الحالة لم يظهر فيضان لا من النوع الأول ولا من النوع الثاني،

لنأخذ مثال آخر نوضح فية عملية الطفحان ،



عملية الجمع التالية تحتمل تفسير واحد لان كلا العددين موجبان  $(0111)^2$  ،  $(0111)^2$  ، لاحظ أن الناتج هو عدد سالب والسبب في ذلك هو ان الخانة الاكثر أهمية MSB تحمل القيمة واحد ، نتيجة لذلك يظهر لدينا طفحان بأشارة Overflow ،

لقد كان المُبرمجين قديماً يتناقضون برامجهم المكونة من سلسلة من الأرقام والوحدات عن طريق أوراق Sheets وذلك لبرمجة اجهزتهم Devices ، أن الشكل Figure 1.4 يوضح مجموعة من الأرقام والوحدات التي تُمثل برنامج مكتوب باللغة الثنائية او لغة الماكينة Machine Codes ،

شفرة ثنائية	
0000:11010001	0101:11010001
0001:11010001	0110:11010001
0010:01101101	0111:01101101
0011:11001100	1000:11001100
0100:11001101	1001:11001101

Figure 1.4

هذا الأسلوب أدى الى ظهور شوائب كثيرة Bugs نتيجة إخطاء في عملية نقل البرنامج ، مما دفعهم الى إختراع نظام جديد يسهل عملية التحويل من النظام الثنائي الى النظام الجديد ، وذلك لصعوبة التحويل من النظام الثنائي الى النظام العشري وبالعكس ، فظهر النظام السادس عشر Hex.Decimal ، هذا النظام أساسه Radix هو 16 ، الأساس ستة عشر يعني أن هذا النظام مكون من ستة عشر عدداً Digit لتمثيل الأعداد هي 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F حيث إن A=10,B=11,C=12,D=13,E=14,F=15 تسمى هذه الأعداد Digits معاملات Coefficient النظام السادس عشر ،

## Hex.Decimal to Binary

## التحويل من النظام السادس عشر إلى النظام الثنائي وبالعكس

بما أن  $2^4=16$  أي يمكن تقسيم العدد الثنائي الى مجاميع Group كل مجموعة مكونة من 4-bit حسب الجدول Table 1.8 ،

عدد ثنائي	المكافئ بالنظام السادس عشر
0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	A
1011	B
1100	C
1101	D
1110	E
1111	F

Table 1.8

## ❖ مثال 1.15

حول العدد  $(0001\ 0110)^2$  الى النظام السادس عشر  $^{16}(?)$ ؟

الحل:

نجزء العدد  $(0001\ 0110)^2$  الى جزئين كل جزء مكون من 4-bit ، أي 0110 و 0001 ثم نجد مكافئ العدد 0110 بالنظام السادس عشر الذي هو 6 ، ونجد مكافئ العدد 0001 بالنظام السادس عشر الذي هو 1 ، اذن مكافئ العدد  $(0001\ 0110)^2$  هو  $(16)^{16}$ .

## ❖ مثال 1.16

حول العدد  $(1110\ 1111)^2$  الى النظام السادس عشر  $^{16}(?)$ ؟

الحل:

نجزء العدد  $(1110\ 1111)^2$  الى جزئين كل جزء مكون من 4-bit ، أي 1111 و 1110 ثم نجد مكافئ العدد 1111 بالنظام السادس عشر الذي هو F ، ونجد مكافئ العدد 1110 بالنظام السادس عشر الذي هو E ، اذن مكافئ العدد  $(1110\ 1111)^2$  هو  $(EF)^{16}$ .

## ❖ مثال 1.16

حول العدد  $(2E)^{16}$  الى النظام الثنائي  $^2(?)$ ؟

الحل:

نجزء العدد  $(2\ E)^{16}$  الى جزئين، أي E و 2 ثم نجد مكافئ العدد E بالنظام الثنائي الذي هو  $(1110)^2$ ، ونجد مكافئ العدد 2 بالنظام الثنائي الذي هو  $(0010)^2$ ، اذن مكافئ العدد  $(2E)^{16}$  هو  $(00101110)^2$ .

## ❖ مثال 1.17

حول العدد  $(AB)^{16}$  الى النظام الثنائي  $^2(?)$ ؟

الحل:

نجزء العدد  $(AB)^{16}$  الى جزئين، أي B و A ثم نجد مكافئ العدد B بالنظام الثنائي الذي هو  $(1011)^2$ ، ونجد مكافئ العدد A بالنظام الثنائي الذي هو  $(1010)^2$ ، اذن مكافئ العدد  $(AB)^{16}$  هو  $(10101011)^2$ .

بهذا أصبح للبرمجين نظام جديد يُسهل عليهم التحويل من النظام الثنائي الى النظام السادس عشر ، مباشراً ، وسهل النقل ، أنظر للشفرة في الشكل 1.5 Figure

شفرة بالنظام السادس عشر	
0: D 1	5: D 1
1: D 1	6: D 1
2: 6 D	7: 6 D
3: C C	8: C C
4: C D	9: C D

Figure 1.5



## Programmable Devices Architecture

## بُنِيَة الأنظمة القابلة للبرمجة

الأنظمة القابلة للبرمجة هي تلك الأنظمة التي لها صيغة محددة ومنظمة ، والتي يعتمد عملها اعتماداً كبيراً على مجموعة تعليمات تصاغ بطريقة معينة، يتم تخزينها داخل تلك الأنظمة لاداء مهام معينة ،ولكي نفهم مبدء عمل هذه الأنظمة دعنا نلقي نظرة على انواع الأنظمة الرقمية الالكترونية ،

### Hard wire connection

### أنظمة الربط الصلب

وهي الأنظمة المعتمدة على ربط مجموعة من الرقائق الرقمية Chips لتأدية مهمة معينة....

### Programmable System

### الانظمة القابلة للبرمجة

وهو نظام رقمي ذو صيغة مُحددة ومنظمة، يتكون من ،

- ❖ وحدة معالجة مركزية (CPU) Central processing unit
- ❖ ذاكرة Memory
- ❖ وحدات إدخال،وحدات أخراج Input and Output Unit

الشكل 2.1 Figure يوضح المخطط الصندوقي block diagram للنظام القابل للبرمجة Programmable Device

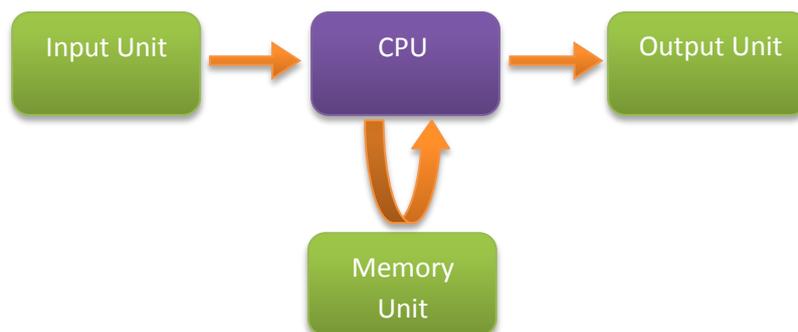


Figure 2.1

تقوم وحدة المعالجة المركزية CPU بجلب وتنفيذ البرنامج المخزن في الذاكرة Memory او بصيغة اخرى وضع التعليمات المخزنة في الذاكرة Memory قيد التنفيذ ، وذلك لتأدية المهمة المطلوبة من قبل شريحة المسيطر الدقيق Microcontroller ، لكي ينفذ المعالج CPU التعليمات المخزنة في الذاكرة Memory فأنة يمر بمرحلتين او دورتين

١- مرحلة الجلب Fetching cycle

٢- دورة التنفيذ Execution cycle

### Fetching cycle

### دورة الجلب

يقوم المعالج CPU خلال هذه الفترة بجلب التعليمات المخزنة في الذاكرة Memory ، وذلك عن طريق عنوان الموقع المطلوب من الذاكرة باستخدام عداد خاص يدعى عداد البرنامج (PC(program counter) ، عند بداية التشغيل يحمل العداد PC القيمة صفر ، أي انه يُوْشِر على الموقع الاول من الذاكرة ، ثم تنقل محتويات هذا الموقع الى مستوعب(سجل) خاص special register يدعى سجل التعليمات instruction Register(IR) ، وذلك لتهيئة التعليمات instruction لكي يتم تمييزها وتنفيذها ، وبعد أن تنفذ التعليمات يزداد عداد البرنامج PC بمقدار واحد وذلك لكي يُوْشِر الى الموقع التالي ثم تنقل محتويات الموقع الى سجل IR ثم تُمَيِّز التعليمات وتكرر هذه العملية الى ان ينتهي البرنامج، اذن دورة الجلب تتم كالاتي

- ❖ نقل محتويات عداد البرامج PC الى الذاكرة memory لكي يُوْشِر الموقع المطلوب desired location
- ❖ نقل محتويات الذاكرة Memory الى سجل التعليمات IR

الشكل Figure 2.2 يوضح الترابط بين سجل التعليمات IR ، وعداد البرنامج PC ، والذاكرة Memory

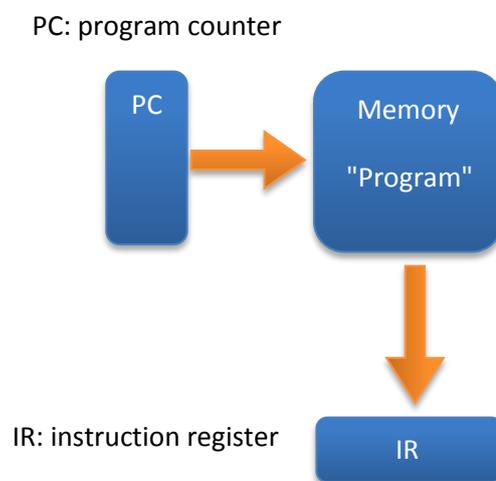


Figure 2.2

### Execution cycle

### دورة التنفيذ

تبدء هذه الدورة بعد ان تنتهي دورة الجلب fetch cycle ، تقوم هذه الدورة بأخذ التعليمات من سجل التعليمات IR ، وتنقل هذه التعليمات الى دوائر خاصة لكي يتم الكشف عن التعليمات والتعرف عليها وتنفيذها ، أي بأختصار تقوم دورة التنفيذ بتفويض التعليمات ، الشكل Figure 2.3 يوضح وحدة التنفيذ execution unit مع وحدة الجلب fetching unit

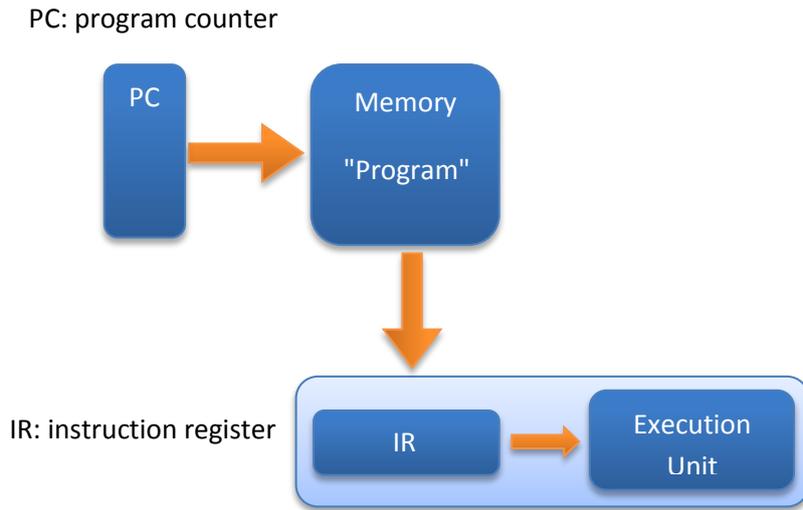


Figure 2.3

تتنوع التعليمات instruction حسب طبيعة الوظيفة التي تؤديها ، أي يمكن تقسيم التعليمات الى مجاميع

arithmatic instruction	❖ مجموعة التعليمات (الايجازات) الحسابية
moving instruction	❖ مجموعة ايعازات النقل
jump instruction	❖ مجموعة ايعازات القفز

تسمى مجموعة الايعازات هذه بطقم التعليمات instruction set ، سيتم شرح هذه الايعازات بشكل مفصل في فصول لاحقة ، غالب التعليمات التي تنفذ من قبل وحدة المُعالجة المركزية هي تعليمات حسابية ، تحتاج هذه التعليمات الحسابية الى دوائر جمع وطرح ودوائر منطقية logic circuit لاداء الوظائف المنطقية مثل And,Or,Not الوحدة المسؤولة عن اجراء هذه العمليات هي وحدة الحساب والمنطق ALU(arithmetic and logic unit)

## Arithmetic and logic unit

## وحدة الحساب والمنطق

وهي الوحدة المسؤولة عن إجراء العمليات الحسابية والمنطقية ، تحتوي هذه الوحدة كما قلنا على مجموعة دوائر جمع ، طرح ، دوائر منطقية لاجراء العمليات المنطقية ، في الحقيقة ترتبط مع هذه الوحدة مجموعة مساحات خزنية صغيرة تدعى Register File ، لنوضح فكرة Register file ، لنفرض اني أردت منك أن تجري العملية  $5+6$  ، أنت عرفت أن العملية جمع وذلك لوجود اشارة (+) هذا الجزء يسمى الايعاز instruction ، سيقوم عقلك بالتهيئة لاجراء عملية الجمع ، لنفرض أن في عقلك وحدة الحساب والمنطق ALU هي التي ستقوم بالعملية، الايعاز وحدة لا يكفي لاجراء عملية الجمع ، فأنت تحتاج الى معاملات Operand هذه المعاملات هما العددان (5),(6) اللذان سيجمعان في وحدة الحساب والمنطق ، بعد ان تتم عملية الجمع تقوم وحدة ALU بخزن الناتج في Register file ، أذن Register file تخزن معاملات الرياضية التي سيتم معالجتها وكذلك النواتج من وحدة ALU، كذلك Register file تستخدم في وظائف أخرى ، مثلاً تخصص مواقع معينة لخزن البيانات Data القادمة من أطراف الادخال ، او تخصيص مواقع ترتبط مباشرة مع وحدات الاخراج ، بحيث اني أي قيمة تحملها هذه المواقع ستظهر على وحدات الاخراج ، الشكل Figure 2.4 يوضح وحدة الحساب والمنطق ALU مع Register file ،

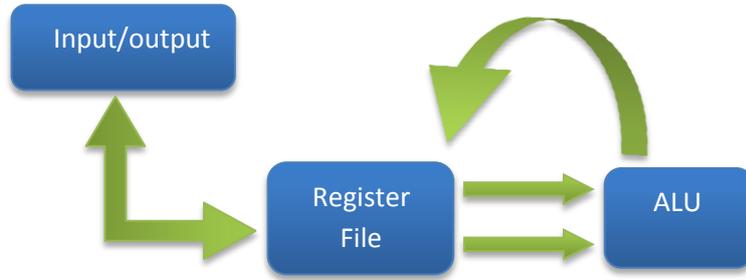


Figure 2.4

## Control unit

## وحدة السيطرة

السؤال الذي يتبادر الى الذهن ما هي الدائرة المسؤولة عن ؟

- ❖ نقل محتويات العداد PC الى السجل الخاص بتحديد عنوان الذاكرة memory والذي هو السجل MAR(memory address register)
- ❖ نقل محتويات الذاكرة Memory الى سجل التعليم IR
- ❖ زيادة محتوى العداد PC وذلك لكي يؤشر الى الموقع التالي من الذاكرة next location of memory
- ❖ فك شفرة التعليم وتنفيذها execution and decoding في وحدة الحساب والمنطق او أي وحدة اخرى ،

في الحقيقة الوحدة المسؤولة عن هذه العمليات هي وحدة السيطرة Control Unit ، وحيثما تسمى وحدة توليد دورة الماكينة Machine cycle generation ، وحدة السيطرة ترسل اشارات سيطرة Control Signal الى باقي وحدات وحدة المعالجة المركزية CPU ، وذلك لكي تضمن التزامن Synchronization في عمل هذه الوحدات الشكل Figure 2.5 يبين مخطط لوحدة توليد دورة الماكينة،

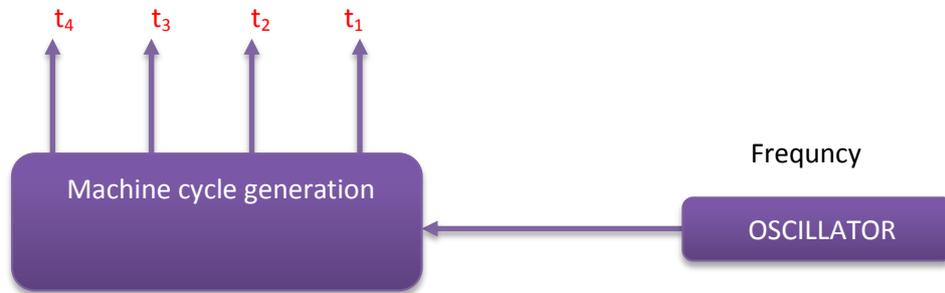


Figure 2.5

وحدة السيطرة (وحدة توليد دورة الماكينة) تحتي على اربع اشارات Signal أو أكثر حسب نوع المعالج Processor ، هذه الاشارات تسمى  $t_1, t_2, t_3, t_4$  ، عند اول نبضة للمذبذب Oscillator ، فإن وحدة السيطرة ترسل اشارة فقط عبر  $t_1$  اما بقية الاشارات تكون غير مفعلة ، تستخدم هذه الاشارة لنقل محتويات عداد البرنامج PC الى وحدة الذاكرة Memory ، عندما تأتي النبضة الثانية من المذبذب فإن وحدة السيطرة ترسل اشارة فقط عبر  $t_2$  كذلك بقية الاشارات تكون غير مفعلة ، تستخدم هذه الاشارة لنقل محتويات وحدة الذاكرة memory الى سجل التعليم ، عندما تأتي النبضة الثالثة من المذبذب فإن وحدة السيطرة ترسل اشارة فقط عبر  $t_3$  وتكون بقية الاشارات غير مفعلة ، تستخدم هذه الاشارة لزيادة محتويات عداد البرنامج pc بواحد ولذلك لكي يؤشر الى الموقع التالي من الذاكرة ، وسبب زيادة عداد البرنامج خلال الفترة  $t_3$  هو عندما تأتي  $t_1$  خلال الدورة التالية الجديدة ، فانه العداد PC سيكون مؤشر الى الموقع التالي وحيث أن  $t_1$  تقوم بنقل محتويات العداد PC الى موقع الذاكرة memory لكي يتم تحديد موقع الذاكرة الذي عنوانه هو محتويات العداد PC ، عندما تأتي النبضة الرابعة من المذبذب فإن وحدة السيطرة ترسل اشارة فقط عبر  $t_4$  وتكون بقية الاشارات غير مفعلة ، تستخدم هذه الفترة لفك شفرة التعليم وتنفيذها وحدة السيطرة هذه مبسطة في الحقيقة تحتاج فك شفرة التعليم وتنفيذها اكثر من فترة زمنية  $t$  ، عندما تأتي النبضة الخامسة من المذبذب فإن وحدة

السيطرة ترسل اشارة فقط عبر  $t_1$  وبقية الاشارات غير مفعلة ، أي تعاد الدورة من جديد ، أذن نستنتج من ذلك ان كل اربع نبضات من المذبذب تنفذ تعليمة كاملة ، وكلما زادت سرعة المذبذب زادت سرعة تعاقب الفترات الزمنية  $t$  ، الجدول Table 2.1 يوضح فترات time وحدة السيطرة،

المذبذب	T <sub>1</sub>	T <sub>2</sub>	T <sub>3</sub>	T <sub>4</sub>	العملية خلال الفترة الزمنية المفردة T	الحالة
النبضة ١	1	0	0	0	نقل محتويات عداد البرنامج الى وحدة الذاكرة	دورة ماكنة الاولى ، نتيجتها تنفيذ تعليمة كاملة
النبضة ٢	0	1	0	0	نقل محتويات الذاكرة الى سجل التعليمه	
النبضة ٣	0	0	1	0	زيادة محتويات عداد البرنامج بواحد	
النبضة ٤	0	0	0	1	فك شفرة التعليمة وتنفيذها	
النبضة ٥	1	0	0	0	نقل محتويات عداد البرنامج الى وحدة الذاكرة	دورة ماكنة الثانية ، نتيجتها تنفيذ تعليمة كاملة
النبضة ٦	0	1	0	0	نقل محتويات الذاكرة الى سجل التعليمه	
النبضة ٧	0	0	1	0	زيادة محتويات عداد البرنامج بواحد	
النبضة ٨	0	0	0	1	فك شفرة التعليمة وتنفيذها	
ما لا نهاية						

Table 2.1

## ❖ مثال 2.1

مذبذب oscillator ذو تردد 1 KHZ ربط الى معالج دورة الماكنة تستغرق اربع فترات زمنية لكي يتم تنفيذ تعليمة واحدة فما هو الوقت الذي تستغرقه التعليمة الواحدة لكي تنفذ وما هو عدد التعليمات التي يمكن تنفيذها في الثانية الواحدة ؟

الحل:

بما أن التردد مقلوب الزمن أي

$$\text{Time} = \frac{1}{\text{Frequency}}$$

أي أن الزمن

$$\text{Time} = \frac{1}{1000} = 0.001 \text{ second}$$

أي أن النبضة  $t$  تستغرق فترة زمنية مقدارها 0.001 من الثانية ، اذن اربع نبضات تستغرق  $4 \times 0.001 = 0.004$  من الثانية ، أي ان فترة تنفيذ تعليمة واحد هي 0.004 من الثانية ، اما عدد التعليمات التي يمكن تنفيذها في الثانية الواحدة هي ،

$$\text{Num. of instruction per second} = \frac{\text{Frequency}}{\text{number of (t) of machine cycle}}$$

عدد التعليمات التي تنفذ في الثانية الواحدة هي 250 تعليمة في الثانية،

$$\text{Num. of instruction per second} = \frac{1000 \text{ HZ}}{4} = 250$$

الشكل 2.6 يوضع المخطط الصندوقي نموذجي مبسط للمعالج CPU ، أما الشكل 2.7 يوضع المخطط الصندوقي مفصل للمعالج CPU هناك وحدة المكس Stack ساترق لها في مواضيع لاحقة انشاء الله لم اضيفها هنا وذلك لتسهيل عملية الفهم ، وكذلك لتسلسل الافكار ،

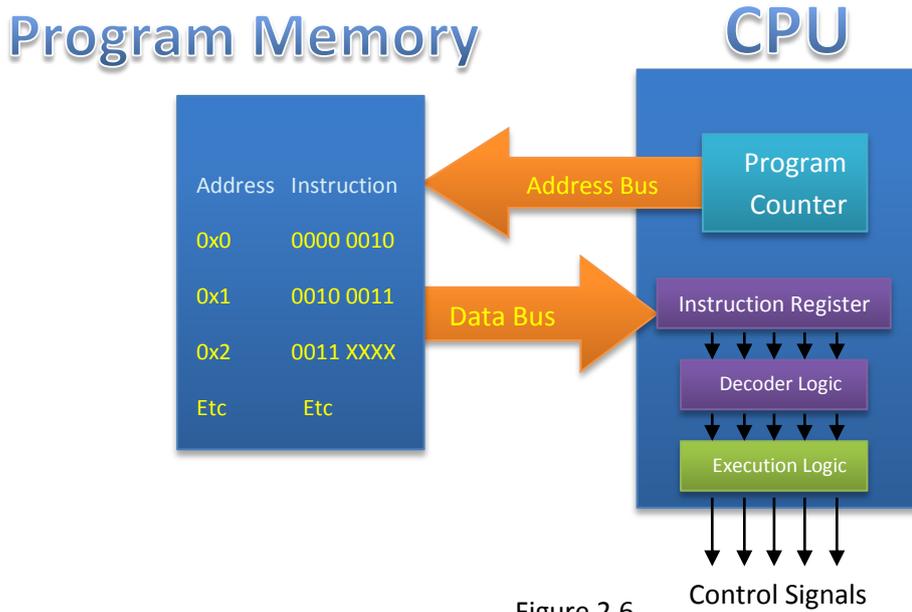


Figure 2.6

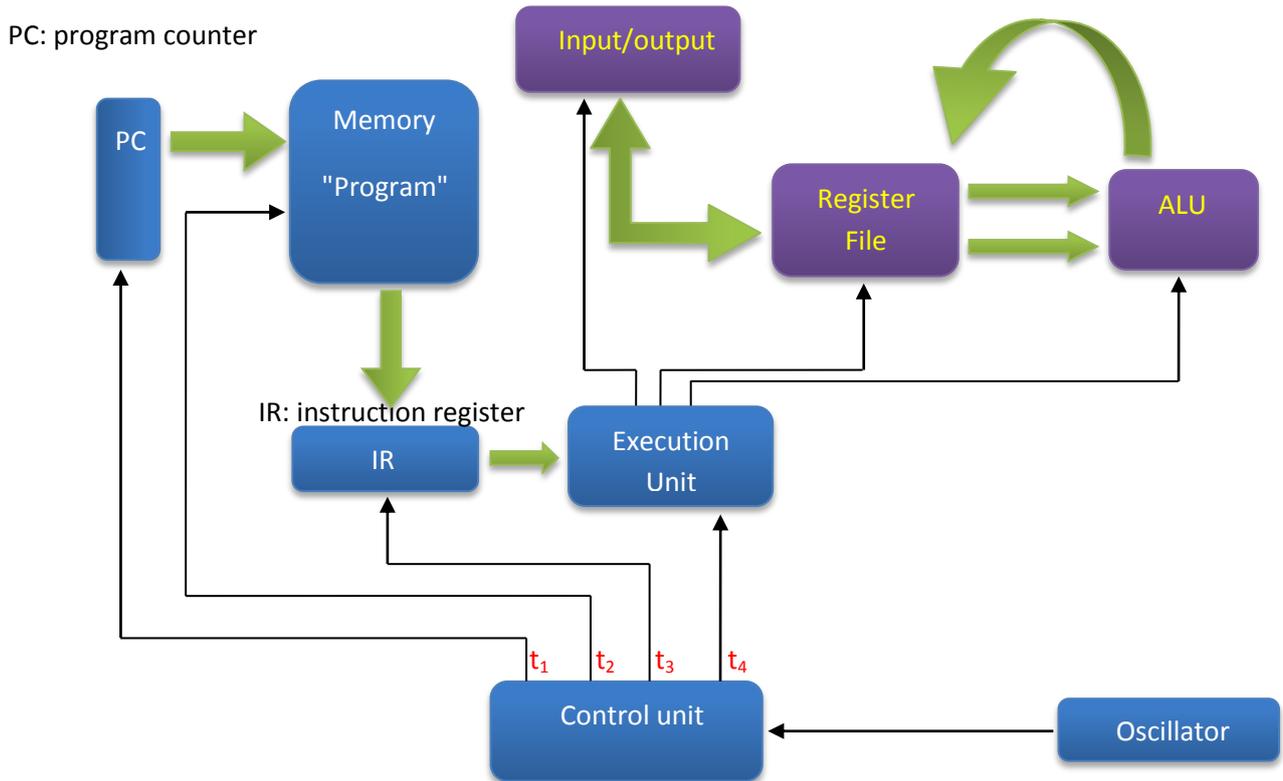


Figure 2.7

لاحظ ان وحدة التنفيذ execution unit ترسل اشارة الى وحدة ALU ، وذلك لكي تخبر ALU بنوع العملية التي سيتم تنفيذها داخل وحدة ALU ، وكذلك وحدة التنفيذ ترسل اشارة الى Register file وذلك لتحديد عناوين السجلات المراد اجراء العملية عليها وارسالها الى وحدة

ALU ، كما ترسل وحدة التنفيذ execution unit اشارات الى أطراف الادخال والاخراج وذلك لتهيئتها لعملية ادخال البيانات Data او اخراجها ، من الشكل اعلاة تلاحظ كذلك هناك ثلاث انواع من الناقل البيانات Buses هي

- ❖ ناقل العناوين Address Bus : وهو الناقل الذي ينقل عناوين Address ، مثل الناقل الذي يربط عداد البرنامج PC بوحدة الذاكرة ،
- ❖ ناقل البيانات Data Bus : هو الناقل الذي ينقل البيانات Data ، مثل الناقل الذي يربط وحدة الذاكرة مع سجل التعليم IR ، وكذلك الناقل الذي يربط وحدة ALU مع Register file ،
- ❖ ناقل السيطرة Control Buss : وهو الناقل الذي يرسل اشارات السيطرة ليضمن عمل وتزامن الوحدات الاخرى ، مثل هذه الاشارات هي اشارات وحدة السيطرة  $t_1, t_2, t_3, t_4$  ، اشارات السيطرة موضحة في الشكل اعلاة بشكل اسهم Arrow

لكي يؤدي CPU عملة ، يحتاج الى توجيه ، عملية التوجيه تتم عن طريق سلسلة من الأوامر instruction تُسمى برنامج .. هذا البرنامج يخزن داخل ذاكرة memory .....

## Memory

## وحدة الذاكرة

وهي الوحدة التي يخزن فيها البرنامج الذي سينفذ من قبل وحدة المعالجة المركزية CPU، وهناك نوعان رئيسيان من الذاكرة

### RAM(Random Access memory)

### ١- ذاكرة الولوج العشوائي

وهي ذاكرة تفقد معلوماتها بعد انقطاع التيار الكهربائي عنها ، عادة تستخدم لخرن البيانات او المعطيات التي تعالج من قبل وحدة المعالجة المركزية CPU.

### ROM(Read only memory)

### ٢- ذاكرة القراءة فقط

وهي ذاكرة لا تفقد معلوماتها حتى بعد انقطاع التيار الكهربائي عنها وهي على أنواع أهمها ذاكرة EPROM ، وهي ذاكرة قابلة للمسح وتحفظ ببياناتها حتى بعد انقطاع مصدر التغذية عنها ، لذلك تستخدم هذه الذاكرة لخرن البرنامج Program ..

## Internal Structure of Memory

## البنية الداخلية للذاكرة

تتكون الذاكرة بصورة عامة من مجموعة مواقع locations لكل موقع من هذه المواقع عنوان محدد Address، عن طريق هذا العنوان يمكن الوصول الى الموقع المطلوب واخذ البيانات من الموقع المحدد، يُرسل العنوان عن طريق ناقل العناوين address bus ، تخرج البيانات المطلوبة Data من الذاكرة عن طريق ناقل يسمى ناقل البيانات Data Bus كما اشرنا الية سابقاً ، كما أن هناك اشارات تستخدم لتحديد العملية التي تجري على الذاكرة مثل اشارة Write التي تعني بها اننا نريد خزن المعلومات داخل الذاكرة ، و اشارة Read التي تعني بها أننا نريد قراءة محتويات الذاكرة ، الشكل Figure 2.8 يوضح المخطط الصندوقي النموذجي للذاكرة،

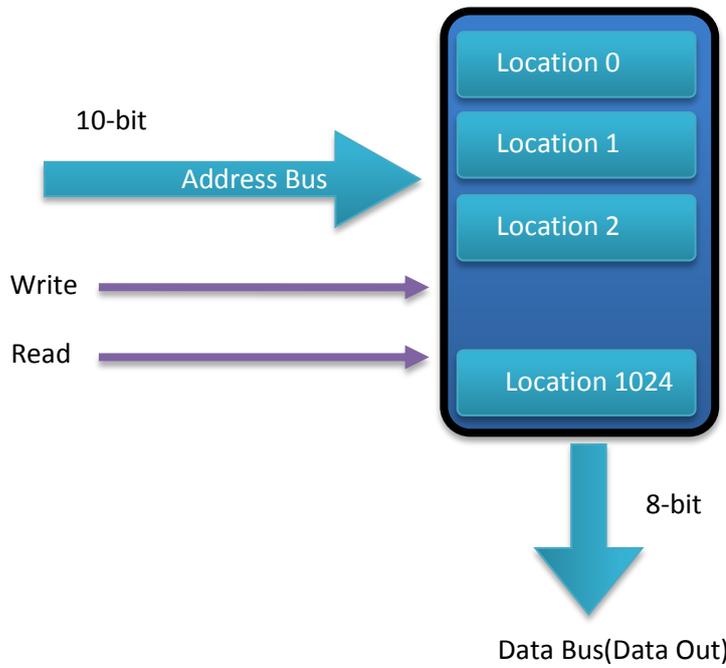


Figure 2.8

كما أن هناك علاقة بين عدد البتات لناقل العناوين وعدد مواقع الذاكرة التي هي ،

$$\text{Number of Locations} = 2^{\text{Number of bits of Address Bus}}$$

مثلاً اذا عدد بتات ناقل العناوين 14-bit فإن عدد المواقع هو 1024 موقع ،

$$\text{Number of Locations} = 2^{10\text{-bit}} = 1024$$

## Von-Neumann vs Harvard structure

## بنية فون نيومان وبنية هارفارد

تقسم الذاكرة حسب نوع البيانات المخزنة الى قسمين،

program memory  
data memory

❖ ذاكرة البرنامج  
❖ ذاكرة المعطيات

قد اقترح العام Von-Neumann على أن البرنامج Program والمعطيات Data ، تقع في نفس رقاقة الذاكرة ، هذا النظام متبع في الحواسيب الشخصية Personal computer ، أما العالم Harvard أقترح على أن البرنامج Program والمعطيات Data مفصولة عن بعضهما البعض ، أي أن كل قسم يقع على رقاقة ، هذا النظام متبع في المسيطرات الدقيقة Microcontroller ، الشكل Figure 2.9 يوضح ذلك ....

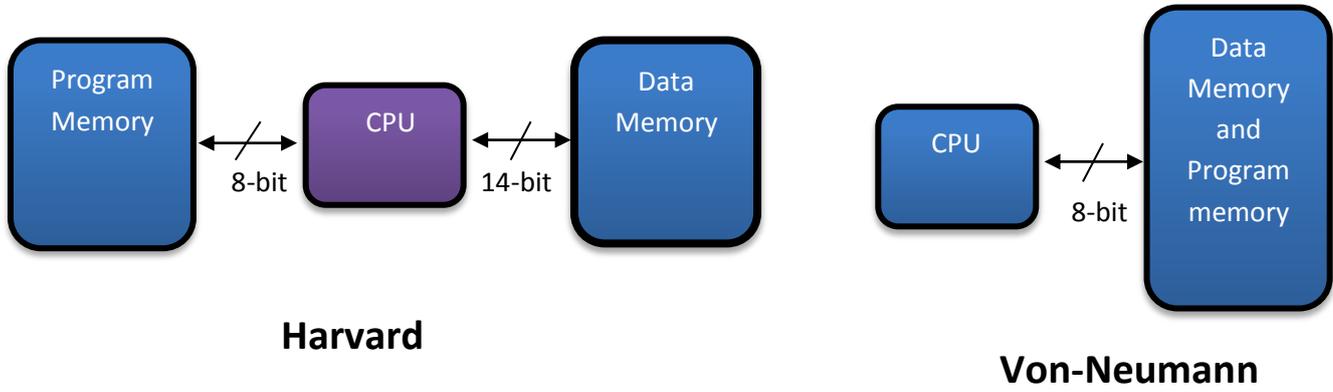


Figure 2.9

ملاحظة:

ذاكرة البرنامج program memory هي الذاكرة التي يخزن فيها التعليمات sinstruction ، وذاكرة البيانات Data memory ، هي الذاكرة التي تخزن فيها المعطيات Operand او المعاملات المراد معالجتها ، وغيرها من البيانات الضرورية ، مثلاً تخزين القيم ثابتة مثل النسبة الثابتة  $\pi$  التي قيمته 3.14 وذلك للاستفادة منها اثناء معالجة البرنامج ،

## Input Unit

## وحدة الإدخال

وهي الوحدة التي من خلالها ، تدخل الاشارات الرقمية Digital Signal او البيانات Data الى النظام وذلك للاستفادة من هذه البيانات ومعالجتها، اشارات الادخال ممكن أن تأتي من متحسسات Sensors أو مفاتيح Switch ، او من وحدات التحويل التماثلي الى الرقمي Analog to Digital Converter (ADC) الشكل Figure 2.10 يوضح تركيب مبسط لوحدة الادخال ، لاحظ أن البيانات الداخلة تخزن في File Register ،

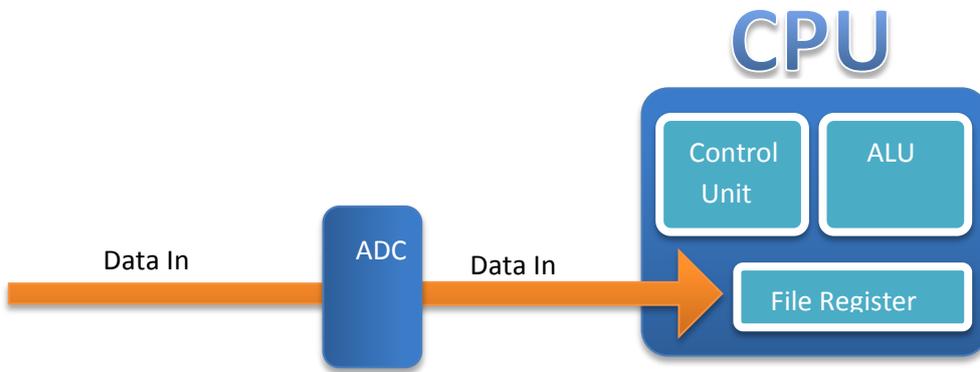


Figure 2.10

## Output Unit

## وحدة الإخراج

وهي الوحدة التي من خلالها ، تخرج الاشارات الرقمية Digital Signal او البيانات Data من النظام الى وحدات الإخراج مثل الثنائيات الباعثة للضوء LED او العارضات ذات السبع قطع 7 Segment ، او الاشارات الرقمية ممكن أن تربط الى محولات الرقمية الى تماثلية Digital to Analog Converter(DAC) وذلك لتغذية وحدات اخراج تعمل على الاشارات التماثلية ، لاحظ في الشكل Figure 2.11 البيانات الخارجة تخزن في File Register ،

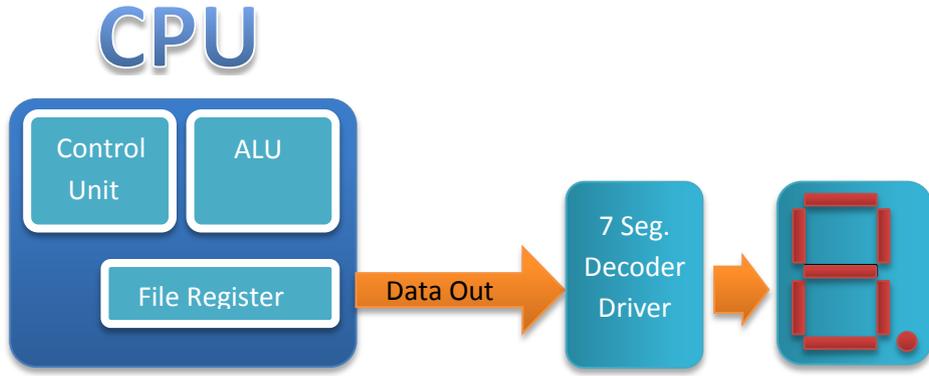


Figure 2.11

في المسيطرات الدقيقة Microcontroller تجد منفذ واحد يستخدم في عمليات الادخال وفي عمليات الاخراج ، عند التعامل مع هذه المنافذ ، يجب اولاً تهيئة المنفذ Configuring the ports ، أي اخبار المنفذ هل هو منفذ ادخال أو منفذ أخراج أو جزء منه ادخال والجزء الاخر اخراج حسب متطلبات الحاجة ، عادة يوجد موقع خاص ضمن File Register هذه الموقع تستطيع من خلاله تحديد نوع المنفذ ، لاحظ الشكل Figure 2.12

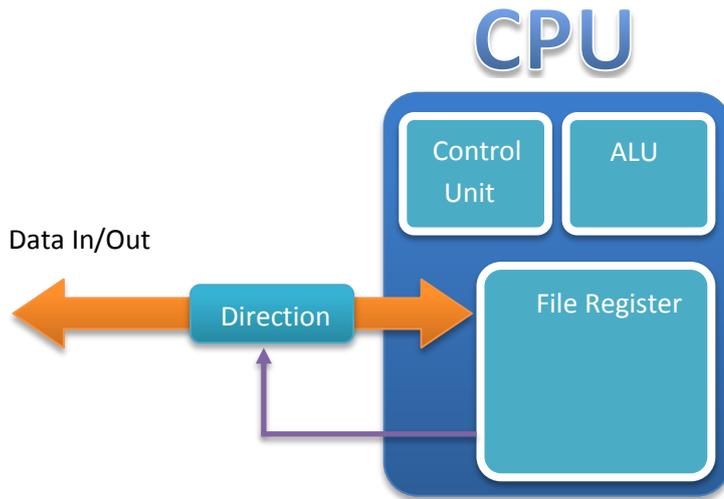
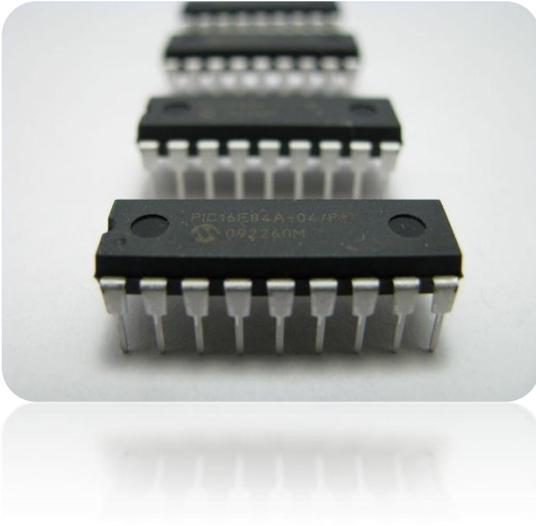


Figure 2.12



## PIC16f84A

## بُنْية المُسَيِّطِر الدَّقِيق

مُقدمة:

في هذه الوحدة سوف نتعرف على بُنية المُسَيِّطِر الدَّقِيق Architecture of Microcontroller وعلى تَنظِيم ذاكرة المُسَيِّطِر الدَّقِيق Memory Organization ، كما سنتعرف من خلال هذه الوحدة على كيفية تهيئة المسيطر الدقيق للعمل ، أرجوا من القارئ الكريم قراءة هذه الوحدة بتركيز ، لأن هذه الوحدة تُعتبر أساس الفهم الصحيح لبرمجة المُسَيِّطِر الدَّقِيق و عملة .

## ما هو المُسيطر الصغري Microcontroller ؟

المسيطر الدقيق Microcontroller عبارة عن شريحة الكترونية Chip ، مضمنة Embedded داخل منظومة system أو دائرة لتأدية وظائف معينة ، يُستخدم المسيطر الدقيق في حياتنا اليومية بشكل واسع مثل ؟

- ❖ أجهزة الالعاب اللاكترونية Video Games
- ❖ أجهزة الكهريائية مثل Split, Air-conditions
- ❖ الأجهزة المكتبية مثل Digital Clocks
- ❖ أجهزة السيطرة الرقمية PID,PLC

يتوفر المُسيطر الدقيق في السوق بشكل دائرة متكاملة (IC) integrated circuit ، بأبعاد عدة سنتيمترات Centimeter ، الشكل Figure 3.1 يوضح شريحة مُسيطر دققيق ، يوجد داخل هذه الدائرة المتكاملة IC ، معالج Processor ،

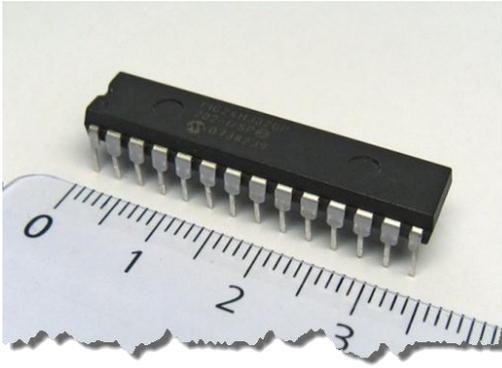


Figure 3.1

وذاكرة لخرن البرنامج ، وتحتوي على أطراف لأدخال وأخراج البيانات Data او الاشارات Signals ، في أنواع معينة توجد محولات أشارات تماثلية الى رقمية Analog to Digital Converter (ADC) ، ومحولات أشارات رقمية الى تماثلية Digital to Analog Converter (DAC) ، كما يحتوي المسيطر الدقيق على وظائف إضافية مثل المؤقتات Timer والعدادات Counter ، ومعدلات الاشاره Pulse Width Modulation (PWM) لاستخدامها في عمليات تضمين الاشارات وذلك بغية ارسالها .

سوف نتحدث في هذه الوحدة عن بُنية أحد انواع الأجهزة القابلة للبرمجة وهو المُسيطر الدقيق Microcontroller من شركة Microchip ، تنتج شركة Microchip سيلة من المُسيطرات الدققيقة Microcontroller تحت أسم PIC(Programmable interface Controller) ، من هذه السلسلة هو المتحكم PIC16F84A ، يتصف هذا المسيطر بالمواصفات التالية،

- ❖ يحتوي هذا المسيطر على 35 تعليمة فقط .
- ❖ جميع التعليمات تأخذ دورة ماكنة واحدة One machine cycle ما عدا تعليمات التفرع jump instruction فأنها تأخذ دورتان ماكنة.
- ❖ يعمل على تردد 4 Mega Hertz ، دورة الماكنة لهذا المسيطرة مكونة من 4 نبضات لاتمام تنفيذ تعليمة كاملة ، أي أن فترة الزمنية التي تستغرقها التعليمة الواحدة هو 4 micro-second .
- ❖ يحتوي على ذاكرة برنامج program memory قدرها 1 kilo-byte
- ❖ يحتوي على مؤقت Timer يستخدم في عمليات التوقيت Timer والعد Counting
- ❖ يحتوي على ذاكرة EEPROM لخرن البيانات Data واسترجاعها.

بُنْيَة المُسَيِّطِر الدَّقِيق لا تختلف كثيراً عن بُنْيَة باقي الأجهزة القابلة للبرمجة ، مثل Microprocessor وغيرها ، مُعْظَم الأجهزة القابلة للبرمجة تتكون من شقين

Hardware  
Software

❖ البنية الصلبة  
❖ بنية البرمجيات

## Hardware

## البُنْيَة الصَّلْبَة

وهي المكونات المادية الملموسة Physical Component من الاجهزة القابلة للبرمجة ، مثل جهاز الحاسوب يتكون من لوحة المفاتيح ، المعالج ، الذاكرة ، هذه المواد كلها تعتبر مكونات صلبة Hardware ، اما نظام التشغيل والبرمجيات فتعتبر مكونات البرمجيات Software التي تدير وتشرف على عمل المكونات الصلبة Hardware ، في هذه الوحدة سنتطرق لبُنْيَة المُسَيِّطِر الدَّقِيق الصَّلْبَة Hardware اما البرمجيات فسنتركها لوحدها لاحقة ، الشكل Figure 3.2 يوضح بنية المسيطر الدقيق PIC16f84A ،

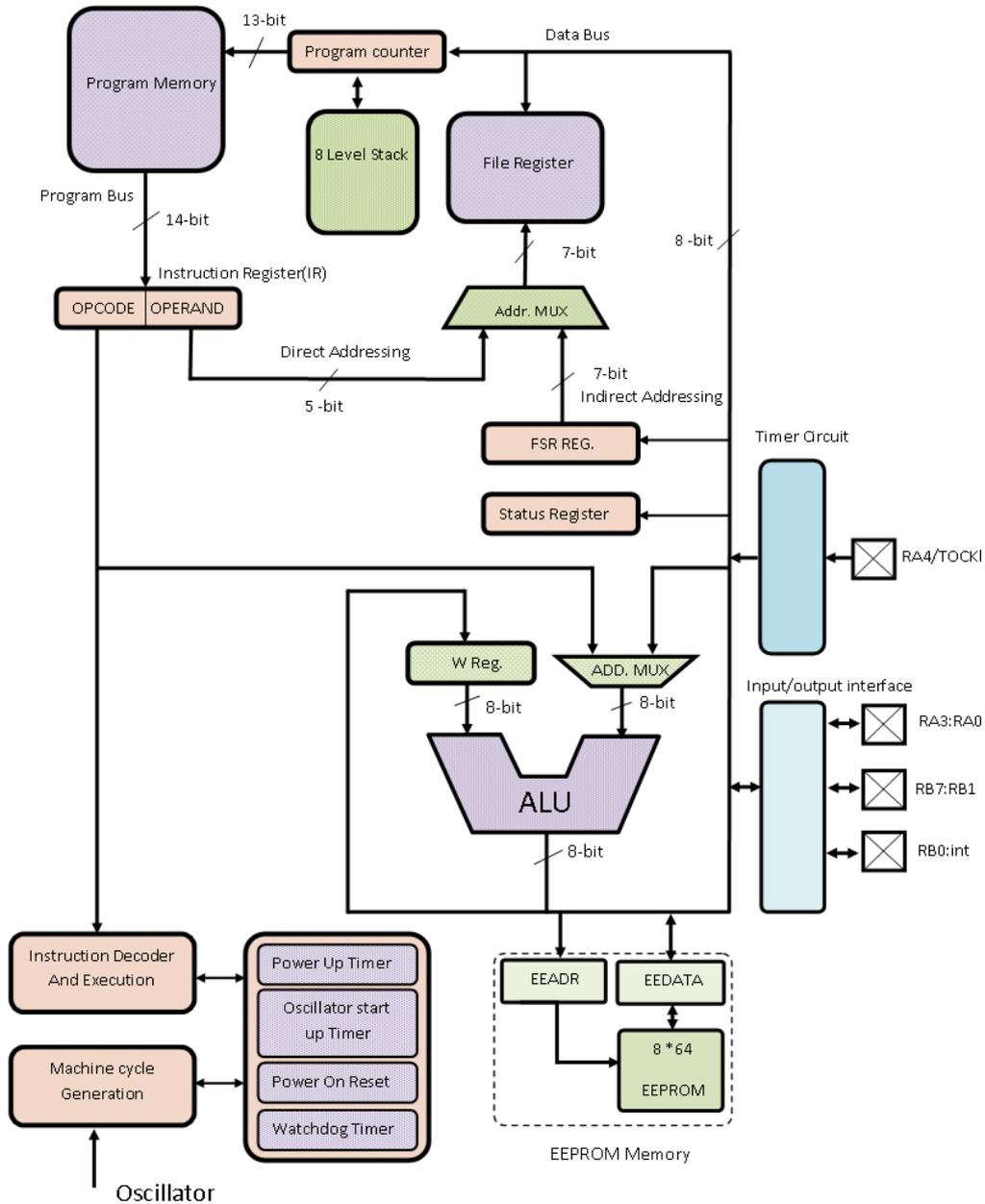


Figure 3.2

وحدة المعالجة المركزية CPU كما مر علينا في الوحدة السابقة هي قلب المُسيطر الدقيق Microcontroller ، وهي المسؤولة عن جلب وفك شفرة وتنفيذ التعليمات Instruction ، كما عرفنا سابقاً أن دورة الماكينة تبدأ بعملية Fetch أي جلب التعليمات من ذاكرة البرنامج program memory لتخزن في مسجل التعليمات IR(instruction Register) وفق الخطوات التالية :

١- يقوم عداد البرنامج PC(Program Counter) بعنوانه الذاكرة كما هو موضح في الشكل Figure 3.2 ، وذلك لالتقاط التعليمات من الذاكرة ، لاحظ أن المسيطر PIC16f84A طول عداد البرنامج PC له هو 13-bit .

٢- التقاط التعليمات من ذاكرة البرنامج Program Memory التي عنوانها هو محتويات عداد البرنامج PC ، لينتهي بها المطاف في مسجل التعليمات Instruction Register ، لاحظ أن طول الكلمة Word التي تخرج من ذاكرة البرنامج هو 14-bit ، لاحظ الشكل Figure 3.2 ، أن الغاية الأساسية لاغلب التعليمات في المسيطر الدقيق Microcontroller هو استهداف أو الوصول File Register ، هناك طريقتان لعنونة أو الوصول الى File Register هما ؟

١- العنونة المباشرة Direct Addressing

٢- العنونة الغير مباشرة Indirect Addressing

## Direct Addressing

## العنونة المباشرة

أن File Register ، عبارة عن وحدة خزنية ، تستخدم لخزن البيانات المهمة والمراد معالجتها ، وكذلك تحتوي على مواقع لها علاقة باعدادات Configuration وحدات الإدخال والأخراج وغيرها ، موضوع File Register سيتم مناقشته باسهاب لاحقاً ، لنأخذ أول ايعاز لنا في المسيطر الدقيق Microcontroller الذي هو ،

BSF Reg. , b

تقوم هذه التعليمات أو الأيعاز برفع احد بتات مسجل Register الى القيمة واحد ، تتكون هذه التعليمات من جزئين ،

- ١- الجزء الأول وهي شفرة العملية Opcode (operation code) التي هي BSF وهي اختصار لكلمة Bit set flag ، التي تخبر المعالج أو المسيطر الدقيق بالوظيفة التي نريد اجراءها التي هي جعل إحدى بتات bit مسجل Register قيمة واحد.
- ٢- الجزء الثاني هو المُعاملات Operand الذي هو Reg. الذي يقصد به المسجل الهدف الذي نريد رفع احد بتاتة الى القيمة واحد ، والجزء b والذي هو رقم البت المراد جعل او رفع قيمته الى الواحد من المسجل المستهدف ،



ملاحظة:

أن سعة المسجلات Registers الموجوده في File Register هو 8-bit ، لذلك يطلق على المسيطر الدقيق PIC16f84A أنه نظام ذو 8-bit ،

### ❖ مثال 3.1

أحد المسجلات الموجودة في file Register هو المسجل الذي عنوانه 0x0C بالنظام السادس عشر، المطلوب جعل قيمة البت bit الثالث تساوي واحد من هذا المسجل ؟

الحل:

```
BSF 0x0C , 2 ; 0000 0100
```

ملاحظة: 

توضع السابقة 0x قبل الرقم وذلك للدلالة على أن الرقم بالنظام السادس عشر مثل 0x02,0x08,0x0D Hex

لاحظ أن المسجل 0x0C قد تم الوصول اليه مباشرة من التعليمة BSF 0x0C,3 وهذا ما يسمى بالعنونة المباشرة ، لنعود الى الشكل Figure 3.2، ان طول الكلمة الخارجة من الذاكرة 14-bit وهي التي ستحمل التعليمة BSF 0x0C,3 الى مسجل التعليمة IR ، وعند وصولها تنشط الى جزئين كما هو موضح في الشكل Figure 3.2 ، جزء شفرة العملية Opcode ليذهب الى وحدة فك الشفرة التنفيذ Instruction Decoder And Execution ، وجزء المعاملات Operand حاملاً المعاملات التي تمثل عنوان المسجل المراد اجراء العملية عليه عبر ناقل العنونة المباشرة Direct Addressing الى file Register لاحظ في الشكل Figure 3.2 هناك شيء اسمه Addr. MUX وهو اختصار Address multiplexing أي مازج العنوان وهو مثل المفتاح الذي يحول بين العنونة المباشرة والعنونة الغير مباشرة ، في عملية العنونة المباشرة سيرتبط File Register مع ناقل العنونة المباشر Direct Addressing عن طريق المازج Addr. MUX،

## Direct Addressing

## العنونة الغير المباشرة

يقصد بالعنونة الغير مباشرة هو أن عنوانه أو الوصول الى أحد مسجلات File Register لا يتم عن طريق التعليمة instruction ، بل يتم عن طريق مسجل آخر خاص وهو المسجل FSR(file select Register) لاحظ الشكل Figure 3.3 ، اذا حمل هذا المسجل على سبيل المثال بالقيمة FSR = 0X0C فان هذه المسجل FSR يشير بذلك الى المسجل الذي عنوانه 0X0C من File Register ، وتضهر محتويات المسجل 0X0C الذي عنون عن طريق المسجل FSR على مسجل يسمى INDF(indirect register flag) الذي هو موجود ضمن File Register ليتم معالجتها ، هذا الموضوع سيتم مناقشته بشكل اوسع في وحدات لاحقة ،

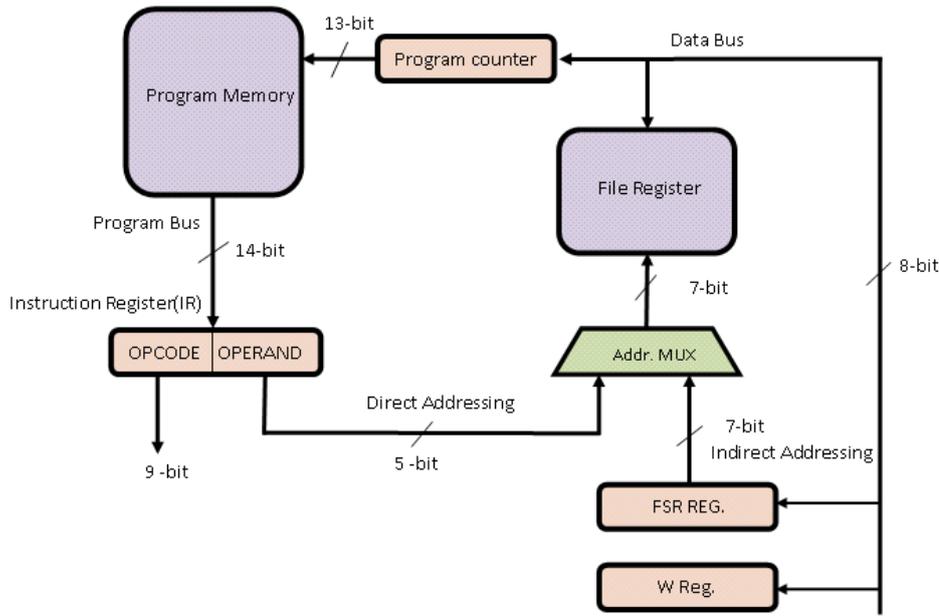


Figure 3.3

الشكل Figure 3.3 هو جزء من بنية معالج الميسر PIC16f84A ، عمدت الى تقسم بنية المعالج وذلك لكي يسهل عملية فهمها واستيعابها ، لفهم ما هو مسجل العمل Working Register او W Reg. الموجود في الشكل Figure 3.2 يجب علينا فهم آلية عمل وحدة الحساب والمنطق ALU في الميسر PIC16f84A ،

## ALU (Arithmetic and logic unit)

## وحدة الحساب والمنطق

وهي الوحدة المسؤولة عن إجراء العمليات الحسابية والمنطقية ، وأهم ما في وحدة الحساب والمنطق هو مسجل العمل W(work register) ، وهو مسجل يستخدم بكثرة في الميسر PIC16f84A ، لكي نفهم آلية عمل وحدة الحساب والمنطق ALU ، يجب ان ننتبه الى ،

- ❖ لاحظ في الشكل Figure 3.4 ، أن أحد مُعاملات أو مدخلات وحدة الحساب والمنطق ALU هو مسجل العمل W Reg. ، أي عند إجراء عملية الجمع أو أي عملية داخل وحدة الحساب والمنطق ALU يجب أن يكون مسجل العمل W Reg. مُحمل بقيمة أحد معاملات العملية الحسابية ،
- ❖ لاحظ في الشكل Figure 3.4 ، ان الطرف الثاني او المعامل الثاني لوحدة ALU ، يحمل من اتجاهين ، المعامل الثاني أما ان يكون رقم Literal قادم من التعليمه مباشرة ، او ان المعامل الثاني قادم من أحد مسجلات File Register .
- ❖ لاحظ ان ناتج وحدة ALU أما أن يخزن في مسجل العمل ، وذلك عند ما تكون الاشارة D=0 او يخزن في احد مسجلات File Register عندما تكون D=1 ، يتم تعيين قيمة D من التعليمه مباشرة .

كما ان لمسجل العمل فوائد كثيرة ، حيث انه الوسيط في اكثر تعليمات instruction الميسر PIC16f84A ، وهذا ما سوف نكتشفه في وحدة البرمجة أنشاء الله.

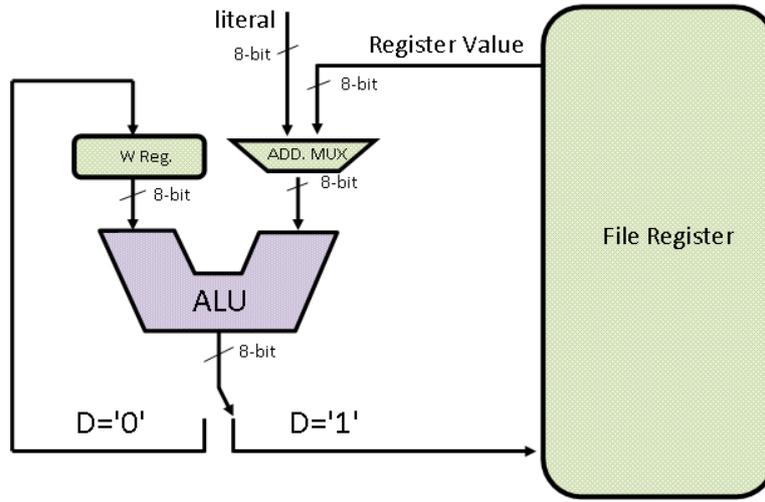


Figure 3.4

## Status Register

## مسجل الحالة

هناك مسجل ضمن بنية المُسَطَّر الدَّقِيق PIC16f84A يُسمى مُسَجَل الحالة Status Register أو مُسَجَل الأَعْلَام Flag register وسمي بِمُسَجَل الحالة لِأَنَّهُ يُسَجَل حالة أو نَتَاج العَمَلِية التي أُودِيت داخِل وِحدة الحِساب والمَنْطِق ALU ، وهو مُسَجَل مَكون من 3-bit كُل بَت يَشِير إلى حالة من الحَالات التَّالِية ،

- ١- عَلم التَّصْفِير Zero flag : عَندما يَصبح نَتَاج عَمَلِية حِسابِية داخِل وِحدة ALU يَساوي صَفر فإن هَذا العَلم يَرفع إلى الوَاحِد وإلا فِانهُ يَصبح صَفر .
- ٢- عَلم المَحمَل Carry flag : يَرفع هَذا العَلم إلى الحَالة وِاحِد عَند ظَهور مَحمَل Carry ، من الخَانة الأَكثَر أَهمِية MSB وإلا اصَبح قِية هَذا العَلم صَفر .
- ٣- عَلم المَحمَل التَّانوي Decimal carry flag : يَرفع هَذا العَلم إلى الحَالة وِاحِد عَند ظَهور مَحمَل من البَت التَّالِث إلى البَت الرَّابِع وإلا تَصبح قِية هَذا العَلم صَفر .



حَالات الأَعْلَام التَّالِثة هَذه تَجدُه مَضمَنة داخِل مَسَجَل في File Register يُسمى مَسَجَل الحالة Status ، الشَكل Figure 3.5 يَبيِن أَشارَات خارجة من وِحدة ALU ذاهِبة إلى مَسَجَل الحالة Status Register ،

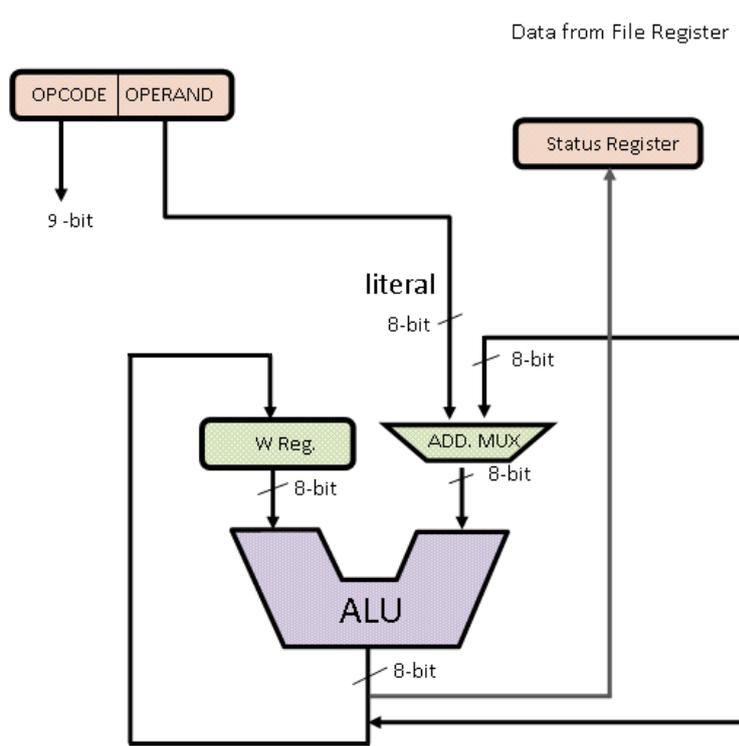


Figure 3.5

## Control Unit

## وحدة السيطرة

بعد أن تتم عملية الجلب Fetch cycle تصل التعلية الى مسجل التعلية IR تنتشر الى جزئين جزء المعاملات Operand الذي يذهب عن طريق ناقل العنوان المباشرة Direct Addressing ليصل الى Register File عن طريق مزج العنوان Address Multiplexer ، والجزء الأخر هو الأيعاز او شفرة العملية Opcode الذي يذهب الى وحدة فك الشفرة لكي تميز الشفرة وتنفذ Decoding and execution ، ان الوحدة المسئولة عن كل هذه العمليات هي وحدة السيطرة او وحدة توليد دورة الماكنة Machine Cycle Generation ، أن وحدة السيطرة في المُسيطر الدقيق PIC16f84A تتكون من أربع نبضات تسمى  $Q_0, Q_1, Q_2, Q_3$  الشكل Figure 3.6 يبين مخطط صندوقي بسيط لوحدة توليد دورة الماكنة ،

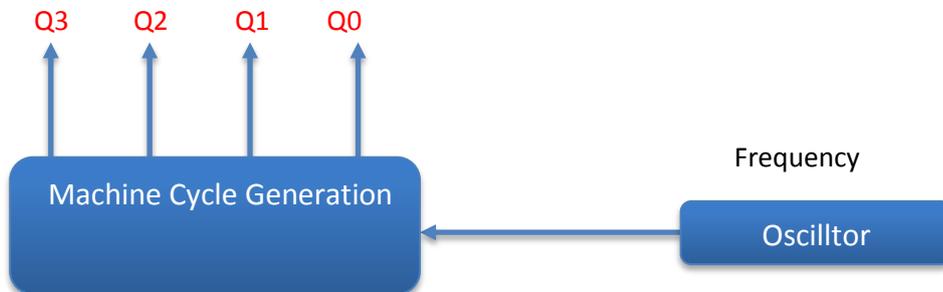


Figure 3.6



## General Purpose Register(GPRs)

تستخدم هذه المسجلات للأغراض العامة ، مثل خزن المعاملات Operands ، خزن نواتج العمليات Result ، وغيرها من العمليات تبدأ هذه المسجلات عند bank 0 من الموقع 0Ch وتنتهي عند الموقع 4Fh ، أما عند bank 1 فإنها تبدأ من الموقع 8Ch وتنتهي عند الموقع CFh .

## مسجلات الأغراض الخاصة

## Special Purpose Register(SPRs)

تستخدم هذه المسجلات للأغراض الخاصة ، مثل تهيئة أطراف Peripheral المسيطر الدقيق PIC16f84A ، تهيئة المؤقتات والعدادات وغيرها من العمليات ، لكل موقع من هذه المواقع له وظيفة معينة ستتعرف عليها خلال مواصلتك لقراءة الكتاب ، تبدأ هذه المسجلات عند bank 0 من الموقع 00h وتنتهي عند الموقع 0Bh ، أما عند bank 1 فإنها تبدأ من الموقع 80h وتنتهي عند الموقع 8Bh .

ملاحظة : 

لأخذ أنه عند bank0 فإن المواقع من 50h الى 7Fh فهي غير منفذة unimplemented ، أي متروكة وكذلك بالنسبة لـ bank1 فإن المواقع من D0h الى FFh ،

## Program Memory

## ذاكرة البرنامج

يحتوي المسيطر الدقيق PIC16f84A عداد برنامج Program Counter ذو سعة 13-bit ، أي أنه يستطيع عنونة ذاكرة سعتها  $2^{13}=8191$  أي 8K byte ما يقابلها بالنظام السادس عشر هو 1FFF ، في الحقيقة العناوين المنفذة implemented من الذاكرة كما هو موضح في الشكل Figure 3.8 هي من (000-3FF) فقط أما العناوين (4FF-1FFF) فهي غير منفذة unimplemented أي أنه لا يمكن تخزين برنامج ضمن هذه المواقع ، كما يرتبط مع عداد البرنامج PC بصورة وثيقة ذاكرة المكسد Stack التي تستخدم لخرن قيمة عداد البرنامج PC في تعليمات القفز jump instruction، انتبه الى العنوان 0000h من ذاكرة البرنامج Program memory يسمى هذا العنوان شعاع التصفير Reset Vector أي عند تشغيل المسيطر الدقيق أو إعادة اقلاعة Restart فإنه يبدأ التنفيذ من هذا العنوان ، وانتبه أيضاً الى العنوان 0004h فإنه يسمى شعاع مقاطعة الأطراف Peripheral Interrupt Vector أي عند حدوث مقاطعة للمسيطر الدقيق فإنه يبدأ التنفيذ من هذا العنوان ، سيتم شرح المقاطعات في فصول لاحقة أن شاء الله .

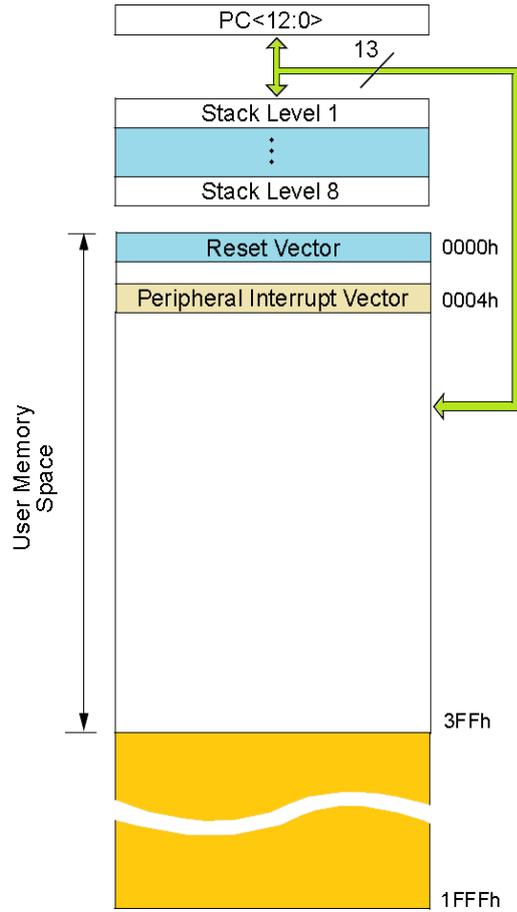


Figure 3.8

## Power Up Timer (PWRT)

## مؤقت بداية اقلاع الطاقه

هناك دائرة مؤقت Timer circuit داخل وحدة المُسيطر الدقيق Microcontroller ، تقوم هذه الدائرة بتوفير فترة تأخير مقدارها 72 ملي ثانية قبل ان ينهض (يعمل) المسيطر الدقيق microcontroller وذلك لكي تضمن استقرار الطاقة لان عدم استقرار مصدر الطاقة يؤثر على عمل المسيطر الدقيق .

## Oscillator Start Up Timer(OST)

## مؤقت بداية اقلاع المذبذب

تعمل دائرة المؤقت هذه عمل power up timer ، حيث توفر هذه الدائرة فترة تأخير زمنية قبل ان تعمل وحدة السيطرة Control Unit التي يغذيها المذبذب وذلك لكي تضمن استقرار الذبذبة لان عدم استقرار الذبذبة يؤثر على عمل وحدة السيطرة وبذلك عمل المُسيطر الدقيق.

عند تجهيز المسيطر الدقيق Microcontroller بالطاقة لغرض تشغيله Running ، يجب التأكد من أن مستوى الطاقة المطلوب مزود بشكل صحيح لكي يعمل المسيطر الدقيق بشكل صحيح ، لحسن الحظ يحتوي المسيطر الدقيق Microcontroller على دائرة تتأكد من ان تجهيز الطاقة وصل للحد المطلوب والا أعاد الجهاز الإقلاع من جديد Reset تسمى هذه الدائرة Power On Reset(POR).

### مؤقت الحراسه

### Watchdog timer

ماذا يحدث لو علق Stuck المسيطر الدقيق، هل هناك زر restart لاعادة تشغيل المسيطر الدقيق ، طبعاً لا ،فماذا نفعل لو كان المسيطر الدقيق في موقع عمل field ودخل في حالة العلق Stuck ، توجد دائرة داخل المسيطر الدقيق Microcontroller تقوم بعملية مراقبة المسيطر الدقيق ، اذا علق Stuck المسيطر الدقيق تقوم هذه الدائرة باعادة اقلع المسيطر الدقيق Microcontroller تسمى هذه الدائرة بمؤقت الحراسة Watchdog timer ، سيتم شرح هذا الموضوع بشكل مفصل في فصول لاحقة .

### المكدس

### Stack Memory

يحتوي المسيطر الدقيق PIC16f84A على مكدس Stack ذو سعة 8 مواقع ، يستخدم في عمليات القفز ،

### الذاكرة EEPROM

### EEPROM Memory

يحتوي المسيطر الدقيق PIC16f84A على ذاكرة EEPROM قابلة للقراءة والكتابة ذات سعة 64 موقع ، وكل موقع ذو طول كلمة 8-bit ، تستخدم لخرن المعلومات ، حيث ان هذه المعلومات ستضل مخزونة حتى بعد فصل مصدر التغذية عن الشريحة ، ترتبط مع هذه الذاكرة مسجلين ، مسجل العنوان EEADR عند تحميل هذا المسجل بقيمة معينة فاننا بذلك نشير الى موقع من مواقع ذاكرة EEPROM الـ 64 ، عند خزن بيانات عند موقع الذاكرة المعنون عن طريق مسجل EEADR فاننا نحمل البيانات Data المراد خزنها الى المسجل EEDATA ، وكذلك عند قراءة معلومات من الذاكرة EEPROM المعنون عن طريق مسجل EEADR فاننا نجد البيانات Data قد حملت الى المسجل EEDATA ، عمليات القراءة والكتابة تتم عن طريق تعليمات واعدادات خاصة سنتناولها في فصول لاحقة ان شاء الله .

### المؤقت

### Timer

يحتوي المسيطر الدقيق على مؤقت يستخدم في عمليات التوقيت Timing كما يستخدم في عمليات العد Counting ، سيتم شرح الموضوع في فصول لاحقة .

### أطراف الادخال والايخراج

### Input/Output Unit

يحتوي المسيطر الدقيق على منفذين Ports لأدخال أو أخراج البيانات الى او من المسيطر الدقيق ، المنفذ الاول يسمى Port A وهو منفذ مكون من 5-bit ممكن أن يستخدم هذا المنفذ كمنفذ أدخال أو أخراج، وعناوين هذا المنفذ هي RA0,RA1,RA2,RA3,RA4 ، والمنفذ الاخر يسمى Port B وهو منفذ مكون من 8-bit ، وكذلك يمكن ان يستخدم هذا المنفذ كمنفذ أدخال او أخراج،وعناوين هذا المنفذ هي RB0,RB1,RB2,RB3,RB4,RB5,RB6,RB7 ، كما أن هناك أطراف لها اكثر من وظيفة مثل الطرف RA4 يستخدم كطرف قدح للمؤقت Timer أو للعداد ويسمى TOCKI ، كما يستخدم الطرف RB0 كوظيفة المقاطعة Interrupt ،

## PIC16F84A

## الوصف الدقيق لأطراف المسيطر

يتم تغليف Packaging المسيطر الدقيق ، بشكل دائرة متكاملة IC(Integrated circuit) مكونة من ١٨ طرف، الشكل Figure 3.9 يوضح اطراف المسيطر PIC16f84A .

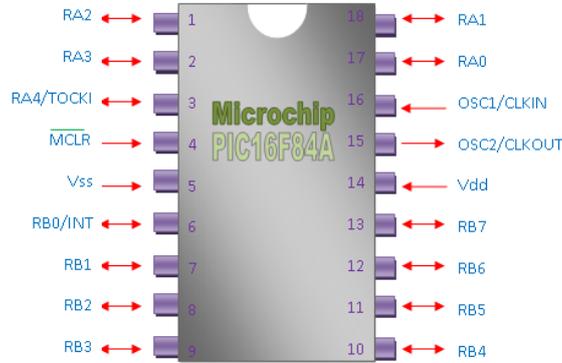


Figure 3.9

من الشكل Figure 3.9 نلاحظ أن أطراف المسيطر الدقيق PIC16f84A هي ،

### ❖ RA0-RA4

المرفئ A وهو مرفئ مكون من 5-bit التي هي RA0,RA1,RA2,RA3,RA4 ، ممكن أن يستخدم هذا المرفئ في عمليات الإدخال أو عمليات الإخراج ،

### ❖ RB0-RB7

المرفئ B وهو مرفئ مكون من 8-bit التي هي RB0,RB1,RB2,RB3,RB4,RB5,RB6,RB7 ، ممكن أن يستخدم هذا المرفئ في عمليات الإدخال أو عمليات الإخراج ،

### ❖ VSS & VDD

أطراف التغذية للشريحة VDD=+5 VDC , VSS=Ground

### ❖ OSC2/CLKOUT & OSC1/CLKIN

أطراف المذبذب Oscillator للمسيطر PIC16f84A، التردد ضروري لعمل وحدة السيطرة او عمل المسيطر الدقيق اذا لم تربط مذبذب لا يُنفذ المسيطر الدقيق البرنامج المخزن في ذاكرته ، يعمل المسيطر الدقيق PIC16f84A على تردد 4 MHZ ،

### ❖ MCLR(memory clear)

عند تسليط جهد منخفض على هذا الطرف يتم مسح البرنامج من الشريحة،



ملاحظة :

الأطراف التالية لها وظائف أخرى تحدد أثناء عملية Configuration

❖ RB0

له وظيفة المقاطعة interrupt لبرنامج المسيطر الدقيق

❖ RA4

ممكن ان يعمل هذا الطرف كطرف قرح لعداد Counter أو مؤقت Timer .

## Memory Clear

## طرف تصفير الشريحة

عند تسليط جهد منخفض على هذا الطرف يتم مسح برنامج الشريحة ، لذلك نربط هذا الطرف الى الجهد الموجب +5 Vdc ، كما هو موضح في الشكل Figure 3.10

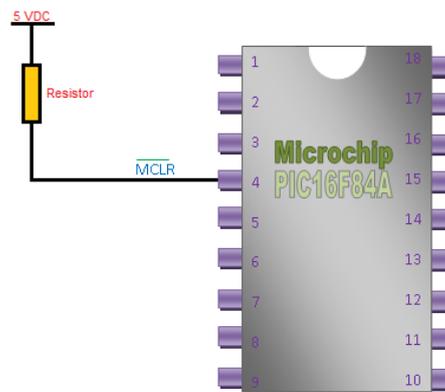


Figure 3.10

## Oscillator

## المذبذب

المذبذب عنصر مهم لعمل المسيطر الدقيق ، وأن اختيار نوع وقيمة المذبذب تؤثر إيجاباً أو سلبياً على عمل المسيطر الدقيق ، ، يتم الحصول على نبضات أو نبضات الضرورية لعمل المسيطر الدقيق بطريقتين

❖ باستخدام مذبذب بلوري XT Oscillator

❖ باستخدام مقاومة ومنتسعة لتشكيل دائرة رنين RC Oscillator

## XT Oscillator

## المذبذب بلوري

المذبذب البلوري Quartz Oscillator عبارة عن قطعة معدنية بطرفين ، تقوم هذه القطعة المعدنية بتوليد الذبذبة الضرورية لعمل المسيطر PIC16f84A ، يكتب على الغلاف الخارجي للقطعة المعدنية قيمة التردد الذي يولده المذبذب البلوري ، تذكر أن PIC16f84A يعمل على تردد 4 MHz ، عادة تربط متسعات سيراميكية C1 ، C2 الى أطراف المذبذب البلوري ذات قيمة 15 بيكو فاراد ، الشكل 3.11 Figure 3.11 يبين كيفية ربط مذبذب بلوري الى طرفي 16,15 للمُسيطر الدقيق ، سنكتفي بذكر طريقة المذبذب البلوري لانه اكثر استقرارية من المذبذب RC .

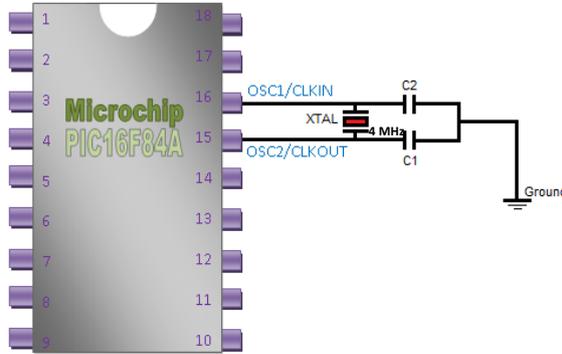


Figure 3.11

الشكل 3.12 Figure 3.12 يوضح كيفية توصيل المذبذب البلوري ذو قيمة 4 MHz الى طرفين 16,15 من PIC16f84A ، وكذلك توصيل الطرف الموجب لمصدر التغذية الذي قيمته 5 VDC الى طرف VDD الذي هو الطرف 14 ، بينما يربط الطرف السالب لمصدر التغذية الى الطرف VSS الذي هو الطرف 5 ، مع توصيل المتسعات Capacitors ، متسعتان بقيمة 15 pico-farad على طرفي المذبذب ، ومتسعة بقيمة 100 micro-farad لأغراض الحماية من الضجيج Noise ، تذكر ربط الطرف 4 الذي هو طرف MCLR الى الطرف الموجب للبطارية.

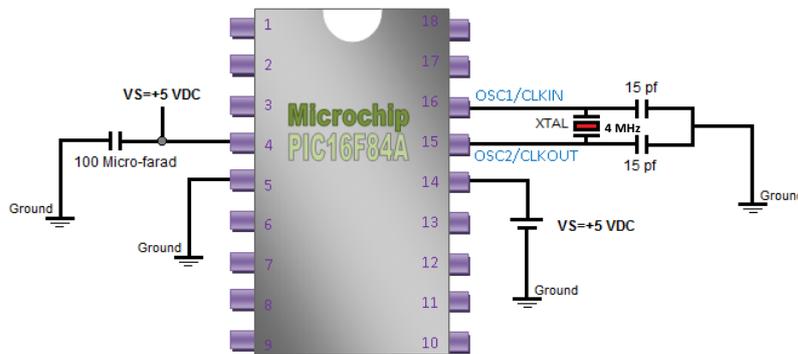


Figure 3.12



## File Register

## السجل

مقدمة:

يعتبر File Register هو قلب المسيطر الدقيق PIC16f84A ومعظم العمليات تجري على File Register ، وهي ذاكرة مكونة من 256 موقع او FFh بالنظام السادس عشر Hex ، كل موقع متكون من 8-bit ، هذه الذاكرة مقسمة الى جزئين Two bank هما Bank 0 الذي يبدأ من العنوان 00h الى العنوان 7Fh و Bank 1 الذي يبدأ من العنوان 80h الى العنوان FFh ، أي ان كل bank يتكون من 128 موقع ، كما هو موضح في الشكل Figure 4.1



## TRISA(Tri-State Buffer)

وهو سجل مكون من 8-bit (المستخدم منها فقط 5-bit ) لانه يتعامل مع port A المكون من 5-bit ، وظيفته تحديد اتجاه نقل البيانات Data ، أي تحديد طرف الإدخال Input أو الأخراج Output، يوجد هذا المسجل في الموقع 0x85 من bank1 ، القيمة صفر تعني الطرف المحدد من port a هو طرف أذخال input ، أما القيمة واحد فتعني ان الطرف هو طرف أخراج Output ، لكي نجعل porta كلة منفذ أخراج ما علينا سوى تحميل المسجل trisa بالقيمة 0x00

```
Trisa = 0x00 // the binary is 0000 0000
```

ولكي نجعل porta كلة منفذ أذخال ما علينا سوى تحميل المسجل trisa بالقيمة 0xFF

```
Trisa = 0xff // the binary is 1111 1111
```

ولكي نجعل اول طرف RA0 من porta طرف ادخال Input وباقي الاطراف هي اطراف خروج Output ما علينا سوى تحميل المسجل trisa بالقيمة 0xFE

```
Trisa = 0xFE //the binary is 1111 1110
```



ملاحظة:

راجع الوحدة الاولى النظام السادس عشر Hex. Decimal system

## TRISB(Tri-State Buffer)

يعمل هذا السجل نفس عمل المسجل trisa ولكنة مسؤول عن المنفذ portb ، وهو سجل مكون من 8-bit وظيفته تحديد أي بت من portb هو الادخال او الاخراج ، يوجد هذا المسجل في الموقع 0x86 من bank1

## PORTA

وهو سجل مكون من 8-bit وظيفته استقبال البيانات في حالة الكون المنفذ A أذخال ، أو أخراج البيانات الى العالم الخارجي في حالة كون المنفذ A أخراج ، ويوجد هذا المسجل في الموقع 0x05 من bank0

## PORTB

وهو سجل مكون من 8-bit وظيفته استقبال البيانات في حالة الكون المنفذ B أذخال ، أو أخراج البيانات الى العالم الخارجي في حالة كون المنفذ B أخراج ، ويوجد هذا المسجل في الموقع 0x06 من bank0

# Status Register

## مُسجل الحالة

وهو مسجل مكون من 8-bit كل بت له وظيفة معينة الشكل Fig 4.2 يبين مسجل الحالة ، ويوجد هذا المسجل في الموقع 0x03 من bank0 وكذلك يوجد عن الموقع 0x83 من bank 1

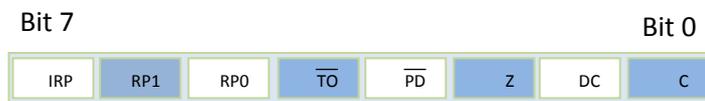


Figure 4.2

❖ علم المحمل  
علم المحمل يصبح هذا البت "1" عندما تكون النتيجة الحسابية الحالية لوحدة ALU موجبة ، اما اذا كانت سالبة سيتحول هذا البت الى صفر

❖ علم المحمل الثانوي  
يظهر عند حوث محمل من البت الثالث الى البت الرابع في النتيجة الحسابية الحالية لوحدة ALU

❖ علم التصفير  
يظهر عندما يكون الناتج صفر في النتيجة الحسابية الحالية لوحدة ALU.

❖ طرف تحديد bank  
من المعروف أن المسيطر الدقيق يتكون من two bank ، ولكي نتحول بين هذين banks ، نستخدم الطرف Rp0 ، حيث يستخدم الطرف Rp0 لتحديد أي bank من File Register سيتم التعامل معه حسب

Bit 0 = select "bank 0"

Bit 1 = select "bank 1"

❖ انخفاض الطاقة  
Power Down(PD)

يرفع هذا الطرف الى القيمة واحد عند بداية تشغيل المسيطر الدقيق ، ويصفر reset عندما يدخل المسيطر الدقيق في نمط حفظ استهلاك الطاقة مثل تنفيذ تعليمة Sleep ، كما يحصل في الحاسب الشخصي عند تركة لفترة بدون ان تقوم بأي عمل ، او عند تنفيذ التعليمة CLRWDT وذلك لتصفير مؤقت الحراسة.

Bit 0 = execute "Sleep" or "CLRWDT" instruction

Bit 1 = Power On

## ❖ أنقضاء الوقت

### Timer Out (TO)

ترفع هذا الطرف الى القيمة واحد عند بداية تشغيل المسيطر الدقيق او عند تنفيذ تعليمة CLRWDT او تعليمة Sleep ، ويصفر هذا الخانة عندما ينتهي مؤقت الحراسة من العد WatchDog Timer مشيراً مشيراً لحدث غير طبيعي اصاب المسيطر الدقيق أي أن التعليمة الحالية لم تنفذ ،

Bit 0 = WatchDog Timer End

Bit 1 = Power On, execute "Sleep" or "CLRWDT" instruction



ملاحظة:

الطرفين RP1,IRP مهمة في المسيطر الدقيق Pic16f84a

## Option Register

## المُسجل

وهو مسجل مكون من 8-bit كل بت له وظيفة معينة الشكل Fig 4.3 يبين مسجل option ، ويوجد هذا المسجل في الموقع 0x81 من bank1



Figure 4.3

### Ps0,Ps1,Ps2

### ❖ أطراف تحديد مقسم التردد Prescaler

أذا كنت من متخصصي علم الالكترونيات الرقمية ، وقد استخدمت عداد تصاعدي باستخدام مجموعة نطاقات Flip-flop ، فاكيد أنت تعرف أن العدادات تستخدم كمقسمة للتردد وذلك بأخذ طرف من أطراف مخارج العداد ، لاحظ الشكل Figure 4.4 تجد عداد يسمى 8-bit prescaler خرج هذا العداد ربط الى مازج multiplexer من نوع 8-1 Mux لاحظ ان اشارت تحديد دخل المازج هي ps0,ps1,ps2 اي أن خرج العدد سيقسم بشكل موزون اي عند الخرج الاول للعداد ستجد أن قيمة التردد هي نفسها تردد دخل العداد وعند الخرج الثاني ستجد أن خرج العداد هو التردد مقسوم على ٢ وعند الطرف الثالث سيكون التردد مقسوم على ٤ ، وهكذا ، يستخدم المازج كمفتاح Selector .

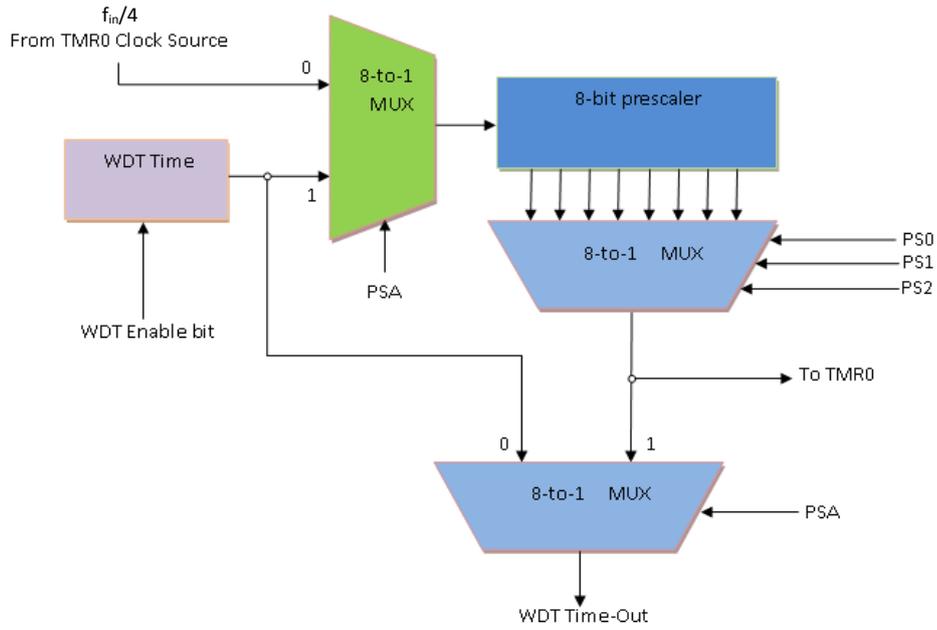


Figure 4.4

ملاحظة:

لمزيد من المعلومات في كيف استخدام العدادات كمقسمة للتردد اقرأ أحد كتب الالكترونيات الرقمية .

الجدول التالي يبين نسبة تقسيم التردد لكل من المؤقت timer ومؤقت الحراسة Watchdog timer

Ps2,ps1,ps0	نسبة التقسيم timer	نسبة التقسيم watchdog
000	1/2	1=frequency in
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

Table 4.0

### PSA(prescaler assignment)

### ❖ الطرف PSA

في المسيطر الدقيق pic16f84a يوجد نوعان من المؤقتات وهما مؤقت الحراسة watchdog timer والمؤقت timer وكلا المؤقتان عدادان اعتياديان يحتاجان الى تردد frequency لكي يعملان ، وايضاً يمكن تقسيم هذا التردد بالاعتماد على قيم ps0,ps1,ps2 لاحظ الجدول table 4.0، اذن الخيار PSA يقوم باسناد المقسم الى مؤقت الحراسة عندما يكون PSA=1 ويسند المقسم الى المؤقت timer عندما تكون PSA=0 ونسبة التقسيم كما علمنا تعتمد على ps0,ps1,ps2 ، أعلم ان تردد مؤقت الحراسة WDTMR هو نفسة تردد المذبذب Oscilator وتردد المؤقت هو Oscilator/4 اي تردد المصدر مقسم على اربعة.

الشكل التالي Figure 4.5 يمثل الدائرة الداخلية لمؤقت الحراسة Watchdog timer والمؤقت Timer

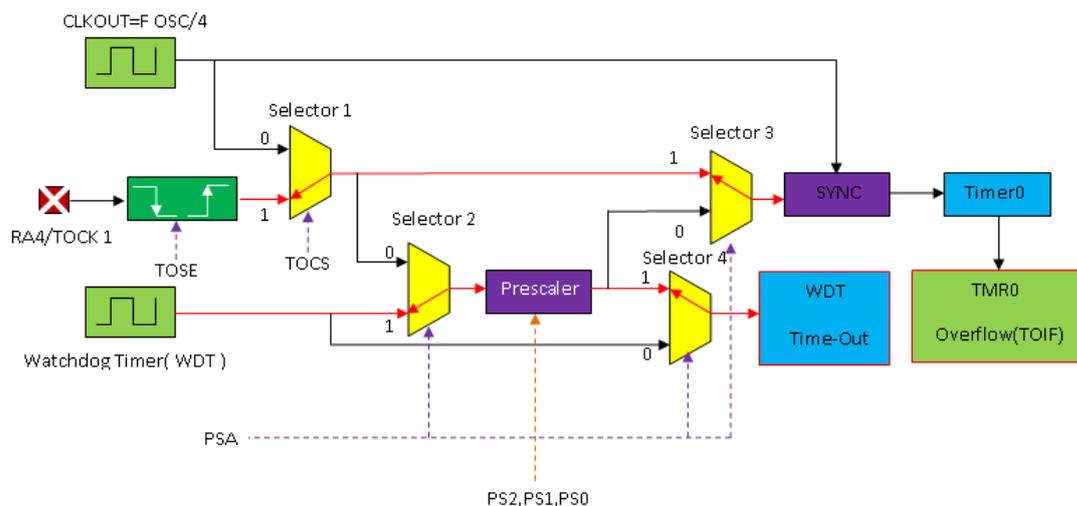


Figure 4.5

ملاحظة:



الغاية الأساسية لاستخدام مقسم التردد prescaler هو لتحديد سرعة العد Counter

**TOCS(TIMER OUT,COUNTER)**

❖ الطرف TOCS

يحتوي المسيطر الدقيق على وظيفة ، المؤقت timer ، العداد counter ، باستخدام هذه الخانة يمكن التنقل بين النمطين ، اذا كانت قيمة الخانة صفر فنمط المؤقت timer هو المفعول في المسيطر الدقيق، اما اذا كانت قيمة الخانة واحد فنمط العداد counter هو المفعول،

**TOSE(timer out,counter trigger edge)**

❖ الخانة TOSE

بالنسبة للعداد، لكي يزداد يحتاج الى عملية قرح من احدى اطراف المسيطر، يتم ذلك عند طريق الطرف 1 RA4/TOCK ، من خلال هذه الخانة يمكننا تحديد نوع جبهة القرح من الحافة الصاعدة او الحافة النازلة حسب..

**Tose=logic 0** , rising edge

**Tose=logic 1** , falling edge

اما بالنسبة لنمط المؤقت، فيمكن بدء المؤقت Start timing عن طريق الطرف RA4

**INTEDG(interrupt edge)**

❖ الخانة INTEDG

تستخدم لتحديد حافة القرح لعملية المقاطعة interrupt ، سيتم شرح المقاطعات في فصول لاحقة ، وتكون الحافة

**INTEDG** = logic 0 ,falling edge

**INTEDG** = logic 1 ,rising edge

## RBPU(Register 'B' pull up Resistor enable)

## ❖ الخانة RBPU

عند كل طرف من اطراف المرفأ portb ، يوجد مقاومة سحب pull up resistor ، هذه المقامات ممكن تفعيلها عندما تكون قيمة هذه الخانة صفر، او ابطالها عندما تكون قيمة هذه الخانة واحد،

ملاحظة: 

لمعلومات اكثر عن مقاومات pull up,pull down راجع اي كتاب عن الالكترونيات الرقمية .

## FSR Register (File Select Register)

## مُسجل FSR

وهو مسجل مكون من 8-bit يقع عند العنوان 0x04 من bank 0 والعنوان 0x84 من bank 1 ، عند تحميل هذا المسجل بقيمة معينة ، فانه يعتبر هذه القيمة عنوان Address لمسجل ، اي أن قيمة مسجل FSR تشير الى عنوان مسجل اخر ، مثلاً لو حملنا المسجل FSR بالقيمة 0x05 اي اننا حملنا المسجل بعنوان ProtA ، فيشير المسجل FSR الى المسجل Porta ، وتظهر محتويات المسجل porta عند المسجل INDF الذي سنشرحه في الفقرة التالية .

## INDF Register (Indirect Addressing Register)

## مُسجل INDF

يقوم هذا المسجل بخزن محتويات مسجل تم الاشارة اليه عن طريق المسجل FSR ، يقع عند العنوان 0x00 من bank 0 والعنوان 0x80 من bank 1 ، تسمى طريقة العنوانه هذ بالعنوانه الغير مباشرة ،لاحظ الشفرة التالية التي تبين العلاقة بين المسجل FSR والمسجل INDF ،

هذه الشفرات للتوضيح فقط ولا تتبع قواعد برمجية معينة

```
porta = 0x22
FSR = 0x05 // address of porta
INDF = 0x22 //data of porta
```

## TMR0 Register (Timer Register)

## مُسجَل TMR0

وهو مسجل مكون من 8-bit ، يخزن قيمة العد الحالية او قيمة الوقت المنقضي للموقت عند تفعيل المؤقت او العداد ، ويق هذا المسجل عن العنوان 0x01 من bank 0 .

## Program Counter Latch low & Latch High

## مُسجَل PCL & PCLATH

ذاكرة البرنامج program memory في المسيطر PIC16f84a المنفذ منها فقط 1 Kilo والباقي مهمل ، هذه المساحة 1 Kilo تسمى صفحة page كل صفحة عادة تتكون من 1 Kilo في مسيطرات اخرى مثل المسيطر PIC16f877a يتكون من أكثر من صفحة ويستخدم المسجل PCL والمسجل PCLATH للتنقل بين هذه الصفحات Page وكذلك التغلب على مشكلة الالتفاف warpping حول الصفحة ، لانهم كثيرة عن هذه المشكلة وكذلك التنقل بين الصفحات فالمسيطر PIC16f84a ببساطة يتكون من صفحة واحدة ، اي يمكنك التنقل خلال الصفحة الواحدة فقط عن طريق هذين المسجلين ، لنعود الى عداد البرنامج Program Counter(PC) الذي هو مسجل مكون من 13-bit ، الـ byte السفلي (<math>PC<7:0></math>) يأتي من المسجل PCL وهو مسجل قابل للقراءة والكتابة يقع عند العنوان 0x02 من bank 0 والعنوان 0x82 من bank1 أما باقي العنوان (<math>PC<12:8></math>) فيمكن الكتابة اليه فقط بصورة غير مباشرة عن طريق المسجل PCLATH وهو مسجل مكون من 5-bit فقط الذي يقع عند العنوان 0x0A من bank 0 والعنوان 0x8A من bank1 ، بشكل طبيعي يتم زيادة عداد البرنامج بشكل ذاتي وذلك لتنفيذ البرنامج المخزن في ذاكرة البرنامج Program memory ، ويتم تغيير محتويات عداد البرنامج عن تنفيذ تعليمات القفز واستدعاء الدوال والمقاطعات ، ويمكن ايضاً القفز الى اي موقع عن طريق تحميل هذين المسجلين ، ولكن كن حذراً عند التعامل معهما لكي لا يختل سير البرنامج ، الشكل Figure 4.6 يبين العلاقة بين عداد البرنامج Program counter والمسجلين PCL & PCLATH

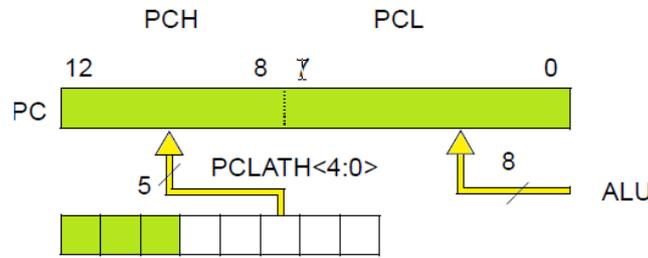


Figure 4.6

# INTCON(Interrupt Controller Register)

# مُسجَل INTCON

يتكون هذا المسجل من 8-bit ، كل خانة لها وظيفة معينة ، هذا المسجل مسئول عن تنفيذ او تفعيل خدمة المقاطعات في المتحكم الدقيق microcontroller ، يقع هذا المسجل عند العنوان 0x0B من bank0 والعنوان 0x8B من bank1 ، الشكل Fig 6.7 يبين مسجل INTCON



Figure 4.7

الوظائف التي يقوم بها المسجل INTCON

## GIE(Global Interrupt Enable)

## ❖ الخانة GIE

عندما ترفع هذه الخانة الى الواحد فانه يمكن خدمة المقاطعات بجميع انواعها ، اما عندما تصفر هذه الخانة فانه المسيطر الدقيق يبطل خدمة المقاطعات .

## PEIE(Programmable Earsable ROM Interrupt Enable)

## ❖ الخانة PEIE

عندما ترفع هذه الخانة الى القيمة واحد فان المسيطر الدقيق يمكن خدمة مقاطعة الذاكرة EEPROM والتي تشير الى انتهاء عملية الكتابة Writting ، أما عندما تصفر هذه الخانة فانه يتم ابطال خدمة المقاطعة لاكمال عملية الكتابة للذاكرة EEPROM .

## TMR0IE(Timer Interrupt Enable)

## ❖ الخانة TMR0IE

عندما ترفع هذه الخانة الى القيمة واحد فان المسيطر الدقيق يمكن خدمة مقاطعة المؤقت Timer والتي تحدث انتهاء الوقت times ، أما عندما تصفر هذه الخانة فانه يتم ابطال خدمة مقاطعة المؤقت Timer .

## INTE(Interrupt Enable)

## ❖ الخانة INTE

عندما ترفع هذه الخانة الى القيمة واحد فان المسيطر الدقيق يمكن خدمة مقاطعة بواسطة الطرف INT ، أما عندما تصفر هذه الخانة فانه يتم ابطال خدمة مقاطعة INT .

## RBIE( Portb RB(7:4) Interrupt Enable)

## ❖ الخانة RBIE

عندما ترفع هذه الخانة الى القيمة واحد فان المسيطر الدقيق يمكن خدمة مقاطعة بواسطة الاطراف RB7,RB6,RB5,RB4 ، أما عندما تصفر هذه الخانة فانه يتم ابطال خدمة مقاطعة عن طريق هذه الاطراف .

## INTF(Interrupt Flag)

## ❖ الخانة INTF

يرفع هذا العلم بشكل ذاتي عند حدوث مقاطعة عند الطرف INT .

## RBIF (Register portb Interrupt Flag)

## ❖ الخانة RBIF

يرفع هذا العلم الى القيمة واحد بشكل ذاتي عند حدوث مقاطعة من الطرف RB4-RB7.

## TMR0IF (Timer Out Interrupt Flag)

## ❖ الخانة TMR0IF

يرفع هذا العلم الى الحالة واحد عند طفحان المؤقت Timer0 ، مشير الى انتهاء الزمن المحدد للمؤقت.

## EEADR(EEPROM Address Register)

## مُسجل EEADR

يستخدم هذا المسجل المكون من 8-bit لتحديد عنوان Address الذاكرة Eeprom لكي يتم القراءة او الكتابة الى هذا الذاكرة ، يقع هذا المسجل عند الموقع 0x09 من bank0 .

## EEDATA(EEPROM Data Register)

## مُسجل EEDATA

يستخدم هذا المسجل المكون من 8-bit لارسال البيانات Data الى الذاكرة eeprom ليتم خزن البيانات بالاعتماد على العنوان الذي يتم تحديده بواسطة المسجل EEADR ، كذلك يقوم هذا المسجل باستقبال البيانات التي يتم قراءتها من ذاكرة eeprom ، يقع هذا المسجل عند الموقع 0x08 من bank0 .

## EECON(EEPROM Controller Register)

## مُسجل EECON1

يتكون هذا المسجل من 8-bit المستغل منها فقط 5-bit ، يقوم هذا المسجل بالتحكم بعمليات القراءة والكتابة الى الذاكرة eeprom ، يقع هذا المسجل عند العنوان 0x88 من bank 1 ، يبين الشكل Fig 4.8 المسجل eecon1



Figure 4.8

الوظائف التي يقوم بها المسجل EECON1

## RD (Read Operation)

## ❖ الخانة RD

عند رفع هذه الخانة الى القيمة واحد تمكن عملية القراءة من الذاكرة eeprom ، ولا يمكن تصفيرها لانها تصفر بعد أن تتم عملية القراءة.

## WR (Write Operation)

## ❖ الخانة WR

عند رفع هذه الخانة الى القيمة واحد تبدء عملية الكتابة الى الذاكرة eeprom ، ولا يمكن تصفيرها لانها تصفر بعد أن تتم عملية الكتابة.

## WREN (Write Enable Operation)

## ❖ الخانة WREN

عند رفع هذه الخانة الى القيمة واحد تمكن عملية الكتابة الى الذاكرة eeprom ، وعند تصفيرها لانها تصفر تبطل عملية الكتابة.

### WRERR (Write Error)

### ❖ الخانة WRERR

عندما ترفع هذه الخانة الى القيمة واحد فانه تشير بذلك الى حدوث خطأ في عملية الكتابة ، أما عندما تصفر فذلك يدل على أن عملية الكتابة تمت .

### EEPGD (Eeprom Program Memory Or Data Memory)

### ❖ الخانة EEPGD

عندما ترفع هذه الخانة الى القيمة واحد فانه تشير بذلك الى ذاكرة البرنامج eeprom program memory ، أما عند تصفير هذه الخانة فانه يتم التاثير الى ذاكرة البيانات eeprom data memory

## EECON(EEPROM Controller Register)

## مُسجل EECON 2

هذا الموقع غير منفذ Unimplemented .



## Instruction Set

## طَقَم التَّعليمات

مُقدمة:

في هذه الوحدة سوف نتعرف على مجموعة التَّعليمات Instruction Set المتوفرة في المتحكم الدَّقيق PIC16f84A ، تُعتبر هذه الوحدة كمرجع لك تُعود الية عند الحاجة الى التَّعرف الى وَضيفة تَعليمة مُعينة ، عند قراءتك لهذه الوحدة سوف تسأل نفسك ما الفائدة أو الغاية من تَعليمة ما ، لا تقلق ، فالتعليمة بحد ذاتها بدون فائدة ما لم تتكامل مع مجموعة تَعليمات لتؤدي وَضيفة مُعينة مكونة ما يُسمى بالبرنامج Program ، مع تقدمك في فصول الكتاب وكتابة برامج سوف تتضح الفكرة اكثر، من الجيد معرفه أن عدد التَّعليمات المتوفرة في المايكروكونترولر PIC16f84A هو 32 تَعليمة ، منها ما يستغرق فترة زمنية مقدارها 1 مايكروثانية ، ومنها ما يستغرق فترة مقدارها 2 مايكروثانية.

عبارة عن سلسلة من التعليمات أو الأيعازات instruction المُتتابة ، التي توجه عمل المايكروكونتروالر PIC16f84A لغرض تأدية وظيفه معينة .

## طَقَم التَّعليمات

## Instruction Set

لقد قلنا أن البرنامج عبارة عن مجموعة من التعليمات Instructions الممتابعة ، أذن العنصر الأساسي لتكوين البرنامج هو التعليمه ، تقسم التعليمات في المايكروكونتروالر PIC16f84A إلى ،

❖ التعليمات على مستوى البت	Bit Orientation Operations
❖ التعليمات على مستوى البايت	Byte Orientation Operation
❖ تعليمات السيطرة والثوابت	Lateral and Control Operations

## التعليمات على مستوى البت

## Bit Orientation Operations

وهي التعليمات التي تجري على بت معين ضمن مُسجل مُعين ، مثل تغيير قيمة بت من 1 الى 0 أو من 0 الى 1 ، عادةً يشار الى البت المراد إجراء العملية عليه بالرمز "b" بينما يُرمز للمُسجل المراد تغيير أحد بتاته بالرمز "f" ، الجدول Table 5.1 يوضح مجموعة التعليمات على مستوى البت المتوفرة في المايكروكونتروالر PIC16f84A ،

Mnemonic, Operands	Description	Cycles
<b>BIT-ORIENTED FILE REGISTER OPERATIONS</b>		
BCF f, b	Bit Clear f	1
BSF f, b	Bit Set f	1
BTFSC f, b	Bit Test f, Skip if Clear	1 (2)
BTFSS f, b	Bit Test f, Skip if Set	1 (2)

Table 5.1

## Bit Clear Flag

## التعليمه BCF

تقوم هذه التعليمه بتصفير أحد بتات مُسجل Register ، وتأخذ التعليمه الصيغة التالية ،

BCF f, b

حيث :

f: يُمثل المُسجل المراد تصفير أحد بتاته،

b: يُمثل رقم البت المراد تصفيره من المُسجل f ،

تستهلك هذه التعليلة دورة ماكنة واحدة ، أي أن هذه التعليلة تستغرق فترة زمنية مقداره 1 مايكروثانية ،

## Bit Set Flag

## التعليلة BSF

تقوم هذه التعليلة برفع أحد بتات مسجل Register الى القيمة واحد ، وتأخذ التعليلة الصيغة التالية ،

BSF f, b

حيث :

f: يُمثل المُسجل المُراد رَفَع أحد بتاتة Bit إلى القيمة واحد،

b: يُمثل رَقم البت Bit المُراد رَفَع قيمته إلى الواحد من المُسجل f ،

ملاحظة:

تستهلك هذه التعليلة دورة ماكنة واحدة ، أي أن هذه التعليلة تستغرق فترة زمنية مقداره 1 مايكروثانية ،

## Bit Test Flag Skip If Set

## التعليلة BTFSS

لكي نفهم وظيفة هذه التعليلة ، دعنا نأخذ الشفرة Code التالية،

1 BTFSS 0x0C, 2  
2 BSF 0x0C, 2  
3 BCF 0x0C, 3

تقوم التعليلة الأولى والتي هي BTFSS 0x0C,2 بفحص البت Bit الثاني من المسجل 0x0C فإذا كان واحد ، يُحمل عداد البرنامج PC بعنوان التعليلة الثالثة التي هي BCF 0x0C,3 لينفذها ، بصيغة أخرى عندما تُنفذ التعليلة BTFSS وكان البت المراد فحصه يساوي واحد يتم قفز التعليلة التي تلي تعليلة BTFSS ليتم تنفيذها أي سيتم قفز التعليلة الثانية الى التعليلة الثالثة والتي هي BCF 0x0C,3 ليقوم المعالج بتنفيذها ، أما إذا كان البت Bit المراد فحصه يحمل القيمة صفر فيتم تنفيذ التعليلة التي تلي التعليلة BTFSS مباشرة أي سيتم تنفيذ التعليلة الثانية BCF 0x0C,3 ويستمر تنفيذ البرنامج ' أي سيتم تنفيذ التعليلة الثالثة أيضاً والرابعة وهكذا ، اذن التعليلة BTFSS تعتبر من تعليمات القفز Jump وتأخذ التعليلة الصيغة التالية ،

BTFSS f, b

حيث:

f: يُمثل المُسجل المُراد فحص أحد بتاتة ،

b: يُمثل رقم البت المُراد فحصه من المُسجل f ،

ملاحظة:



تستهلك هذه التعلّمة دورة ماكنة واحدة إذا لم يتحقق الشرط أي إذا كان البت المُراد فحصه صفر ، أن هذه التعلّمة في هذه الحالة تستغرق فترة زمنية مقدارها 1 مايكرو ثانية ، أما إذا تحقق الشرط أي عندما يكون البت المراد فحصه يحمل القيمة واحد ، فستأخذ هذه التعلّمة دورتان ماكنة أي ان الفترة الزمنية التي تستغرقها التعلّمة ستكون 2 مايكروثانية.

## Bit Test Flag Skip If Clear

## التعلّمة BTFSC

تعمل هذه التعلّمة عكس عمل تعلّمة BTFSS حيث أن تعلّمة BTFSC تقفز Jumping إذا كان البت Bit المُراد فحصه يحمل القيمة صفر ، أما اذا كان البت المراد فحصه يحمل القيمة واحد فسيتم تنفيذ التعلّمة التي تلي تعلّمة BTFSC مباشرة ، تأخذ التعلّمة الصيغة التالية ،

BTFSC f , b

حيث:

f: يُمثل المُسجل المُراد فحص أحد بتاتة ،

b: يُمثل رقم البت المُراد فحصه من المُسجل f ،

ملاحظة:



تستهلك هذه التعلّمة دورة ماكنة واحدة إذا لم يتحقق الشرط أي إذا كان البت المُراد فحصه واحد ، أن هذه التعلّمة في هذه الحالة تستغرق فترة زمنية مقدارها 1 مايكرو ثانية ، أما إذا تحقق الشرط أي عندما يكون البت المراد فحصه يحمل القيمة صفر ، فستأخذ هذه التعلّمة دورتان ماكنة أي ان الفترة الزمنية التي تستغرقها التعلّمة ستكون 2 مايكروثانية.

وهي التعليمات التي تسيطر على سير عمل البرنامج مثل تعليمات القفز Jump ، وكذلك تتضمن التعليمات الخاصة بتحميل قيم ثابتة K الى مسجل معين ، الجدول Table 5.2 يوضح مجموعة تعليمات السيطرة والثوابت ،

Mnemonic, Operands	Description	Cycles
<b>LITERAL AND CONTROL OPERATIONS</b>		
ADDLW k	Add literal and W	1
ANDLW k	AND literal with W	1
CALL k	Call subroutine	2
CLRWDT -	Clear Watchdog Timer	1
GOTO k	Go to address	2
IORLW k	Inclusive OR literal with W	1
MOVLW k	Move literal to W	1
RETFIE -	Return from interrupt	2
RETLW k	Return with literal in W	2
RETURN -	Return from Subroutine	2
SLEEP -	Go into standby mode	1
SUBLW k	Subtract W from literal	1
XORLW k	Exclusive OR literal with W	1

Table 5.2

### MOV Lateral to W Reg.

### التعليمة MOVLW

تقوم هذه التعليمة بنقل قيمة ثابتة Lateral الى مسجل العمل W Reg. ، وتأخذ التعليمة الصيغة التالية ،

MOVLW K

حيث:

K: يمثل الرقم الثابت المراد إجراء نقلة مسجل العمل W Reg. .

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

### ADD Lateral to W Reg.

### التعليمة ADDLW

تقوم هذه التعليمة بجمع قيمة ثابتة Lateral مع قيمة مسجل العمل W Reg. ، وتخزن النتيجة في مسجل العمل W Reg. ، تأخذ التعليمة الصيغة التالية ،

ADDLW K

حيث:

K: يمثل الرقم الثابت المراد جمعة مع مسجل العمل W Reg. .

## AND Lateral and W Reg.

## التعليمة ANDLW

تقوم هذه التعليمة بأجراء عملية AND المنطقية بين مسجل العمل W Reg. وقيمة ثابتة ، وتأخذ التعليمة الصيغة التالية ،

```
ANDLW K
```

حيث:

K: يمثل الرقم الثابت المراد أجراء عملية AND المنطقية عليه .

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## CALL Statement

## التعليمة CALL

تقوم هذه التعليمة باستدعاء برنامج ثانوي Subroutine مخزون في موقع اخر من الذاكرة ، تأخذ التعليمة الصيغة التالية ،

```
CALL K
```

حيث:

K: يمثل أسم البرنامج الثانوي Subroutine المراد استدعاءه .

ملاحظة: 

تستهلك هذه التعليمة دورتان ماكنة، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 2 مايكرو ثانية

## Clear Watchdog Timer

## التعليمة CLRWDT

تقوم هذه التعليمة بتعطيل وضيفة مؤقت الحراسة ، ، تأخذ التعليمة الصيغة التالية.

```
CALL K
```

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية

## Goto Statement

## التعليمة Goto

تقوم هذه التعليمة بالقفز الى موقع آخر من الذاكرة ، تأخذ التعليمة الصيغة التالية ،

```
Goto K
```

حيث:

K: يمثل عنوان موقع الذاكرة المراد القفز اليه .

ملاحظة: 

تستهلك هذه التعليمة دورتان مكنة، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 2 مايكرو ثانية ،

## Exclusive OR Lateral With W Reg.

## التعليمة XORLW

تقوم هذه التعليمة بأجراء عملية Ex-OR المنطقية بين مسجل العمل W Reg. وقيمة ثابتة ، وتأخذ التعليمة الصيغة التالية ،

```
XORLW K
```

حيث:

K: يمثل الرقم الثابت المراد إجراء عملية Ex-OR المنطقية عليه .

ملاحظة: 

تستهلك هذه التعليمة دورة مكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Inclusive OR Lateral With W Reg.

## التعليمة IORLW

تقوم هذه التعليمة بأجراء عملية OR المنطقية بين مسجل العمل W Reg. وقيمة ثابتة ، وتأخذ التعليمة الصيغة التالية ،

```
IORLW K
```

حيث:

K: يمثل الرقم الثابت المراد إجراء عملية OR المنطقية عليه .

## Subtract Lateral from W Reg.

## التعليمة SUBLW

تقوم هذه التعليمة بطرح قيمة ثابتة Lateral من قيمة مسجل العمل W Reg. ، وتخزن النتيجة في مسجل العمل W Reg. ، تأخذ التعليمة الصيغة التالية ،

SUBLW K

حيث:

K: يمثل الرقم الثابت المراد طرحه من مسجل العمل W Reg. .

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## SLEEP Instruction

## التعليمة SLEEP

تقوم هذه التعليمة بأدخال المسيطر الدقيق في وضع توفير الطاقة standby Mode حيث يتوقف المعالج عن العمل ، وعند ورود أي اشارة مقاطعة الى المسيطر يعود المسيطر للنهوض من جديد ، وتأخذ التعليمة الصيغة التالية ،

SLEEP

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Return flag interrupt enable instruction

## التعليمة RETFIE

لكي نفهم عمل هذه التعليمة ، يجب ان نفهم ما هي المقاطعة ، مبدء المقاطعة هو مقاطعة او توقف المعالج عن تنفيذ البرنامج الحالي او الرئيس Main Program لتنفيذ برنامج أخر يسمى برنامج خدمة المقاطعات (ISR) interrupt service routine ، وبعد الانتهاء من برنامج خدمة المقاطعات ، يجب ان يعود التنفيذ الى البرنامج الرئيسي Main Program ، أذن تعليمة RETFIE تقوم بنقل التنفيذ من برنامج خدمة المقاطعات الى البرنامج الرئيسي ، او بصيغة اخرى هي تعليمة العودة الى البرنامج الرئيسي ، سيتم مناقشة هذا الموضوع بشكل مفصل لاحقاً، وتأخذ التعليمة الصيغة التالية .

RETFIE

ملاحظة: 

تستهلك هذه التعليمة دورتان ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 2 مايكرو ثانية ،

## Return Instruction

## التعليمة RETURN

تستخدم هذه التعليمة للعودة من برنامج الفرعي Subroutine ، الى البرنامج الرئيس ، التعليمة تأخذ الصيغة التالية .

```
RETURN
```



ملاحظة:

تستهلك هذه التعليمة دورتان ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 2 مايكرو ثانية ،

## Return with lateral to W Reg.

## التعليمة RETLW

تستخدم هذه التعليمة للعودة من برنامج فرعي Subroutine ، الى البرنامج الرئيس ، ولكن عند العودة تحمل قيمة مسجل العمل W Reg. بقيمة ثابتة Lateral value ، التعليمة تأخذ الصيغة التالية .

```
RETLW K
```

حيث:

K: يمثل الرقم الثابت المراد نقله الى مسجل العمل W Reg. عند العودة من البرنامج الفرعي subroutine .



ملاحظة:

تستهلك هذه التعليمة دورتان ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 2 مايكرو ثانية ،

## Byte Orientation Operations

## التعليمة على مستوى البايت

وهي التعليمة التي تجري على مستوى Byte ، معظم هذه التعليمة تتعامل مع وحدة الحساب والمنطق ALU ، مثل إجراء العمليات الحسابية والمنطقية وغيرها ، عادة يشار الى البايت أو المسجل المراد إجراء العملية عليه بالرمز "f" بينما يرمز للمكان الذي ستخزن فيه النتيجة بالرمز "d" حيث إذا كان "d=0" سيتم تخزين النتيجة في مسجل العمل W Reg. اما اذا كانت "d=1" فسيتم تخزين النتيجة في المسجل f ، لاحظ الشكل 5.1 Figure

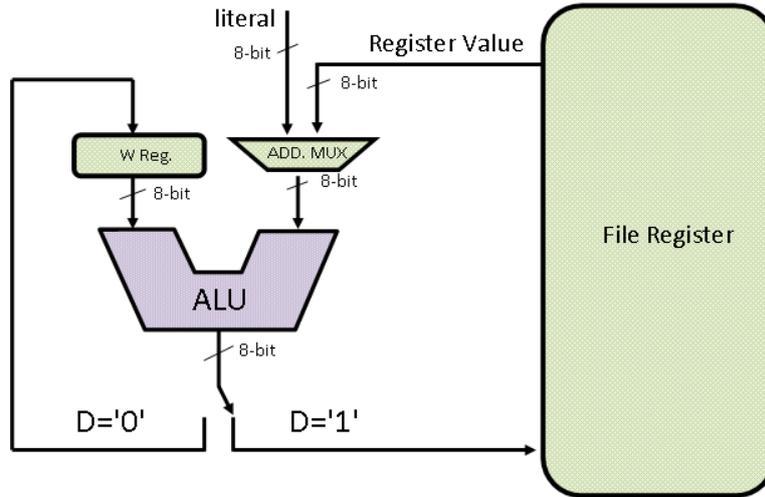


Figure 5.1

، الجدول Table 5.3 يوضح مجموعة التعليمات على مستوى البت المتوفرة في المسيطر PIC16f84A ،

Mnemonic, Operands	Description	Cycles
<b>BYTE-ORIENTED FILE REGISTER OPERATIONS</b>		
ADDWF f, d	Add W and f	1
ANDWF f, d	AND W with f	1
CLRF f	Clear f	1
CLRW -	Clear W	1
COMF f, d	Complement f	1
DECF f, d	Decrement f	1
DECFSZ f, d	Decrement f, Skip if 0	1(2)
INCF f, d	Increment f	1
INCFSZ f, d	Increment f, Skip if 0	1(2)
IORWF f, d	Inclusive OR W with f	1
MOVF f, d	Move f	1
MOVWF f	Move W to f	1
NOP -	No Operation	1
RLF f, d	Rotate Left f through Carry	1
RRF f, d	Rotate Right f through Carry	1
SUBWF f, d	Subtract W from f	1
SWAPF f, d	Swap nibbles in f	1
XORWF f, d	Exclusive OR W with f	1

Table 5.3

## ADD W Reg. to flag

## التعليمة ADDWF

تقوم هذه التعليمة بأجراء عملية الجمع بين مسجل العمل W Reg. وأي مسجل آخر f ، تاخذ هذه التعليمة الصيغة التالية ،

**ADDWF** f, d

حيث:

f: يمثل المسجل المراد جمعة مع مسجل العمل ،

d: تأخذ قيمتان اما 0 او 1 ، اذا d=0 سيتم تخزين ناتج الجمع في مسجل العمل  $W=W+f$  ، اما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f أي  $f=f+W$  ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

مثال 5.1

قم بإضافة العدد 4 الى محتويات المسجل 0Ch ؟

الحل:

```
MOVLW 0x04  
ADDWF 0x0C , 1
```

نقوم أولاً بتحميل مسجل العمل بالقيمة 0x04 عن طريق التعليمة MOVLW 0x04 ، ثم نقوم بأجراء عملية الجمع بين قيمة مسجل العمل W وقيمة المسجل 0x0C عن طريق التعليمة ADDWF 0x0C,1 والقيمة واحد ليتم خزن ناتج الجمع في المسجل 0x0C ،

Subtract W Reg. from flag

التعليمة SUBWF

تقوم هذه التعليمة بطرح قيمة مسجل العمل W Reg. من مسجل آخر f ، تأخذ هذه التعليمة الصيغة التالية ،

```
SUBWF f, d
```

حيث:

f: يمثل المسجل المراد طرح قيمة مسجل العمل W Reg. منه،

d: تأخذ قيمتان اما 0 او 1 ، اذا كانت d=0 سيتم تخزين ناتج الطرح في مسجل العمل  $W=f-W$  ، اما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f أي  $f=f-W$  ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Swap W Reg. With Flag

## التعليمة SWAPF

تقوم هذه التعليمة بعملية تبديل Swapping الاربع بتات السفلية low nipple للمسجل f ، مع اربع بتات العلوية High nipple ، وتأخذ التعليمة الصيغة التالية

**SUBWF** f, d

حيث:

f: يمثل المسجل المراد طرح قيمة مسجل العمل W Reg. منه،

d: تأخذ قيمتان اما 0 او 1 ، اذا كانت d=0 سيتم تخزين ناتج الطرح في مسجل العمل  $W=f-W$  ، اما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f أي  $f=f-W$  ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## AND W Reg. with flag

## التعليمة ANDWF

تقوم هذه التعليمة بأجراء عملية AND المنطقية بين مسجل العمل W Reg. وأي مسجل آخر f ، تأخذ هذه التعليمة الصيغة التالية ،

**ANDWF** f, d

حيث:

f: يمثل المسجل المراد اجراء عملية AND المنطقية عليه مع مسجل العمل ،

d: تأخذ قيمتان اما 0 او 1 ، اذا كانت d=0 سيتم تخزين ناتج عملية AND المنطقية في مسجل العمل، أما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

تقوم هذه التعليمة بأجراء عملية AND المنطقية بين مسجل العمل W Reg. وأي مسجل آخر f ، تاخذ هذه التعليمة الصيغة التالية ،

**XORWF** f, d

حيث:

f: يمثل المسجل المراد اجراء عملية AND المنطقية عليه مع مسجل العمل ،

d: تاخذ قيمتان اما 0 او 1 ، اذا d=0 سيتم تخزين ناتج عملية EX-OR المنطقية في مسجل العمل، أما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

تقوم هذه التعليمة بأجراء عملية OR المنطقية بين مسجل العمل W Reg. وأي مسجل آخر f ، تاخذ هذه التعليمة الصيغة التالية ،

**IORWF** f, d

حيث:

f: يمثل المسجل المراد اجراء عملية OR المنطقية عليه مع مسجل العمل ،

d: تاخذ قيمتان اما 0 او 1 ، اذا d=0 سيتم تخزين ناتج عملية OR المنطقية في مسجل العمل، أما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Clear Flag

## التعليمة CLRF

تقوم هذه التعليمة بتصفير قيمة مسجل معين ، تأخذ التعليمة الصيغة التالية ،

```
CLRF f
```

حيث:

f: يمثل المسجل المراد تصفيره .



ملاحظة:

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## No Operation Instruction

## التعليمة NOP

لا تقوم هذه التعليمة بأي وظيفة ، تستخدم عادة في عمليات التأخير Delay ، حيث أن هذه التعليمة تأخذ دورة ماكنة واحدة لذلك تستغرق هذه التعليمة فترة زمنية مقدارها 1 مايكرو ثانية، تأخذ التعليمة الصيغة التالية ،

```
NOP
```

❖ مثال 5.1

قم بتوليد تأخير مقداره 3 مايكرو ثانية ؟

الحل:

```
NOP  
NOP  
NOP
```

لقد قمنا بكتابة ثلاث تعليمات NOP كل تعليمة تستغرق فترة زمنية مقدارها 1 مايكرو ثانية بدون تأدية أي وظيفة ، اذا المحصلة الكلية لهذه التعليمات الثلاثة 3 مايكرو ثانية

## Clear Working Register

## التعليمة CLRW

تقوم هذه التعليمة بتصفير قيمة مسجل العمل W Reg. ، تأخذ التعليمة الصيغة التالية ،

```
CLRW
```

ملاحظة:



تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Compliment Flag Instruction

## التعليمة COMF

تقوم هذه التعليمة بأيجاد متمم Compliment مسجل f ، تأخذ التعليمة الصيغة التالية ،

COMF f,d

حيث:

f: يمثل المسجل المراد أيجاد متممه،

d: إذا كانت d=0 سيتم تخزين ناتج متمم المسجل f في مسجل العمل W Reg.، أما إذا كانت قيمة d=1 فسيتم تخزين ناتج المتمم في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Decrement Flag

## التعليمة DECF

تقوم هذه التعليمة بأنقاص قيمة مسجل f بالقيمة واحد ، وتأخذ التعليمة الصيغة التالية

DECF f,d

حيث:

f: يمثل المسجل المراد أنقاص قيمته بواحد،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg.، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Increment Flag

## تعلیمة INCF

تقوم هذه التعلیمة بزيادة قيمة مسجل f بالقيمة واحد ، وتأخذ التعلیمة الصیغة التالي

INCF f,d

حيث:

f: يمثل المسجل المراد زيادة قيمته بواحد،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg.، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعلیمة دورة مكنة واحدة ، أي أن هذه التعلیمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Rotate Right Through Carry

## تعلیمة RRF

تقوم هذه التعلیمة بأزاحة Shift ودوران Rotate محتويات مسجل f باتجاه اليمين ، وعند خروج البت الأقل أهمية LSB من المسجل f ، يخزن في خانة المحمل carry من مسجل الحالة Status register ، وعند خروج قيمة المحمل Carry يخزن في خانة الاكثر اهمية MSB من المسجل f أي أن المحمل Carry يدخل في عملية الدوران Rotation ، لاحظ الشكل Figure 5.2

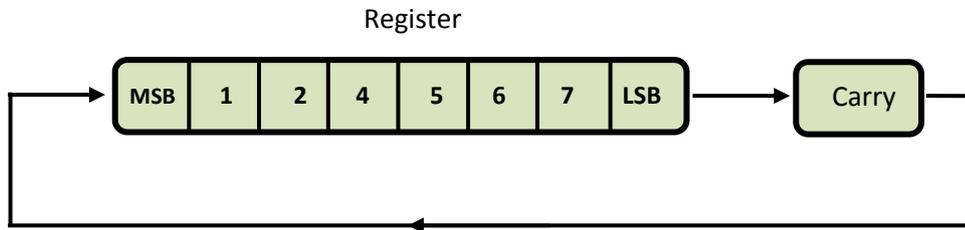


figure 5.2

تأخذ التعلیمة الصیغة التالية ،

RRF f,d

حيث:

f: يمثل المسجل المراد دوران ،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg.، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

تستهلك هذه التعليمات دورة ماكنة واحدة ، أي أن هذه التعليمات تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Rotate Left Through Carry

## تعليمات RLF

تقوم هذه التعليمات بأزاحة Shift ودوران Rotate محتويات مسجل f باتجاه اليسار ، وعند خروج البت الأكثر أهمية MSB من المسجل f ، يخزن في خانة المحمل carry من مسجل الحالة Status register ، وعند خروج قيمة المحمل Carry يخزن في خانة الأقل أهمية LSB من المسجل f ، لاحظ الشكل Fig 5.3

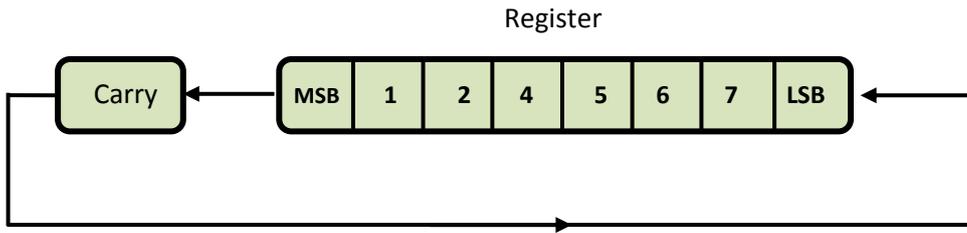


figure 5.3

تأخذ التعليمات الصيغة التالية ،

RLF f,d

حيث:

f: يمثل المسجل المراد دوران محتوياته باتجاه اليسار ،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg. ، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

تستهلك هذه التعليمات دورة ماكنة واحدة ، أي أن هذه التعليمات تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Move Flag

## تعليمات MOVF

تقوم هذه التعليمات بنقل محتويات المسجل f الى مسجل العمل W Reg. ، أو نقل محتويات المسجل f الى نفسه؟؟ سنسأل ما الغاية من نقل مسجل الى نفسه ، نستفاد من هذه الحالة اننا نريد ان نعرف مثلاً هل قيمة هذا المسجل صفر أم لا ، نحن نعلم ان مسجل الحالة status Register يتحدث update بعد كل عملية تجلاني في وحدة الحساب والمنطق ALU ، اذن عند نقل مسجل الى نفسه عن طريق التعليمات MOVF سيتحدث علم الصفر Zero Flag الموجود في مسجل الحالة Status Register ، تأخذ التعليمات الصيغة التالية

```
MOVF f,d
```

حيث:

f: يمثل المسجل المراد إجراء العملية عليه،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

❖ مثال 5.2

حدث Update مسجل الحالة Status Register بحيث يتضمن حالة المسجل 0Eh ؟

الحل:

```
MOVF 0x0E,1
```

Move W Reg. to Flag

تعليمة MOVWF

تقوم هذه التعليمة بنقل محتويات مسجل العمل W Reg الى مسجل f ،تأخذ التعليمة الصيغة التالية

```
MOVWF f
```

حيث:

f: يمثل المسجل المراد نقل محتويات مسجل العمل W Reg اليه،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليمة دورة ماكنة واحدة ، أي أن هذه التعليمة تستغرق فترة زمنية مقداره 1 مايكرو ثانية ،

## Increment Flag Skip if Zero

## تعليلة INCFSZ

تقوم هذه التعليلة بزيادة محتويات مسجل f بالقيمة واحدة ، ثم تفحص النتيجة اذا كانت اكبر من الصفر يتم تنفيذ التعليلة التي تلي تعليلة INCFSZ اما اذا كانت النتيجة صفر فسيتم قفز التعليلة التي تلي تعليلة INCFSZ ، هذه التعليلة مشابهة تقريباً لعمل تعليلة BTFSZ في عملية القفز، تأخذ التعليلة الصيغة التالية

INCFSZ f,d

حيث:

f: يمثل المسجل أجراء العملية عليه،

d: إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg.، أما اذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليلة دورة ماكنة واحدة ، إذا كانت قيمة المسجل بعد الزيادة اكبر من صفر أي لا تتم عملية القفز Jump ، تستغرق التعليلة في هذه الحالة فترة زمنية مقداره 1 مايكرو ثانية ، اما اذا كانت قيمة المسجل بعد الزيادة تساوي صفر فتستغرق وحدة السيطرة دورتان ماكنة أي ستستغرق التعليلة فترة زمنية مقدارها 2 مايكروثانية

❖ مثال 5.3

تتبع trace التعليلات التالية ؟

```
1 MOVLW 0xFF
2 MOVWF 0x0C
3 INCFSZ 0x0C,1
4 DECF 0x0C,1
5 INCF 0x0C,1
```

الحل:

تقوم أول تعليلتان بنقل القيمة fffh الى مسجل العمل W عن طريق تعليلة MOVLW 0xFF وذلك بغية نقلة الى مسجل الاغراض العامة 0Ch عن طريق تعليلة MOVWF 0x0C، ثم تأتي التعليلة الثالثة التعليل INCFSZ لتقوم بزيادة محتويات المسجل 0Ch بواحد بما أن المسجل 0Ch يحتوي الآن على القيمة FFh وعند زيادته بواحد سيتم تصفير المسجل وذلك لأن سعة المسجل 0Ch القصوى هي 8-bit أي القيمة القصوى هي FFh ، راجع الوحدة الاولى عملية الفيضان Overflow ، بما أن المسجل 0Ch يحمل القيمة صفر الآن ، ستقوم التعليلة INCFSZ بالقفز الى التعليلة الخامسة INCF 0x0C,1 لتقوم بزيادة محتويات المسجل 0Ch بواحد لتصبح محتويات المسجل 0Ch هي القيمة 1 .

تقوم هذه التعليلة بانقاص محتويات مسجل f بالقيمة واحدة ، ثم تفحص النتيجة اذا كانت اكبر من الصفر يتم تنفيذ التعليلة التي تلي تعليلة DECFSZ اما اذا كانت النتيجة صفر فسيتم قفز التعليلة التي تلي تعليلة DECFSZ ، تأخذ التعليلة الصيغة التالية

DECFSZ f,d

حيث:

f: يمثل المسجل المراد إجراء العملية عليه،

d : إذا كانت d=0 سيتم تخزين الناتج في مسجل العمل W Reg. ، أما إذا كانت قيمة d=1 فسيتم تخزين الناتج في المسجل f ،

ملاحظة: 

تستهلك هذه التعليلة دورة ماكنة واحدة ، إذا كانت قيمة المسجل بعد النقصان اكبر من صفر أي لا تتم عملية القفز Jump ، تستغرق التعليلة في هذه الحالة فترة زمنية مقداره 1 مايكرو ثانية ، اما اذا كانت قيمة المسجل بعد النقصان تساوي صفر فتستغرق وحدة السيطرة دورتان ماكنة أي تستغرق التعليلة فترة زمنية مقدارها 2 مايكروثانية،

❖ مثال 4.1

تتبع trace التعليلات التالية ؟

```
1 MOVLW 0x01
2 MOVWF 0x0C
3 DECFSZ 0x0C,1
4 DECF 0x0C,1
5 INCF 0x0C,1
```

الحل:

تقوم أول تعليلتان بنقل القيمة 01h الى مسجل العمل W عن طريق تعليلة MOVLW 0x01 وذلك بغية نقلة الى مسجل الاغراض العامة 0Ch عن طريق تعليلة MOVWF 0x0C، ثم تأتي التعليلة الثالثة التعليلتان DECFSZ لتقوم بانقاص محتويات المسجل 0Ch بواحد بما أن المسجل 0Ch يحتوي الآن على القيمة 01h وعند انقاصه بواحد سيتم تصفير المسجل، بما أن المسجل 0Ch يحمل القيمة صفر الآن ، ستقوم التعليلة DECFSZ بالقفز الى التعليلة الخامسة INCF 0x0C,1 لتقوم بزيادة محتويات المسجل 0Ch بواحد لتصبح محتويات المسجل 0Ch هي القيمة 1 .



## Programming Concept

## مبادئ البرمجة

مقدمة:

لكي يعمل المسيطر الدقيق يجب أن نرشدة ولكي يتم أرشاد المسيطر الدقيق Microcontroller يجب خزن مجموعة من التعليمات المتناسقة والتي تعمل معاً لاداء مهمة معينة ، في هذه الوحدة سوف نتعرف على كيفية كتابة البرامج وعلى هيكلية البرامج ، والتعرف على مبادئ البرمجة Programming Concept .

لقد تعرفنا على البرنامج وقلنا انه مجموعة من التعليمات المتتابعة لتأدية مهمة معينة ، يتم خزن البرنامج داخل ذاكرة المسيطر الصغري بهيئة ثنائية او سلسلة من الأصفار والواحدات ، لكتابة البرنامج نحتاج لغة برمجة والتي هي عبارة عن بروتوكول او صيغة متفق عليها لكتابة البرنامج ، كما هو الحال في لغات البشر فهناك اللغة الانكليزية والفرنسية والعربية ولكل من هذه اللغات قواعد معينة لتشكل جمل سليمة ، وهكذا هو الحال مع الانظمة الرقمية ، تتم عملية كتابة البرامج بطريقتين ،

## اللغات النصية

## textual language

هذه اللغات توفر لنا كتابة التعليمات باستخدام مجموعة من الرموز والارقام والكلمات القريبة الى اللغة الانكليزية ، يتم ترجمة هذا السلاسل من الكلمات الى سلسلة من الاصفار والواحدات تفهما الماكنة تسمى Machine language باستخدام مجمع خاص يسمى Assembler في لغات واطئة المستوى Low level language مثل لغة التجميع Assembly او يسمى مترجم Compiler في لغات عالية المستوى High level language مثل لغات C,Basic,Pascal وغيرها من لغات الحاسوب.

```
Read x;
If x>4
    Out=x*4;
Else
    Out=x
```

## اللغات المرئية

## Visual language

وهي اللغات التي يكتب البرامج فيها بشكل مخططات انسيابية flow codes كما هو موضح في الشكل في Fig 6.1

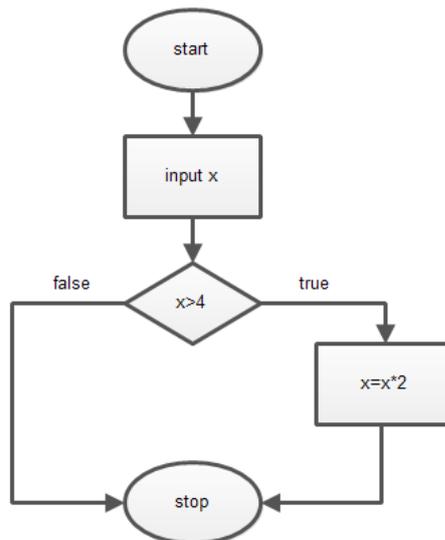


Figure 6.1

## Assembly language

هي الصيغة السهلة القراءة للبشر المقابلة للغة الآلة. فلغة الآلة عبارة عن تتابع من البتات (bits) تمثل عملية حاسوبية أو أمر للحاسوب، تصبح أسهل للقراءة عندما تستبدل برموز تعبر عنها.

ولكل معالج لغة تجميع خاصة به ، وتحتاج لغة التجميع ما يسمى المجمع Assembler وهو الذي يقوم بتحويل لغة التجميع التي يستطيع البشر قراءتها والتعديل فيها إلى لغة الآلة التي يستطيع المعالج تنفيذها. وتستخدم هذه اللغة الآن من قبل البشر وذلك لبرمجة أجزاء من نظم التشغيل أو برمجة أنظمة الزمن الحقيقي Real time system .

و تتكون اسطر برامج لغة التجميع من ثلاثة أجزاء:

- ❖ العلامة (Label) وهو ما يتم به الإشارة لسطر ما ليتم استدعاه في عمليات القفز والاستدعاء .
- ❖ الأمر (Instruction) وهو يكون مناظر في الغالب لأمر في المعالج وهو ما سيقوم المعالج بتنفيذه عند الوصول لهذا السطر أثناء تنفيذ البرنامج.
- ❖ المعامل (Operand) وهو المتغير الذي سيتم تطبيق الأمر عليه.

أمثلة على الأوامر:

```
START: BSF 0x05,5 ;initialize
```

من مميزاتهما

- ❖ تستهلك اقل مساحة
- ❖ السيطرة على البرنامج هي للمبرمج وليسيت للمجمع
- ❖ اللغة التي تخاطب المكونات المادية مباشرة ، فتسهل عملية ادارة الأجهزة.

قبل كتابة التعليمات هناك توجيهات directive يجب كتابتها مع البرنامج هذه التوجيهات توجه المعالج CPU اين هي بداية البرنامج واين هي نهاية البرنامج ، كما توجه المعالج CPU الى ترجمة البرنامج الى تعليمات يفهمها المسيطر المقصود خزن البرنامج فيه .

يتكون هيكل البرنامج من دالة رئيسية main function تكتب ضمنها التعليمات بلغة التجميع Assembly Language، كما تحتوي على تعليمة القفز goto وذلك لكي يدخل المسيطر الدقيق في حلقة تكرار لا منتهية، لينفذ البرنامج بشكر مستمر، لاحظ الشفرات التالية

```
list p=16f84a
include <p16f84a.inc>
main
    ;instructions
    ;instructions
goto main
end
```

التعليمتان الاولى والثانية تسمى توجيهات Directive تستخدم لتوجيه المترجم Assembler ليترجم البرنامج بما يتناسب مع المسيطر الدقيق pic16f84a ما التعليمة الثالثة main فهي عبارة عن عنوان label الذي ترجع الية التعليمة Goto main لتشكل حلقة لا متناهية ومن خلال هذه الحلقة نكتب التعليمات التي سنتكرر بشكل مستمر، اما التعليمة الاخيرة End فهي تدل على نهاية البرنامج.

يتكون المسيطر الدقيق pic16f84a من 13 منفذ ولكي نستخدم هذه المنافذ يجب تهيئتها، اي تحديد اي طرف من اطراف المنافذ هي طرف دخول او طرف خروج، عادة عملية التهيئة تتم مرة واحدة عند بداية تشغيل المسيطر الدقيق، لانها تنفذ مرة واحدة فقط لذلك نكتبها خارج الدالة الرئيسية main كالآتي

```
list p=16f84a
include <p16f84a.inc>
BSF 0x03,5 ; Go to Bank 1 "Bit Set Flag" Register, Bit
MOVLW 0xff ; mov xxx1 1111 to W Register
MOVWF 0x85 ; Move W to Trisa
MOVLW 0x00 ; mov 0000 0000 to W Register
MOVWF 0x86 ; Mov W to Trisb
BCF 0x83,5 ; Go to Bank 0 "Bit Clear Flag" Register, Bit

main
    ;instructions
    ;instructions
goto main
end
```

يتم تهيئة المنافذ عن طريق المسجل trisa, trisb وكما هو معلوم من الوحدة الخامسة أن هذين المسجلين يقعان في bank 1 من file register لذلك يجب ان نتحول الى bank 1 ليتم تحميلهما بالبيانات المطلوبة ويتم ذلك عن طريق المسجل status وبالتحديد البت

الخامس Rp0 برفعة الى القيمة واحد عن طريق التعليمة Bsf 0x03,5 وبعد ان تحولنا الى bank1 نتسطيع تحميل المسجل trisa بالقيمة 0xff اي ما يقابلها بالنظام الثنائي  $(1111\ 1111)^2$  والتي تعني ان المنفذ porta هو منفذ ادخال بيانات input ويتم ذلك عن طريق التعليمة movlw 0xff والتعليمة movwf 0x05 ثم نهيء المنفذ port b بنفس الطريقة ثم بعد ذلك نعود الى bank 0 وذلك بتصفير البت الخامس Rp0 من المسجل Status عن طريق التعليمة bcf 0x83,5 ، لمزيد من المعلومات عن استخدام التعليمات راجع الوحدة الخامسة.

## Writing Operations

## عملية الكتابة

عملية الكتابة Writing الى المنفذ او الأخراج output عملية مهمة وذلك لأخراج البيانات الى العالم الخارجي ، وذلك للاستفادة منها في تشغيل المصابيح LED او سوق المرحلات Relays ، البرنامج التالي يبين عملية الكتابة writing الى المنفذ port a عن طريق البت الثالث.

```
list p=16f84a
include <p16f84a.inc>
BSF 0x03,5 ; Go to Bank 1 "Bit Set Flag" Register, Bit
MOVLW 0xfb ; mov xx11 1011 to W Register
MOVWF 0x85 ; Move W to Trisa
BCF 0x83,5 ; Go to Bank 0 "Bit Clear Flag" Register, Bit
Main
    BSF 0x05,2
    goto main
end
```

قمنا اولا بعملية تهيئة المنفذ porta ليكون البت الثالث هو طرف اخرج output عن طريق تحميل السجل trisa بالقيمة 0xFB ، ثم قمنا بكتابة تعليماتنا داخل الدالة الرئيسية main function وذلك برفع البت الثالث من porta بالقيمة واحد وذلك عن طريق التعليمة bcf 0x83,5 ، تذكر ان عنوان porta هو 0x05 في file register ، اذا وصلت دايود باعث للضوء LED ستجده قد اشتغل.

## Reading Operations

## عملية القراءة

عملية القراءة Reading من المنافذ مهمة لقراءة البيانات من المحيط الخارجي مثلاً لمعرفة حالة المفاتيح Switches او المتحسسات Sensors ، البرنامج التالي يبين عملية القراءة Reading من المنفذ port b عن طريق البت السادس ، اذا كان البت السادس في حالة توصيل نشغل البت الرابع من portb .

```
list p=16f84a
include <p16f84a.inc>
BSF 0x03,5 ; Go to Bank 1
MOVLW 0x40 ; move 0100 0000 to W Register
MOVWF 0x86 ; Mov W to Trisb
BCF 0x83,5 ; Go to Bank 0
Main
    BTFSS 0x06,6 ; "Bit Test Flag Skip if Set " Register,bit
    GOTO bit_is_reset
    GOTO Main
bit_is_reset BSF 0x06,4
    goto Main
end
```

قمنا اولاً بعملية تهيئة المنفذ portb ليكون البت السادس هو طرف دخول input عن طريق تحميل السجل trisb بالقيمة 0x40 ، ثم قمنا بكتابة تعليماتنا داخل الدالة الرئيسية main function وذلك عن طريق فحص البت السادس من portb وذلك عن طريق التعليمة btfs وبعد تحقق الشرط قمنا برفع البت الرابع من portb ، راجع الوحدة الخامسة لمزيد من المعلومات حول التعليمات instructions .

## Programming Concept

## مبادئ البرمجة

توفر لغة البرمجة مجموعة من اللبنيات الأساسية للاستناد عليها خلال عملية تكوين البرنامج ومجموعة من القواعد التي تمكن من التعامل مع الشفرات وتنظيمها بشكل متناسق لغرض أداء العمل المطلوب.

### Variable

### المتغيرات

في عملية برمجة الميسطرات الدقيقة ، المتغيرات هي مواقع خزنية داخل ذاكرة الميسطر الصغري Pic16f84a لهذه المواقع الخزنية أسم محدد ، ويمكن تعريف المتغيرات في الميسطر الصغري على غرار باقي لغات البرمجة ، وتكون الصيغة كآلاتي

```
Name_Of_variable Set Value
```

أمثلة على المتغيرات

```
Level set 50  
Set_point set 20
```

ويمكن كتابة المتغيرات بالصيغة التالية،

```
Variable Name_Of_Constant = Value
```

أمثلة على المتغيرات

```
Variable level= 50  
Variable Set_point = 20
```

## Constant

هي قيم مسندة لها اسماء معرفة لتسهيل التعامل مع القيم Values, يمكن استخدام تعليمة EQU لاسناد أسماء للقيم Values ، وتكون الصيغة كآلاتي

```
NameOf_Constant EQU Value
```

أمثلة على الثوابت

```
Trisa EQU 0x85
Porta EQU 0x05
```

ويمكن كتابتها بالصيغة التالية

```
Constatnt Name_Of_Constant = Value
```

أمثلة على الثوابت

```
Constatnt trisa= 0x85
Constatnt trisb= 0x05
```

## Comment

وهي عبارة عن نصوص غير قابلة للترجمه ولا يفسرها المعالج تستخدم للإرشاد والتذكير وما يميز التعليقات انها مسبوقة بالرمز ";" وتأخذ الصيغة التالية

```
; # this is comment #
BCF 0x0f,1 ;this is comment to clear bit 1 of register 0x0f
```

تقوم هذه التعليمة بأسناد نص text الى قيمة Value ، وتأخذ الصيغة التالية

```
#DEFINE TEXT VALUE
```

Example:

```
#DEFINE Input 1
```

```
#DEFINE Output 0
```

البرنامجات التالية متكافئان ، كتب البرنامج الاول بصيغة منظمة باستخدام المعرفات ، اما البرنامج الثاني فقد كتب بصيغة مباشرة

### ❖ مثال 6.1

سنتعلم في هذا المثال كيفية إجراء عملية الضرب في الميسطر الصغري ، لنفرض لدينا متغيران وهما X,Y والمتغير Z ونريد إجراء العملية الآتية

```
Z = X * Y
```

عملية الضرب ببساطة هي عملية جمع ، أي أن عملية الضرب X\*Y لنفرض نريد إجراء العملية X\*5 فالعملية هي

```
Z = X + X + X + X + X ;adding X five times
```

لقد قمنا بعملية جمع المتغير X خمس مرات وذلك لان قيمة المتغير Y هي 5، عملية الضرب في هذه الحالة تصلح فقط عندما تكون الاعداد صحيحة والناتج اقل من 255 أي أن عملية الضرب يكون ناتجها 8-bit، هناك خوارزميات أعقد يمكنك الاستعانة بها ، لكي يكون الشرح واضح لم اتطرق لها ،

لاحظ الشفرة التالية ، التي قمنا بتعريف ثلاث متغيرات int1,int2,product وحملنا المتغيرات int1=8,int2=7 ثم قمنا بعملية تصفير كل من المتغيران product,Wreg ، بعد ذلك قمنا بعمل حلقة تكرر بقدرة قيمة int2 ، وذلك عن طريق التعليمة Decfsz int2 والتي تطرح القيمة واحد من المتغير int2 اذا لم تساوي صفر يقفز الى العنوان MULT\_LOOP ليقوم بعملية جمع المتغير int1 مع قيمة المسجل WREG الذي ناتج عملية الجمع السابقة وهكذا

```
list p=16f84a
include <p16f84a.inc>
#define int1    0x0D
#define int2    0x0E
#define product 0x0F
main
    movlw    0x08    ;    Multiply 8
    movwf    int1
    movlw    0x07    ;    by 7
    movwf    int2
; MULTIPLY_SETUP
    clrw
    clrf    product
MULT_LOOP
    movfw    int1
    addwf    product    ;    Move int1 to W, then add
                        ;    to product
    decfsz   int2
    goto    MULT_LOOP    ;    Every time we go through the loop
                        ;    we decrement int2, until it's Zero
goto main
end
```

## التعليمة

## ORG

تقوم هذه التعليمة بتحديد بداية موقع تخزين البرنامج في الذاكرة ، وبشكل افتراضي يتم تخزين البرنامج بشكل افتراضي عند الموقع 0x00 ان لم تكتب التعليمة ORG 0x00

```
Start org 0x00
movlw 0xFF
movwf PORTB
```

## التعليمة

## CBLOCK

تقوم هذه التعليمة بتعيين اسماء لقيم ولكن بشكل متسلسل ، حيث تقوم بأعطاء قيمة الاولية لاول ثابت Constant ، وبقية الثوابت يتم تحديد قيمها ذاتيا بالاعتماد على القيمة الاولية ، لاحظ المثال التالي

```
Cblock 0x02
First, second, third ;first=0x02, second=0x03, third=0x04
endc
```

لاحظ انك اسندت القيمة 0x02 الى الثابت first اما الثابت second سيكون قيمته first+1 والثابت third سيكون second+1 ، ويمكن زيادة القيمة بأكثر من واحد ، لاحظ المثال التالي.

```
Cblock 0x02
First:4, second:2, third ;first=0x06, second=0x08, third=0x09
endc
```

فمنا باسندت القيمة 0x02 الى الثابت first وبما ان قيمة الزيادة هي 4 نتيجة وجود اللاحقة 4: ستكون قيمة 0x02+0x04=0x06 اي أن قيمة first=0x06 ، اما الثابت second سيكون قيمته first+2 والثابت third سيكون second+1.

## Macro Instruction

## الماكرو

عبارة عن مجموعة من التعليمات تكتب بشكل اجراء لها اسم محدد تسهل عمل المبرمج اثناء البرمجة ، لكن اثناء ترجمة البرنامج الى لغة الالة سيتم تبديل كل استدعاءات الماكرو بالشفرة الفعلية للماكرو، وتأخذ الصيغة التالية ،

```
My_micro_name macro
; instructions
endm
```

## Subroutine

## الاجراءات

الاجراءات Subroutine او البرامج الفرعية عبارة عن برامج-او اجزاء برامج-ثنائية يتم استخدامها لاداء غرض معين ومن فوائدها:

- تقليل وتلافي التكرار في بناء البرامج مرة اخرى.
- تقليل الوقت المطلوب لبناء البرامج والمشارع.
- التقليل من الذاكرة المطلوبة لشفرات وبيانات المشروع.

```
list p=16f84a
include <p16f84a.inc>
  main

  CALL subroutine
  Goto main
Subroutine
  ;Instruction
Return
end
```

يجب أن تكون برامجك منظمة ومنسقة ومدعمة بالتعليقات ليسهل على الآخرين قراءتها وفهما ، كما تسهل عليك تذكرها عند الرجوع اليها مستقبلاً ، أنظر الى البرنامج التالي كتب بصيغة منظمة بأستخدام الاجراءات والماكرو والمعرفات ،

```
list p=16f84a
include <p16f84a.inc>
;***** my first program *****
;code of subroutine
Goto   Initialized
      bank0  macro
          bsf 0x03,5
      endm
      bank0  macro
          bcf 0x83,5
      endm
Initialized
;give identifiers and configures ports
      Portb EQU 0x06
      Porta EQU 0x05
      Trisa  EQU 0x85
      Trisb  EQU 0x86
;set port b input
      Bank1
      Movlw 0xff
      Movwf Trisb
;set port a output
      Clrf Trisa
      Bank0
Main
      ; instruction goes here
end
```

تستخدم تعليمات التفرع للقفز jumping خلال سير البرنامج لتحقيق اغراض معينة او قفز تعليمات معينة او تكرار تعليمات وهكذا،

## if statement

## تعلية اذا الشرطية

تعتبر تعليمة اذا الشرطية if statement من التعليمات المهمة في لغات البرمجة ، وتأخذ الصيغة العامة التالية

```
If ( Condition )
    ;do something if condition is true;
Else
    ; do something if condition is false;
```

ولنوضح تعليمة اذا الشرطية فلنأخذ المثال التالي .

### ❖ مثال 6.2

قم بعملية فحص البت الخامس من مسجل الغرض العام 0x0E اذا كان صفر يتم تحميل المتغير Val بالقيمة 3 ، ام اذا كان البت الخامس واحد نحمل المتغير Val بالقيمة 5

### الحل :

تعتمد عملية المقارنة في هذا المثال على التعليمة Btfss والتي البت الخامس من المسجل 0x0E اذا كان واحد يقفز الى التعليمة التي تسند القيمة 5 الى المتغير val ، ثم يواجه التعليمة Goto end\_if ليقفز الى نهاية تركيبه اذا الشرطية ، لتعاد عملية المقارنو من جديد ، أما اذا كانت قيمة البت الخامس من المسجل 0x0E هي صفر تنفذ التعليمة القفز Goto bin\_is\_reset ليقفز الى التعليمة 3 Val set ليتم تحميل المتغير Val بالقيمة 3 وهكذا .

```
list p=16f84a
include <p16f84a.inc>

Variable Val = 0
main
    BTFSS 0x0E,5
    GOTO bin_is_reset
    Val Set 5
    GOTO end_if

bin_is_reset Val Set 3
end_if

    goto main

end
```

## While Loop

تستخدم التعلیمة While لتكرار مجموعة من التعلیمات ، ويمكن استخدام تعلیمة التكرار في لغة التجميع بالاسلوب الآتي

```
list p=16f84a
include <p16f84a.inc>
main
    MOVLW 0x0A
    MOVWF 0x0E
    LOOP DECFSZ 0x0E
    GOTO still_in_loop
    GOTO end_loop
    still_in_loop inst.
    Inst.          ;instruction to be executed inside loop
    Inst.
    GOTO LOOP

    end_loop

goto main
end
```

قمنا في البرنامج السابق بعملية تكرار loop تقوم بتكرار مجموعة تعلیمات ١٠ مرات ، اولا قمنا بتحميل العدد ١٠ او ما يقابلها بالنظام السادس عشر 0x0A الى مسجل الاغراض العامة 0x0E ثم قمنا بعملية طرح هذا المسجل وفحصه الى ان يتم تصفير المسجل 0x0E وذلك عن طريق التعلیمة Decfsz . ويمكن ايضاً استخدام الصيغة التالية لتنفيذ حلقة التكرار

```
Variable x=0
While x<10
    ;Instructions
endw
```

اكتب برنامج لجعل LED يومض Flashing بين فترة واخرى باستخدام الاجراءات Subroutine

الحل:

المطلوب هو جعل دايود باعث للضوء LED يومض بين فترة واخرى كما هو مبين في الشكل Fig 6.2 ، وذلك عن طريق الطرف RB7

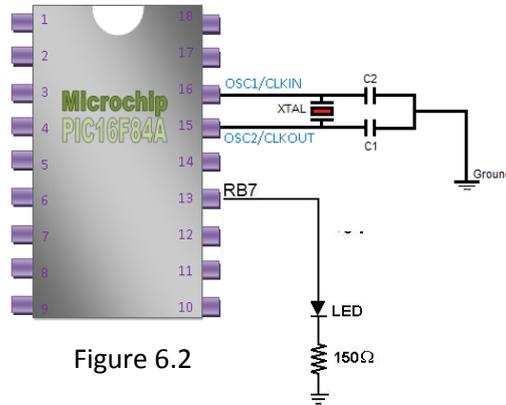


Figure 6.2

قمنا بكتابة اجراء Subroutine اسمة Delay يحتوي هذا الاجراء على حلقة تكرار ، قمنا في بداية الاجراء بتحميل القيمة 0xff الى مسجل الاغراض العامة 0x0E وقمنا بطرح هذه المسجل ومقارنة مع القيمة صفر عن طريق التعليمة Decfsz فاذا كانت قيمة المسجل 0x0E تساوي صفر يخرج المسيطر من الاجراء delay والاتكرر العملية الى أن تصبح قيمة المسجل 0x0E تساوي صفر ، أما الدالة الرئيسية main function فهي تحتوي على تعريف المنفذ portb والذي اصبح منفذ أخراج ، وقمنا بعملية أخراج إشارة الى البت السابع من portb عن طريق التعليمة bsf portb,7 ثم تالها تأخير delay ثم تالها أطفاء bcf portb,7 وهكذا

```
list p=16f84a
include <p16f84a.inc>
STATUS_BANK0 EQU 03h
STATUS_BANK1 EQU 83h
trisb EQU 86
portb EQU 06
    BSF STATUS_BANK0 ,5
    MOVLW 0x00          ; b"0000 0000"
    MOVWF trisb
    BCF STATUS_BANK1 ,5

main
    BSF portb,7
    CALL Delay
    BCF portb,7
    CALL Delay
    goto main

Delay
    MOVLW 0xff
    MOVWF 0x0E
Repeat
    Decfz 0E,1
    GOTO Repeat
RETURN
```

## ❖ مثال 6.4

أكتب برنامج يقوم بفحص قيمة الموقع 0x0E هل هي اكبر من العدد 4 أو اصغر من العدد 4 أو تساوي العدد 4 .

الحل:

الفكرة هي بطرح العد المطلوب فحصه أو مقارنة من العدد 4 وذلك يتم عن طريق اجراء عملية الطرح ثم فحص مسجل الحالة Status اذا كان العدد الاول يساوي العدد الثاني فان قيمة zero flag =1 اما اذا كان العدد الاول اكبر من العدد الثاني فان carry flag=1 اما اذا كان العدد الاول اصغر من العدد الثاني فان carry flag=0 ، راجع الوحدة الاولى الطرح بأستخدام المتمم الثاني ،ويتم برمجة العملية كآلاتي .

```
list p=16f84a
include <p16f84a.inc>
;identifiers
my_location EQU 0x0E
Status EQU 0x03
main
    Movlw 0x04           ;move literal to Work register value
    SUBWF my_location   ;subtract w reg. from specified register
    BTFSC Status,2
    Goto equ_           ;goto equal procedure
    Btfsc Status,0
    GOTO less_than_    ;goto less than procedure
    GOTO greater_than_ ;goto to greater then procedure
    Equ_
        instruction.
        Instruction.
        GOTO main
    less_than_
        instruction.
        Instruction.
        GOTO main
    greater_than_
        instruction.
        Instruction.
        goto main
end
```

يقصد بالعنونة المباشرة هي الوصول Accessing الى المسجل مباشرة ، كما هو مبين في المثال الآتي

```
list p=16f84a
include <p16f84a.inc>
main
    DECF 0x0E,1
    RLF 0x0E,1

goto main
end
```

قمنا في المثال السابق بطرح محتويات المسجل 0x0E مباشرة فقط بذكر عنوان المسجل ، ثم قمنا بتزحيف المزجل بت واحد الى اليمين ، هذه الطريقة تسمى العنونة المباشرة .

يقصد بالعنونة الغير المباشرة هي الوصول Accessing الى المسجل ما يتم بطريقة غير مباشرة ، وذلك باستخدام المسجل FSR ، كما هو مبين في المثال الآتي ، راجع الوحدة الخامسة لمزيد من المعلومات عن المسجل FSR ، والمسجل INDF .

```
list p=16f84a
include <p16f84a.inc>
main
    MOVLW 0x0E
    MOVWF 0x04 ;FSR
    DECF 0x00 ;INDF
    RRF 0x00

goto main
end
```

## ❖ مثال 6.5

البرنامج التالي يستخدم للسيطرة على مستوى خزان ، وذلك عن طريق الفحص المستمر للمتحمس السفلي ، عندما تصبح قيمة المتحمس السفلي صفر فهذا يدل على ان الخزان فارغ ، نقوم بفتح الصمام ، وننتظر الى ان تصبح قيمة المتحمس العلوي تساوي واحد والتي تدل على ان الخزان قد ملئ وتعاد العملية ، الشكل Fig 6.3 يبين رسم توضيحي للعملية .

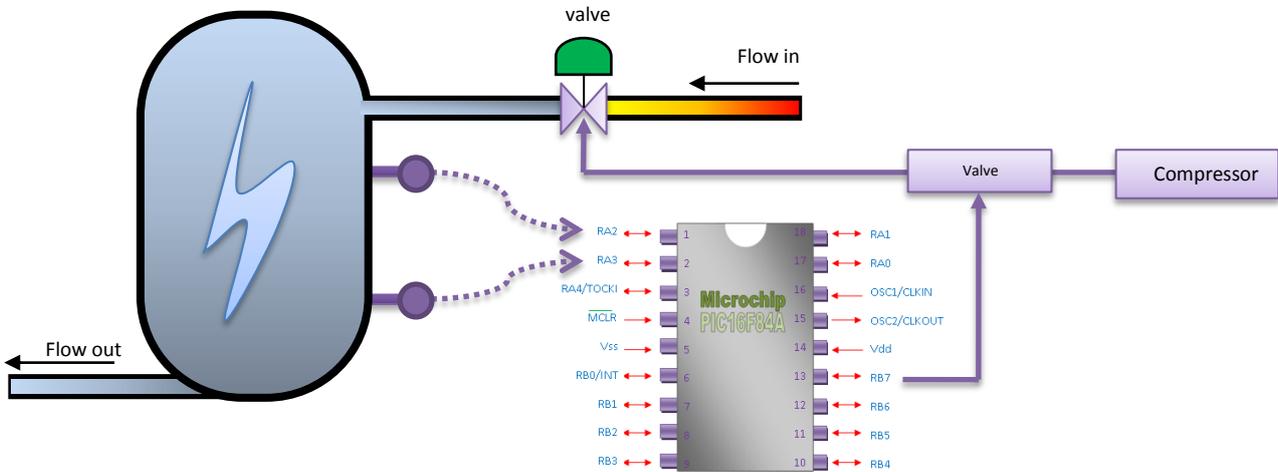


Figure 6.3

```
list p=16f84a
include <p16f84a.inc>

;##### Control tanl level #####

STATUS_BANK0 EQU 03h
STATUS_BANK1 EQU 83h
trisa EQU 0x85
trisb EQU 0x86
porta EQU 0x05
portb EQU 0x06
BSF STATUS_BANK0 ,5
MOVLW 0x0Ch ; xxxx 11xx bit3 =LL bit2=HL
MOVWF trisa
MOVLW 0x00h ; 0xxx xxxx
MOVWF trisb
BCF STATUS_BANK1 ,5

MAIN BTFSF porta ,3 ; test the tank if empty
GOTO MAIN
BSF portb ,7 ;1xxx xxxx turn on bump
REPEAT BTFSF porta ,2 ; test the tank if full
GOTO REPEAT
BCF portb ,7 ;1xxx xxxx turn on bump

GOTO MAIN

end
```

إذا أردت ان ترمج لوحة مفاتيح تُربط الى المُسيطر الصغري ، بالطبع سوف تقوم بفحص Scan ، كل طرف من أطراف المُسيطر الدقيق الذي وصل الى لوحة المفاتيح ، للتعرف على الزر المضغوط ، المثال التالي يوضح ذلك،

```
Portb equ 0x06
Loop   BTFSC portb,0
        ; action for key_0
      BTFSC portb,1
        ; action for key_1
      BTFSC portb,2
        ; action for key_2
      Goto Loop
        ;rest of program
```

هذا النوع من الفحص Scan، يُسمى الانتخاب polling، السؤال الذي يتبادر الى الذهن؟؟

كيف سيتم تنفيذ برنامج آخر مخزون داخل شريحة المُسيطر الصغري ، في الوقت الذي تجري فيه عملية الانتخاب Polling ، طبعا مُحال لأننا داخل حلقة مغلقة close loop لفحص أطراف المُسيطر المربوطة الى لوحة المفاتيح وعند تحسس أي طرف نقوم بتأدية المهمة المطلوبة ، أذن نستنتج من ذلك،

❖ المُسيطر لا يستطيع تنفيذ أكثر من مهمة ، و اذا حاولت وتمكنت، فستحاول تنفيذ المهمة ثم ترجع لتفحص الأطراف Scanning بشكل متكرر، تخيل لو أن المُسيطر ينفذ المهمة الأخرى ، وفي نفس الوقت ضغط شخص على لوحة المفاتيح بسرعة ثم ترك release المفاتيح ، عندها المُسيطر عندما يعود لفحص الأطراف لا يجد أي طرف يشير الى وجود زر مضغوط؟؟

❖ ممكن إن المُسيطر لا يتحسس أو يفقد الإشارة في عملية الانتخاب Polling..

## فلسفة المقاطعة

## interrupt philosophy

معنى المقاطعة في اللغة وهو مقاطعة شيء أو عمل ما لأداء عمل أكثر أهمية ثم العودة الى العمل السابق .. وهو فعلاً ما يحدث في المُسيطر الصغري حيث أن المقاطعة interrupt ، تقاطع عمل المُسيطر الحالي لتأدية مهمة أخرى، بعد إتمام المهمة يعود المُسيطر لوضعه الطبيعي.

## Mechanism of interrupt

## آلية عمل المقاطعات

المقاطعة تقريباً مشابهة لعمَل الإجراءات Subroutine ، حيث يوجد برنامج فرعي يستدعى عند حدوث المقاطعة، هذا البرنامج الفرعي يُسمى برنامج خدمة المقاطعات (ISR(interrupt service routine) ، يتم استدعاءه بطريقتين

من داخل البرنامج الرئيسي Main program باستخدام تعليمات خاصة وهو ما يسمى بـ software interrupt. ❏

عن طريق أطراف المسيطر الدقيق وهو ما يسمى بـ Hardware interrupt ❏

الشكل Figure 6.5 يبين برنامج خدمة المقاطعة مع البرنامج الرئيس main function

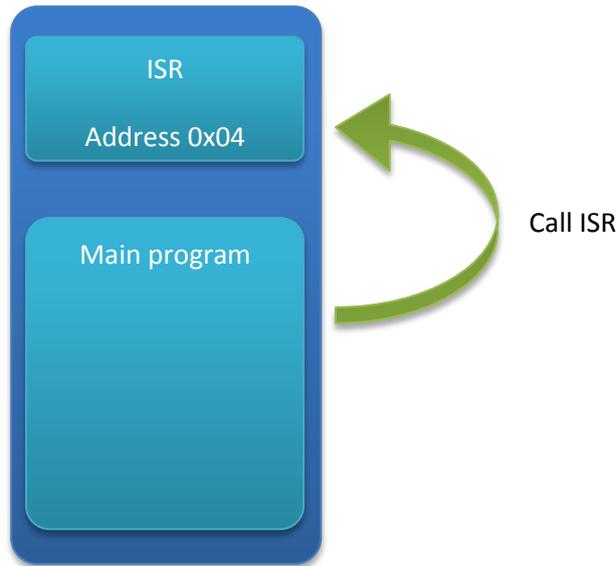


Figure 6.5

حسناً ... دعني نلقي الضوء على موضوع استدعاء خدمة ISR عن طريق أطراف المسيطر الصغري ، عند النظر الى المسيطر الصغري عند الطرف RB0 ستجد ان لة وَصيفة إضافية ألتى هي INT ، أي انة ممكن ان يُستخدم هَذَا الطرف لخدمة المقاطعة الخارجية ، وكذلك RB4-RB7 ممكن تفعيل خدمة المقاطعة عليها ، قبل استخدام المقاطعات يجب علينا ،

- ❖ اخبار المسيطر الصغري microcontroller باننا سنستخدم خدمة المقاطعة.
- ❖ يجب تحديد أي طرف سيستخدم لخدمة المقاطعة واي طرف سيستخدم كخدمة I/O.

لتفعيل خدمة المقاطعة عند الطرف RBO ، نتبع الخطوات التالية

١- يمكن خدمة المقاطعات العامة GIE من المسجل INTCON كالاتي

BSF 0x0b,7

٢- يمكن خدمة المقاطعة عند الطرف RBO بتمكين الخانة INTE من المسجل INTCON كالاتي

BSF 0x0b,4

عند هذه اللحظة ، المسيطر الدقيق سيكون لة العلم متى سيذهب لبرنامج خدمة المقاطعة ISR ، وذلك عند تحسس اشارة عند الطرف RBO ، فقط بقي شئ واحد مهم هل المقاطعة ستتم عندما !!؟

❖ الطرف RBO يتحول من المنطق الواحد الى المنطق الصفر falling edge

❖ الطرف RBO يتحول من المنطق الصفر الى المنطق الواحد rising edge

هذا يتم تحديده من خلال مسجل خاص داخل المسيطر الدقيق يدعى Option ، الخانة التي تهمننا حالياً هي الخانة السادسة حيث

Intedg = bit 1 (rising edge 0-5V) default value  
Intedg = bit 0 (falling edge 5V-0)

## Interrupt flag

## علم المقاطعة

يوجد هذا العلم في السجل INTCON في الخانة (1) bit ، وظيفه هذا البت هو عند حدوث مقاطعة ويدخل المسيطر الصغري الى برنامج خدمة المقاطعة ISR يرفع هذا العلم الى القيمة واحد وذلك لمنع أي مقاطعة تقاطع عمل المسيطر الصغري داخل ISR ، لسوء الحظ ان المسيطر الصغري عند خروجه من خدمة ISR فإن هذا العلم لا يرجع الى حالة الصفر أي أن المسيطر لا يقبل أي مقاطعة طالما أن هذا البت واحد ، لذلك يجب تصفير reset هذه الخانة برمجياً عند الخروج من خدمة المقاطعة .



ملاحظة:

عند بداية تشغيل المسيطر الصغري بشكل طبيعي يؤشر عداد البرنامج PC على الموقع 0x00 من الذاكرة ، عند حدوث مقاطعة فان عداد البرنامج يحمل بالقيمة 0x04 مؤدياً الى جعل المسيطر الصغري يؤشر الى موقع الذاكرة 0x04 لذلك يجب اخبار PIC باننا سوف نحجز الموقع 0x04 لبرنامج خدمة المقاطعة ISR ونفصله عن البرنامج الرئيسي كالاتي ،

```

ORG 0000h ; المسيطر سيذهب الى هذا العنوان عند بداية التشغيل
GOTO Main ; القفز الى دالة البرنامج الرئيسي
ORG 0004h ; هذا العنوان هو بداية برنامج خدمة المقاطعة

; التعليمات المراد تنفيذها عن حدوث مقاطعة

RETFIE ; العودة من برنامج خدمة المقاطعات

Main

; التعليمات التي تريد تنفيذها داخل الدالة الرئيسية

Goto Main

End

```

#### ❖ مثال 6.6

في هذا المثال سوف نقوم بتفعيل خدمة المقاطعة عند الطرف Rb0 ، حيث اذا كان هناك إشارة عند الطرف Rb0 ، يتم استدعاء برنامج خدمة المقاطعات ISR بشكل ذاتي وذلك لكي يرفع البت الاول من المسجل porta الى القيم واحد .

```

list p=16f84a
include <p16f84a.inc>

ORG 0000h ; بيده المسيطر التنفيذ من هذا العنوان
goto init
ORG 0004h ; بداية برنامج خدمة المقاطعات
    bsf 0x05,0 ;0x05 is porta
    BCF 0x0b,1 ;INTF=0 of INTCON

RETFIE

Init ; عمليات تهيئة المسيطر الصغري
    bsf 0x03,5
    clrf 0x86
    bsf 0x86,0
    bcf 0x85,0
    bcf 0x83,5

Main ; بداية البرنامج الرئيسي
    bsf 0x0b,7 ;GIE=1 of INTCON
    bsf 0x0b,4 ;INTE=1 of INTCON

Goto Main

End

```

قبل ان ندخل في موضوع المؤقت ، دعنا نتحدث عن وحدة سيطرة قليلاً ، وحدة السيطرة او الوحدة التي تولد دورة الماكنة machine cycle generation هي في الحقيقية عبارة عن عداد حلقي مكون من 4-bit كل دورة كاملة لهذا العداد تشير الى نهاية تعليمة ، سرعة هذا العداد الحلقي او سرعة دورة الماكنة تعتمد على سرعة المذبذب oscillator , الشكل Fig 6.5 يبين وحدة السيطرة

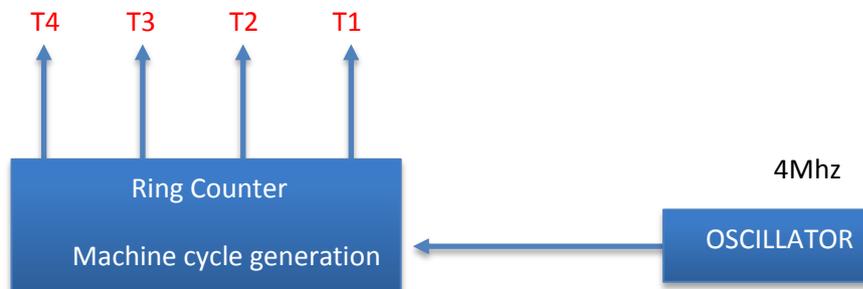


Figure 6.5

يتم خلال الاربع دورات،

١- T1 : خلال هذه الدورة يتم نقل محتويات PC الى الذاكرة لجلب التعليمة الى مسجل IR ، ونفس الوقت يزداد عداد البرنامج بواحد ليشير الى التعليمة التالية .

٢- T2 , T3 , T4 : دورة تنفيذ التعليمة وهي متغيرة تعتمد على نوع التعليمة .

مجمل الأربع نبضات هذه التي يتم فيها جلب وتنفيذ التعليمة ، تسمى دورة التعليمة machine cycle مع ملاحظة انه بعض التعليمات تحتاج الى اكثر من دورة ماكنة one machine cycle .

كلنا نعلم ان التردد مقلوب الزمن أي

$$t=1/4\text{MHZ}=0.25 \text{ micro second}$$

وبما ان التعليمة تحتاج اربع نبضات من دورة الماكنة لكي تنفذ بشكل كامل

$$\text{Instruction cycle}=0.25 \text{ micro second} * 4 \text{ pulse} = 1 \text{ micro second}$$

أي ان التعليمة تحتاج 1 micro second لكي تنفذ.

ملاحظة:



كل التعليمات في المسيطر الصغري تاخذ دورة ماكنة واحدة أي 1 micro باستثناء التعليمات التي فيها عملية القفز jump .

اذن نستنتج من ذلك بإمكان ان نحسب الوقت الذي يستغرقه المسيطر الدقيق لاتمام تنفيذ برنامجنا مثلا البرنامج التالي

MOVLW 0x02	1 micro
MOVWF 0x0E	1 micro
INCF 0xE	1 micro

اذن الوقت المستغرق لتنفيذ البرنامج السابق هو 3 micro ، لاحظ البرنامج التالي ،

	MOVLW 0x0A	1 micro
	MOVWF 0x0E	1 micro
Loop	DECFSZ 0x0E,1	1 micro or 2 micro = 1*9 + 2 =11 micro second
	GOTO loop	2 micro 2*9 microsecond =18 microsecond
	End	1 micro

الوقت المستغرق هو 18+11+3=32 microsecond

نستنتج من ذلك ان بإمكانك ان تكتب دالة او اجراء تاخير حسب المدة التي ترغب بها بالاعتماد على زمن كل تعليمة،

## Timer modules

## وحدة المؤقت

المؤقت في المسيطر الصغري PIC16f84A هو مؤقت ذو سعة 8-bit يسمى TMRO والذي يمكن استخدامه كمؤقت أو عداد ، عند استخدام هذا المؤقت كعداد Counter ، يقوم هذا المؤقت بالزيادة عند كل نبضة تأتي الى الطرف TOCK1 ، أما عند استخدام كمؤقت Timer يزداد هذا المسجل بشكل ذاتي بنسبة تعتمد على مصدر التردد ومقسم التردد prescaler assignment الموجود في المسجل option ، لتفعيل المؤقت Timer يجب

## Timer Enable

## تفعيل خدمة المؤقت

عرفنا ان المؤقت من الوحدات السابقة عبارة عن مسجل ذو 8-bit وهذا المسجل تعتمد سرعة الزيادة فيه بنسبة تعتمد على تردد الدخل بالإضافة الى مقسم التردد ، ماذا يحدث فعلاً ، عندما يصل مسجل المؤقت الى 256 وهي اقصى قيمة له لانه مسجل ذو 8-bit وعند زيادة هذه القيمة بواحد يصبح المسجل يحمل القيمة 0 مشيراً بذلك الى حدوث حالة طفحان ، هذه الحالة تستدعي خدمة المقاطعة عند تفعيل مقاطعة المؤقت ، لتنفيذ برنامج خدمة المقاطعات ISR ، ما يهما الان هي العلاقة التي من خلالها سنعرف زمن استدعاء خدمة المقاطعة وهي

$$\text{timer frequency} = (f/4) \times \text{Prescaler} \quad \text{eq. (1)}$$

$$\text{Overflow time} = 1/\text{timer frequency} \times (256 - \text{tmr0}) \quad \text{eq. (2)}$$

حيثُ أن قيمة tmr0 تمثل قيمة مسجل المؤقت ، ومقسم التردد prescaler تمثل نسبة التقسيم والتي هي موضحة في الجدول التالي

Ps2,ps1,ps0	نسبة التقسيم timer
000	1/2
001	1/4
010	1/8
011	1/16
100	1/32
101	1/64
110	1/128
111	1/256

Table 6.1

### ❖ مثال 6.7

لنفرض أنك حملت المسجل tmr0 بالقيمة 0x64 وقيمة مقسم التردد هي ps2,ps1,ps0 هي 010 فما هو زمن حدوث حالة الطفحان اذا كان التردد 4 Mhz

الحل:

نحول القيمة 0x64 التي هي بالنظام السادس عشر الى ما يقابلها بالنظام العشري والتي هي 100 ، وبما أن مقسم التردد حمل بالقيمة 010 اذن نسبة التقسيم هي 1/8 من الجدول Table 6.1، وكان تردد المصدر هو 4 Mhz .

$$\text{timer frequency} = (4 \text{ Mhz}/4) \times 1/8 = 0.125 \text{ Mhz}$$

$$\text{Overflow time} = 1/0.125 \text{ Mhz} \times (256-100) = 1024 \mu\text{s}$$

لنقوم الآن بتعديل بسيط على المعادلات أعلاة ، لنعوض المعادلة رقم (1) في المعادلة رقم (2) فنحصل

$$\text{overflow time} = \frac{1}{\frac{f}{4} \times \text{prescaler}} * (256 - \text{tmr0})$$

$$\text{overflow time} = \frac{1}{\frac{f}{4}} * \frac{1}{\text{prescaler}} * (256 - \text{tmr0})$$

$$\text{overflow time} = 4 * T * \frac{1}{\text{prescaler}} * (256 - \text{tmr0}) \quad \text{eq. (3)}$$

حيثُ أن الزمن T هو مقلوب تردد المصدر

$$T = \frac{1}{f}$$

الآن نريد معرفة القيمة التي سنحمل بها المسجل tmr0 عند زمن طفحان معين ، سنقوم بتعديل المعادلة رقم (3) لتصبح بالشكل التالي

$$tmr0 = 256 - \frac{\text{overflow time}}{4 * T * \frac{1}{\text{prescaler}}} \quad \text{eq. (4)}$$

الجدول Table 6.2 يبين قيم مختلفة للمسجل Tmr0 عند قيم مختلفة من overflow ، القيم الفارغة دلالة أن القية الناتجة هي قيمة سالبة ولا يجوز تحميل المسجل Tmr0 بالقيم السالبة .

Time to Overflow(μs)	Prescaler Values							
	2	4	8	16	32	64	128	256
100	206	231	243	250	253	254	-	-
200	156	206	231	243	250	253	254	-
300	106	181	218	237	246	251	253	255
400	56	156	206	231	243	250	253	254
500	6	131	193	224	240	248	252	254
600	-	106	181	218	237	16	251	253
700	-	81	168	212	234	245	250	253
800	-	56	156	206	231	243	250	253
1000	-	3	131	193	225	240	248	252
5000	-	-	-	-	100	178	77	236
10000	-	-	-	-	-	100	178	217
20000	-	-	-	-	-	-	100	178
30000	-	-	-	-	-	-	-	139
40000	-	-	-	-	-	-	-	100
50000	-	-	-	-	-	-	-	60
60000	-	-	-	-	-	-	-	21

Table 6.2

## Timers interrupt

## مقاطعة المؤقت

ليتم مقاطعة ال Timer والذهاب الى خدمة المقاطعة ISR يجب أن يدخل مسجل المؤقت timer0 في حالة طفحان overflow أي أن قيمة المسجل Timer0 تنتقل من القيمة 0xff الى القيمة 0x00 ، ويتم ذلك

١- اخبار المسيطر الدقيق باننا نريد تفعل خدمة المؤقت وذلك عن طريق السجل option عند القيمة TOCS حيث أن

TOCS=0 timer mode

TOCS =1 counter mode

٢- تحديد فترة المؤقت وهي بشكل افتراضي 16 micro second اذا كانت ps0=0,ps1=0,ps2=0، لكن أعلم ان prescaler يتشارك مع شيئين watchdog و Timer ويتم تحديد أي العملية المطلوبة عن طريق الطرف (Pin assignment) PSA حيث

PSA =0 Timer prescaler

PSA =1 Watchdog prescaler

## WATCHDOG TIMERS (WDT)

## مؤقت الحراسة

سؤال يطرح نفسه!!؟ ماذا يحدث لو علق Stuck المُسيطر الدقيق، هل هناك زر restart لاعادة تشغيل المُسيطر الدقيق، طبعاً لا، فماذا نفعل لو كان المُسيطر الدقيق في موقع عمل field ودخل في حالة العلق Stuck، السطور التالية سنتعرف بها على كيفية التغلب على هذه الحالة،، كيف يعلم المُسيطر الدقيق انه علق Stuck!!؟

نحن نعلم أن دور الماكينة تبدأ بعملية الجلب Fetch التي يتم فيها جلب التعليمات من الذاكرة الى سجل التعليمات Instruction register، حالة العلق Stuck هو ان وحدة المعالجة المركزية CPU لم تستلم تعليمات من الذاكرة مما يسبب خلل في سير البرنامج، وهو فعلاً ما يحدث في حاسباتنا الشخصية، فنظراً الى إعادة تشغيل الحاسوب،

لقد قلنا أن المُسيطر الدقيق لا يحتوي على زر إعادة التشغيل Restart، الفكرة التي ابتكروها المهندسون هو وضع مؤقت، يبدأ هذا المؤقت عند دورة الجلب بالعد Fetch cycle، أي عندما يقوم المعالج بارسال طلب التعليمات الى الذاكرة، اذا أنتهت قيمة المؤقت ولم يستلم المعالج اي تعليمات من الذاكرة الى مسجل التعليمات IR، يقوم المُسيطر الدقيق باعادة تشغيل نفسه ذاتياً، هذا المؤقت يسمى Watch Dog Timer ويختصر WDT،

في الحقيقة مؤقت الحراسة هو عبارة عن عداد ذو ثمان خانات 8-bit يعد من القيمة 0x00 الى القيمة 0xff، بشكل ذاتي، وعند نهاية كل عملية عد أي عند القيمة 0xff، يزداد مؤقت الحراسة ليصبح 0x00، مؤدياً ذلك لطفحان مؤقت الحراسة، ليبدء العد من جديد، متى يبدأ مؤقت الحراسة بالعد!!؟

يبدء مؤقت الحراسة بالعد خلال مرحلة الجلب fetch cycle من دورة الماكينة machine cycle، إذا لم يستلم CPU أي تعليمات خلال فترة الجلب Fetching وقد وصل مؤقت الحراسة بالعد الى القيمة 0xff، وقد طفح المؤقت أي تحول من القيمة 0xff الى القيمة 0x00، عندها سيتم عمل reset لشريحة المُسيطر الدقيق.. السؤال الذي يتبادر الى الذهن!!؟

✘ اذا طلبت العد من الصفر الى العشرة من شخصين طبعاً سرعة العد ستختلف فمثلاً الشخص الاول يعد كل ثانية، اما الشخص الثاني فمثلاً يعد كل ثانيتين، أذن ما هي الفترة التي يزداد فيها مؤقت الحراسة!!؟ بشكل افتراض المؤقت يعد كل ١٨ مايكرو ثانية، وهذه القيمة قابلة للتغيير باستخدام سجل option ..

✘ اخبار المُسيطر الدقيق بتفعيل enable او ابطال disable خدمة مؤقت الحراسة WDT

## watchdog timer preparation

## خطوات تهيئة مؤقت الحراسة

يُجب تحديد الفترة الزمنية التي يُعد فيها مؤقت الحراسة ، وذلك عن طريق المسجل option ، عن طريق الخانات ps0,ps1,ps2 حسب الجدول Table 6.3

Ps0	Ps1	Ps2	Rate	WDT Time
0	0	0	1:1	18 ms
0	0	1	1:2	38 ms
0	1	0	1:3	72 ms
0	0	1	1:4	144 ms
1	0	0	1:5	288 ms
1	1	1	1:6	576 ms
1	1	0	1:7	1.1 second
1	1	1	1:8	2.2 second

Table 6.3

تذكر !!! أن الخانات PS0,PS1,PS2 مشتركة بين المؤقت Timer0 ومؤقت الحراسة WDT ، لذلك يجب تحديد الوظيفة التابعة لهذه الخانات ، وذلك عن طريق الخانة PSA من مسجل Option ، حيث اذا رفعة هذه الخانة PSA الى القيمة واحد فهذا يعني أن قيمة التقسيم PS0,PS1,PS2 هي لمؤقت الحراسة WDT ، بمجرد ما ترفع قيمة الخانة PSA الى الواحد يتم تفعيل مؤقت الحراسة، حسب الشفرة التالية

```
Bsf 0x03,5
```

```
Bsf 0x81,3
```

```
BCF 0x83,5
```

يمكنك أن تصفر مؤقت الحراسة وكذلك قيم PS0,PS1,PS2 عن طريق التعليمة،

```
CLRWDWT ;Clear WatchDog Timer
```

إذا اردت أن تغير قيم prescaler ، الشفرة التالية توضح ذلك

```
movlw    0x0D ;This is b'0000 1101' in Hex
movwf    81h   ;This is the Option Register
```

عند النظر للشكل Figure 6.1 تجد ان هناك مصدرين للتردد Clock generator المصدر الذي يجهز المؤقت CLKOUT والذي يساوي تردد المذبذب البلوري مقسوم على اربعة. المصدر الذي يجهز مؤقت الحراسة WDT وهو مذبذب مستقل.

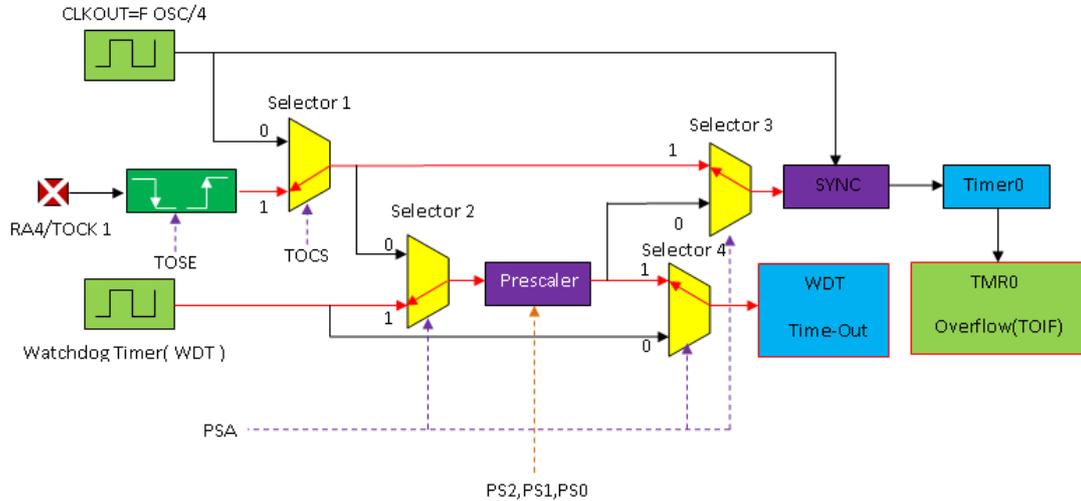


Figure 6.1

عندما ترفع قيمة PSA الى الواحد ، هذا يؤدي الى أن كل من selector 2 ، selector 4 التان هما عبارة عن منتقي بيانات multiplexer ، سيشيران الى البيانات القادمة من الطرف "1" من multiplexer ، وهذا يؤدي الى ان المذبذب WDT سيعبر عن طريق selector 2 ذاهباً الى المقسم Prescaler ، الذي يقسم التردد بالاعتماد على قيمة كل من PS0,PS1,PS2 ، ليخرج التردد المقسم الى selector 4 ، الذي يفتح الطريق الى مؤقت الحراسة WDT(Time-Out) ليقوم بعملية العد.



## Toolkit & Simulation

## العدد والمحاكاة

مقدمة:

في هذه الوحدة سوف نتعرف على الادوات Toolkit المستخدمة في عملية البرمجة ، وكذلك على عملية المحاكاة للمسيطر الدقيق بأستخدام برامج الحاسوب Computer Software وذلك للكشف عن العلل Bugs وتصحيحها قبل تحميل Download البرنامج الى ذاكرة المسيطر الدقيق

## عملية البرمجة

ليتم برمجة المسيطر الدقيق Microcontroller يجب أن يربط الى جهاز حاسوب (PC) Personal Computer وبم ذلك عن طريق مبرمجة خاصة يتم توصيلها الى منافذ الحاسوب سواء كان هذا المنفذ Eathernet او منفذ Rs232 أو منفذ USP وذلك بالاعتماد على نوع المبرمجة ، كما في الشكل ادناه



## المبرمجة :

تتوفر في السوق أنواع مختلفة من المبرمجات ، من هذه المبرمجات عبارة عن عدة متكاملة مثل المبرمجة من شركة Microchip هذه المبرمجة مزودة بقواعد Socket لمختلف انواع PIC microcontrolrs ، وكذلك تم توصيل مفاتيح ومصباح للفحص ، وجهزت بوحدات 7 segment, lcd character 2\*16 , graphic lcd ، وغيرها من الوحدات مثل الذاكرة وحدات دخل تماثلية والكثير ما عليك سوا شرائها من السوق وقراءة الدليل المرفق مع المبرمجة لتعرف اكثر عن مزاياها .

## عملية توصيل المبرمجة easypic7 بجهاز الحاسوب

لكي يتم توصيل المبرمجة easypic7 بجهاز حاسوب تحتاج الى

- ☒ كيبيل توصيل من نوع Usb شبيهة بكيبيل الطابعة الكيبيل مرفق مع المبرمجة، لاحظ الشكل Figure 7.1
- ☒ تحتاج الى سواقة Driver لتعريف المبرمجة الى نظام الحاسوب ، هذه التعريف مرفق مع المبرمجة اسم التعريف هو mikroProg For PIC Drivers v2.00 فقط حدد نوع النظام Windows Version

- ☒ تحتاج الى برنامج mikroProg يقوم هذا البرنامج بعملية نقل الملف hex. File الذي تم توليدته باستخدام بيئة البرمجة وذلك ليتم نقله الى ذاكرة المسيطر الدقيق microcontroller

بعد أن تقوم بتوصيل المبرمجة الى جهاز الحاسوب عن طريق كيبيل Usb ، وقمت بتنصيب التعاريف Drivers وبرنامج Mikroprog ، للتأكد من أن المبرمجة تم توصيلها بشكل صحيح الى جهاز الحاسوب ، قم بتشغيل برنامج Mikroprog واجهة البرنامج مبينة في الشكل Figure 7.2 تجد في اسفل واجهة البرنامج أيقومة بشكل رمز منفذ USP عندما تكون باللون الرمادي فأنة المبرمجة تكون في حالة عدم توصيل أما اذا كانت باللون الاحمر فهذا يدل على ان المبرمجة موصلة بشكل صحيح لاحظ الشكل Figure 7.3



Figure 7.1



Figure 7.2

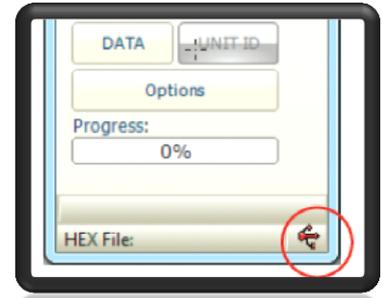


Figure 7.3

## كتابة الشفرة Code وتوليد الملف hex.

لكي تولد ملف hex. الذي سيتم خزنة في ذاكرة المسيطر الدقيق ، تحتاج الى مترجم يقوم هذا المترجم بترجمة السلاسل النصية Strings الى لغة تفهمها الانظمة الرقمية بشكل ملف يسمى hex. File ، توفر شركة microchip مترجم يسمى mplab هذا المترجم يستقبل سلاسل نصية مكتوبة بصيغة لغة التجميع Assembly language ،

## خطوات توليد ملف hex. File

❖ قم بتنصيب برنامج Mplab ، الشكل Figure 7.4 يبين واجهة البرنامج

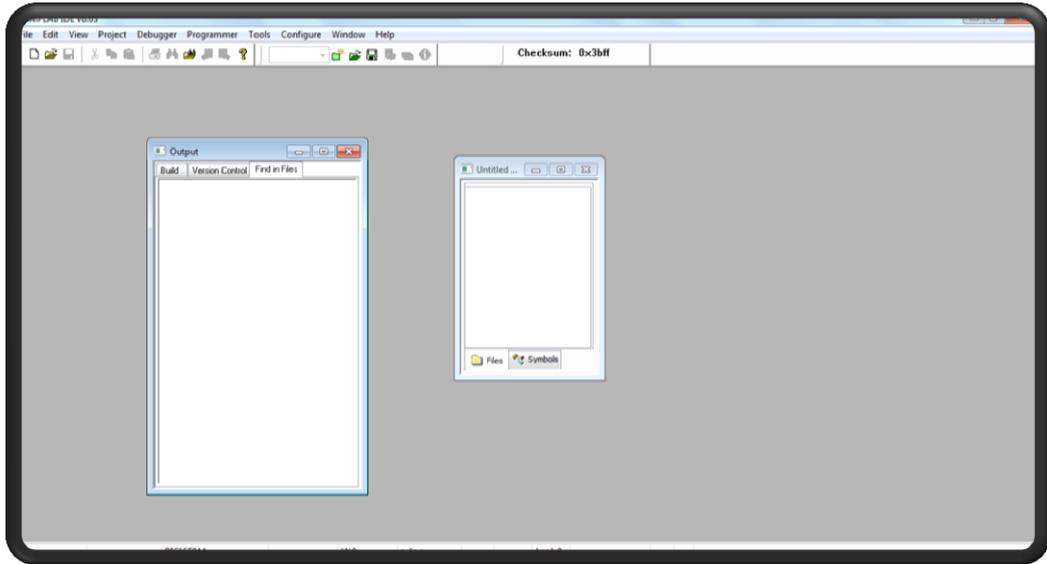


Figure 7.4

❖ من قائمة project wizard أختَر project wizard سنفتح النافذة المبية في الشكل Figure 7.5

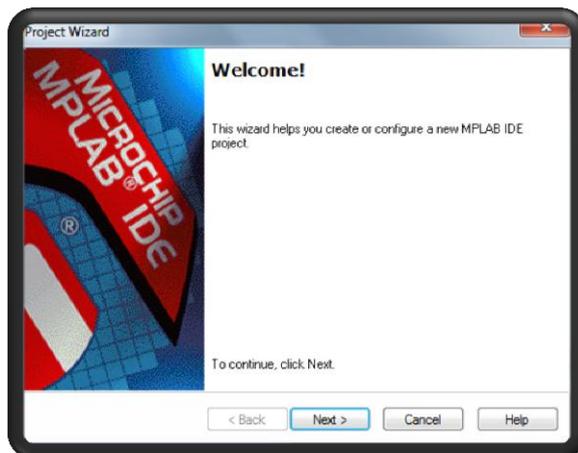


Figure 7.5

❖ من النافذة project wizard انقر على زر next  
❖ تظهر نافذة جديدة أختَر منها نوع المسيطر الدقيق Device type كما هو موضح في الشكل Figure 7.6

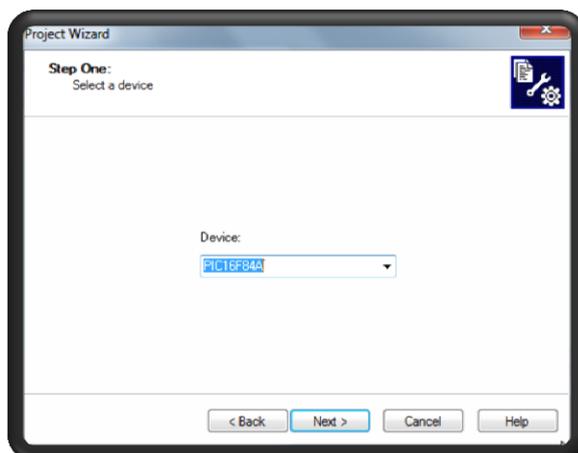


Figure 7.6

❖ عند النقر على زر next تظهر النافذة التالية Figure 7.7 والتي تحدد فيها مسار المترجم Assembler اترك الاعدادات الافتراضية لها ثم انقر على زر next

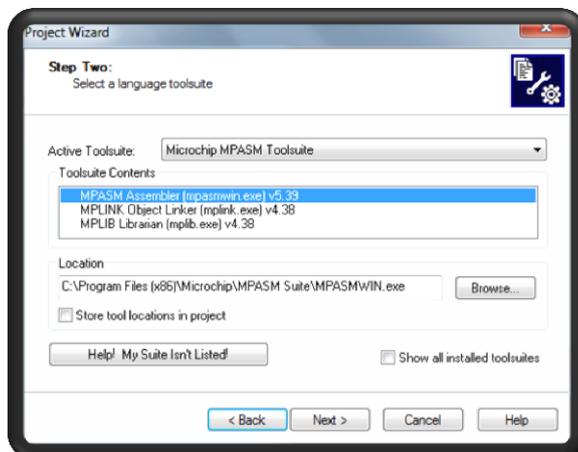


Figure 7.3

❖ بعد النقر على زر next تظهر النافذة التالية Figure 7.8 والتي تحدد من خلالها أسم والمسار الذي تريد حفظ ملف hex. فية ثم انقر على زر next ، كما هو موضح في الشكل

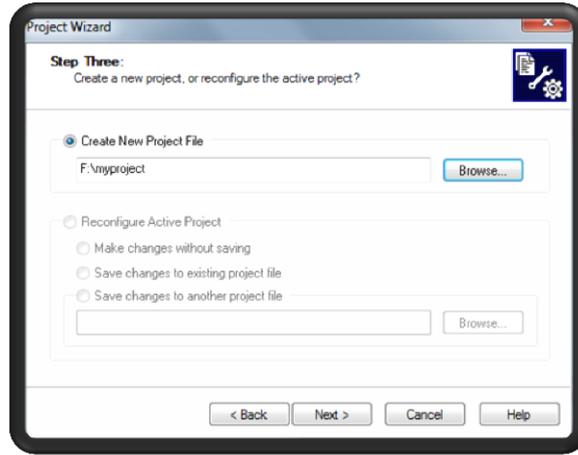


Figure 7.8

❖ تظهر لك النافذة الموضحة في الشكل Figure 7.9 والتي تطلب منك إضافة ملفات خارجية الى مشروعك، لا تقم باضافة اي ملف فقط اضغط زر next .

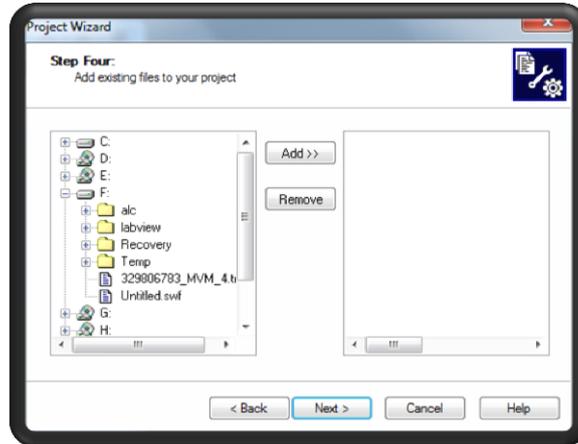


Figure 7.9

❖ تظهر لك النافذة التالية Figure 7.10 والتي تلخص الخطوات التي قمت بها ، فقط أضغط على زر finish

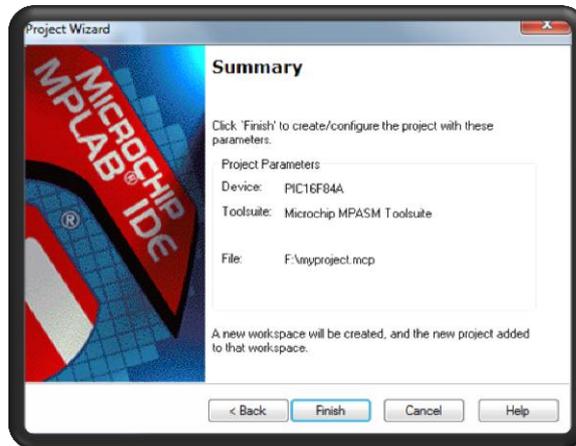


Figure 7.10

بعد أن تم تحديد نوع المسيطر الدقيق Microcontroller type وتم تحديد اسم ومسار الملف hex. ، يتم إضافة ملفات المشروع كما هو موضح في الشكل Figure 7.11

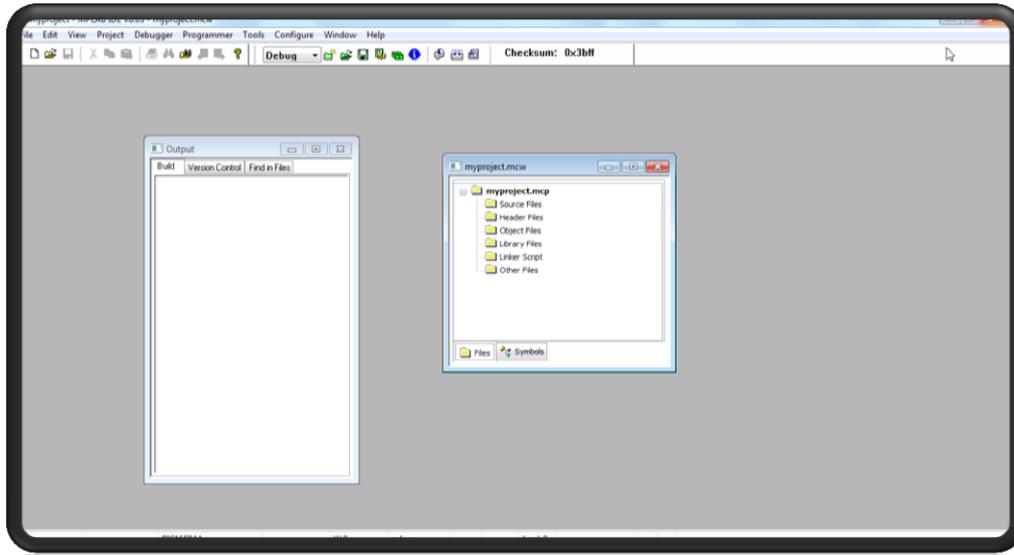


Figure 7.11

❖ من القائمة file أختار new ستنبثق نافذة لتحرير الاوامر Codes قم بكتابة البرنامج الذي تريد ترجمته كما هو موضح في الشكل Figure 10.12

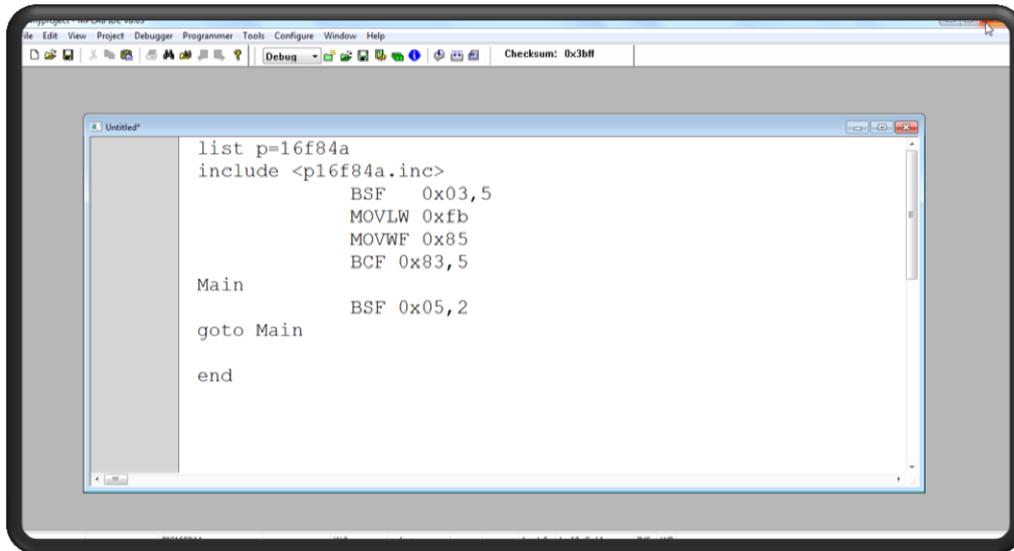


Figure 7.12

❖ من القائمة file أختَر Save وذلك لحفظ الملف في اي مكان تريد وليكن المسار F:\myproject\untitled.asm ستتغير النصوص الى  
 Figure 10.13 نصوص ملونة كما في الشكل

```

list p=16f84a
include <p16f84a.inc>
        BSF    0x03,5
        MOVLW 0xfb
        MOVWF  0x85
        BCF    0x83,5

Main
        BSF    0x05,2
goto Main
end
  
```

Figure 7.13

❖ من النافذة myproject.mcw المبنية في الشكل Figure 10.14 ، أضغط بالزر الايمن على الملف source files ستنبثق نافذة فرعية pop-up menuهنا أختَر منها Add Files... ستنتفخ نافذة مستكشف الملفات أختَر الملف الذي قمت بحفظه في الفقرة السابقة والذي هو untitled.asm

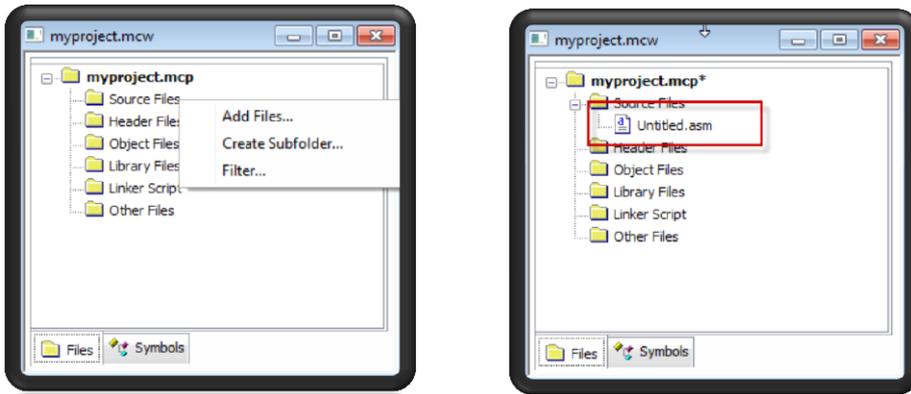
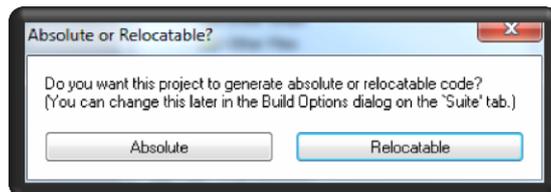


Figure 7.13

❖ اذهب الى القائمة project وأختَر الأمر build all ستنبثق نافذة تطلب منك تحديد نوع طريقة تخزين البرنامج ، اذا اخترت الامر Absolute فانك تولد ملف مطلق اي ثابت عند عنوان معين ولا يتغير موقع البرنامج داخل الذاكرة ، أما اذا أخترت الامر Relocatable فان موقع البرنامج سيكون نسبي وقابل للتغيير في داخل الذاكرة كما هو موضح في الشكل، أختَر الامر Absolute اي أننا نود أن يكون البرنامج محدد عند 0x00 من الذاكرة



❖ إذا كانت العمليات التي قمت بها صحيحة فستظهر لك النافذة التالية Figure 10.14 والتي تشير الى أن عملية توليد ملف hex تمت بطريقة صحيحة .

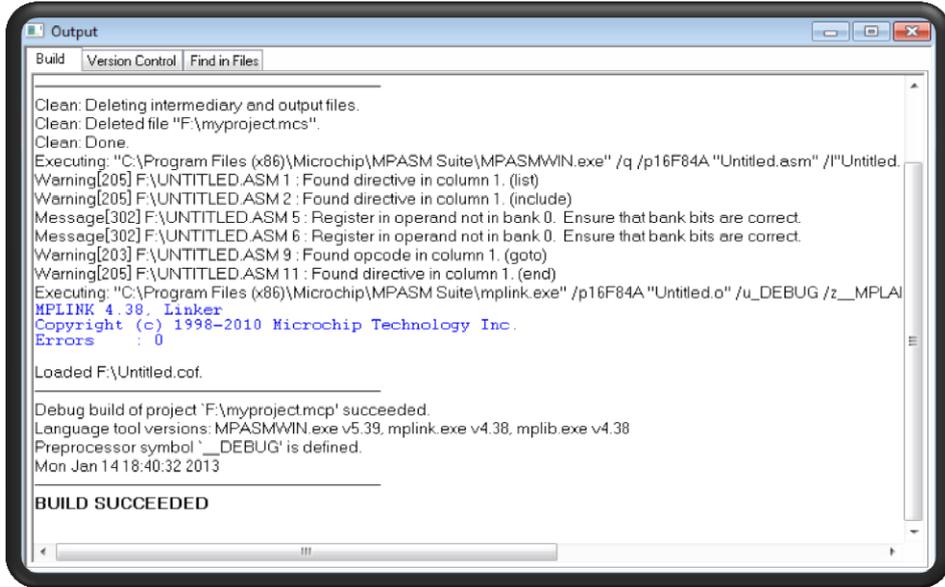


Figure 7.14

عملية نقل الملف hex الى ذاكرة المسيطر الدقيق microcontroller لكي تتم عملية نقل الملف hex المتولد من برنامج mplab ، نحتاج الى برنامج microprog عند تشغيل البرنامج تشاهد نافذة البرنامج

Figure 10.15 التالية

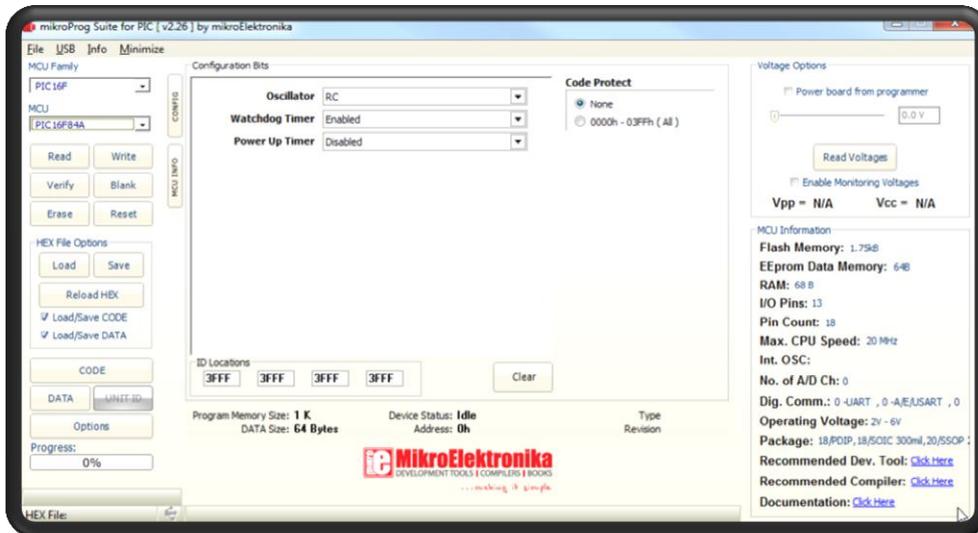


Figure 7.14

نلاحظ من الشكل أنه عن طريق القائمة list box المعنونة MCU Family نستطيع تحديد من خلالها عائلة المسيطر الدقيق مثلاً العائلة PIC16F ثم عن طريق القائمة list box المعنونة MCU نستطيع تحديد نوع المسيطر الدقيق وليكن مثلاً المسيطر PIC16F84A ، بد أن تمت عملية تحديد نوع المسيطر الدقيق microcontroller علينا تحميل الملف hex الى البرنامج وذلك عن طريق المجموعة Hex file option ثم النقر على الزر load لتحميل الملف Hex ، ما علينا الان سوى نقل البرنامج الى المسيطر الدقيق microcontroller عن طريق الزر Write .

## Simulation using proteus

## المحاكاة بأستخدام برنامج proteus

يوجد هناك العديد من برامج المحاكاة المستخدمة لمحاكاة الدارات الإلكترونية ولكن ما يميز برنامج Proteus هو الدقة والسهولة في محاكاة المشاريع وخصوصاً أنه يحتوي على مكاتب كثيرة لكافة القطع الإلكترونية وهو يستطيع أن يحاكي الدارات التي تحتوي على المتحكمات بسهولة جداً مما يتيح للمصمم بأن يختبر عمل المتحكم قبل أن يتم حقن البرنامج في المتحكم Microcontroller مما يسهل عملية التطوير في البرامج بسهولة وإضافة لذلك يستطيع المتدرب أو الذي يدرس برمجة المتحكمات أن يتم إختبار برامجه ويعدل عليها دون أن يتم تطبيقها .

### واجهة البرنامج 7 proteus

بعد أن قمت بتوليد الملف hex. File عن طريق برنامج mplab ، افتح برنامج 7 proteus واجهة البرنامج موضحة في الشكل Figure10.15

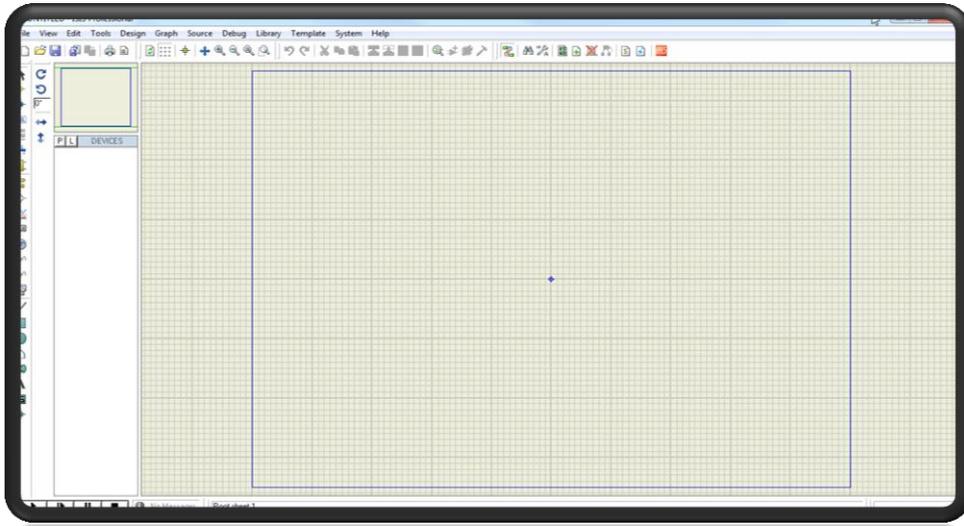


Figure 7.15

❖ بعد أن قمت بفتح البرنامج قم بفتح مكتبة العناصر Device كما هو موضح في الشكل Figure 10.16، تحتوي المكتبة على كافة العناصر الإلكترونية من المقاومات والمتسعات الى المعالجات الدقيقة microprocessor والحاكمات الدقيقة microcontroller

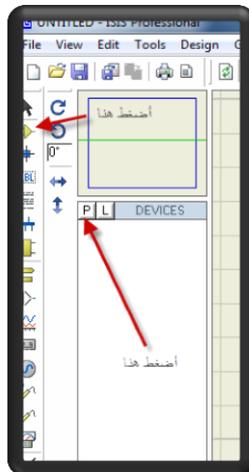


Figure 7.16

❖ سنتفتح نافذة المكتبة نافذة pick devices انتقي القائمة MICRO ثم انتقي المسيطر الصغري PIC16f84a بالنقر المزدوج كما هو

موضح في الشكل Figure 10.17

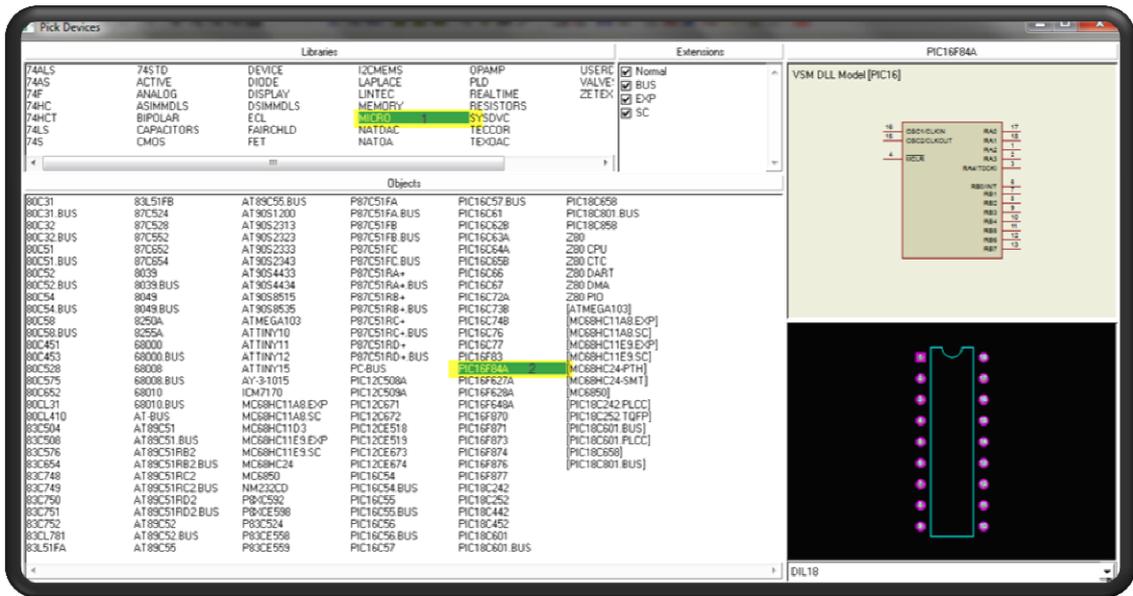


Figure 7.17

❖ بعد أن قمت بربط المكونات مع المسيطر الصغري كما هو موضح في الشكل Figure 10.18 أبدأ

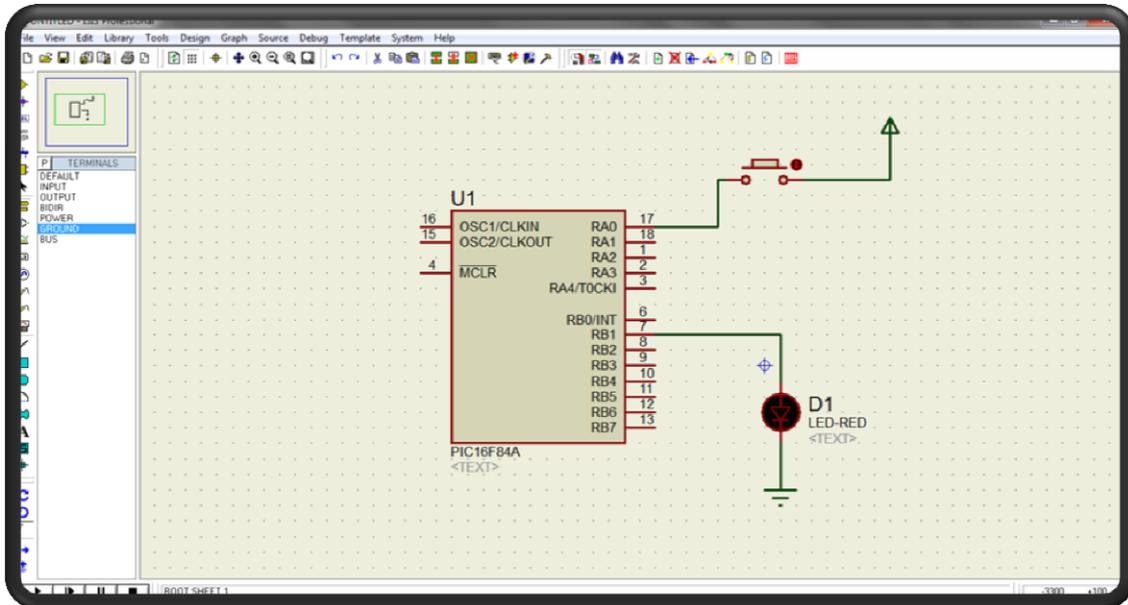


Figure 7.18

❖ انقر نقرأ مزدوجاً على المسيطر الصغري microcontroller سنتفتح نافذة Figure 10.19 تطلب من تحديد الملف Hex الذي

قمت بتوليدته من برنامج mplab

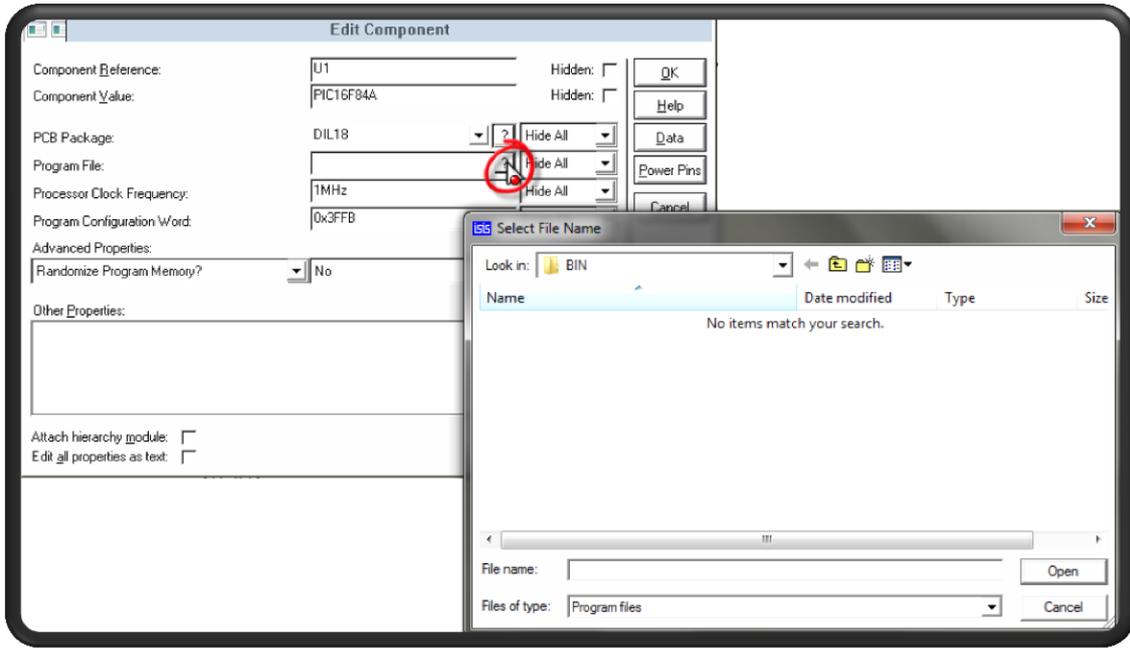


Figure 7.19

بعد أن قمت بعملية تحميل برنامج proteus بمسار الملف hex file قم بتشغيل المحاكاة Simulation وذلك من القائمة Debug ثم من القائمة أختار Start/Restart debugging

**1- PIC microcontrollers *for beginners,too!***

Author: Nebojsa Matic

**2- Microchip PIC16F84A Data Sheet**

**3- Some images are from Google search engine**

تم بحمد الله