The PIC MicroController

- PIC16f84A 8-bit enhanced with EEPROM

بَرمجة المتحكمات الدقيقة بلغة التجميع

Programming PIC Microcontroller with Assembly language

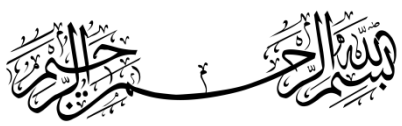
Include

- Introduction to Digital System

المهندس

بسام أحمد صالح

Email: CET.ENG2012@yahoo.com



﴿ وَاللَّهُ أَخْرَجَكُم مِّن بُطُونِ أُمَّهَاتِكُمْ لَا تَعْلَمُونَ شَيْئًا وَجَعَلَ لَكُمُ السَّمْعَ وَالْأَبْصَارَ وَالْأَفْئِدَةَ لَعَلَّكُمْ تَشْكُرُونَ ﴾

الحمـــــدُ لله رب العالمين ، وأشهـــــــد أن لا إله إلا الله وحده لا شَريك له ، وأشهد أن محـــــــــمداً عَبـــده ورسوله وصَفيه وخَليله ، صلوات ربي وســلامه عليه وعلى آله وأصـــــــــحابه أجمعين ، والتـــــــــابعين ومن تبعهم بإحسان إلى يوم الدين . .

أما بـــــعد :

نعيشُ اليوم في عَصر تَكنلوجيا المَعلومات ، حيثُ يشهد عَصرنا هَذا تطوراً سريعاً لاسِيما فيما يخصُ مَجال تقنيات معَـــــالجة البيانات وكثرة الأجهزة التي تم اختراعُها في هذا المَـــجال مثل Microprocessor , PLC , Microcontroller ، فالقرن العشرين هو بلا شكٍّ قرن الإلكترون والصناعات الحاسوبية وبخاصة في العقود الأربعة الأخيرة , فاختراع الترانزيستور شكّل قفزةٍ نوعيةٍ في الصناعة الالكترونية , وغدا هذا العنصر فيما بعد عنصراً أساسيّاً في صناعة الدوائر الالكترونية. نقطة التحول الثانية في الصناعة الالكترونية بعد الترانزيستور تمثلت بنجاح العلماء في صناعة دوائر متكاملة (IC) متناهية الصغر ، وخصوصاً الاختراع المتحكم الصُغري 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 الصُغري يختلف عن بقية الدارات المتكاملة في أنه دارة 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 الالكترونية.

هذا الكتاب سيكون مدخلك نحو الفهم العميق لمبدء عمل المتحكم الصُغري وطريقة برمجة ، حيثُ يتكون هذا الكتاب من سبعة وحدات ، في الوحدة الاولى تناولنا موضوع الأنظمة العَددية وكيفيها تمثيلها ، اما الوحدة الثانية فكانت مقدمة عن بنية الانظمة القابلة للبَرمجة ، وفي الوحدة الثالثة تناولنا البنية الصلبة Hardware للمتحكم الصُغري Microcontroller ، اما الوحدة الرابعة تناولت موضوع مسجل البيـانات File Register ، وفي الوحدة الخامسة تم شرح طقم التعليمات instruction set الخاصة بالمتحكم الصُغريMicrocontroller ، وفي الوحدة السادسة سوف تتعلم مبادى البرمجة وطرق برمجة المتحكم الصُغري Microcontroller ، اما الوحدة الاخيرة فخصصتها عن الادوات Toolkit والبرامج Softwares المستخدمة في عملية البرمجة .

المهندس

بسام أحمد صالح

الفهرس

الوحدة الاولى — الانظمة العددية Numbrics Systems

الوحدة الثانية بنية الانظمة القابلة للبرمجة Digital System Archetecture

---

الوحدة الثالثة بنية المتحكم الصغري MCU Archetecture

الوحدة الرابعة File Register

الوحدة الخامسة طقم التعليمات Instructions set

الوحدة السادسة مبادء البرمجة Programming Concept

الوحدة السابعة — العدد والمحاكاة — Toolkit & Simulation

اللهم أكتب هذا العمل خالصاً لي ولوالدي

أدعوا لي ولوالدي بالرحمة

Dedication

الى كل مُسلم يُريد عِزة الأسلام ونبينا محمد (علية الصلاة والسلام)

الى كل شخص يُريد رفع رأس الوطن وكرامته العراق الحبيب

الى أعز شخصين الذين اوصلاني الى ما انا عليه الوالد والوالدة

الى الشخص الذي كان السبب في أكمال هذا الكتاب زوجتي العزيزة

الى أخوتي واصدقائي ....

27/11/2013

Numerical System

الأنظمة العَددية

مقدمــة:

هذه الوحدة هي مُقدمة للنظام العُشري Decimal ، الثُنائي Binary، السادس عَشر Hex. Decimal، كَيفية تَمثيلها وكيفية التَحويل بين هذه الانَظمة، في الحَقيقة أن الغاية الأساسيه لتقديم هذه الوحدة هي التَعرف على بعَض المَفاهيم الأساسية مثل كيفية تمثيل الأعداد الموجَبة والسالبة في النِظام الثُنائي، والتعرف على حالة الطفحان Overflow ، حالة المُحَمل Carry ، وغيرها من المفاهيم الواجب التعرف عليها قبل الدخول في عالم المُسيطر الدقيق Microcontroller..

Decimal System

النِظام العُشري

إن النِظام ألاكثر استخداماً في حَياتنا اليَومية في عَمليات العَد و الحِساب هو النِظام العشري Decimal system ، هذا النِظام أساسه Radix هو 10 ، الأساس عَشرة يعني أن هذا النظام مكون من عَشر أعداد Digit لتَمثيل القِيم Values هي 0,1,2,3,4,5,6,7,8,9 تسمى هذه الأرقام مُعامِلات Coefficient النِظام العشري، فمثلاً العَدد 123 يمكن تَفسيرة بِشكل،

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

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

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

حيثُ أنَ ،

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

ألنِظام الثُنائي Binary system

وهو النِظام الذي يُستخدم في الأجهزة الرَقمية Digital Devices لتَمثيل الأعداد Numbers و تمثيل الأِشارات Signals ، هَذا النِظام أساسة Radix هو "2" ، الأساس أثنان يعني أن هذا النِظام مكون من رَقمين Two Digit لتمثيل الأعداد هما "1" , "0" يُسمى هذان الرَقمان مُعاملات Coefficient النِظام الثنائي، فمثلاً العدد "100"b تمثل العدد "4" بألنظام العُشري ، والعدد "1101"b تمثل العَدد "13" بالنظام العشري وهَكذا ، أي أن هذا النِظام مكون من سلسلة من الأصفار والوحدات فقط ..

ملاحظة:

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

الشكل Figure 1.1 يوَضح عَدد بألنِظام الثُنائي مُكون من ثمانِ مَراتب 8-bit ،

Figure 1.1

التَحويل بين الأنظمة System Conversion

أذا كُنت تود برمجة مُسَيطر دَقيق Microcontroller ، يَجب أن تَعرف كيفية التعامل مع النِظام الثنائي Binary والنِظام العشري Decimal وكيفية التَحويل بين هذه الأنظمة ، لأنك ستحتاج عملية التحويل بكثرة اثناء عملية البرمجة programming .

❖ التَحويل من النِظام الثُنائي إلى النِظام العُشري Binary to Decimal Conversio

للتَحويل من النِظام الثُنائي الى النِظام العشري نَضرب كل bit في أساس النِظام Radix الذي هو '2' مَرفوعة للأُس رَقم المَرتبة أي ،

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

ثم نَجمع كل b_n لأستخراج الناتج أي $b_2+b_1+b_0$ وهكذا ، مع مُلاحظة أن مَرتبة الأحاد LSB تبدء بالقيمة صفر أي أن position := 0

- مثال 1.1

حول العدد $(1001)^2$ بالنظام الثنائي الى ما يكافئة بالنظام العشري ؟

الحل:

$1X2^3 + 0X2^2 + 0X2^1 + 1X2^0 = 8+0+0+1 = (9)^{10}$

- مثال 1.2

حول العدد $(0110)^2$ بالنظام الثنائي الى ما يكافئة بالنظام العشري ؟

الحل:

$0X2^3 + 1X2^2 + 1X2^1 + 0X2^0 = 0+4+2+0 = (6)^{10}$

❖ التَحويل من النِظام العُشري إلى النِظام الثُنائي Decimal to Binary Conversion

عند التَحويل من النِظام العُشري الى النِظام الثُنائي نَتبع الصيغة التالية ، نقسم العَدد الذي نريد تَحويلة الى النظام الثنائي على أثنان" 2" ،مثلاً العدد "10" ،

10 ÷ 2 = quotient is 5 reminder = 0 LSB

ناتج قسمة quotient العدد 2÷10 هو "5" ، والمتبقي هو "0" ، الذي يعتبر هو الخانة الأقل أهمية LSB ، ثم نقسم العدد 2÷5 ونستخرج باقي القسمة وهكذا ... وتستمر العملية الى أن يصبح العدد المقسوم أصغر من المقسوم علية لنتوقف ،

5 ÷ 2 = quotient is 2 reminder = 1 bit.1

2 ÷ 2 = quotient is 1 reminder = 0 bit.2

1 ÷ 2 = invalid operation reminder = 1 MSB

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

- مثال 1.3

حول العدد $(14)^2$ بالنظام العشري الى ما يكافئة بالنظام الثُنائي ؟

الحل:

14 ÷ 2 =	quotient is 7	reminder	=	0	LSB
7 ÷ 2 =	quotient is 3	reminder	=	1	bit.1
3 ÷ 2 =	quotient is 1	reminder	=	1	bit.2
1 ÷ 2 =	invalid operation	reminder	=	1	MSB

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

- مثال 1.4

حول العدد $(15)^2$ بالنظام العشري الى ما يكافئة بالنظام الثُنائي ؟

الحل:

15 ÷ 2 =	quotient is 7	reminder	=	1	LSB
7 ÷ 2 =	quotient is 3	reminder	=	1	bit.1
3 ÷ 2 =	quotient is 1	reminder	=	1	bit.2
1 ÷ 2 =	invalid operation	reminder	=	1	MSB

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

الجدول table 1.1 يوَضح القيِم العُشرية Decimal وما يقابلها بالنظام الثنائي Binary System

العَدد العشري	العَدد 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

وحدات القياس Units of Measurement

كثير ما نسمع مصطلحات Kilo byte, Mega byte , Giga byte التي نُشير بها الى سِعات خَزنية ، مثلاً نقول أن لديَ قرص صَلب Hard disk ذو سعة 120 Giga byte ، نَحن بِذلك نَذكر وحدة القياس الخاصة بالأنظمة الرَقمية ، حيثُ أنَ أصغر وحدة قياس في النِظام الثنائي تسمى bit والتي تَحمل قيمتان أما صفر أو واحد ، مجموع ثَمان خانات 8-bit تسمى byte ، أي أن :

1 byte = 8-bit

2 byte = 2 x 8-bit = 16-bit

الشكل Figure 1.2 يمثل الـ byte ،

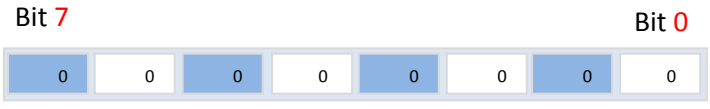


Figure 1.2

أقصى عدد يُمكن أن يَستوعبة البايت الواحد 1 byte هو $(1111\ 1111)^2$ أي ما يكافئها بالِنظام العُشري $(255)^{10}$ ، هناك عِلاقة تستطع من خِلالها مَعرفة أقصى عدد يمكن أن تَستوعبة 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 تَستطيع من خِلالة معَرفة عدد البتات bit numbers اذا عُلم أعظم رَقم Maximum Value وذلك بأستخدام اللوغارتمات حيثُ،

$$\text{Number of bit(N)} = \text{Log}_2 X$$ 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 عَدد يُمكن تَحمله هو 65535

- مثال 1.6

أذا كان لَديك مستوعب Register أعظم رقم يسَتوعبة هو العَدد $(32)^{10}$ فما هو عدد البتات Bit Number الذي يتكون منة المستوعب ؟

الحل:

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

أذن عدد البتات هي 5-bit.

الجَدول Table 1.2 يُبين وحدات القياس ،

الوحدة	عَدد 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

الأعداد الموجَبة والسالبة Singed and Unsigned Value

قلنا أن النظام الثنائي مكون من سِلسلة من الأصفار والواحدات Ones and Zeros ، و أن هذه السلسلة من الأصفار والواحدات عِبارة عن أشارات كهربائية Electrical Signals ، السؤال الذي يتبادر الى الذهن كيف يتم تمثل الاعداد السالبة في النظام الثنائي ؟!!

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

- ❖ الأعداد بدون الأشارة Unsigned value
- ❖ الاعداد بأشارة Signed value

❖ الاعداد بدون الأشارة Unsigned Value

وهي الأعداد التي لا تحتوي على أشارة 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 .

❖ الأعداد بأشارة Signed value

وهي الأعداد التي تحتوي على أشارة سالبة مثل العدد 22- ، 10- ، بما أن الأجهزة الرَقمية تتعامل مع الأشارات الكهربائية المكونة من سِلسلة من الأصفار والواحدات ، كيف سيتم تمثيل الرمز السالب (-) ؟!!

هناك ثلاث طرق لتمثيل الأعداد السالبه والموجبة ،

١- طريقة الأشارة والمُقدار Sign and Magnitude method
٢- طَّريقة المُتمم الاول 1's Compliment method
٣- طَّريقة المُتمم الثاني 2's Compliment method

الأشارة والمقدار Signe and Magnitude

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

Sign (+,-)	Magnitude (value)

ملاحظة:

عيوب Disadvantage أستخدام الأعداد السالبة Signed value هو خَسارة المدى Range ، وذلك بتخصيصنا خانة للأشارة حيث،

Range of 8-bit $2^{8-1} = 2^7 = 128$

الجَدول Table 1.3 يوضـــــح عَــدد مكون من 3-bit ممـــثل بصيغة عدد غير بـدون إشارة Unsigned وصـيـغة عدد بأشارة Signe-Magnitude

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

Table 1.3

لاحظ انة في حالة استخدام عدد بأشارة Signed value لعدد مكون من 3-bit سنجد انة 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 أشارتة سالبة فالصفر اشارتة موجـــبة 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:

المدى Range لعدد مكون من 8-bit بدون إشارة هو $2^8 = 256$

بعد تخصيص خانة للاشارة يصبح المدى $2^{8-1} = 2^7 = 128$

أي أن مدى الاعداد الموجبة التي يكون فيها MSB هو 0 هو 0 to 127

ومدى الاعداد السالبة التي يكون فيها MSB هو 1 هو 128 to 256

ملاحظة :

في الحَقيقة الحاسبات الألكترونية والأجهزة الرَقمية Digital Device لا تَستَخدم أسلوب الأشارة والْمقدار Sign-Magnitude لتَمثيل الأعداد السالبة وذلك بسبب؟

- صعوبة معالجة Processing الأعداد السالبة بأستخدام طريقة الاشارة والمُقدار.
- تَكرر حالة الصفر ، أو وجود عدد صفري سالب (0-).

المتمم ألاول 1's Compliment

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 $(110)^2$ ليصبع 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،

العَدد الثَنائي	عَدد بدون إشارة	الأشارة- والمقدار	المُتمم الأول
000	0	0	111 = 7 = -0 قيمة خطأ
001	1	1	110 = 6 = -1
010	2	2	101 = 5 = -2
011	3	3	100 = 4 = -3
100	4	-0 قيمة خطأ	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 أستخدام طريقة المتمم الاول 1's Compliment.

- ❖ وجود حالة السالب الصفري (0-).

أيجابيات Advantage أستخدام طريقة المتمم الاول 1's Compliment.

- ❖ سهولة معالجة الارقام السالبة .

- مثال 1.8

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

الحل:

- ❖ نَقلب كل خانة من العدد $(0010\ 0011)^2$ ليكون لَدينا $(1101\ 1100)^2$
- ❖ نحول العَدد $(1101\ 1100)^{10}$ الىَ النِظام العَشري $(220)^{10}$ 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:

- ❖ نقلب كل خانة من العدد ليكون $(0010)^2$ لدينا $(1101)^2$
- ❖ نحول العدد $(1101)^{10}$ الى النظام العشري $(13)^{10}$ أي أن سالب العدد $(2)^{10}$ هو $(220)^{10}$

المُتَمم ألثاني 2's Compliment

وهي الطَريقة الاكثر استخداماً في مُعالجة وتَمثيل الأعداد السالبة في الحاسبات والاجهزة الرقمية ،وذلك كونها تتغلب على حالة الصفر المُكرر وكذلك عدم وجود عدد صفري سالب، مثلاً لأيجاد سالب العدد $(3)^{10}$ بطريقة المتمم الثاني نتبع الخَطوات التالية،

- ❖ نحول العدد $(3)^{10}$ 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 العَدد $(3)^{10}$ بالنظام الثنائي هو $(1111\ 1101)^2$ أي ما يكافئة بالنظام العشري $(253)^{10}$.

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

عَدد ثنائي	عَدد بدون إشارة	إشارة ـ ومقدار	المُتمم الاول	المُتمم الثاني
000	0	0	111 = 7 = -0 قيمة خطأ	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	-0 قيمة خطأ	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 أستخدام طريقة المتمم الثاني 2's Compliment.

- ❖ سهولة معالجة Processing الأرقام السالبة ،
- ❖لا وجود لحالة الصفر السالب (-0)

- مثال 1.7

جد سالب العدد $(11)^{10}$ بأستخدام المتمم الثاني ؟

الحل:

- ❖ نحول العدد $(11)^{10}$ الى النظام الثنائي $(1011)^{2}$
- ❖ نقلب كل خانة bit ليكون لدينا $(0100)^{2}$
- ❖ نظيف واحد الى الناتج ليكون $(0101)^{2}$

أذن سالب العَدد $(11)^{10}$ بالنظام الثنائي هو $(0101)^{2}$ أي ما يكافئة بالنظام العشري $(5)^{10}$

- مثال 1.8

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

العَمليات الحِسابية على النِظام الثُنائي Arithmetic Operations

تَجري على النِظام الثنائي العَمليات الحِسابية نفسها التي تَجري على النِظام العُشري ، مثل عَملية الجَمع Adding ، الطَرح Subtracting ، وغيرها من العَمليات بالأِضافة الى عَمليات أخرى تُدعى العمليات المنطقية Logical Operation ، مثل عملية AND,OR,NOT وغيرها....

عَملية الجَمع Addition Arithmetic Operation

أذا كان لدينا عَددان كل عَدد مكون من 4-bit ، لنرمز للعدد الأول A أي ان هناك A_0 A_1 A_2 A_3 ، ونرمز للعدد الثاني B أي أن هناك B_0 B_1 B_2 B_3، سَتجري عملية الجمع بين العددين A + B كما تَجري عَملية الجَمع في النِظام العُشري كما هو موضح في الشكل Figure 1.2

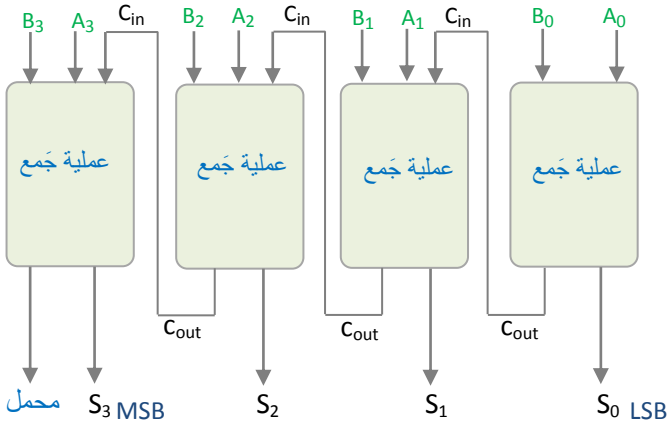


Figure 1.2

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

C_{in}	C_{in}	C_{in}	C_{in}		
	A_3	A_2	A_1	A_0	
	B_3	B_2	B_1	B_0	+
Carry	S_3	S_2	S_1	S_0	

لاحظ الجَـــدول Table 1.6 الذي يبُين حـــالات الأدخال C_{in} , B_n, A_n وحالات الأخـــــراج، التي هي ناتج عملية الَجمع Sum والمُحمل الخارج C_{out} ،

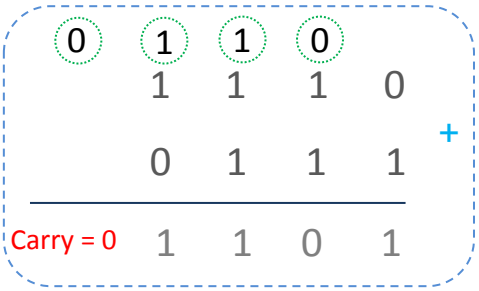
A_n	B_n	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 1.6

- مثال 1.6

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

الحل:

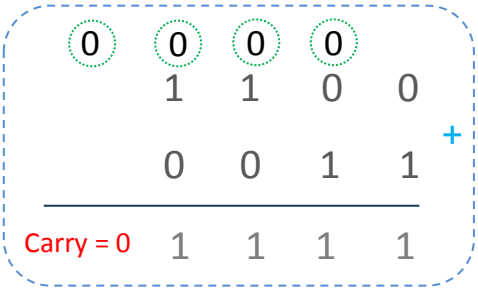


أذن ناتج جمع العدد $(6)^{10}$ مع العدد $(7)^{10}$ هو العدد $(13)^{10}$ الذي يكافئة بالنظام الثنائي $(01101)^2$

- مثال 1.7

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

الحل:



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

الطرح بأستخدام المتمم الثاني — Subtracting Using 2's Compliment

اذا أشتريت حاسبة علمية Scientific Calculator أو أستخدمت الحاسبة العلمية المرفقة مع نظام التشغيل Windows ، لتقوم بعملية طرح عَددين ، وذلك كي توفر الوقت والعناء بجعل الاله Machine تقوم بعملية الطرح عوضاً عنك، الية الطرح في هذه الأجهزة تتم باستخدام المُتمم الثاني كما سنلاحظة في السطور التالية ، مثلاً أذا أردت طرح العدد $(13)^{10}$ من العدد $(5)^{10}$ ، باستخدام الحاسبة العلمية ، كما قلنا سابقاً ان النظام الذي تتعامل معة اللألات الرقمية Digital Devices مثل الحاسبات العلمية والحاسبات الالكترونية personal computer والمسيطرات الدقيقة Microcontroller ، هو النظام الثنائي binary System ، وذلك بوجود دوائر خاصة تقوم بعملية الطـرح ، لنرى كيف تتم عملية الطرح نقوم أولاً بتحول العدد $(5)^{10}$ الى النـظام الثنائي سنـجد أن مكافـئ العـدد $(5)^{10}$ هو $(0101)^2$، ومكافـئ الـعدد $(13)^{10}$ هو $(1101)^2$ ، أي أن العمـلية التي نريد أن نجـريها هي (5 - 13)، أو (0101-1101)، الطريقة المستخدمة في عمليات الطرح هو بأيجاد المتمم الثاني 2's compliment للعدد المطروح ، بما أننا سنجد المتمم الثاني للعدد $(5)^{10}$ أو $(0101)^2$ والذي نقصد فية اننا نجد سالب العدد $(-5)^{10}$ أو $(1011)^2$ لتصبح العملية ((-5) + 13) ، أو (1101+1011)، أنظر كيف أن عملية الطرح ألآن أصبحت عملية جمع ، أي أن الدائرة التي تقوم بعملية الطرح هي نفسها التي تقوم بعملية جمع وهذة هي الغاية من أستخدام المتمم الثاني ، الشكل Figure 1.3 يوضح مخطط صندوقي Block Diagram لدائرة جمع و طرح باستخدام المتمم الثان

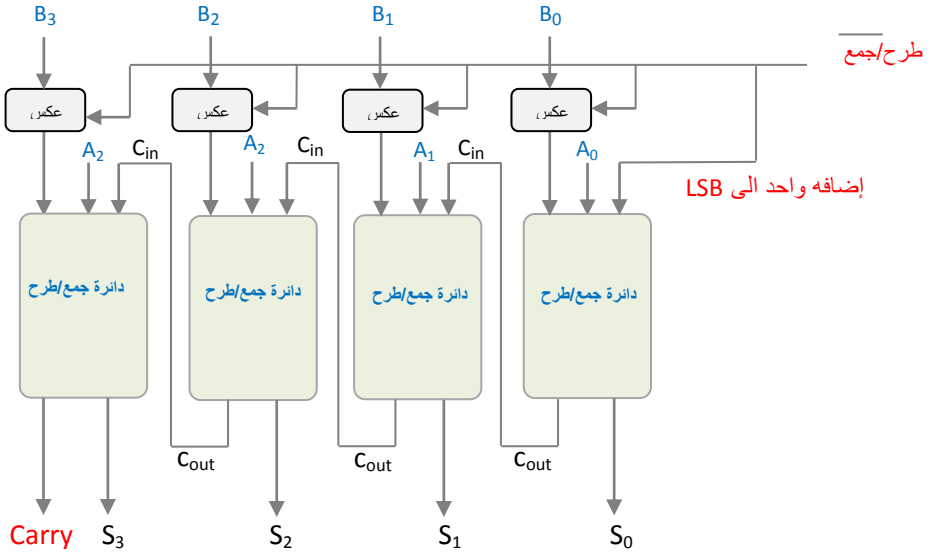
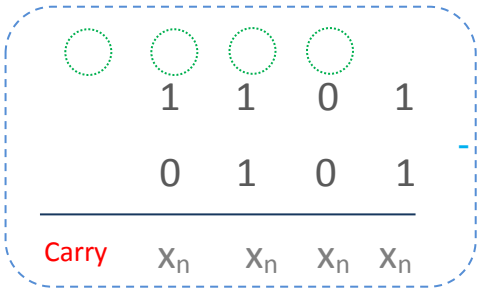


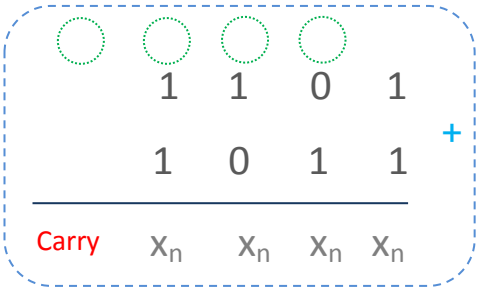
Figure 1.3

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

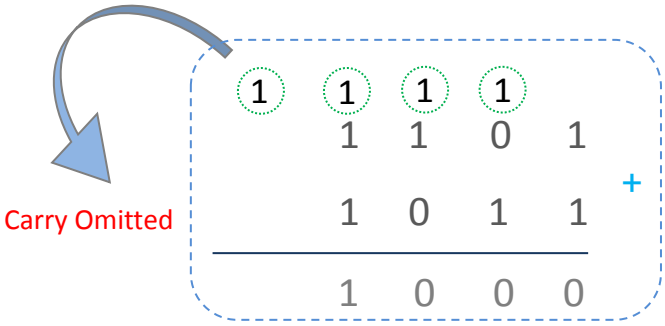


خطوات ايجاد المتمم الثاني للعدد المطروح للتهيئة لأجراء عملية الطرح :

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



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



ملاحظة :

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

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

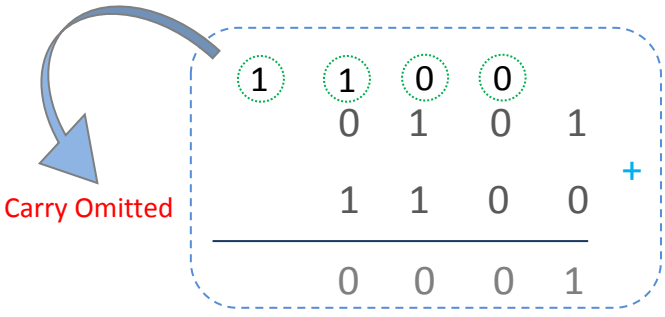
- مثال 1.9

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

الحل:

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

- ❖ نجد المتمم الثاني للعدد $(4)^{10}$ ليكون لدينا $(1100)^2$
- ❖ نجمع متمم العدد $(4)^{10}$ مع العدد $(5)^{10}$



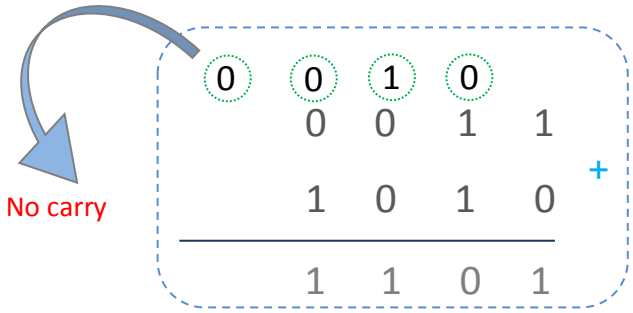
- مثال 1.10

أطرح العدد $(0110)^2$ من العدد $(0011)^2$ ؟

الحل:

العملية التي نريد ان نجريها هي (6-3) أي ما يقابلها بالنظام الثنائي $(0110)^2$ - $(0011)^2$

- ❖ نجد المتمم الثاني للعدد $(6)^{10}$ ليكون لدينا $(1010)^2$
- ❖ نجمع متمم العدد $(6)^{10}$ مع العدد $(3)^{10}$

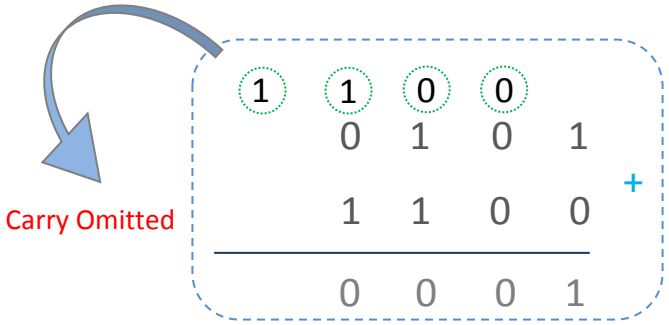


حالة المُحمل Carry State

المُحمل هو الخانة الثنائية Bit التي تَخرج من الخانة الأكثر أهمية MSB ، هذا المُحمل له فوائد كثيرة منها على سَبيل المِثال ، عند مُقارنة عَددين ونُريد معرفة أي العَدد هو الاكبر ، لنأخذ مثلاً العدد $(4)^{10}$ والعددة $(5)^{10}$ عند أجراء عملية الطرح التالية ،

Y = 5 - 4

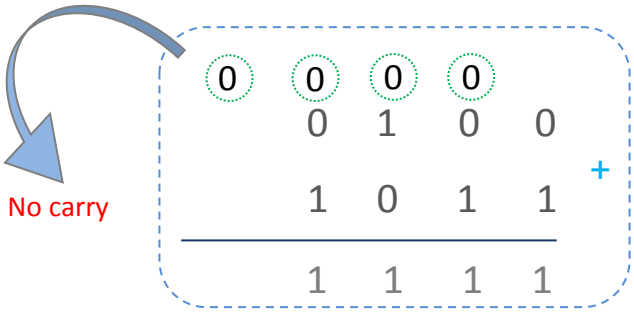
نقوم أولاً بأيجاد المكافئ الثنائي لكن من العدد $(4)^{10}$ الذي هو $(0100)^2$ والعدد $(5)^{10}$ الذي هو $(0101)^{10}$ ، كما تعلمنا سابقاً أن عملية الطرح تتم بأستخدام المتمم الثاني ، أي اننا سنجد المتمم الثاني للعدد $(4)^{10}$ الذي هو $(1100)^2$ لتصبع العملية كالاتي



لاحظ أن ناتج العملية هو $(1)^{10}$ وهو فعلاً ناتج عملية طرح $(4 - 5)^{10}$ ، المُهم 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 $(1011)^{2}$ لتصبع العملية كالأت



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 ،

أستنتاج:

- عند أجراء عملية طرح بين عددين ، فان المحمل Carry يظهر عندما يكون العدد الاول أكبر من العدد الثاني،
- عند أجراء عملية طرح بين عددين ، فان المحمل Carry لا يظهر عندما يكون العدد الاول أكبر من العدد الثاني،

الأرقام داخل ذاكرة الحاسوب Number in Computer's Memories

لقد تَعلمنا مما سَبق أن أيجاد المُتمم الثاني ، معناة أيجاد سالب العَدد ، مثلاً العَدد $(0110)^{2}$ المُتمم الثاني لة أو سالب العَدد هو $(1010)^{2}$ ، السؤال الذي يتبادر الى الذِهن كيفَ يُميز الحاسوب بين العَدد $(1010)^{2}$ الذي هو $(-6)^{10}$ والعدد $(1010)^{2}$ الذي هو العدد $(10)^{10}$ ؟؟؟

لكي يستَطيع الحاسوب التَمييز بين العَددين ، يَستخدم أسلوب الأشارة والمُقدار بحيث يُخصص الخانة MSB للأشارة ، 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 بالمتمم الثاني فقط للعدد السالب
000	عدد موجب	**0**
001	عدد موجب	**1**
010	عدد موجب	**2**
011	عدد موجب	**3**
100	عددموجب	**-4**
101	عدد سالب	**-3**
110	عدد سالب	**-2**
111	عدد سالب	**-1**

Table 1.7

لاحظ العَدد $(0110)^2$ هو عَدد موجب والسَبب في ذلك هو إن أخر بت bit على جِهة اليسار أي البت الاكثر أهمية MSB يحمل القيمة صفر ، عندما نَجد المُتمم الثاني للعدد $(0110)^2$ والذي هو $(1010)^2$ ، أنظر إلى الخانة Bit الأكثر أهمية MSB أصبحت تَحمل القيمة واحد مشيرة بذلك الى إن العدد هو سالب Negative ..

حالة الطَفَحان Overflow Condition

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 $(255)^{10}$ ، نستنتج من ذلك أن إمكانية التخزين داخل الاجهزة الرقمية محدودة ، تخيل لو أردنا جمع العدد 255 مع العدد 1 ماذا يحدث ؟؟؟

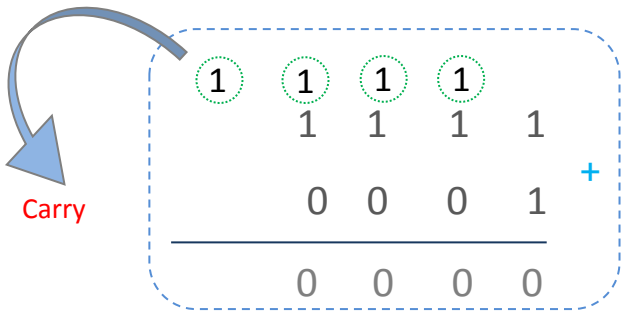
عملية الجمع ستكون ناتجها 256 أي ما يقابلها بالنظام الثنائي $(1\ 0000\ 0000)^2$ ، لقد تجاوز الناتج العدد المَسموح ، الذي هو 255 ، فتظهر حالة الطفحان التي تسبب لنا تصفير المسجل Reset Register .

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

أمثلة على الطفحان Examples

يختلف الطفحان 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 تَفسيرين،

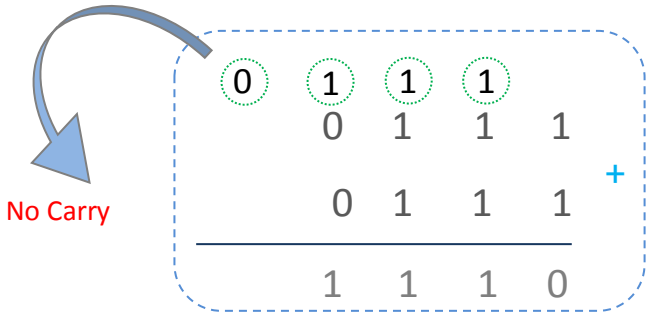
- التفسير الاول :

لنفرض أن العَددان $(1111)^2$ ، $(0001)^2$ هُما من نَوع الأعداد بدون الأشارة Unsigned Numbers ، أي أننا نريد جَمع العَدد $(15)^{10}$ مع العَدد $(1)^{10}$ ليكون الناتج عدد موجَب 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 بالأشارة ليم يظهر في هذه الحالة،

- التفسير الثاني :

لنفرض أن العددان $(1111)^2$ ، $(0001)^2$ هما من نوع الأعداد بأشارة Signed Numbers ، أي أننا نريد جمع العدد $(1111)^{10}$ وهو عدد سالب ، اذا وجدنا المتمم الثاني لة نجد انة $(-1)^{10}$ ، 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 أخر نوضح فية عملية الطفحان ،



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

النظام السادس عشر Hex. Decimal

لقد كان المُبرمجين قَديما يتناقلون برامجهم المكونة من سِلسلة من الأصفار والواحدات عن طريق أوراق Sheets وذلك لبرمجة اجهزتهم Devices ، أن الشكل Figure 1.4 يوضح مَجموعة من الأصفار والواحدات التي تُمثل بَرنامج مكتوب باللغة الثنائية او لغة الماكنة Machine Codes،

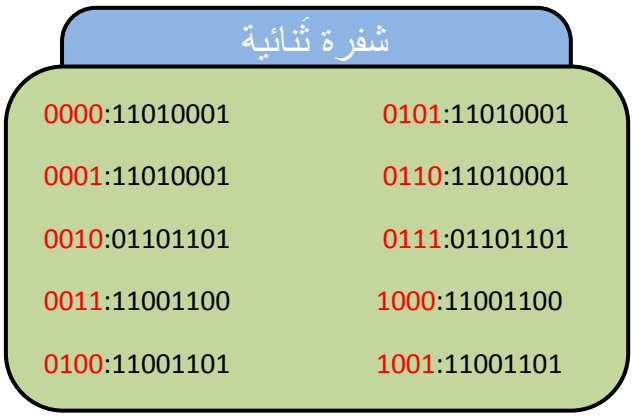


Figure 1.4

هذا الأسلوب أدى الى ظهور شوائب كثيرة Bugs نتيجة إخطاء في عملية نقل البرنامج ، مما دفعهم الى إختراع نِظام جَديد يَسهل عَملية التَحويل من النِظام الثُنائي الى النِظام الجديد ، وذلك لِصعوبة التَحويل من النِظام الثنائي الى النِظام العُشري وبالعكس ، فظهر النظام السادس عشر Hex.Decimal ، هذا النظام أساسة Radix هو 16 ، الأساس ستة عشر يعني أن هذا النِظام مكون من ستة عشر عدداً Digit لتَمثيل 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 النِظام السادس عشر،

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

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

Table 1.8

- مثال 1.11

حول العدد $(0110\ 0001)_2$ الى النظام السادس عشر $(?)_{16}$؟

الحل:

نجزء العدد $(0110\ 0001)_2$ 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 $(0110\ 0001)_2$ هو $(16)_{16}$.

- مثال 1.12

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 1.13

حول العدد $(2E)_{16}$ الى النظام الثنائي $(?)_2$؟

الحل:

نجزء العدد $(E\ 2)_{16}$ 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 1.13

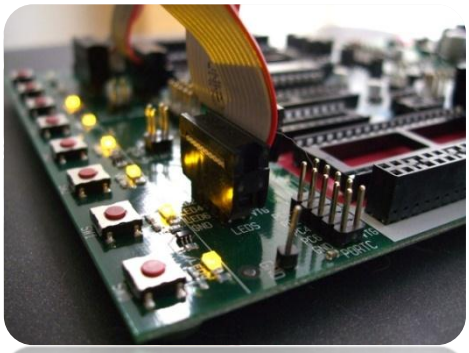
حول العدد $(AB)^{16}$ الى النظام الثنائي $(؟)^2$؟

الحل:

نجزء العدد $(AB)^{16}$ الى جزئين، أي B وA 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 Figure 1.5

Figure 1.5



بُنية الأنظمة القابلة للبرمجة Programmable Devices Archetecture

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

- **أنظمة الربط الصلب Hard wire connection**

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

- **الانظمة القابلة للبرمجة Programmable System**

وهو نِظام رقَمي ذو صيغة مُحددة ومنَظمة،يتكون من ،

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

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

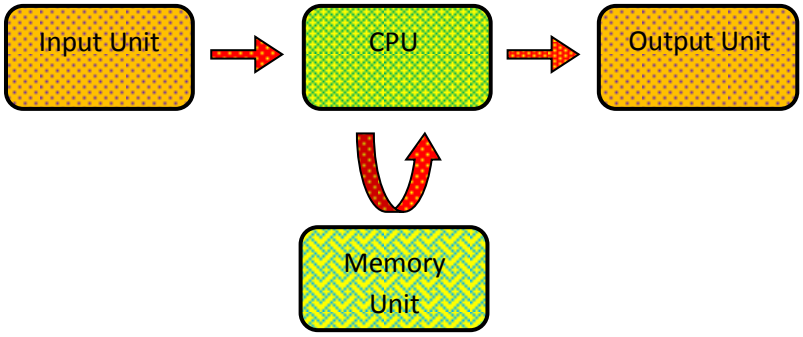


Figure 2.1

وَضيفة وحدة المعالجة المركزية CPU Function

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

١- مرحلة الجلب Fetching cycle
٢- دورة التنفيذ Execution cycle

دورة الجَلب Fetching cycle

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

- نقل محتويات عداد البرامج PC الى الذاكرة memory لكي يؤشر الموقع المطلوب desiresd location
- نقل محتويات الذاكرة Memory الى سجل التعليمة IR

الشكل Figure 2.2 يوضع الترابط بين سجل التعليمة IR ، وعداد البرنامج PC ، والذاكرة Memory

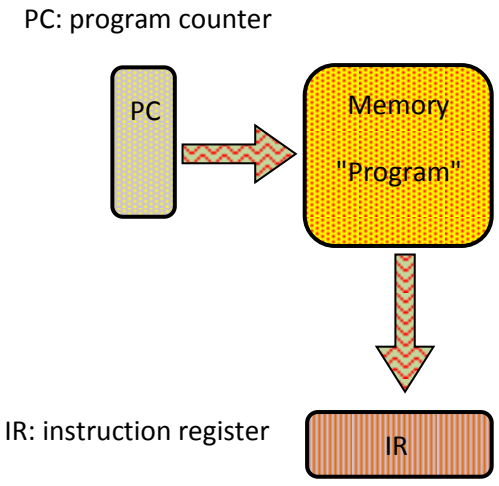


Figure 2.2

دورة التنفيذ Execution cycle

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

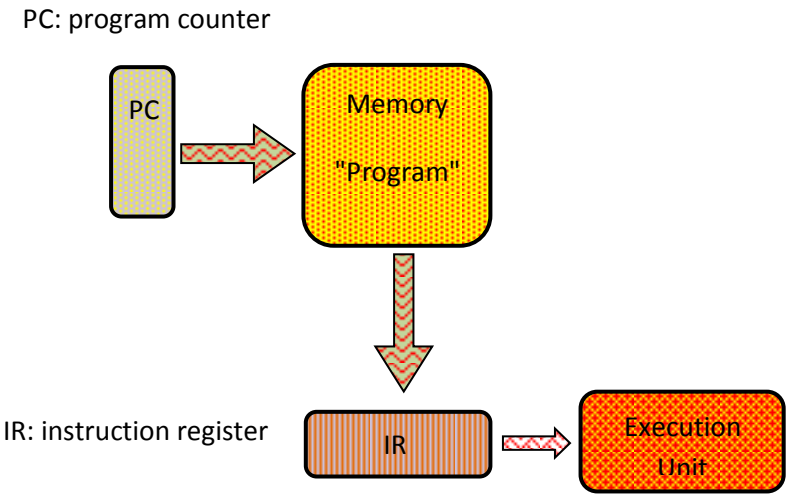


Figure 2.3

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

- مجموعة التعليمات(الايعازات) الحسابية arethmatic instruction
- مجموعة ايعازات النقل moving instruction
- مجموعة ايعازات القفز jump instruction

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

وحدة الحساب والمنطق Arithmatic and logic unit

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 مباشره 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 ،

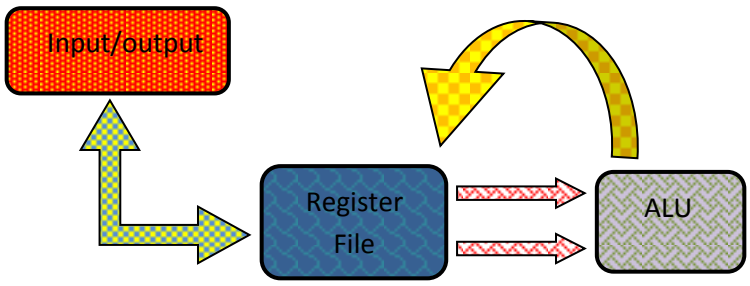


Figure 2.4

السؤال الذي يتبادر الى الذهن ما هي الدائرة المسؤلة عن ؟

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

في الحقيقة الوحدة المسؤلة عن هذه العمليات هي وحدة السيطرة Control Unit ، واحياناً تسمى وحدة توليد دورة الماكنة Machine cycle generation ، وحدة السيطرة ترسل اشارات سيطرة Control Siginal 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 الماكنة،

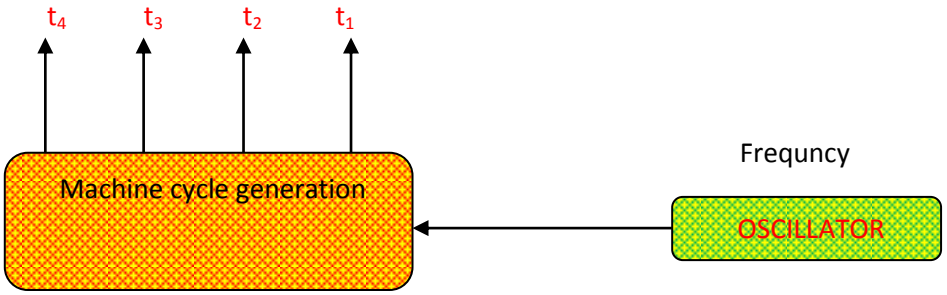


Figure 2.5

وحدة السيطرة(وحدة توليد دورة الماكنة) تحتي على اربع اشارات Siginal أو أكثر حسب نوع المعالج Processor ، هذه الاشارات تسمى t_1,t_2,t_3,t_4 ،عند اول نبظة للمذبذب Oscillator، فأن 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 النبظة الثانية من المذبذب فأن وحدة السيطرة ترسل اشارة فقط عبر t_2 كذلك بقية الاشارات تكون غير مفعلة ،تستخدم هذه الاشارة لنقل محتويات وحدة الذاكرة memory الى سجل التعليمة ، عندما تأتي النبظة الثالثة من المذبذب فأن 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 ، فانة العداد PC سيكون مؤشر الى الموقع التالي و حيث أن t_1 تقوم بنقل محتويات العداد PC الى موقع الذاكرة memory لكي يتم تحديد موقع الذاكرة الذي عنوانة هو محتويات العداد PC، عندما تأتي النبظة الرابعة من المذبذب فأن وحدة السيطرة ترسل اشارة فقط عبر t_4 وتكون بقية الاشارات غير مفعلة ،تستخدم هذه الفترة لفك شفرة التعليمة وتنفيذها وحدة السيطرة هذه مبسطة في الحقيقة تحتاج فك شفرة التعليمة وتنفيذها اكثر من فترة زمنية t ، عندما تأتي النبظة الخامسة من المذبذب فأن وحدة السيطرة ترسل اشارة فقط عبر t_1 وبقية الاشارات غير مفعلة ، أي تعاد الدورة من جديد ، أذن نستنتج من ذلك ان كل اربع نبظات من المذبذب تنفذ تعليمة كاملة ، وكلما زادت سرعة المذبذب زادت سرعة تعاقب الفترات الزمنية t ، الجدول Table 2.1 يوضح فترات time وحدة السيطرة،

الحالة	العملية خلال الفترة الزمنية المفردة T	T_4	T_3	T_2	T_1	المذبذب
دورة ماكنة الاولى ، نتيجتها تنفيذ تعليمة كاملة	نقل محتويات عداد البرنامج الى وحدة الذاكرة	0	0	0	1	النبضة ١
	نقل نقل محتويات الذاكرة الى سجل التعليمه	0	0	1	0	النبضة ٢
	زيادة محتويات عداد البرنامج بواحد	0	1	0	0	النبضة ٣
	فك شفرة التعليمة وتنفيذها	1	0	0	0	النبضة ٤
دورة ماكنة الثانية ، نتيجتها تنفيذ تعليمة كاملة	نقل محتويات عداد البرنامج الى وحدة الذاكرة	0	0	0	1	النبضة ٥
	نقل نقل محتويات الذاكرة الى سجل التعليمه	0	0	1	0	النبضة ٦
	زيادة محتويات عداد البرنامج بواحد	0	1	0	0	النبضة ٧
	فك شفرة التعليمة وتنفيذها	1	0	0	0	النبضة ٨
						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 تستغرقة التعليمة الواحدة لكي تنفذ وما هو عدد التعليمات التي يمكن تنفيذها في الثانية الواحدة ؟

الحل:

بما أن التردد مقلوب الزمن أي

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

أي أن الزمن

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

أي أن النبظة t تسترق فترة زمنية مقدارها 0.001 من الثانية ، اذن اربع نبظات تستغرق 0.004 = 4×0.001 من الثانية ، أي ان فترة تنفيذ تعليمة واحد هي 0.004 من الثانية ، اما عدد التعليمات التي يمكن تنفيذه في الثانية الواحده هي ،

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

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

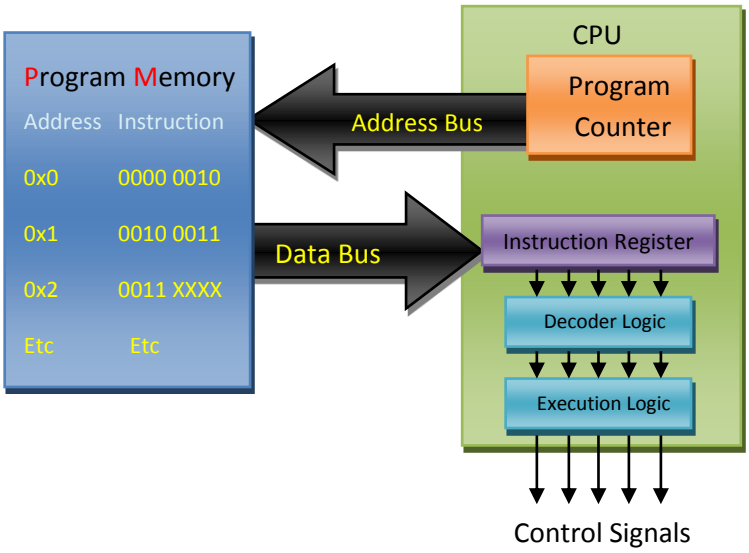


Figure 2.6

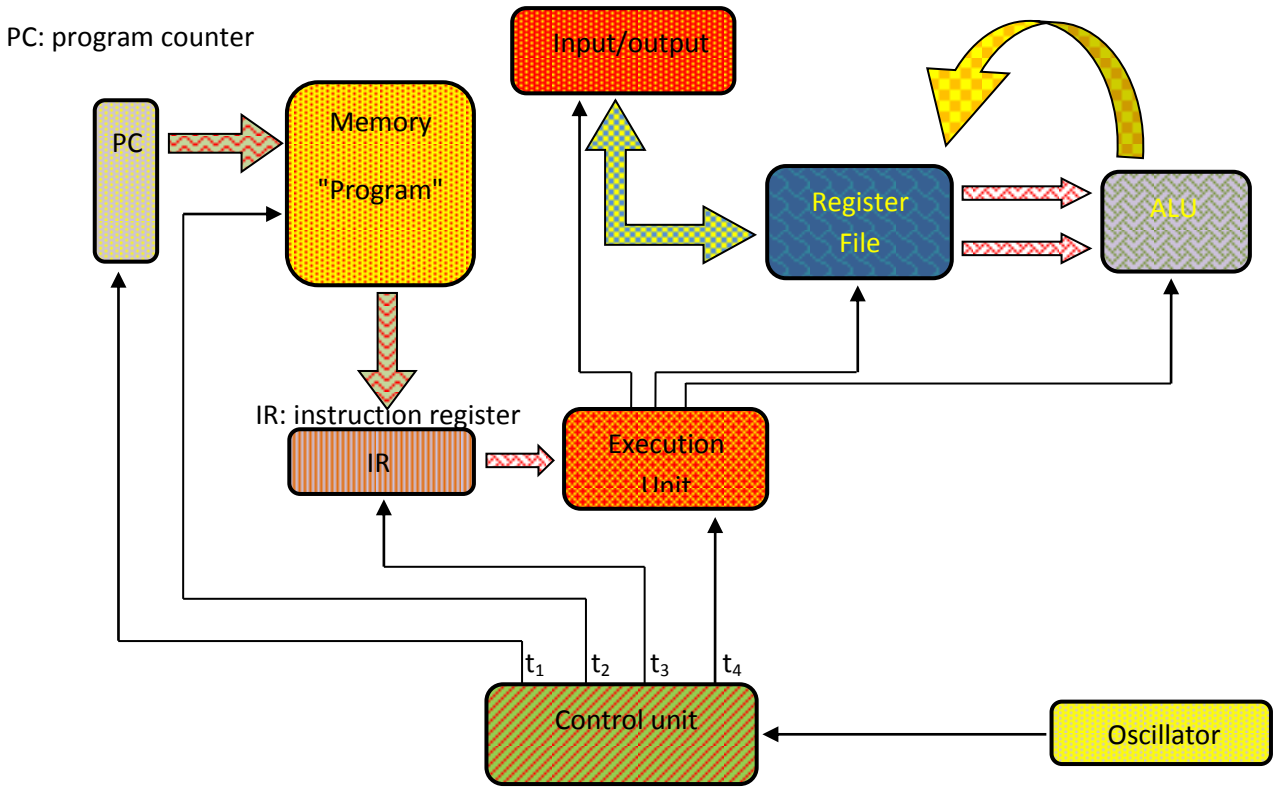


Figure 2.7

لاحظ ان وحدة التنفيذ execution unit ترسل اشاراة 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 عنواين السجلات المراد اجراء العمليه 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 النواقل 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 التعليمة IR ، وكذلك الناقل الذي يربط وحدة ALU مع Register file ،
- نواقل السيطرة Control Buss : وهو الناقل الذي يرسل اشارات السيطرة ليضمن عمل وتزامن الوحدات الاخرى ، مثل هذه الاشارات هي إشارات وحدة السيطرة t_1,t_2,t_3,t_4 ، إشارات السيطرة موضحة في الشكل اعلاة بشكل اسهم Arrow

لكي يؤدي CPU عملة ، يحتاج الى توجية ، عملية التوجية تتم عن طريق سلسلة من ألاوامر instruction تُسمى برنامج .. هذا البرنامج يخزَن داخل ذاكرة memory .....

وحدة الذاكرة Memory

وهي الوحدة التي يخزن فيها البرنامج الذي سينفذ من قبل وحـــدة المعالجة المـــركزية CPU، وهناك نوعان رئيسيان من الذواكرة

١- ذاكرة الولوج العشوائي RAM(Random Access memory)

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

٢- ذاكرة القراءة فقط ROM(Read only memory)

وهي ذاكرة لا تفقد معلوماتها حتى بعد انقطاع التيار الكهربائي عنها وهي على أنواع أهما ذاكرة EPROM ، وهي ذاكرة قابلة للمَسح وتحتفظ ببياناتها حتى بعد انقطاع مصدر التغذية عنها ، لذلك تستخدم هذه الذاكرة لخزن البرنامج Program..

البنية الداخلية للذاكرة Internal Structure of Memory

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 adrdress 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 ، واشارة Read التي نعني بها أننا نريد قراءة محتويات الذاكرة ، الشكل Figure 2.8 يوضح المخطط الصندوقي النموذجي للذاكرة،

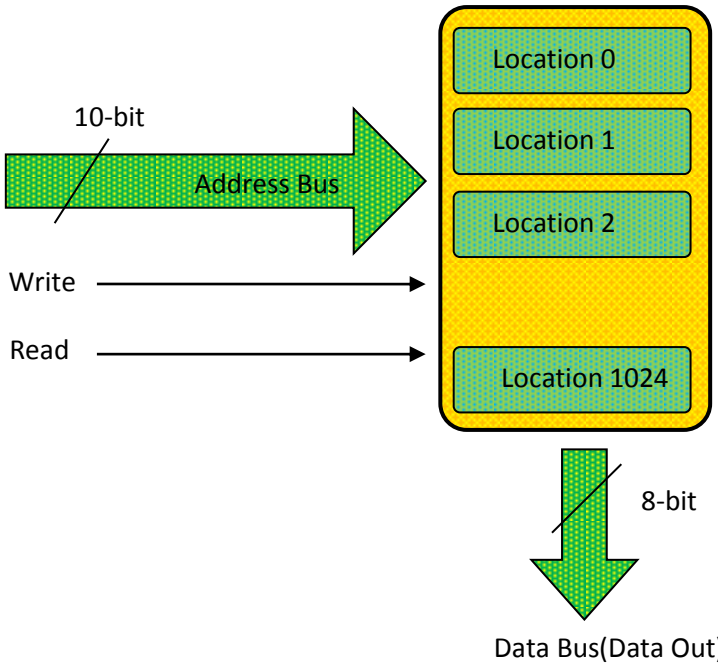


Figure 2.8

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 فأن 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،

- ذاكرة البرنامج program memory
- ذاكرة المعطيات data memory

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 ....

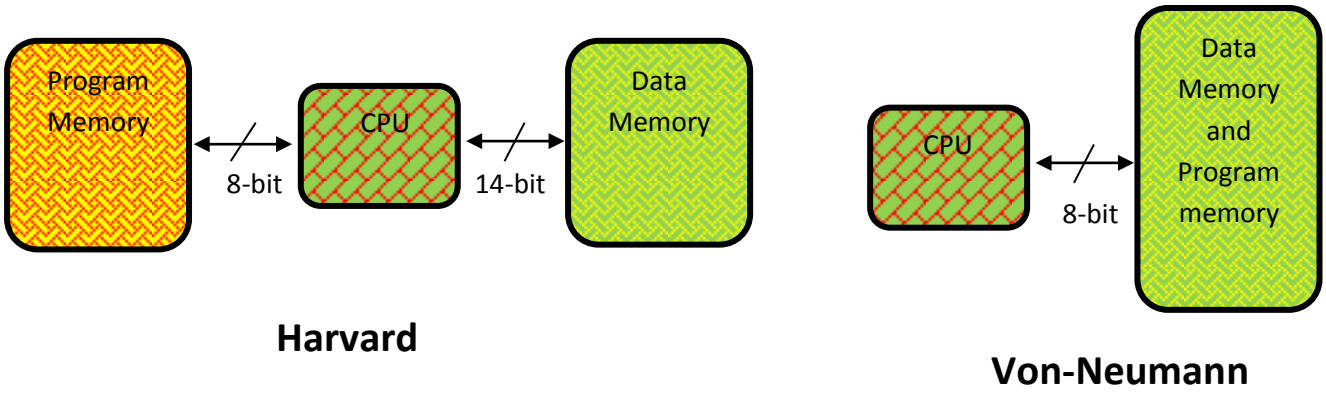


Figure 2.9

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 π التي قيمته 3.14 وذلك للاستفادة منها اثناء معالجة البرنامج ،

Input Unit وحدة الأدخال

وهي الوحدة التي من خلالها ، تدخل الاشارات الرقمية Digital Siginal او البيانات Data الى النظام وذلك للاستفادة من هذه البيانات ومعالجتها، أشارات 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 ،

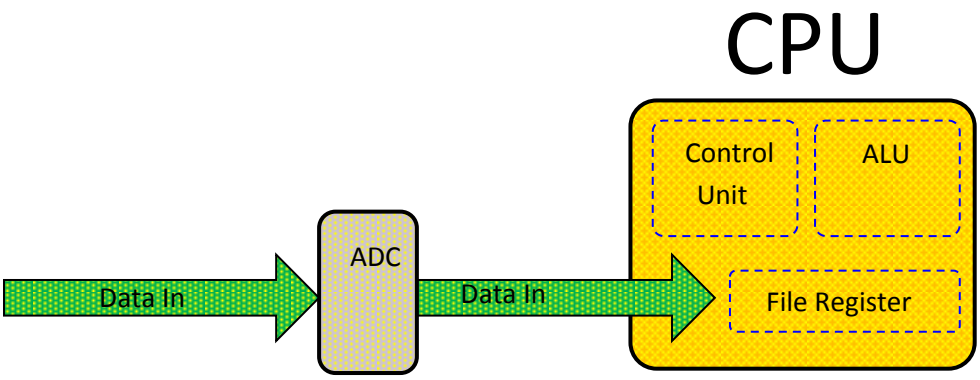


Figure 2.10

Output Unit وحدة الاخراج

وهي الوحدة التي من خلالها ، تخرج الاشارات الرقمية Digital Siginal او البيانات Data من النظام الى وحدات الاخراج مثل الثنائيات الباعثة للضوء LED او العارضات ذات السبع قطع Segmint 7، 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 ،

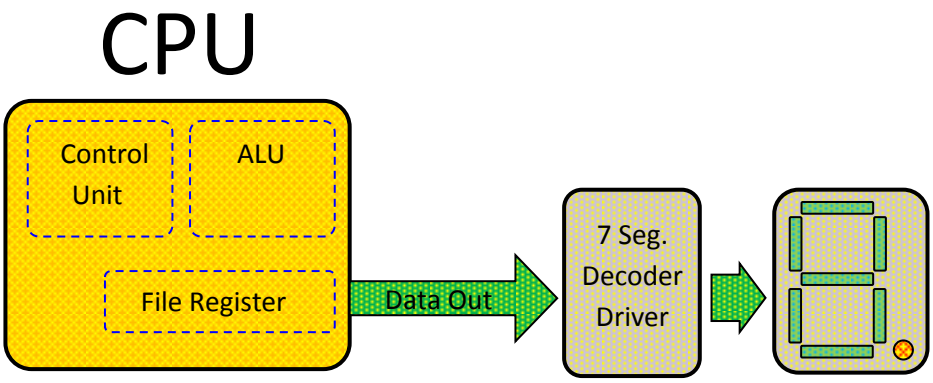


Figure 2.11

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 أوجزء منه أدخال 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

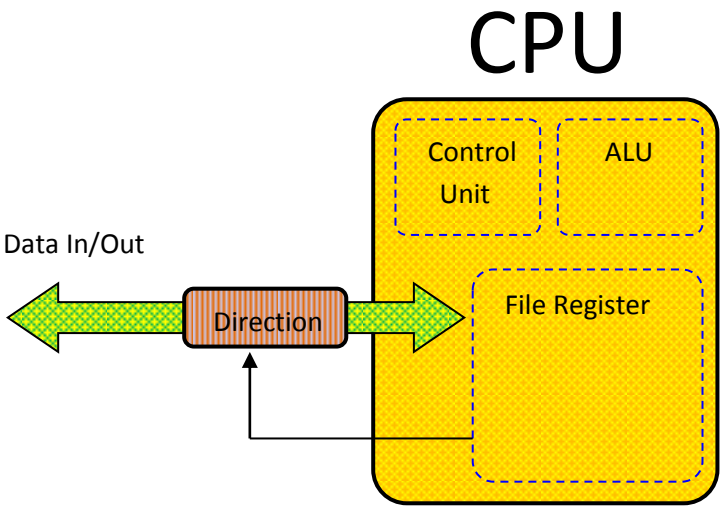
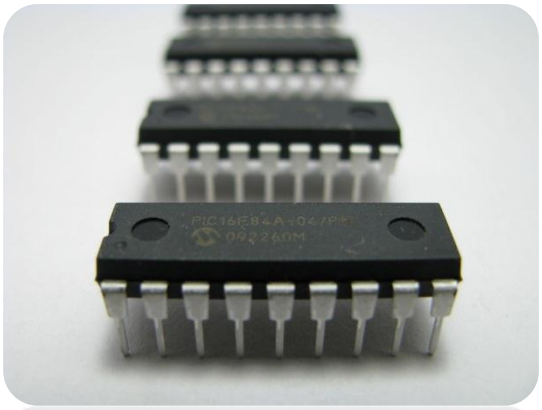


Figure 2.12



بُنية المُسَيطر الصغري

PIC16f84A

مُقدمة:

في هذه الوحدة سوفَ نتعَرف على بُنية المُسيطر الصغري Architecture of Microcontroller وعلى تَنظيم ذاكرة المُسيطر الصغري Memory Organization ، كما سنتعرف من خلال هذة الوحدة على كيفية تهيئة المسيطر الصغري للعمل ، أرجوا من القارء الكَريم قراءة هذه الوحدة بتَركيز ، لأن هذه الوحدة تُعتبر أساس الفهم الصَحيح لبَرمجة المُسيطر الدَقيق 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 الكهربائية مثل Split, Air-conditions
- الأجهزة المكتبية مثل Digital Clocks
- أجهزة السيطرة الرقمية PID,PLC

يتوفر المُسيطر الدَقيق في السوق بشَكل دائرة متكاملة integrated circuit (IC) ، بأبعاد عدة سنتيمترات Centimeter ، الشكل Figure 3.1 يوضح شريحة مُسيطر دَقيق ، يوجد داخل هذه الدائرة المتكاملة IC ، معالج Processor ، وذاكرة لخزن 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 .

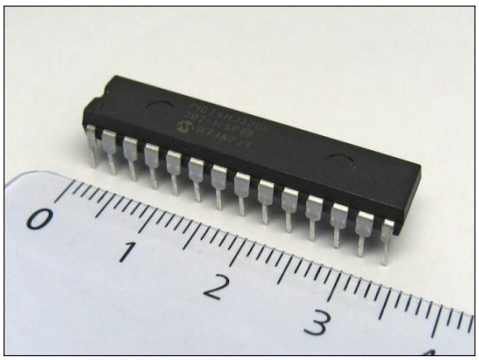


Figure 3.1

سوف نتحدث في هذه الوحدة عن بُنية أحد انواع الأجهزة القابلة للبَرمجة وهو المُسيطر الدَقيق Microcontroller من شَركة Microchip ، تنتج شركة Microchip سِلسِلة من المُسيطرات الدَقيقة Microcontroller تحتَ 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 تَعليمات التفرع jump instruction فأنها تأخذ دورتان ماكنة.
- يعمل على تردد 4 Mega Hertz ، دورة الماكنة لهذا المسيطرة مكونة من 4 نبظات لاتمام تفيذ 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 لخزن البيانات Data واسترجاعها.

بنية المُسيطر الدَقيق لا تختلف كَثيراً عن بُنية باقي الأجهزة القابلة للبَرمجة ، مثل Microprocessor وغيرها ، مُعظم الأجهزة القابلة للبرمجة تتكون من شقين

- البنية الصلبة Hardware
- بنية البرمجيات Software

Hardware البُنية الصلبة

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 لبُنية المُسيطر الدَقيق الصَلبة Hardware اما البَرمجيات فسَنتركها لوحدات لاحقة ، الشكل Figure 3.2 يوضح بنية المسيطر الدقيق PIC16f84A ،

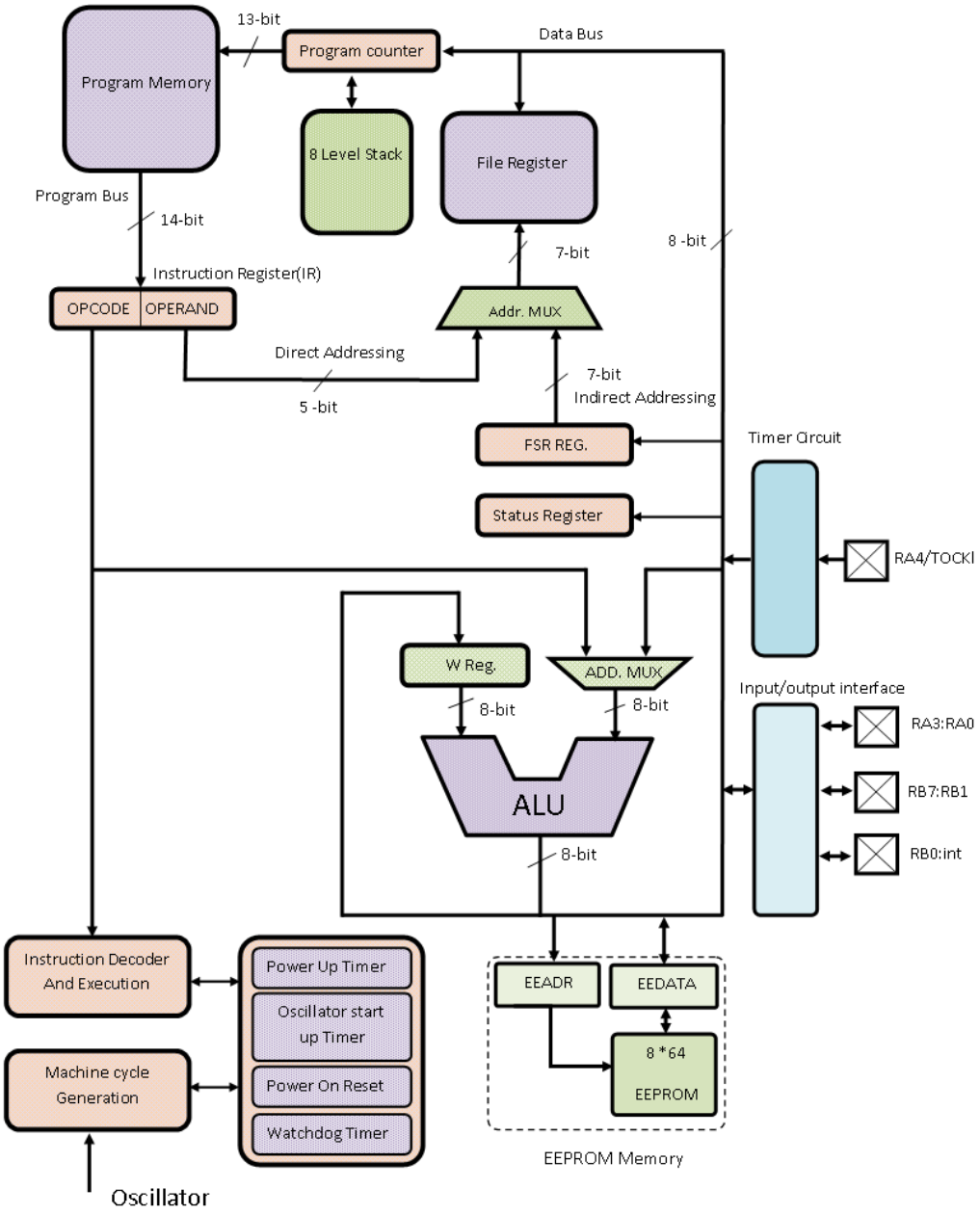


Figure 3.2

وحدة المعالجة المركزية CPU — Central Processing Unit (CPU)

وحدة المُعالجة المَركزية CPU كما مر علينا في الوحدة السابقة هي قلبُ المُسيطر الدَقيق Microcontroller ، وهي المسؤولة عن جَلب وفك شفرة وتنفيذ التعليمة Instruction ، كما عَرفنا سابقاً أن دورة الماكنة تبدء بعملية Fetch أي جَلب التَعليمة من ذاكرة البرنامج program memory لتخزن في مسجل التعليمة IR(instruction Register) وفق الخَطوات التالية :

١- يقوم عداد الرنامج PC(Program Counter) بعنونة الذاكره كما هو موضع في الشكل Figure 3.2، وذلك لالتقاط التَعليمة من الذاكرة ، لاحظ أن المسيطر PIC16f84A طول عداد البرنامج PC له هو 13-bit .

٢- التقاط التعليمة من ذاكرة البرنامج Program Memory التي عنوانها هو محتويات عَداد البرنامج PC ، لينتهي بها المَطاف في مُسجل التَعليمة 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 او الوصل الى File Register هما ؟

١- العَنونة المباشرة Direct Addressing
٢- العَنونة الغير مباشرة Indirect Addressing

العنونة المباشرة — Direct Addressing

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

```
BSF Reg. , b
```

تقوم هذه التعليمة او الأيعاز برفع احد بتات مسجل Register الى القيمة واحد ، تتكون هذه التعليمة من جزئين ،

١- الجزء الأول وهي شفرة العَملية Opcode (operation code) التي هي BSF وهي اختصار لكلمة Bit set flag ، التي تخبر المعالج او المسيطر الدقيق بالوضيفة التي نريد اجراءها التي هي جعل أحدى بتات bit مُسجل Register قيمتة 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 المَوجدوه في File Register هو 8-bit ، لذلك يطلق على المسيطر الدقيق PIC16f84A أنة نظـــــام ذو 8-bit ،

مثال 3.1

أحد المسجلات الموجدة في file Register هو المسجل الذي عنوانة 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 الرَقم وذلك للدلاله على أن الرقم بالنظام السادس عشر Hex مثلاً 0x02,0x08,0x0D

لاحظ أن المسجل 0x0C قد تم الوصول الية مباشرتاً 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 تنشطر 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 علية عبر ناقل العنونة المباشرة Direct Addressing الى file Register لاحظ في الشكل Figure 3.2 هناك شيء اسمة Addr. MUX وهو اختصار Address multiplexing أي مازج العنوان وهو مثل المفتاح الذي يحول بين العنونة المباشرة والعنوانة الغير مباشرة ، في عملية العنونة المباشرة سيرتبط File Register مع ناقل العنونة المباشر Direct Addressing عن طريق المازج Addr. MUX،

العنونة الغير المباشرة Direct Addressing

يقصد بالعنونة الغير مُباشرة هو أن عنونه أو الوصول الى أحد مسجلات File Register لا يتم عن طريق التعليمة instruction ، بل يتم عن طريق مسجل أخر 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 عنوانة 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 مناقشتة بشكل اوسع في وحدات لاحقة ،

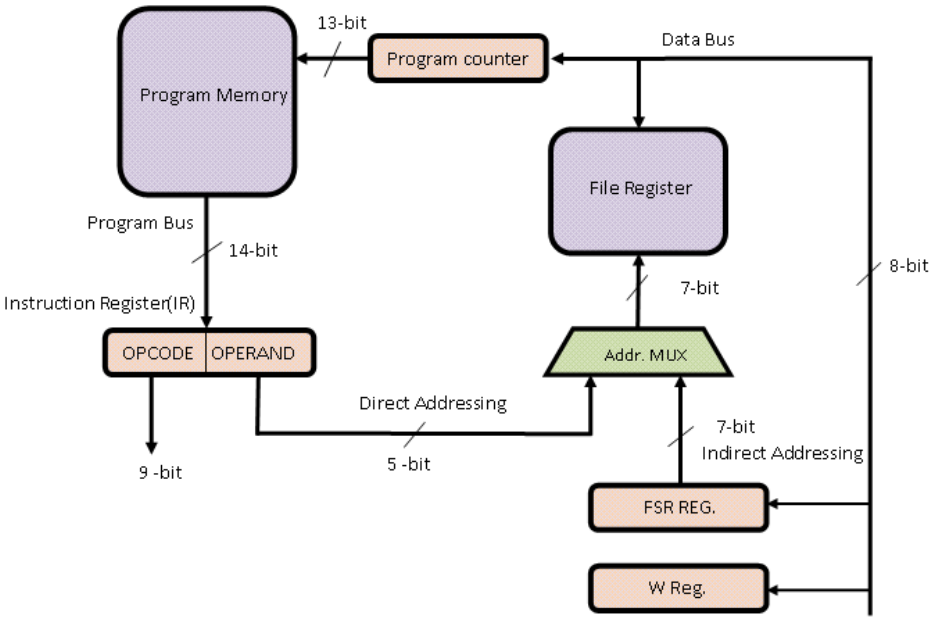


Figure 3.3

الشكل Figure 3.3هو جزء من بنية معالج المسيطر PIC16f84A ، عَمدت الى تقسم بنية المعالج وذلك لكي يسهل عملية فهما واستيعابها ، لنفهم ما هو مسجل العمل Working Register او W Reg. الموجود في الشكل Figure 3.2 يجب علينا فهم ألية عمل وحدة الحساب والمنطق ALU في المسيطر PIC16f84A،

ALU (Arithmetic and logic unit) وحدة الحساب والمنطق

وهي الوحدة المسؤولة عن إجراء العَمليات الحِسابية والمَنطقية ، وأهم ما في وحدة الحساب والمنطق هو مُسجل العَمل W(work register) ، وهو مسجل يستخدم بكثرة في المُسيطر الدَقيق PIC16f84A ، لكي نفهم ألية عمل وحدة الحساب والمنطق ALU ، يجب ان ننتبة الى ،

- لاحظ في الشكل Figure 3.4 ، أن أحد مُعاملات أو مدخلات وحدة الحساب والمنطق ALU هو مُسجل العمل W Reg. ، أي عند اجراء عَملية الجَمع او أي عملية داخل وحدة الحساب والمنطق ALU يجب أن يكون مُسجل العَمل 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 التعليمة مباشرتاً ، 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 التعليمة مباشرتاً .

كما ان لمسجل العمل فوائد كثيرة ، حيث انه الوسيط في اكثر تعلميات instruction المُسيطر الدَقيق ، وهذا ما سوف تكتشفة في وحدة البرمجة أنشاء الله.

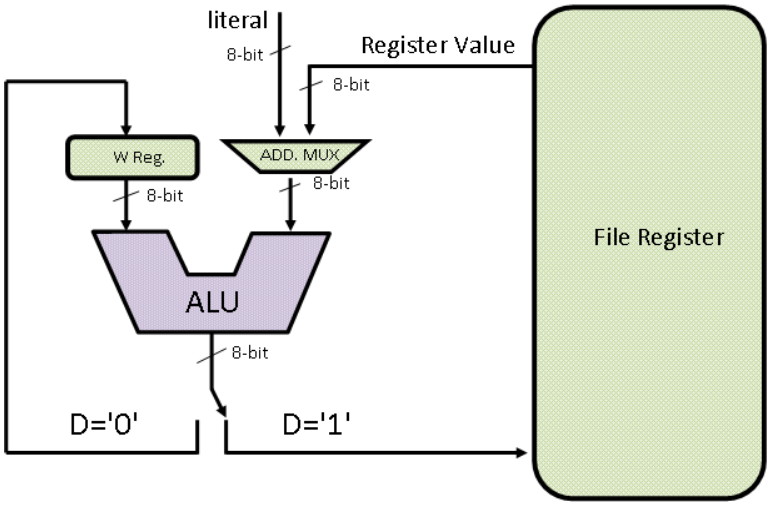


Figure 3.4

Status Register مسجل الحالة

هناك مسجل ضمن بنية المُسطر الدَقيق PIC16f84A يسمى مُسجل الحالة Status Register أو مُسجل الأعلام Flag register وسمي بمُسجل الحالة لانة يُسجل حالة أو ناتج العملية التي أوديت داخل وحدة الحِساب والمَنطق ALU ، وهو مُسجل مكون من 3-bit كل بت يشير الى حالة من الحالات التالية ،

١- علم التصفير Zero flag : عندما يصبح ناتج عَملية حِسابية داخل وحدة ALU يساوي صفر فان هذا العلم يرفع الى الواحد وإلا فانة يصبح صفر .
٢- علم المحمل Carry flag : يرفع هذا العلم الى الحالة واحد عند ظهور محمل Carry ، من الخانة الاكثر أهمية MSB وإلا اصبح قيمة هذا العَلم صفر .
٣- علم المحمل الثانوي Decimal carry flag : يرفع هذا العلم الى الحالة واحد عند ظهور محمل من البت الثالث الى البت الرابع وإلا تصبح قيمة هذا العلم صفر .

ملاحظة:

حالات الأعلام الثلاثة هذه تجده مضمنة داخل مسجل في File Register يسمى مسجل الحالة Status ، الشكل Figure 3.5 يبين أشارات خارجة من وحدة ALU ذاهبة الى مسجل الحالة Status Register ،

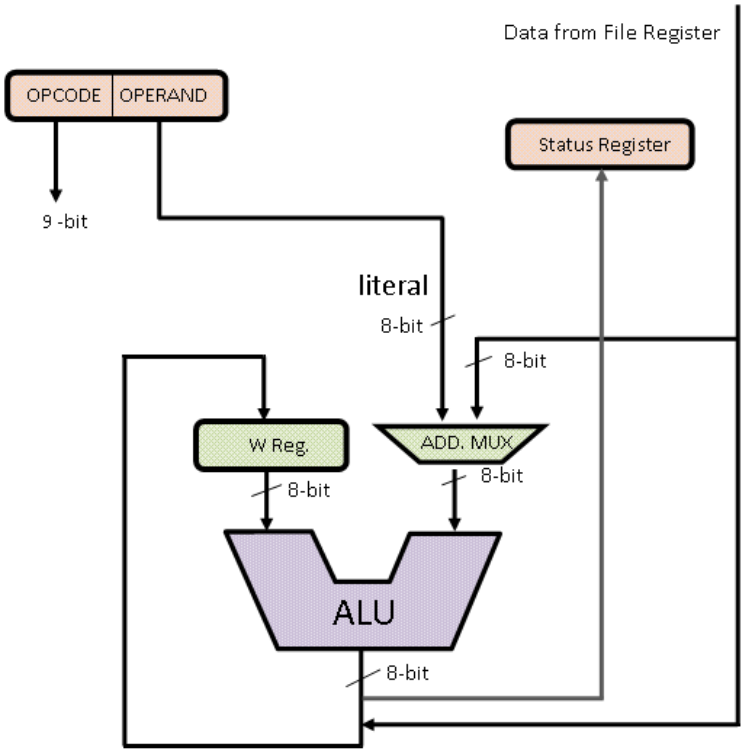


Figure 3.5

Control Unit وحدة السيطرة

بعد أن تتم عَملية الجَلب Fetch cycle تصَل التَعليمة الى مسُجل التَعليمة IR لتنشطر الى جزئين جزء المعاملات Operand الذي يذهب عن طريق ناقل العنونة المباشرة Direct Addressing ليصل الى Register File عن طريق مازج 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 العَمليات هي وحدة السيطرة او وحدة توليد دورة الماكنة Machine Cycle Generation ،أن وحدة السيطرة في المُسيطر الدَقيق PIC16f84A تتكون من أربع نبظات تسمى Q_0,Q_1,Q_2,Q_3 الشكل Figure 3.6 يبين مخطط صندوقي بسيط لوحدة توليد دورة الماكنة ،

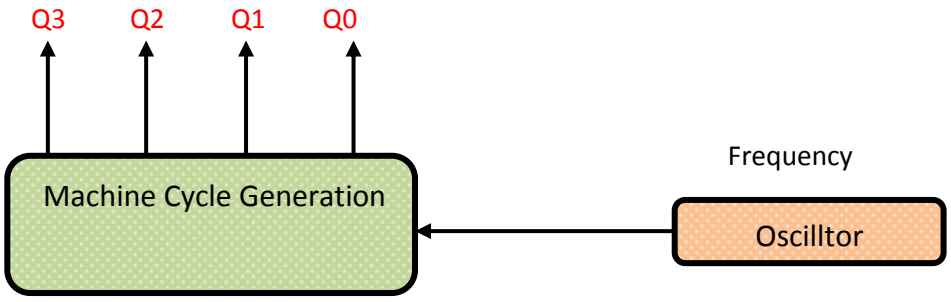


Figure 3.6

Memory Organization تنظيم الذاكرة

هناك ذاكرتان في وحدة المسيطر الدقيق PIC16f84A هما ذاكرة البرنامج Program memory وذاكر البيانات data memory ، لكل من هاتان الذاكرتان نواقلهما Buses الخاصة ، لذلك يمكن قراءة الذاكرتان في نفس الدورة Cycle .

Data memory ذاكرة البيانات

في المسيطر الدقيق PIC16f84A تُسمى ذاكرة البيانات بـ file Register وهي ذاكرة مكونة من 256 موقع او FFh بالنظام السادس عشر Hex. ، كل موقع متكون من 8-bit ، هذه الذاكرة مقسمة الى جـــزئين Two bank هما Bank 0 الـذي يبدء من العنوان 00h الى العنوان 7Fh و Bank 1 الذي يبدء من العنوان 80h الى العنوان FFh ، أي ان كل bank يتكون من 128 موقع ، كما هو موضح في الشكل Figure 3.7

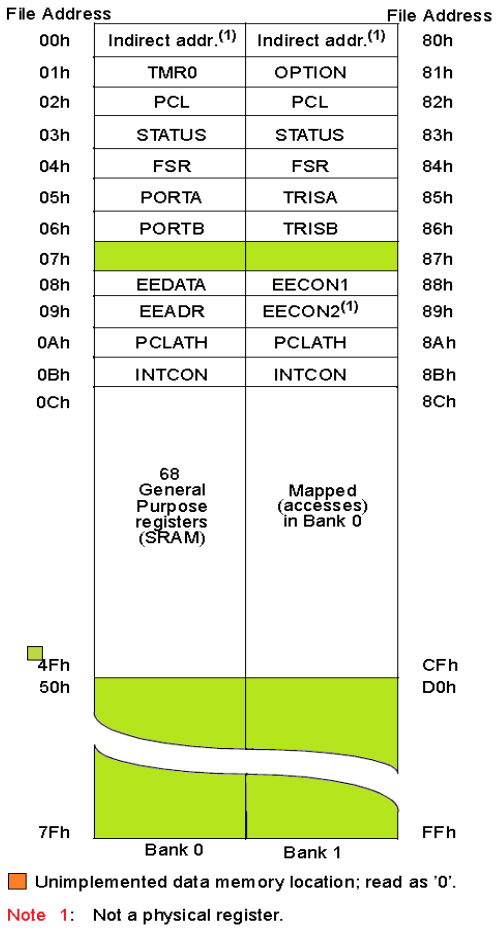


Figure 3.7

تقسم ذاكرة البيانات File register الى قسمين ،

- مجموعة مُسجلات الأغراض الخاصة Special Purpose Registers(SPRs)
- مجموعة مُسجلات الأغراض العامة General Purpose Registers(GPRs)

مسجلات الأغراض العامة General Purpose Register(GPRs)

تستخدم هذة هذه المسجلات للأغراض العامة ، مثل خزن المعاملات Operands ، خزن نواتج العمليات Result ، وغيرها من العَمليات تبدء هذه المسجلات عند bank 0 من الموقع 0Ch وتنتهي عند الموقع 4Fh ، اما عند bank 1 فانها تبدء من الموقع 8Ch وتنتهي عند الموقع CFh .

مسجلات الأغراض الخاصة Special Purpose Register(SPRs)

تستخدم هذة هذه المسجلات للأغراض الخاصة ، مثل تهيئة أطراف Peripheral المسيطر الدقيق PIC16f84A ، تهيئة المؤقتات والعدادات وغيرها من العَمليات ، لكل موقع من هذه المواقع له وضيفة معينة ستتعرف عليها خلال مواصلتك لقراءة الكتاب ، تبدء هذه المسجلات عند bank 0 من الموقع 00h وتنتهي عند الموقع 0Bh ، اما عند bank 1 فانها تبدء من الموقع 80h وتنتهي عند الموقع 8Bh .

ملاحظة :

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

ذاكرة البرنامج Program Memory

يحتوي المُسيطر الدَقيق PIC16f84A عَداد بَرنامج Program Counter ذو سعة 13-bit ، أي انة يستطيع عنونة ذاكرة سعتها 2^{13}=8191 أي 8K byte ما يقابلها بالنظـــــام الســـادس عشر هو 1FFF ، في الحقيـــقة العناوين المنفذة implemented من الذاكرة كما هـــو موضح في الشكل Figure 3.8 هي مـن (000-3FF) فقط اما العناوين (4FF-1FFF) فهي غير منفذة unimplemented أي انة لايمكن تخزين برنامج ضمن هذه المواقع، ، كما يرتبط مع عداد البرنامج PC بصورة وثيقة ذاكرة المكدس Stack التي تستخدم لخزن قيمة عداد البرنامج PC في تعليمات القفز jump instruction، انتبة الى العنوان 0000h من ذاكرة البرنامج Program memory يسمى هذا العنوان شعاع التصفير Reset Vector أي عند تشغيل المُسيطر الدَقيق او أعادة اقلاعة Restart فانه يبدء التنفيذ من هذا العنوان ، وانتبة ايضاً الى العنوان 0004h فانه يسمى شعاع مقاطعة الأطراف Peripheral Interrupt Vector أي عند حدوث مقاطعة للمسيطر الدقيق فانة يبدء التنفيذ من هذا العنوان ، سيتم شرح المقاطعات في فصول لاحقة أن شاء الله .

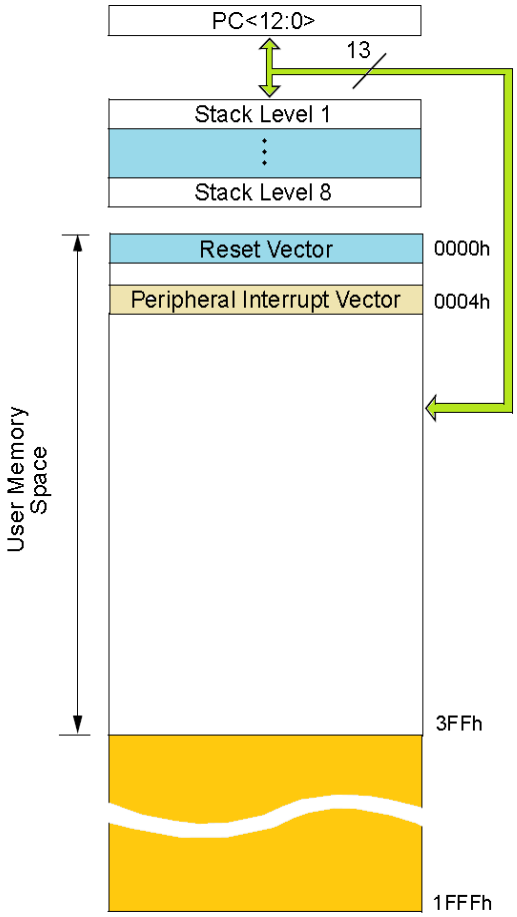


Figure 3.8

Power Up Timer (PWRT)

مؤقت بداية اقلاع الطاقه

هناك دائرة مؤقت Timer circuit داخل وحدة المُسيطر الدَقيق Microcontroller ، تقوم هذه الدائرة بتوفير فترة تأخير مقدارها 72 ملي ثانية قبل ان ينهض (يعمل) المسيطر الدقيقmicrocontroller وذلك لكي تظمن 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 نظمن استقرار الذبذبة لان عدم استقرار الذبذبة يؤثر على عمل وحدة السيطرة وبذلك عمل المُسيطر الدَقيق.

Power On Reset (POR) أعادة ألاقلاع عند بداية تشغيل الطاقة

عند تجهيز المسيطر الدقيق Microcontroller بالطاقه لغرض تَشغيلة Running ، يجب التأكد من أن مستوى الطاقة المطلوب مزود بشكل صحيح لكي يعمل المُسيطر الدَقيق بشكل صَحيح ،لحسن الحَظ يحتوي المُسيطر الدَقيق Microcontroller على دائرة تتأكد من ان تجهيز الطاقه وصل للحد المطلوب والا أعاد الجهاز الاقلاع من جديد Reset تمسى هذه الدائرة Power On Reset(POR).

Watchdog timer مؤقت الحراسه

ماذا يحدث لو عَلق Stuck المُسيطر الدَقيق، هل هناك زر restart لاعادة تشغل 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 المُسيطر الدَقيق ، اذا علق Stuck المسيطر الدقيق تقوم هذه الدائرة باعادة أقلاع المسيطر الدقيق Microcontroller تسمى هذه الدائرة بمؤقت الحراسة Watchdog timer ، سيتم شرح هذا الموضوع بشكل مفصل في فصول لاحقة .

Stack Memory المكدس

يَحتوي المُسيطر الدَقيق PIC16f84A على مكدس Stack ذو سعة 8 مواقع ، يستخدم في عمليات القفز ،

EEPROM Memory الذاكرة EEPROM

يحتوي المسيطر الدقيق PIC16f84A على ذاكرة EEPROM قابلة للقراءة والكتابة ذات سعة 64 موقع ، وكل موقع ذو طول كلمة 8-bit ، تستخدم لخزن 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 .

Timer المؤقت

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

Input/Output Unit أطراف الادخال والاخراج

يحتوي المُسيطر الدَقيق على منفذين Ports لأدخال أو أخراج البيانات الى او من المُسيطر الدَقيق ، 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 منـــفذ 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 وضيفة مثل الطرف RA4 يستخدم كطرف قدح للمؤقت Timer أو للعداد ويسمى T0CKI ، كما يستخدم الطرف RB0 كوضيفة المقاطعة Interrupt ،

يتم تغليف Packaging المُسيطر الدَقيق ، بشكل دائرة متـــــكاملة IC(Integrated circuit) مكونة من ١٨ طـــــــــرف، الشكل Figure 3.9يوضح اطراف المسيطر PIC16f84A .

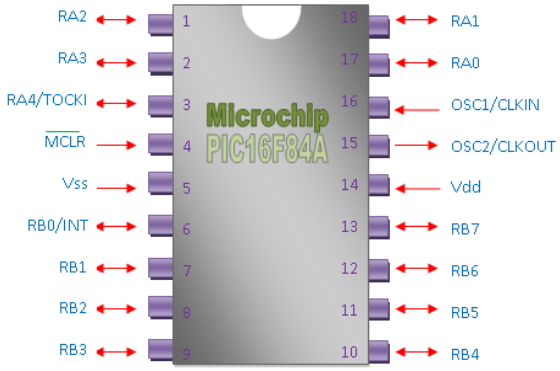


Figure 3.9

من الشكل Figure 3.9 نلاحظ أن أطراف المُسيطر الدقَيق PIC16f84A هي ،

- RA0-RA4

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

- RB0-RB7

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

- 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 المُسيطر الدَقيق اذا لم تربط مذبذب لا يُنفذ المُسيطر الدَقيق البرنامج المخزن في ذاكرتة ، يعمل المسيطر الدقيق PIC16f84A على تردد 4 MHZ ،

- MCLR(memory clear)

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

ملاحظة :

الأطراف التالية لها وظائف أخرى تحدد اثناء عملية Configuration

- RB0

  له وظيفة المقاطعة interrupt لبرنامج المسيطر الدقيق

- RA4

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

طرف تصفير الشريحة Memory Clear

عند تَسليط جهد منخفظ على هَذا الطَرف يتم مسح برنامج الشَريحة ، لذلك نربط هذا الطرف الى الجُــهد الموجب 5+ Vdc، كما هو موضـــــــــــح في الشكل Figure 3.10

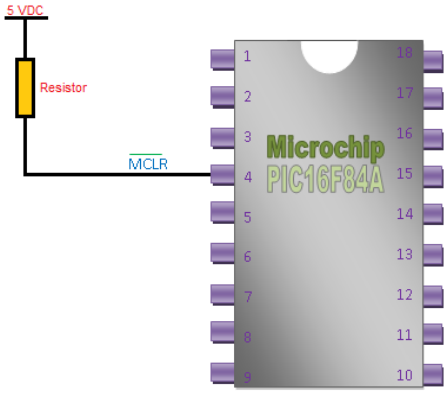


Figure 3.10

المذبذب Oscillator

المذبذب عنصر مهم لعمل المسيطر الدقيق ، وأن أختيار نوع وقيمة المذبذب تؤثر أيجاباً او سلبياً على عمل المسيطر الدقيق ، ، يتم الحصول على ذبذبات أو نبضات الضرورية لعمل المسيطر الدقيق بطريقتين

- بأستخدام مذبذب بلوري XT Oscillator
- بأستخدام مقاومة ومتسعة لتشكيل دائرة رنين RC Oscillator

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 يولدة 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 Figure 3.11 يبين كيفية ربط مذبذب بلوري الى طرفي 16,15 للمُسيطر الدَقيق ، سنكتفي بذكر طريقة المذبذب البلوري لانه اكثر استقرارية من المذبذب RC .

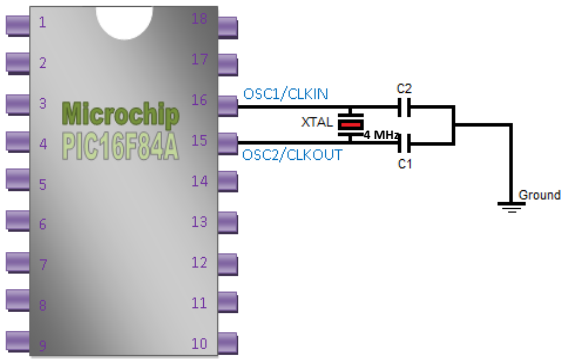


Figure 3.11

الشكل Figure 3.12 يوضع 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 لأغرتض الحماية من الضجيج Noise ، تذكر ربط الطرف 4 الذي هو طرف MCLR الى الطرف الموجب للبطارية.

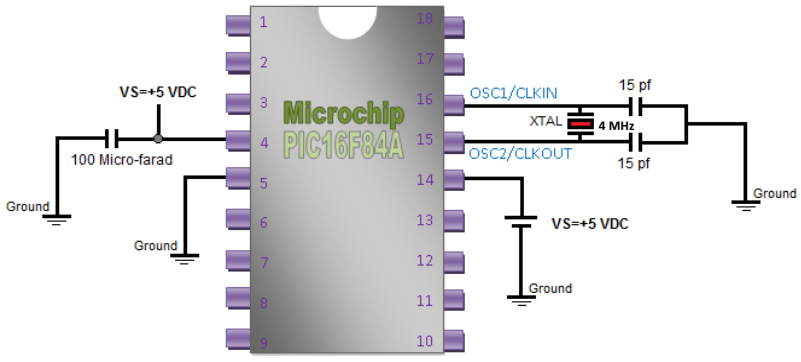


Figure 3.12

File Register

الِسجل

مقدمـــة:

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 يبدء من العنوان 00h الى العنوان 7Fh و Bank 1 الذي يبدء من العنوان 80h الى العنوان FFh ، أي ان كل bank يتكون من 128 موقع ، كما هو موضح في الشكل Figure 4.1

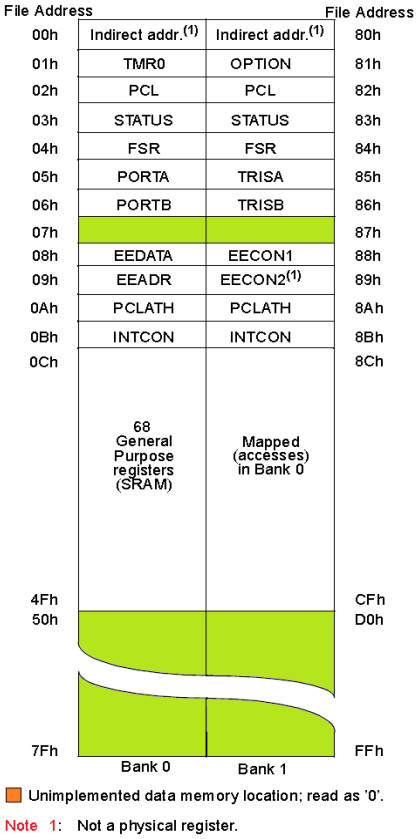


Figure 4.1

تقسم ذاكرة البيانات File register الى قسمين ،

- مجموعة مُسجلات الأغراض الخاصة Special Purpose Registers(SPRs)
- مجموعة مُسجلات الأغراض العامة General Purpose Registers(GPRs)

مُسجلات الأغراض العامة General Purpose Register(GPRs)

تستخدم هذة المسجلات للأغراض العامة ، مثل خزن المعاملات Operands ، خزن نواتج العمليات Result ، وغيرها من العَمليات تبدء هذه المسجلات عند bank 0 من الموقع 0Ch وتنتهي عند الموقع 4Fh ، اما عند bank 1 فانها تبدء من الموقع 8Ch وتنتهي عند الموقع CFh .

مُسجلات الأغراض الخاصة Special Purpose Register(SPRs)

تستخدم هذة هذه المسجلات للأغراض الخاصة ، مثل تهيئة أطراف Peripheral المسيطر الدقيق PIC16f84A ، تهيئة المؤقتات والعدادات وغيرها من العَمليات ، لكل موقع من هذه المواقع له وضيفة معينة ستتعرف عليها خلال مواصلتك لقراءة الكتاب ، تبدء هذه المسجلات عند bank 0 من الموقع 00h وتنتهي عند الموقع 0Bh ، اما عند bank 1 فانها تبدء من الموقع 80h وتنتهي عند الموقع 8Bh ، يتكون File Register من مجموعة مسجلات للأغراض خاصة،عددها ٢٢ سجل،وهي كالأتي

TRISA(Tri-State Buffer) المُسجل

وهو سجل مكون من 8-bit (المستخدم منها فقط 5-bit) لانة يتعامل مع port A المكون من 5-bit ، وضيفتة تحديد أتجاة نقل البيانات Data ، أي تحديد طرف الأدخال Input أو الأخراج Output، يوجد هذا المسجل في الموقع 0x85 من bank1 ، القيمة صفر تعني الطرف المحدد من port a هو طرف أدخال 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 أدخال 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) المُسجل

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

PORTA المُسجل

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

PORTB المُسجل

وهو سجل مكون من 8-bit وضيفتة استقبال البيانات في حالة الكون المنفذ B أدخال ، 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 لة وضيفة معينة الشكل Fig 4.2 يبين مسجل الحالة ، ويوجد هذا المسجل في الموقع 0x03 من bank0 وكذلك يوجد عن الموقع 0x83 من bank 1

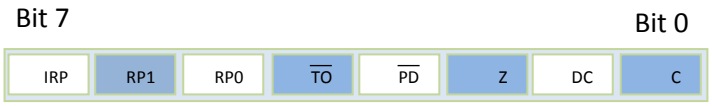


Figure 4.2

- **علم المحمل C(carry flag bit)**

علم المحمل يصبح هذ البت "١" عندما تكون النتيجة الحسابية الحالية لوحدة ALU موجبة ، اما اذا كانت سالبة سيتحول هذا البت الى صفر

- **علم المحمل الثانوي DC(Decimal Carry-Auxiliary carry)**

يضهر عند حوث محمل من البت الثالث الى البت الرابع في النتيجة الحسابية الحالية لوحدة ALU

- **علم التصفير Z(zero flag)**

يضهر عندما يكون الناتج صفر في النتيجة الحسابية الحالية لوحدة ALU.

- **طرف تحديد bank bank select(RP0)**

من المعروف أن المسيطر الدقيق يتكون من two bank ، ولكي نتحويل بين هذين banks ، نستخدم الطرف Rp0 ،حيث يستخدم الطرف Rp0 لتحديد أي bank من File Register سيتم التعامل معه حَسب

Bit 0 = select "bank 0"
Bit 1 = select "bank 1"

- **أنخفاظ 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 مهملة في المسيطر الدقيق Pic16f84a

المُسجل Option Register

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

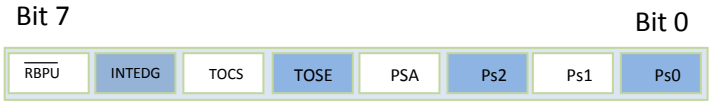


Figure 4.3

- **أطراف تحديد مقسم التردد Prescaler** **Ps0,Ps1,Ps2**

أذا كنت من متخصصي علم الالكترونيات الرقمية ، وقد استخدمت عداد تصاعدي بأستخدام مجموعة نطاطات Flip-flop ، فاكيد أنت تعرف أن العدادات تستخدم كمقسمة للتردد وذلك بأخذ طرف من أطراف مخارج العداد ، لا حظ الشكل Figure 4.4 تجد عداد يسمى prescaler 8-bit خرج هذا العداد ربط الى مازج multiplexer من نوع Mux 8-1 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 .

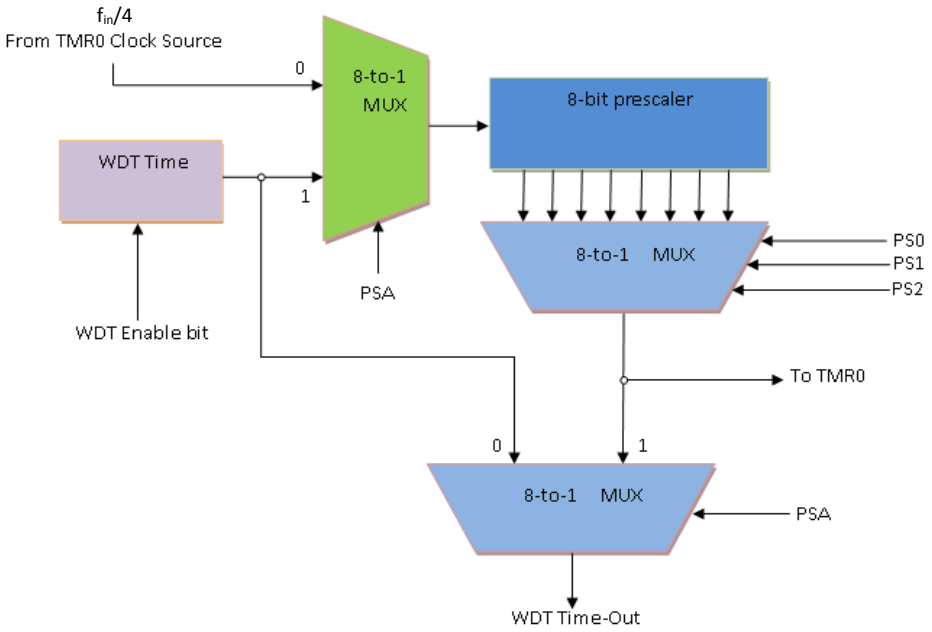


Figure 4.4

ملاحظة :

لمزيد من المعلومات في كيف استخدام العدادات كمقسمة للتردد اقرء 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=frequncy 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

- **الطرف PSA** **PSA(prescaler assignment)**

في المسيطر الدقيق pic16f84a يوجد نوعان من المؤقتات وهما مؤقت الحراسة watchdog timer والمؤقت timer وكلا الموقتان 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,pos1,ps2 ، أعلم ان تردد موقت الحراسة WDTMR هو نفسة تردد المذبب Oscilator وتردد المؤقت هو Oscilator/4 اي تردد المصدر مقسم على اربعة.

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

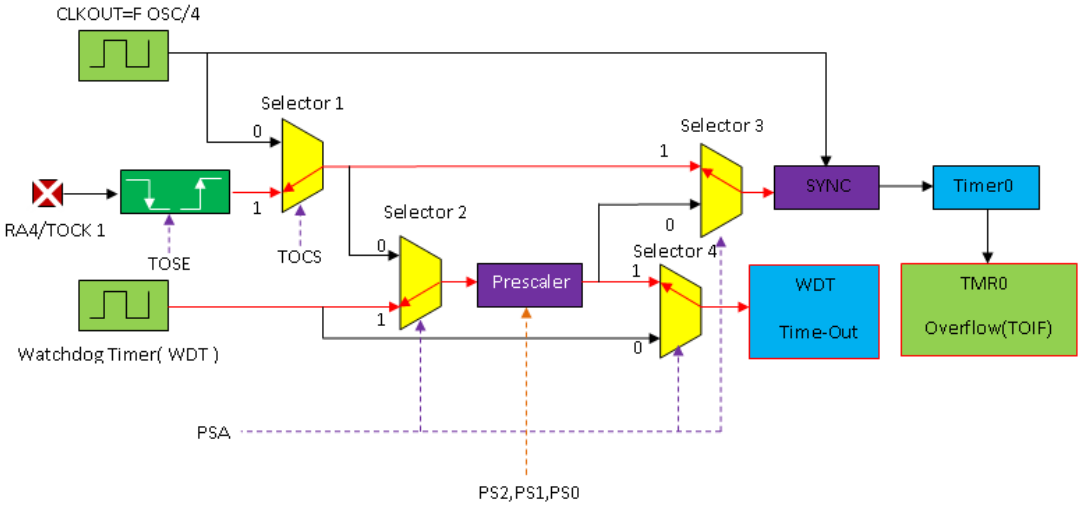


Figure 4.5

ملاحظة :

الغاية الأساسية لاستخدام مقسم التردد prescaler هو لتحديد سرعة العدد Counting

- **الطرف TOCS** **TOCS(TIMER OUT,COUNTER)**

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

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

بالنسبة للعداد،لكي يزداد يحتاج الى عملية قدح من احدى اطراف المسيطر، يتم ذلك عند طريق الطرف RA4/TOCK 1 ، من خلال هذه الخانة يمكنا تحديد نوع جبهة القدح 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

- الخانة INTEDG **INTEDG(interrupt edge)**

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

INTEDG = logic 0 ,falling edge

INTEDG = logic 1 ,rising edge

- الخانة RBPU **RBPU(Register 'B' pull up Resistor enable)**

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 .

مُسجل FSR **FSR Register (File Select Register)**

وهو مسجل مكون من 8-bit يقع عند العنوان 0x04 من bank 0 والعنوان 0x84 من bank 1 ، عند تحميل هذا المسجل بقيمة معينة ، فانة يعتبر هذة 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 ، وتضهر محتويات المسجل porta عند المسجل INDF الذي سنشرحة في الفقرة التالية .

مُسجل INDF **INDF Register (Indirect Addressing Register)**

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

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) مُسجل TMR0

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

Program Counter Latch low & Latch High مُسجل PCL & PCLATH

ذاكرة البرنامج program memory في المسيطر PIC16f84a المنفذ منها فقط 1 Kilo والباقي مهمل ، هذة 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 هذة الصفحات Page وكذلك التغلب على مشكلة الالتفاف warpping حول الصفحة ، لاتهتم كثيرة عن هذة 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 (PC<7:0>) يأتي من المسجل PCL وهو مسجل قابل للقراءة والكتابة يقع عند العنوان 0x02 من bank 0 والعنوان 0x82 من bank1 أما باقي العنوان (PC<12:8>) فيمكن الكتابة الية فقط بصورة غير مباشرة عن طريق المسجل PCLATH وهو مسجل مكون من 5-bit فقط الذي يـقع 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

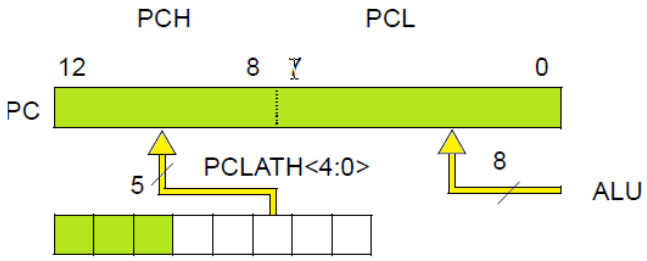


Figure 4.6

INTCON(Interrupt Controller Register) مُسجل INTCON

يتكون هذا المسجل من 8-bit ، كل خانة لها وضيفة معينة ، هذا المسجل مسئول عن تنفيذ او تفعيل خدمة المقاطعات في المتحكم الصغري 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

- **الخانة GIE** **GIE(Global Interrupt Enable)**

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

- **الخانة PEIE** **PEIE(Programmable Earsable ROM Interrupt Enable)**

عندما ترفع هذة الخانة الى القيمة واحد فأن المسيطر الدقيق يمكن خدمة مقاطعة الذاكرة EEPROM والتي تشير الى انتهاء عملية الكتابة Writting ، أما عندما تصفر هذة الخانة فانة يتم ابطال خدمة المقاطعة لاكتمال عملية الكتابة للذاكرة EEPROM .

- **الخانة TMR0IE** **TMR0IE(Timer Interrupt Enable)**

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

- **الخانة INTE** **INTE(Interrupt Enable)**

عندما ترفع هذة الخانة الى القيمة واحد فأن المسيطر الدقيق يمكن خدمة مقاطعة بواسطة الطرف INT ، أما عندما تصفر هذة الخانة فانة يتم ابطال خدمة مقاطعة INT .

- **الخانة RBIE** **RBIE(Portb RB(7:4) Interrupt Enable)**

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

- **الخانة INTF** **INTF(Interrupt Flag)**

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

- **الخانة RBIF** **RBIF (Register portb Interrupt Flag)**

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

- **الخانة TMR0IF** **TMR0IF (Timer Out Interrupt Flag)**

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

مُسجل EEADR EEADR(EEPROM Address Register)

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 .

مُسجل EEDATA EEDATA(EEPROM Data Register)

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

مُسجل EECON1 EECON(EEPROM Controller Register)

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

الوضائف التي يقوم بها المسجل EECON1

- **الخانة RD** **RD (Read Operation)**

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

- **الخانة WR** **WR (Write Operation)**

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

- **الخانة WREN** **WREN (Write Enable Operation)**

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

- **الخانة WRERR** **WRERR (Write Error)**

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

- **الخانة EEPGD** **EEPGD (Eeprom Program Memory Or Data Memory)**

عندما ترفع هذة الخانة الى القيمة واحد فانة تشير بذلك الى ذاكرة البرنامج eeprom program memoey، أما عند تصفير هذة الخانة فأنة يتم التاشير الى ذاكرة البيانات eeprom data memory

مُسجل EECON 2 EECON(EEPROM Controller Register)

هذا الموقع غير منفذ Unimplemented .

Instruction Set

طَقم التَعليمات

مُقدمـــــــــة:

في هذه الوحدة سوفَ نتَعرف على مَجموعة التَعليمات Instruction Set المتوفرة في المتحكم الصغري PIC16f84A ، تُعتبر هذة الوحدة كمرجع لك تَعود الية عند الحاجة الى التَعرف الى وَضيفة تَعليمة مُعينة ، عند قراءتك لهذة الوحدة سوفَ تسأل نفسك ما الفائدة أو الغاية من تعليمة ما ،لا تقلق ، فالتعليمة بحد ذاتها بدون فائدة ما لم تتكامل مع مَجموعة تَعليمات لتؤدي وَضيفة مُعينة مكونة ما يُسمى بالبرنامج Program ، مع تقدمك في فصول الكتاب وكتابة برامج سوف تتَضح الفكرة اكثر، من الجيد معرفه أن عدد التَعليمات المتوفرة في المايكروكونترولر PIC16f84A هو 32 تعليمة ، منها ما يستغرق فترة زمنية مقدارها 1 مايكروثانية ، ومنها ما يستغرق فترة مقدارها 2 مايكروثانية.

البَرنامج Program

عبارة عن سِلسلة من التَعليمات أو الأيعازات instruction المُتتابعة ، التي توجه عَمل المايكروكونترولر PIC16f84A لغرض تأدية وَضيفة معينة .

طَقم التَعليمات Instruction Set

لقد قُلنا أن البرنامج عبارة عن مَجموعة من التَعليمات Instructions الممتابعة ، أذن العنصر الأساسي لتكوين البرنامج هو التَعليمة ، تقسم التعليمات في المايكروكونترولر PIC16f84A إلى ،

- التعليمات على مســتوى البت Bit Orientation Operations
- التعليمات على مستوى البايت Byte Orientation Operation
- تعليمات الســيطرة والثوابت Lateral and Control Operations

التعَليمات على مُستوى البت Bit Orientation Operations

وهي التَعليمات التي تَجري على بت معين ضِمن مُسجل مُعين ، مثل تغيير قيمة بت من 1 الى 0 أو من 0 الى 1 ، عادةً يشار الى البت المراد إجراء العَملية علية بالرمز "b" بينما يُرمرز للمُسجل المُراد تَغيير أحد بتاتة بالرمز "f" ، الجدول Table 5.1 يوَضح مَجموعة التَعليمات على مستوى البت المتوفرة في المايكروكونترولر PIC16f84A ،

Mnemonic, Operands		Description	Cycles
BIT-ORIENTED FILE REGISTER OPERATIONS			
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

التَعليمة BCF Bit Clear Flag

تَقوم هَذه التَعليمة بتَصفير أحد بتات مُسجل Register ، وتأخذ التَعليمة الصيغة التالية ،

```
BCF f,b
```

حيث :

f: يمُثل المُسجل المُراد تَصفير أحد بتاتة،

b: يمثل رقَم البت المراد تَصفيرة من المُسجل f ،

مُلاحظة:

تَستهلك هذه التَعليمة دَورة ماكنة واحدة ، أي أن هذه التَعليمة تَستغرق فَترة زَمنية مقداره 1 مايكروثانية ،

التَعليمة BSF **Bit Set Flag**

تَقوم هذه التَعليمة برفع أحد بتات مسجل Register الى القَيمة واحد ، وتأخذ التَعليمة الصيغة التالية ،

```
BSF  f , b
```

حيث :

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

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

مُلاحظة:

تَستهلك هذه التَعليمة دَورة ماكنة واحدة ، أي أن هذه التَعليمة تَستغرق فَترة زَمنية مقداره 1 مايكروثانية ،

التعليمة BTFSS **Bit Test Flag Skip If Set**

لكي نَفهم وضيفة هَذه التَعليمة ، دَعنا نأخذ الشفرة Code التالية،

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

تقوم التعليمة الأولى والتي هي BTFSS 0x0C,2 بفحص البت Bit الثاني من المسجل 0x0C فأذا كان واحد ، يُحمل عداد البرنامج PC بعنوان التَعليمة الثالثة التي هي BCF 0x0C,3 لينفذها ، بصيغة أخرى عندما تُنفذ التَعليمة BTFSS وكان البت المراد فحصة يساوي واحد يتم قَفز التَعليمة التي تَلي تَعليمة BTFSS ليتم تنفيذها أي سيتم قفز التعليمة الثانية الى التعليمة الثالثة والتي هي BCF 0x0C,3 لقوم المعالج بتنفيذها ، أما أذا كان البت Bit المُراد فحصة يَحمل القيمة صفر فيتم تنفيذ التعليمة التي تلي التعليمة BTFSS مباشرتاً أي سيتم تنفيذ التعليمة الثانية BSF 0x0C,3 ويستمر تنفيذ البرنامج ' أي سيتم تنفيذ التعليمة الثالثة ايضاً والرابعة وهكذا ، اذن التعليمة BTFSS تعتبر من تعليمات القفز Jump وتأخذ التعَليمة الصيغة التالية ،

```
BTFSS  f , b
```

حيث:

f: يُمثل المُسجل المُراد فحص أحد بتاتة ،

b: يُمثل رَقم البت المُراد فَحصة من المُسجل f ،


ملاحظة:

تَستهلك هذه التَعليمة دورة ماكنة واحدة إذا لم يتحقق الشَرط أي إذا كان البت المُراد فَحصة صفر ، أن هذه التَعليمة في هذه الحالة تستَغرق فترة زَمنية مقدارها 1 مايكرو ثانية ، أما إذا تَحقق الشَرط أي عندما يكون البت المراد فحصة يُحمل القيمة واحد ، فستأخذ هذه التَعليمة دورتان ماكنة أي ان الفترة الزَمنية التي تَستغرقها التَعليمة ستكون 2 مايكروثانية.

التَعليمة BTFSC **Bit Test Flag Skip If Clear**

تَعمل هذه التَعليمة عَكس عَمل تَعليمة BTFSS حيثُ أن تعليمة BTFSC تقفز Jumping أذا كان البت Bit المُراد فَحصة يَحمل القيمة صفر ،أما اذا كان البت المراد فحصة يحمل القيمة واحد فسيتم تنفيذ التعليمة التي تلي تَعليمة BTFSC مباشرتاً ، تأخذ التعليمة الصيغة التالية ،

```
BTFSC   f , b
```

حيث:

f: يُمثل المُسجل المُراد فحَص أحد بتاتة ،

b: يُمثل رَقم البت المُراد فَحصة من المُسجل f ،


ملاحظة:

تَستهلك هذه التَعليمة دورة ماكنة واحدة إذا لم يتحقق الشَرط أي إذا كان البت المُراد فَحصة واحد ، أن هذه التَعليمة في هذه الحالة تستَغرق فترة زَمنية مقدارها 1 مايكرو ثانية ، أما إذا تحقق الشرط أي عندما يكون البت المراد فحصة يُحمل القيمة صفر ، فستأخذ هذه التَعليمة دورتان ماكنة أي ان الفترة الزَمنية التي تَستغرقها التَعليمة ستكون 2 مايكروثانية.

Lateral and Control Instruction تعليمات السيطرة والثوابت

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
LITERAL AND CONTROL OPERATIONS			
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

التعليمة MOVLW MOV Lateral to W Reg.

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

```
MOVLW  K
```

حيث:

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


ملاحظة:

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

التعليمة ADDLW ADD Lateral to W Reg.

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 .

التعليمة ANDLW **AND Lateral and W Reg.**

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 علية .

ملاحظة:

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

التعليمة CALL **CALL Statement**

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 استدعاءة .

ملاحظة:

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

التعليمة CLRWDT Clear Watchdog Timer

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

التعليمة Goto **Goto Statement**

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

```
Goto  K
```

حيث:

K: يمثل عنوان موقع الذاكرة المراد القفز الية .

ملاحظة:

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

التعليمة XORLW **Exclusive OR Lateral With W Reg.**

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 أجراء عملية Ex-OR المنطقية علية .

ملاحظة:

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

التعليمة IORLW Inclusive OR Lateral With W Reg.

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 أجراء عملية OR المنطقية علية .

التعليمة SUBLW **Subtract Lateral from W Reg.**

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 طرحة من مسجل العمل .W Reg .


ملاحظة:

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

التعليمة SLEEP **SLEEP Instruction**

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

```
SLEEP
```


ملاحظة:

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

التعليمة RETFIE **Return flag interrupt enable instruction**

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 المقطاعات interrupt service routine(ISR) ، 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، وتاخذ التعليمة الصيغة التالية .

```
RETFIE
```


ملاحظة:

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

التعليمة RETURN **Return Instruction**

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 ،

التعليمة RETLW **Return with lateral to W Reg.**

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 نقلة 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 ،

التعليمات على مستوى البايت Byte Orientation Operations

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

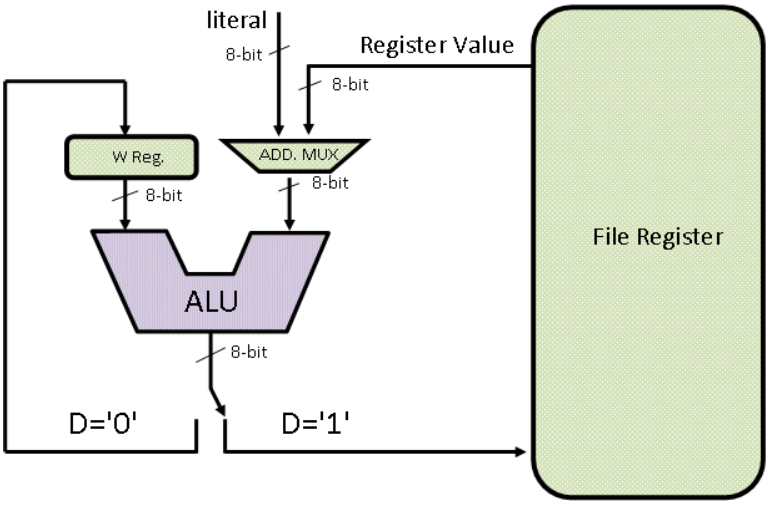


Figure 5.1

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

Mnemonic, Operands		Description	Cycles
BYTE-ORIENTED FILE REGISTER OPERATIONS			
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

التعليمة ADDWF **ADD W Reg. to flag**

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

```
ADDWF  f, d
```

حيث:

f: يمثل المسجل المراد جمعة مع مسجل العمل ،

d: تاخذ 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 بأضافة 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 Reg. وقيمة المسجل 0x0C عن طريق التعليمة ADDWF 0x0C,1 والقيمة واحد ليتم خزن ناتج الجمع في المسجل 0x0C،

التعليمة SUBWF **Subtract W Reg. from flag**

تقوم هذه التعليمة بطرح قيمة مسجل العمل W Reg. من مسجل أخر f ، تاخذ هذه التعليمة الصيغة التالية ،

```
SUBWF   f, d
```

حيث:

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

d: تاخذ 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 ،

التعليمة SWAPF **Swap W Reg. With Flag**

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

```
SUBWF   f, d
```

حيث:

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

d: تاخذ 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 ،

التعليمة ANDWF **AND W Reg. with flag**

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

```
ANDWF   f, d
```

حيث:

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

d: تاخذ قيمتان اما 0 او 1 ، اذا 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 ،

التعليمة XORWF **EX-OR W with flag**

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

```
XORWF   f, d
```

حيث:

f: يمثل المسجل المراد اجراء عملية AND المنطقية علية 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 ،

التعليمة IORWF **Inclusive OR W with flag**

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

```
IORWF   f, d
```

حيث:

f: يمثل المسجل المراد اجراء عملية OR المنطقية علية 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 ،

التعليمة CLRF **Clear Flag**

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

```
CLRF  f
```

حيث:

f: يمثل المسجل المراد تصفيرة .

ملاحظة:

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

التعليمة NOP **No Operation Instruction**

لا تقوم هذه التعليمة بأي وضيفة ، تستخدم عادة في عمليات التاخير Delay ، حيث أن هذه التعليمة تأخذ دورة ماكنة واحدة لذلك تستغرق هذه التعليمة فتره زمنية مقدارها 1 مايكروثانية،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 مايكروثانية بدون تأدية أي وضيفة ، اذا المحصلة الكلية لهذة التعليمات الثلاثة 3 مايكروثانية

التعليمة CLRW **Clear Working Register**

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 ،

التعليمة COMF **Compliment Flag Instruction**

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 اذا 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 ،

التعليمة DECF **Decrement Flag**

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

```
DECF  f,d
```

حيث:

f: يمثل المسجل المراد أنقاص قيمتة بواحد،

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

تعليمة INCF **Increment Flag**

تقوم هذه التعليمة بزيادة قيمة مسجل f بالقيمة واحد ، وتأخذ التعليمة الصيغة التالي

```
INCF f,d
```

حيث:

f: يمثل المسجل المراد زيادة قيمتة بواحد،

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

تعليمة RRF **Rotate Right Through Carry**

تقوم هذه التعليمة بأزاحة Shift ودوران Rotate محتويات مسجل f باتجاة 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

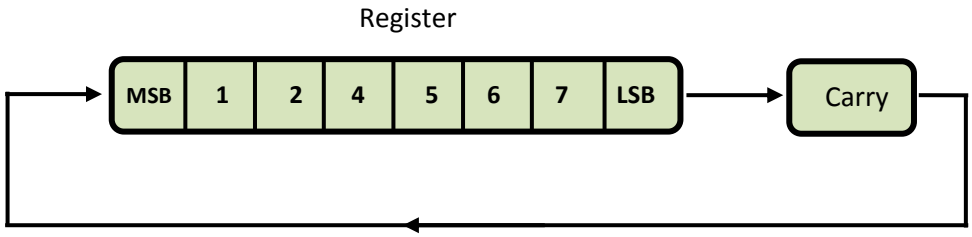


figure 5.2

تأخذ التعليمة الصيغة التالية ،

```
RRF f,d
```

حيث:

f: يمثل المسجل المراد دوران ،

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

تعليمة RLF **Rotate Left Through Carry**

تقوم هذه التعليمة بأزاحة Shift ودوران Rotate محتويات مسجل f باتجاة اليسار ، وعند خروج البت الاكثر 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

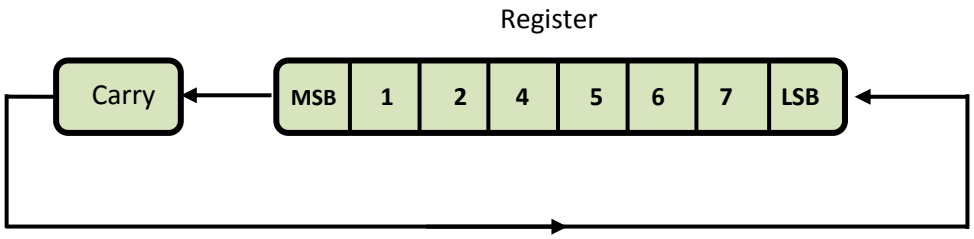


figure 5.3

تأخذ التعليمة الصيغة التالية ،

```
RLF f,d
```

حيث:

f: يمثل المسجل المراد دوران محتوياتة باتجاة اليسار ،

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

تعليمة MOVF **Move Flag**

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

```
MOVF  f,d
```

حيث:

f: يمثل المسجل المراد أجراء العملية علية،

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

مثال 5.2

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

الحل:

```
MOVF  0x0E,1
```

تعليمة MOVWF **Move W Reg. to Flag**

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 الية،

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

تعليمة INCFSZ **Increment Flag Skip if Zero**

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

```
INCFSZ f,d
```

حيث:

f: يمثل المسجل أجراء العملية علية،

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 التعليمة دورة ماكنة واحدة ، أذا كانت قيمة المسجل بعد الزيادة اكبر من صفر أي لا تتم عملية القفز Jump ، تستغرق التعليمة 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 التعليمة فترة زمنية مقدارها 2 مايكروثانية

مثال 5.3

تتبع trace التعليمات التالية ؟

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

الحل:

تقوم أول تعليمتان بنقل القيمة ffh الى مسجل العمل W عن طريق تعليمة MOVLW 0xFF وذلك بغية نقلة الى مسجل الاغراض العامة 0Ch عن طريق تعليمة MOVWF 0x0C، ثم تأتي التعليمة الثالثة التعليمةل INCFSZ لتقوم بزيادة محتويات المسجل 0Ch بواحد بما أن المسجل 0Ch يحتوي ألان على القيمة FFh وعند زيادتة بواحد سيتم تصفير المسجل وذلك لان 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 ألان ، ستقوم التعليمة INCFSZ بالقفز الى التعليمة الخامسة INCF 0x0C,1 لتقوم بزيادة محتويات المسجل 0Ch بواحد لتصبح محتويات المسجل 0Ch هي القيمة 1 .

تعليمة DECFSZ **Decrement Flag Skip if Zero**

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

```
DECFSZ  f,d
```

حيث:

f: يمثل المسجل المراد أجراء العملية علية،

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

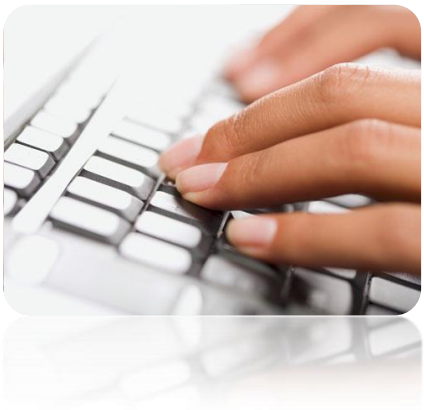
مثال 4.1

تتبع trace التعليمات التالية ؟

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

الحل:

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



Programming Concept
مبادء البَرمجة

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 مبادء البرمجة Programming Concept .

البَرمجة Programming

لقد تعرفنا على البرنامج وقلنا انة 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 Fig 6.1

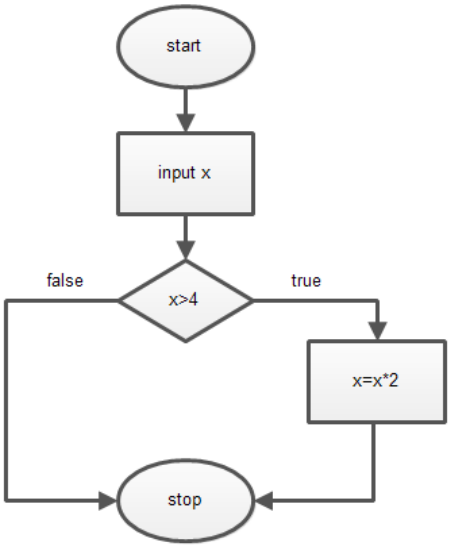


Figure 6.1

لغة التجميع 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 المجمِّع 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 انظمة الزمن الحقيقي Real time system .

و تتكون اسطر برامج لغة التجميع من ثلاثة أجزاء:

- العلامة (Label) وهو ما يتم به الإشارة لسطر ما ليتم استدعاءه 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 مميزاتها

- ☒ تستهلك اقل مساحة
- ☒ السيطرة على البرنامج هي للمبرمج ولسيت للمجمع
- ☒ اللغة التي تخاطب المكونات المادية مباشرتاً ، فتسهل عملية ادارة الأجهزة.

قبل كتابة التعليمات هناك توجيهات directive يجب كتابتها مع البرنامج هذة التوجات توجة المعالج CPU اين هي بداية البرنامج واين هي نهاية البرنامج ، كما توجة المعالج CPU الى ترجمة البرنامج الى تعليمات يفهمها المسيطر المقصود خزن البرنامج فية .

Program Structure هيكل البرنامج

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 لتوجية 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 هذة الحلقة نكتب التعليمات التي ستتكرر بشكل مستمر ، اما التعليمة الاخيرة End فهي تدل على نهاية البرنامج .

PORTA &PORTB Configuration تهيئة المنافذ

يتكون المسيطر الدقيق pic16f84a من 13 منفذ ولكي نستخدم هذة 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 كالأتي

```
 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

التعليـــــــمة 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.

عملية الكتابة Writing Operations

عملية الكتابة Writing الى المنفذ او الأخراج output عملية مهمة وذلك لأخراج البيانات الى العالم الخارجي ، وذلك للاستفاده 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 الرئيسة main function وذلك برفع البت الثالث من porta بالقيمة واحد وذلك عن طريق التعليـــــــمة bsf 0x05,2 ، تذكر ان عنوان porta هو 0x05 في file register ، اذا وصلت دايود باعث للضوء LED ستجدة قد اشتغل.

عملية القراءة Reading Operations

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

Programming Concept مبادء البرمجة

توفر لغة البرمجة مجموعة من اللبنات 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 المسيطرات الدقيقة ، المتغيرات هي مواقع خزنية داخل ذاكرة المسيطر الصغري Pic16f84a لهذة المواقع الخزنية أسم محدد ، ويمكن تعريف المتغيرات في المسيطر الصغري على غرار باقي لغات البرمجة ، وتكون الصيغة كألاتي

```
Name_Of_ variable   Set    Value
```

أمثلة على المتغيرات

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

ويمكن كتابتة المتغيرات بالصيغة التالية،

```
Variable    Name_Of_Constant =    Value
```

أمثلة على المتغيرات

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

Constant الثوابت

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

```
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 التعليقات

وهي عبارة عن نصوص غير قابلة للترجمه ولا يفسرها المعالج تستخدم للارشاد 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 هذة التعليمة بأسناد نص text الى قيمة Value ،وتأخذ الصيغة التالية

```
#DEFINE TEXT  VALUE

Examble:
#DEFINE  Input   1
#DEFINE Output  0
```

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

مثال ٦.١
سنتعلم في هذا المثال كيفية أجراء عملية الضرب في المسيطر الصغري ، لنفرض لدينا متغيران وهما X,Y والمتغير Z ونريد أجراء العملية الاتية

```
Z := X * Y
```

عملية الضرب ببساطة هي عملية جمع ، أي أن عملية الضرب X*Y لنفرض نريد أجراء 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 هذة الحالة تصلح فقط عندما تكون الاعداد صحيحية والناتج اقل من 255 أي أن عملية الضرب يكون ناتجها 8-bit، هناك خوازميات 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 تكرار تكرر بقدر 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

```
#DEFINE  int1      0x0D
#DEFINE  int2      0x0E
#DEFINE  product   0x0F

         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
```

التعليمة ORG

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

```
Start org 0×00
 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 قيمتة first+1 والثابت third سيكون second+1 ، ويمكن زيادة القيمة باكثر من واحد ، لاحظ المثال التالي.

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

قمنا 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 قيمتة 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
```

Suproutine الاجراءات

الاجرائات Subroutine او البرامج الفرعية عبارة عن برامج-او اجزاء برامج-ثانوية 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.

وتكون بالصيغة التالية

```
 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 الاخرين 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
```

branching Instruction تعليمات التفرع

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

if statement تعلمية 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 يواجة التعليمة Goto end_if ليقفز الى نهاية تركيبة 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 Val set 3 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 التعليمة While لتكرار مجموعة من التعليمات ، ويمكن استخدم تعليمة التكرار في لغة التجميع بالاسلوب الأتي

```
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 تعليمات 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 وفحصة الى ان يتم تصفير المسجل 0x0E وذلك عن طريق التعليمة Decfsz .

ويمكن ايضاً استخدام الصيغة التالية لتنفيذ حلقة التكرار

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

مثال 6.3:

اكتب برنامج لجعل LED يومض Flashing بين فترة واخرى بأستخدام 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

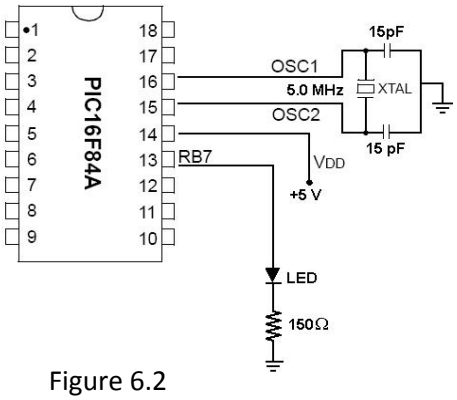


Figure 6.2

قمنا بكتابة أجراء Subroutine اسمة Delay يحتوي هذا الاجراء على حلقة تكرار ، قمنا في بداية الاجراء بتحميل القيمة 0xff الى مسجل الاغراض العامة 0x0E وقمنا بطرح هذة المسجل ومقارنة مع القيمة صفر عن طريق التعليمة Decfsz فاذا كانت قيمة المسجل 0x0E تساوي صفر يخرج المسيطر من الاجراء delay والا تتكر العملية الى أن تصبح قيمة المسجل 0x0E تساوي صفر ،أما الدالة الرئيسية main function فهي تحتوي على تعريف المنفذ protb والذي اصبح منفذ أخراج ، وقمنا بعملية أخراج أشارة الى البت السابع من portb عن طريق التعليمة bsf portb,7 ثم تلالها تأخير delay ثم تلالها 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
  START
               BSF portb,7
               CALL  Delay
               BCF portb,7
               CALL Delay
               GOTO START
  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 ٤ او اصغر من العدد ٤ او تساوي العدد ٤.

الحل:

الفكرة هي بطرح العد المطلوب فحصة او مقارنتة من العدد ٤ 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 فأن carry flag=1 اما اذا كان العدد الاول اصغر من العدد الثاني فأن carry flag=0 ، راجع الوحدة الاولى الطرح بأستخدام المتمم الثاني ،ويتم برمجة العملية كألاتي .

```
 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
```

العنونة المباشرة Direct Addressing

يقصد بالعنونة المباشرة هي الوصول Accessing الى المسجل مباشرتاً ، كما هو مبين في المثال الأتي

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

goto main
end
```

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

العنونة الغير المباشرة UnDirect Addressing

يقصد بالعنونة الغير المباشرة هي الوصول Accessing الى المسجل ما يتم بطريقة غير مباشرة ، وذلك بأستخدام المسجل FSR ،كما هو مبين في المثال الاتي ،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 الطريق الفحص المستمر للمتحسس السفلي ، عندما تصبح قيمة المتحسس السفلي صفر فهذا يدل على ان الخزان فارغ ، نقوم بتشغل بفتح الصمام ، وننتظر الى ان تصبح قيمة المتحسس العلوي تساوي واحد والتي تدل على ان الخزان قد ملئ وتعاد العملية ، الشكل Fig 6.3 يبين رسم توضيحي للعملية .

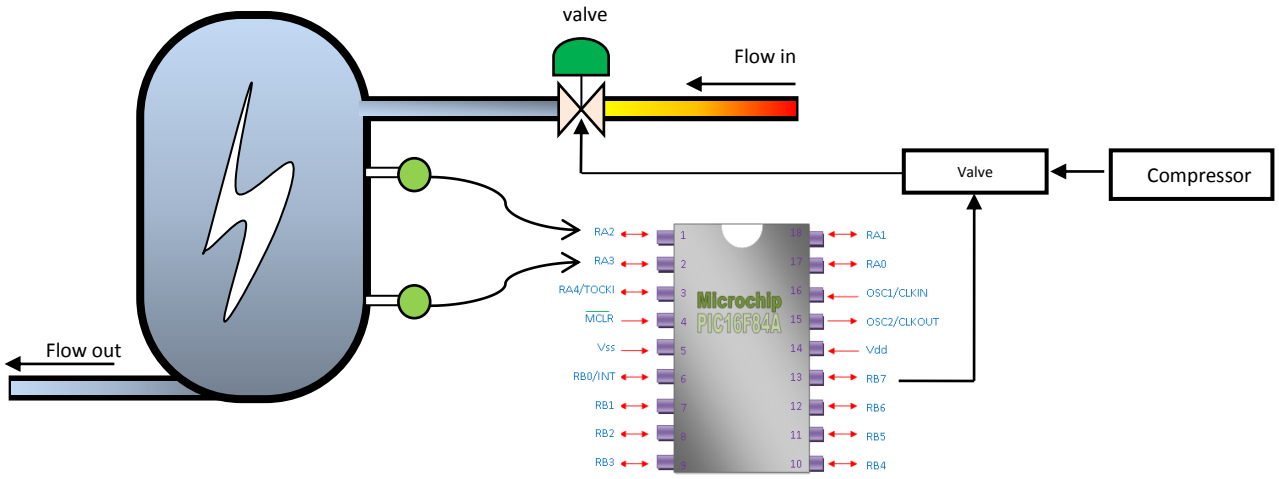


Figure 6.3

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

 end
```

Introduction to interrupt مُقَدمة في المُقاطعات

إذا أردت ان تبرمج لَوحة مفاتيح تُربط الى المُسيطر الصغري ، بالطَبع سوفَ تَقوم بفِحص Scan ، كُل طَرف من أطراف المُسيطر الدَقيق الذي وصل الى لَوحة المفاتيح ، للتَعرف على الزر المَضغوط ، المِثال التالي يوَضح 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
```

هَذا النَوع من الفَحص Scan، يُسمى الأنتخاب polling ،السؤال الذي يَتبادر الى الذِهن ؟!!

كيفَ سَيتم تَنفيذ بَرنامج أخر مَخزون داخل شَريحة المُسيطر الصغري ، في الوَقت الذي نَجري فية عَملية الأنتخاب Polling ، طبعا مُحال لأننا داخل حلقة مغلقة close loop لفَحص أطراف المُسيطر المَربوطة الى لَوحة ألمفاتيح وعند تحسس أي طرف نقوم بتأدية المُهمة المَطلوبة ، أذن نستنتج من ذلك،

- المُسيطر لا يَستطيع تنفيذ اكثر من مهمة ، واذا حاولت وتمكنت، فستحاول تنفيذ المهمة ثم ترجع لتفحَص الأطراف Scanning بشَكل متكرر، تخيل لو أن المسيطر ينفذ المهمة الأخرى ، وفي نفس الوقت ضغط شخص على لوحة ألمفاتيح بِسُرعة ثم تركَ release ألمفتاح ، عندها المُسيطر عندما يعود لفَحص الأطراف لا يَجد أي طرف يشير الى وجود زر مضغوط ؟!!

- ممكن إن المُسيطر لا يتحَسس أو يفقد الأشارة في عملية الأنتخاب Polling..

interrupt philosophy فلسلفة المقاطعة

معنى المقاطعة في اللُغة وهو مقاطعة شيء أو عَمل ما لأداء عمل اكثر أهمية ثمَ العَودة الى العَمل السابق .. وهو فعلاً ما يحدث في المُسيطر الصغري حيث أن المقاطعة interrupt ، تقاطع عمل المسيطر الحالي لتأدية مهمة أخرى ،بعد إتمام المهمة يعود المسيطر لوضعة الطبيعي.

Mechanism of interrupt

الية عمل المقاطعات

المقاطعة تقريباً مشابهة لعَمل الأجراءات Subroutine ، حيث يوجد برنامج فرعي يستدعى عند حدوث المقاطعة، هذا البَرنامج الفَرعي يُسمى برنامج خدمة المقاطعات ISR(interrupt service routine) ، يتم استدعاءة بطريقتين

- ☒ من داخل البَرنامج الرئيسي Main program باستخدام تَعليمات خاصة وهو ما يسمى بـ software interrup.
- ☒ عن طَريق أطراف المسيطر الدقيق وهو مايسمى بـ Hardware interrupt

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

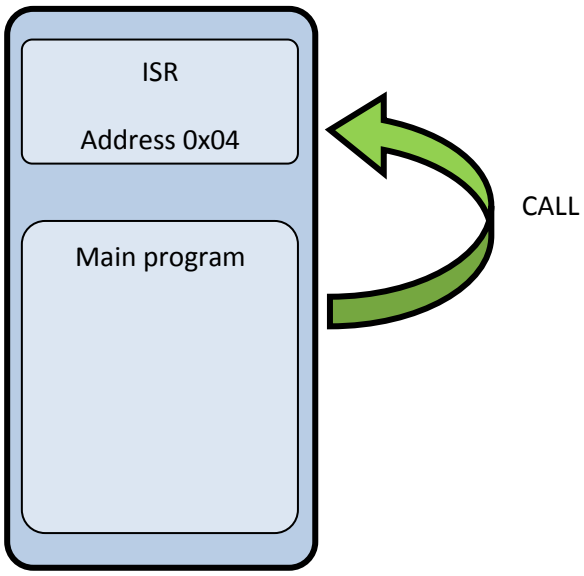


Figure 6.5

حسناً ... دعنى نلقي الضوء على موضوع استدعاء خدمة ISR عن طريق أطراف المسيطر الصغري ، عند النَظر الى المسيطر الصغري عند الطرف RB0 ستَجد ان لة وَضيفة إضافية ألتي هي INT ،أي انة ممكن ان يُستخدم هَذا 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 تَحديد أي طرف سيستخدم لخدمة المقاطعة واي طرف سيستخدم كخدمة I/O.

لتفعيل خدمة المقاطعة عند الطرف RB0 ، نتبع الخطوات التالية

١- نمكن خدمة المقاطعات العامة GIE من المسجل INTCON كالأتي

```
BSF 0x0b,7
```

٢- نمكن خدمة المقاطعة عند الطرف RB0 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 RB0، فقط بقي شئ واحد مهم هل المقاطعة ستتم عندما ؟!!

- الطرف RB0 يتحول من المنطق الواحد الى المنطق الصفر falling edge
- الطرف RB0 يتحول من المنطق الصفر الى المنطق الواحد rising edge

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

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

علم المقاطعة Interrupt flag

يوجد هذا العلم في السجل INTCON في الخانة bit(1) ، وضيفة 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 خروجة من خدمة ISR فأن 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 ونفصلة عن البرنامج الرئيسي كالأتي ،

```
        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 المقاطة

                             ; 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 أذا كان هناك أشارة عند الطرف Rb0 ، يتم استدعاء برنامجة خدمة المقاطعات ISR بشكل ذاتي وذلك لكي يرفع البت الاول من المسجل porta الى القيم واحد .

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

ORG   0000h    ; يبدء المسيطر التفيذ 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

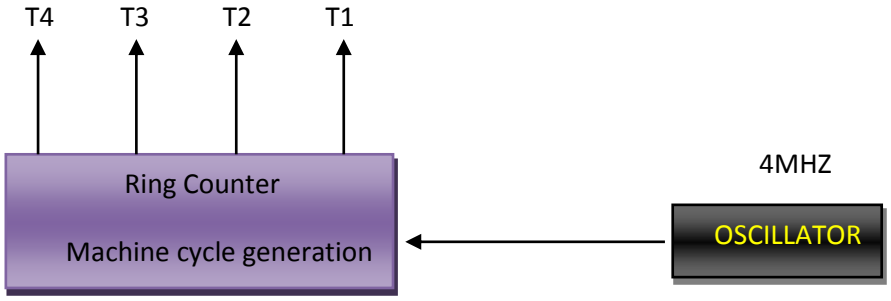


Figure 6.5

يتم خلال الاربع دورات،

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

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

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

كلنا نعلم ان التردد مقلوب الزمن أي

t=1/4MHZ=0.25 micro second

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

Instruction cycle=0.25 mico second * 4 pulse = 1 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 بامكان ان نحسب الوقت الذي يستغرقة 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 انة بامكانك 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 TMR0 والذي يمكن أستخدامة كمؤقت أو عداد ، عند أستخدام هذا المؤقت كعداد Counter ، يقوم هذا المؤقت بالزيادة عند كل نبضبة تأتي الى الطرف TOCK1 ، أما عند أستخدامة 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 فية بنسبة تعتمد على تردد الدخل بالاضافة الى مقسمم التردد ، ماذا يحدث فعلاً ، عندما يصل مسجل المؤقت الى 256 وهي اقصى قيمة لة لانة مسجل ذو 8-bit وعند زيادة هذة القيمة بواحد يصبح المسجل يحمل القيمة 0 مشيراً بذلك الى حدوث حالة طفحان ، هذة الحالة تستدعى خدمة المقاطعة عند تفعيل مقاطعة المؤقت ، لتنفذ برنامج خدمة المقاطعات ISR ، ما يهما الان هي العلاقة التي من خلالها سنعرف زمن استدعاء خدمة المقاطعة وهي

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

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

حيثُ أن قيمة tmr0 تمثل قيمة مسجل المؤقت ، ومقسم التردد prescaler تمثل نسبة التقسيم والتي هي موضحه 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 .

timer frequency =(4 Mhz/4)× 1/8 =0.125 Mhz

Overflow time = 1/0.125 Mhz × (256-100)=1024µs

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

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

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

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

حيثُ أن الزمن T هو مقلوب تردد المصدر

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

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

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

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 المسجدل Tmr0 بالقيم السالبة .

Time to Overflow (µs)	Prescaler							
	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
1,000	–	6	131	193	225	240	248	252
5,000	–	–	–	–	100	178	77	236
10,000	–	–	–	–	–	100	178	217
20,000	–	–	–	–	–	–	100	178
30,000	–	–	–	–	–	–	–	139
40,000	–	–	–	–	–	–	–	100
50,000	–	–	–	–	–	–	–	60
60,000	–	–	–	–	–	–	–	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 ،لاكن أعلم ان prescaler يتشارك مع شيئين whatchdog و Timer ويتم تحديد أي العملية المطلوبة عن طريق الطرف PSA(Pin assignment) حيث

PSA =0 Timer prescaler

PSA =1 Watchdog prescaler

مؤقت الحراسة WATCHDOG TIMERS (WDT)

سؤال يطرح نفسة؟!!

ماذا يحدث لو عَلق Stuck المُسيطر الدَقيق، هل هناك زر restart لاعادة تشغل المسيطر الدقيق ، طبعاً لا ،فماذا نفعل لو كان المسيطر الدقيق في موقع عمل field ودخل في حالة العلق Stuck ، السطور التالية سنتعرف بها على كيفية التغلب على هذه الحالة ،،

كيف يعلم المسيطر الدقيق انة علق Stuck ؟!!

نحن نعلم أن دور الماكنة تبدء بعملية الجَلب Fetch التي يتم فيها جَلب التَعليمة من الَذاكرة الى سِجل التعليمة Instruction register ، حالة العلق Stuck هو ان وحدة المعالجة المركزية CPU لم تستلم تَعليمة من الذاكرة مما يسبب خلل في سير البرنامج ، وهو فعلاً ما يحدث في حاسباتنا الشخصية ، فنظطر الى إعادة تشغيل الحاسوب ،

لقد قلنا أن المسيطر الدقيق لا يحتوي على زر اعادة التشغيل Restart ، الفكرة التي ابتكروها مهندسي الحاسبات هو وضع مؤقت ، يبدء هذا المؤقت عند دورة الجلب بالعد Fetch cycle، أي عندما يقوم المعالج بارسال طلب التعليمة الى الذاكرة ، اذا أنتهت قيمة المؤقت ولم يَستلم المعالج اي تعليمة من الذاكرة الى مسجل التعليمة IR ، يقوم المسيطر الدقيق باعادة تشغيل نفسة ذاتياً ، هذا المؤقت يسمى Watch Dog Timer ويختصر WDT ،

، في الحقيقة مؤقت الحراسة هو عبارة عن عَداد 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 يبدء مؤقت الحراسة بالعد ؟!!

يبدء مؤقت الحراسة بالعد خلال مرحلة الجلب fetch cycle من دورة الماكنة machine cycle، إذا لم يستلم CPU أي تعليمة خلال فترة الجلب وقد وصل مؤقت الحراسة بالعد الى القيمة 0xff ،وقد طفح المؤقت أي تحول من القيمة 0xff الى القيمة 0x00 ،عندها سيتم عمل reset لشريحة المسيطر الدقيق ..

السؤال الذي يتبادر الى الذهن ؟!!

- ✓ اذا طلبت العد من الصفر الى العشرة من شخصين طبعاً سرعة العد ستختلف فمثلا 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، وهذة القيمة قابلة للتغير باستخدام سجل option ..
- ✓ اخبار المسيطر الدقيق بتفعيل enable او ابطال disable خدمة مؤقت الحراسة WDT

watchdog timer preparation

خَطَوات تَهيئة مؤقت الحِراسة

يَجب تَحديد الفَترة الزَمنية ألتي يعُد فيها مؤقت الحِراسة ، 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 الوضيفة التابعة لهذة 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،

```
CLRWDT   ;Clear WatchDog Timer
```

أ ذا 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.

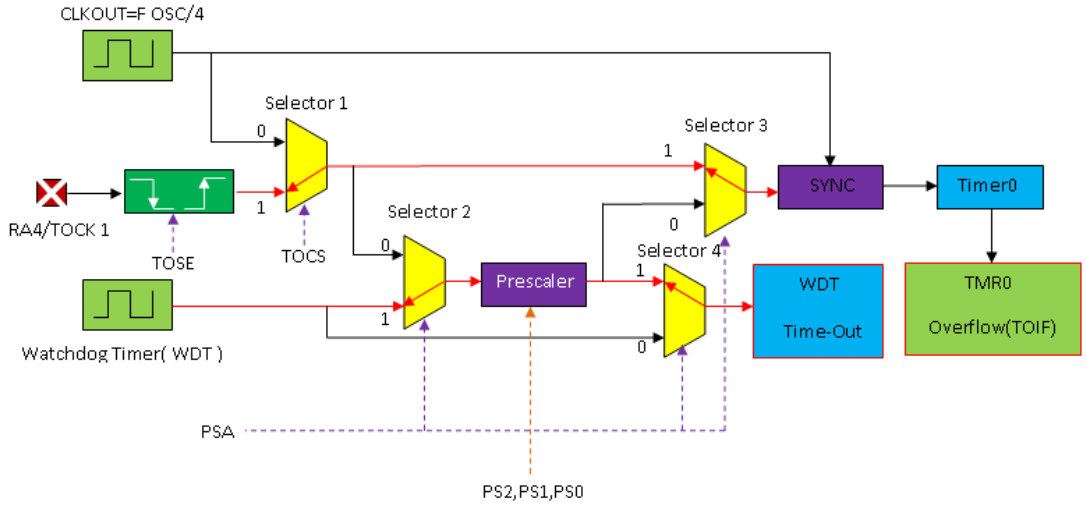
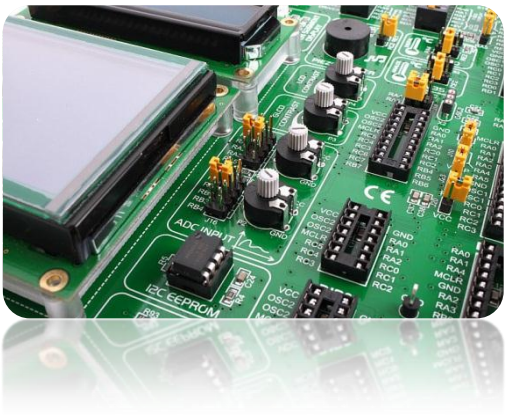


Figure 6.1

عندما ترفع قيمة PSA الى الواحد ، هذا يؤدي الى أن كل من selector 4 , selector 2 التان هما عبارة عن منتقيَ بيانات multiplexer ، سيشيران الى البيانات القادمة من الطرف "1" من multiplexer، وهذا يؤدي الى ان المذبذب WDT سيعبر عن طريق selector 2 ذاهبً الى المقسم Prescaler، الذي يقسم التردد بالأعتماد 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

العُدد والمحاكاة

مُقدمـــــــــة:

في هذه الوحدة سوفَ نتَعرف 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 Personal Computer (PC) ويم 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 ادناة

المبرمجة :

تتوفر في السوق أنواع مختلفة من المبرمجات ، من هذة المبرمجات عبارة عن عدة متكاملة مثل المبرمجة من شركة Microchip هذة المبرمجة مزودة بقواعد Socket لمختلف انواع PIC microcontrolrs ، وكذلك تم توصيل مفاتيح ومصابح 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

- ☒ كيبل توصيل من نوع Usb شبية بكيبل الطابعة الكيبل 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 توليدة باستخدام بيئة البرمة وذلك ليتم نقلة الى ذاكرة المسيطر الدقيق microcontroller

بعد أن تقوم بتوصيل المبرمجة الى جهاز الحاسوب عن ط، كيبل Usb ، وقمت بتنصيب التعاريف Drivers وبرنامج Microprog ، للتأكد من أن المبرمجة تم توصيلها بشكل صحيح الى جهاز الحاسوب ، قم بتشغيل برنامج Microprog واجهة البرنامج مبينة في الشـــــكل Figure 7.2 تجد في اسفل واجهة البرنامج أيقومة بشكل رمز منفذ USP عندما تكون باللون الرمادي فأنة المبرمجة تكون في حالة 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

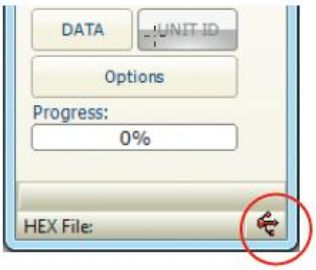


Figure 7.3

كتابة الشفرة Code وتوليد الملف hex. File

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

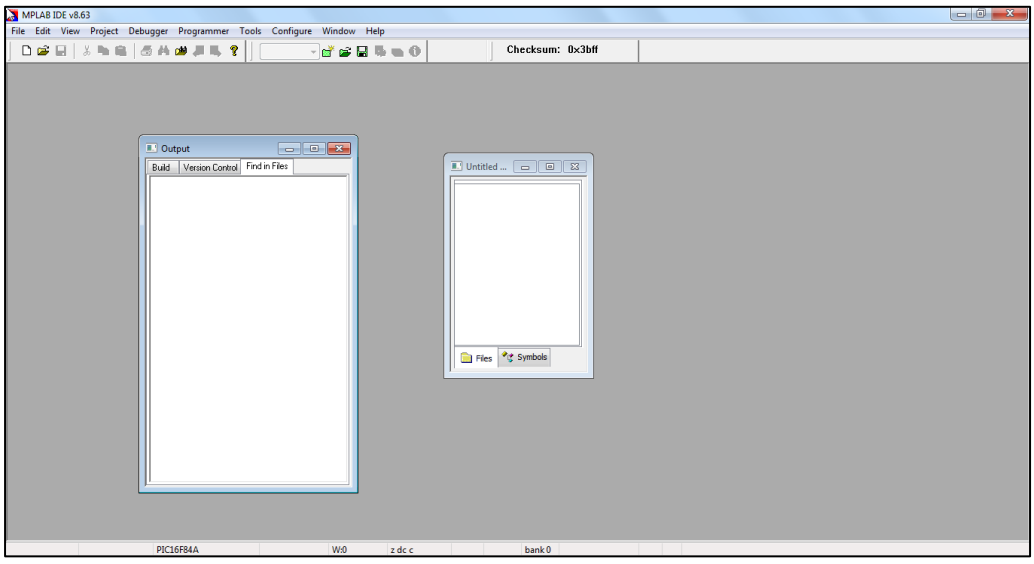


Figure 7.4

- من قائمة project أختر project wizard سنفتح النافذة المبية في الشكل Figure 7.5

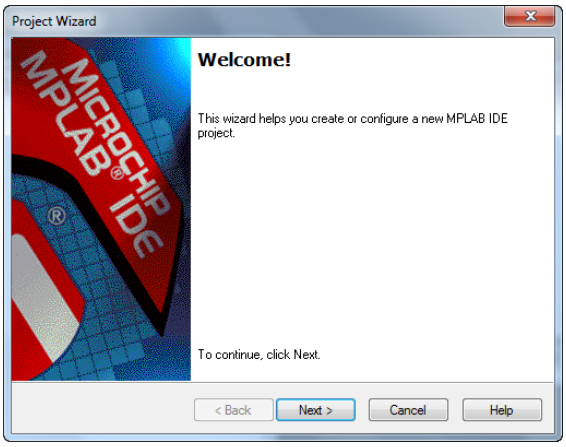


Figure 7.5

- من النافذة project wizard انقر على زر next
- تضهر نافذة جديدة أختر منها نوع المسيطر الدقيق Device type كما هو موضح في الشكل Figure 7.6

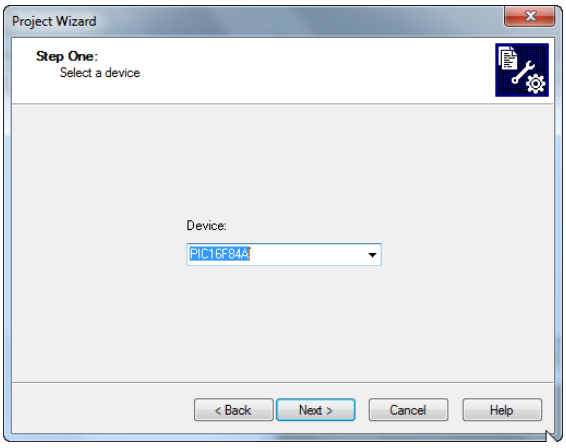


Figure 7.6

- عند النقر على زر next تضهر النافذة التالية Figure 7.7 والتي تحدد فيها مسار المترجم Assember اترك الاعدادات الافتـراصية لـــها ثم انقر على زر next

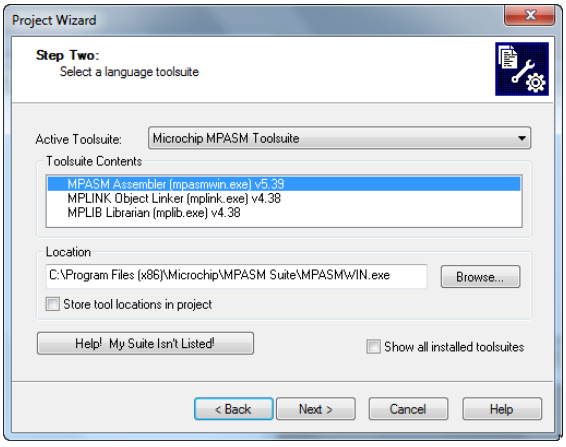


Figure 7.3

- 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

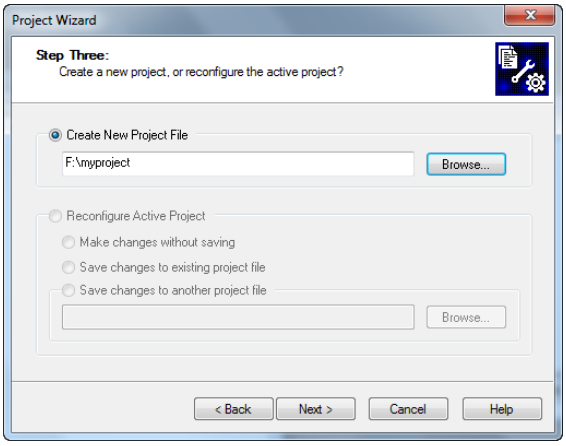


Figure 7.8

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

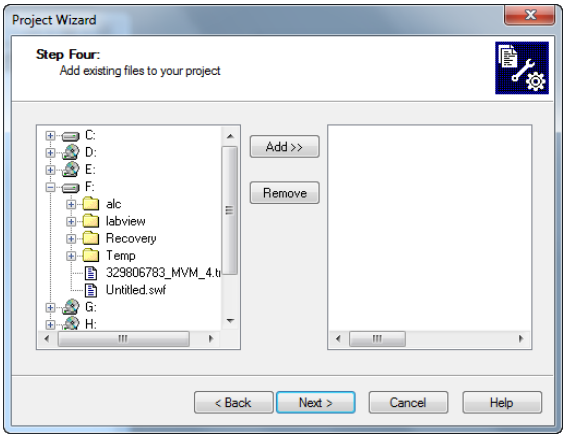


Figure 7.9

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

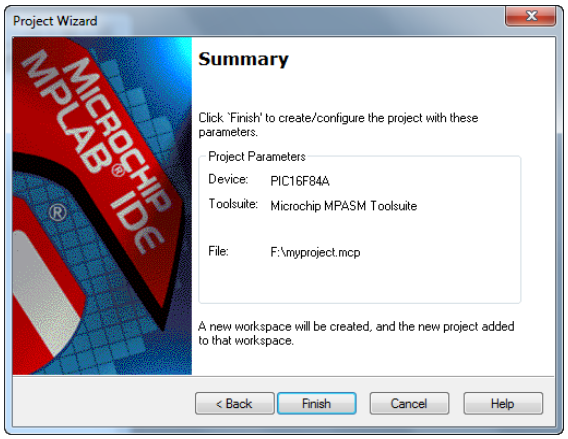


Figure 7.10

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

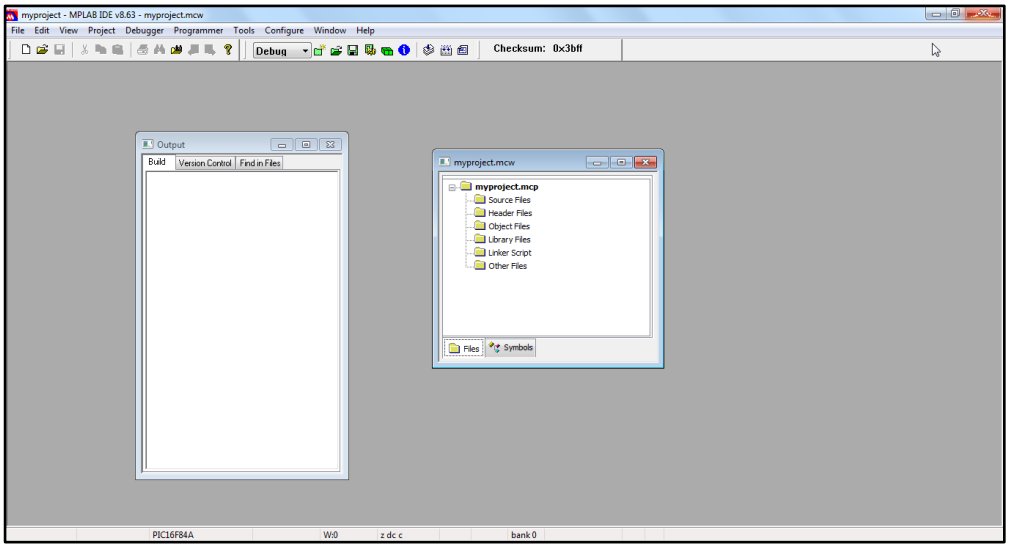


Figure 7.11

- من القائمة file أختر new ستنبثق نافذة لتحرير الاوامر Codes قم بكتابة البرنامج الذي تريد ترجمتة كما هو موضح في الشكل Figure 10.12

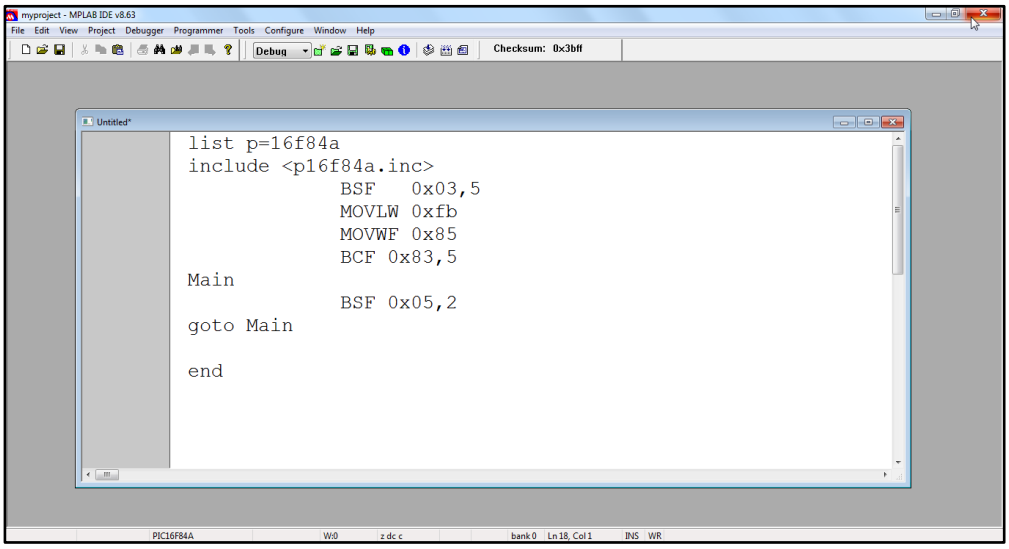


Figure 7.12

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

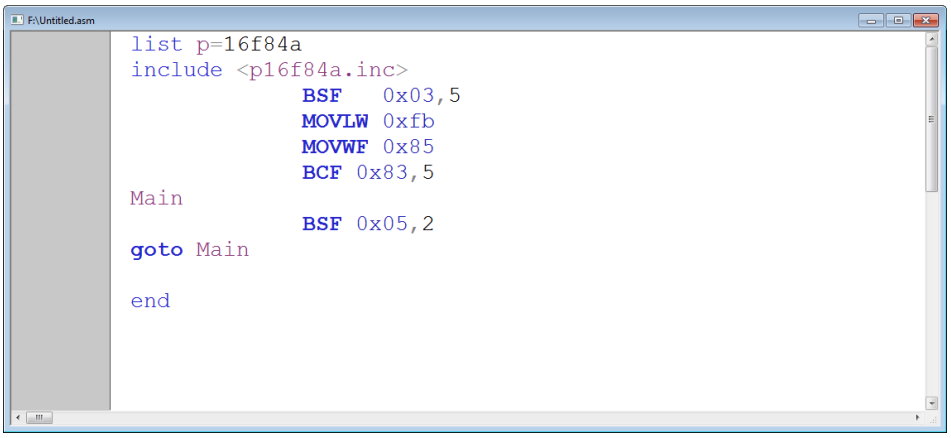


Figure 7.13

- من النافذة myproject.mcw المبينة في الشكل Figure 10.14 ، أضغط بالزر الايمن على الملف source files ستنبثق نافذة فرعية pop-up menue أختر منها Add Fliles... ستنفتح نافذة مستكشف الملفات أختر الملف الذي قمت بحفظة في الفقرة السابقة والذي هو untitled.asm

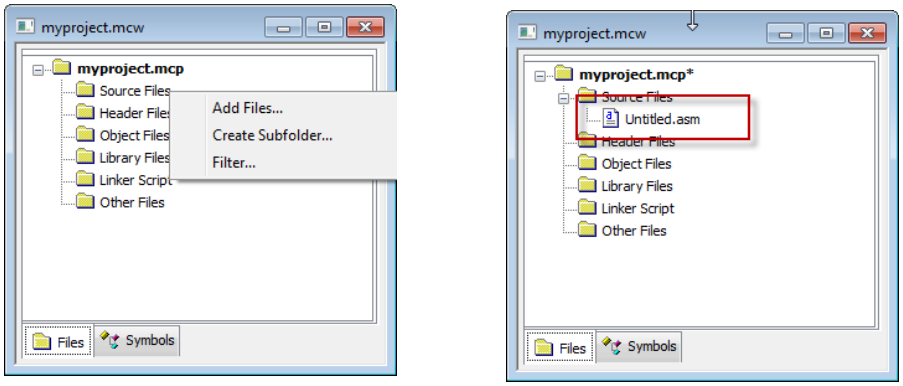
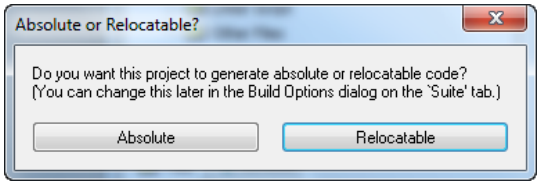


Figure 7.13

- اذهب الى القائمة project وأختر ألامر 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 للتغير في داخل الذاكرة كما هو موضح في الشكل، أختر الامر Absolute اي أننا نود أن يكون البرنامج محدد عند 0x00 من الذاكرة



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

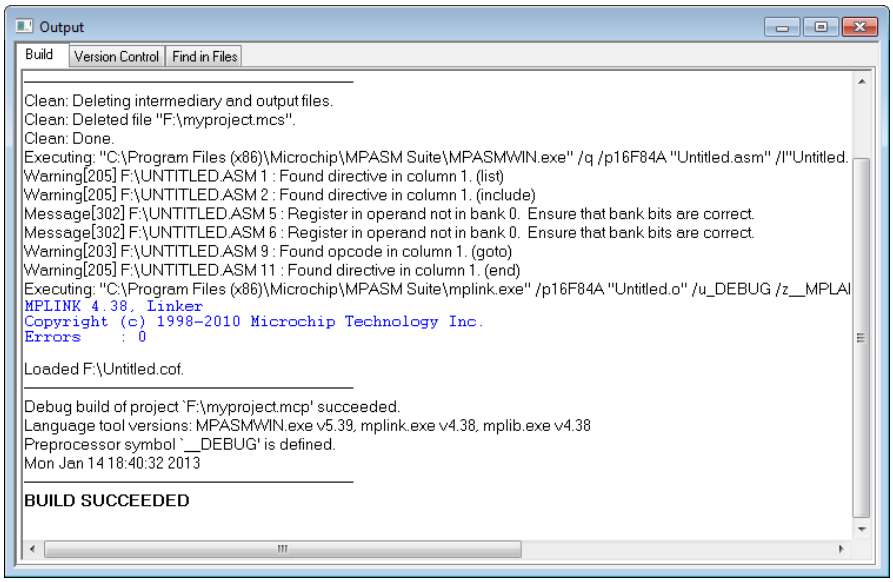


Figure 7.14

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 التالية Figure 10.15

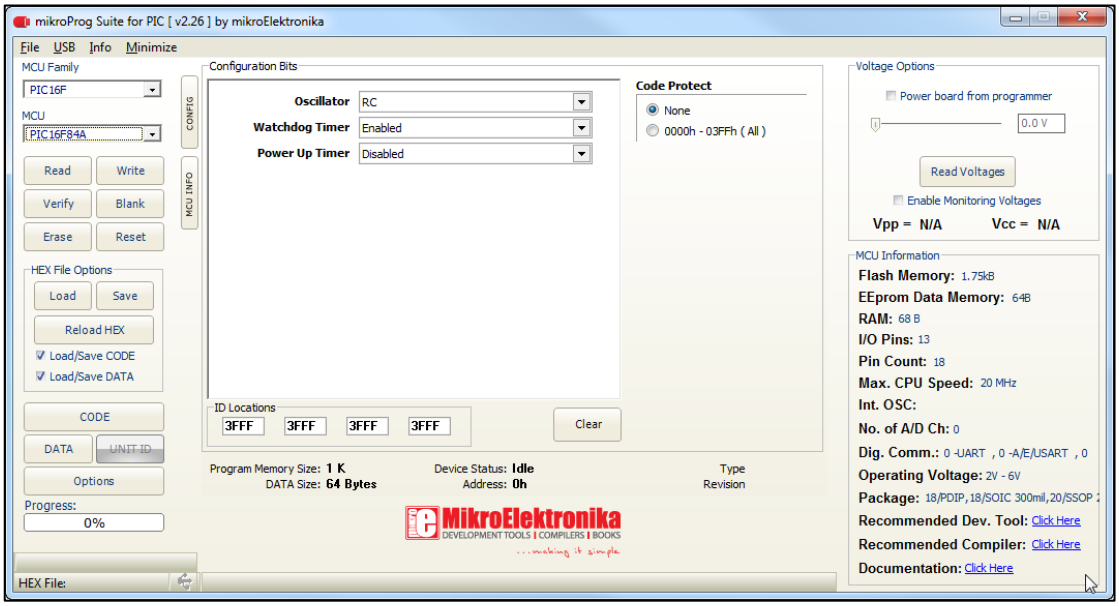


Figure 7.14

نلاحظ من الشكل أنة 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 .

المحاكاة بأستخدام برنامج proteus Simulation using proteus

يوجد هناك العديد من برامج المحاكاة المستخدمة لمحاكاة الدارات الإلكترونية ولكن ما يميز برنامج Proteus هو الدقة والسهولة في محاكاة المشاريع وخصوصا أنه يحتوي على مكاتب كثيرة لكافة القطع الإلكترونية وهو يستطيع أن يحاكي الدارات التي تحتوي على المتحكمات بسهولة جدا مًما 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 .

واجة البرنامج proteus 7

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

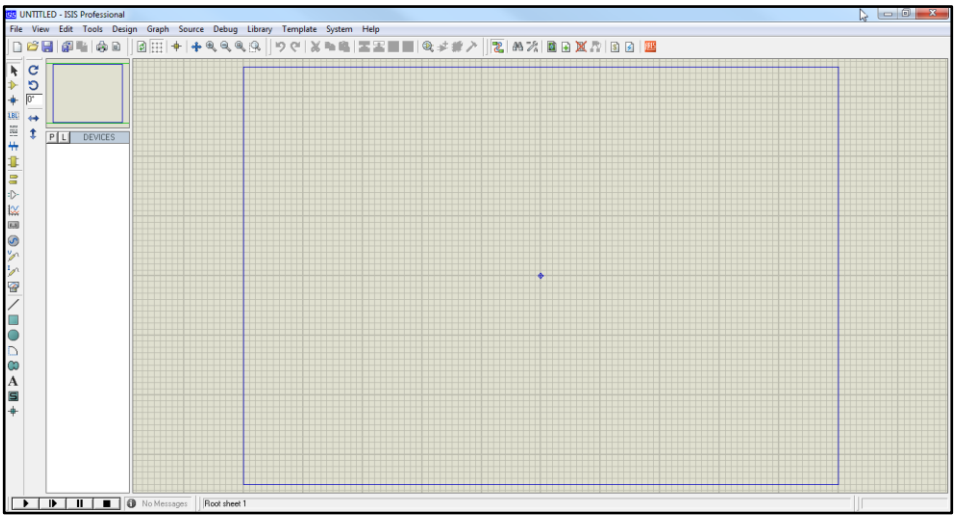


Figure 7.15

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

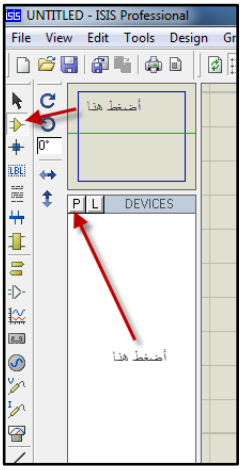


Figure 7.16

- ستنفتح نافذة المكتبة نافذة pick devices انتقي القائمة MICRO ثم انتقي المسيطر الصغري PIC16f84a بالنقر المزدوج كما هو موضح في الشكل Figure 10.17

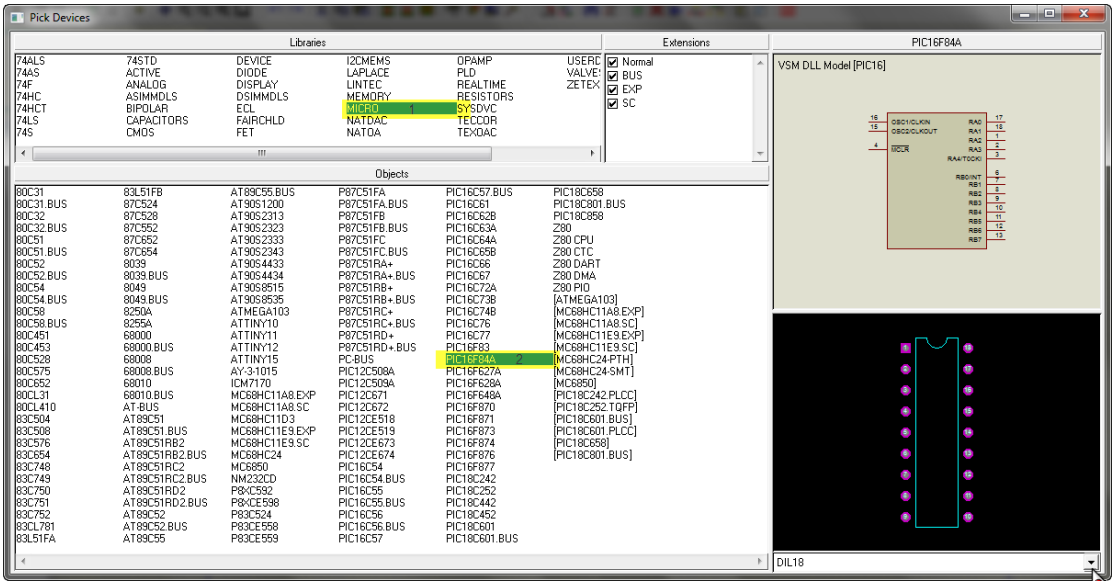


Figure 7.17

- بعد أن قمت بربط المكونات مع المسيطر الصغري كما هو موضح في الشكل Figure 10.18 أدناة

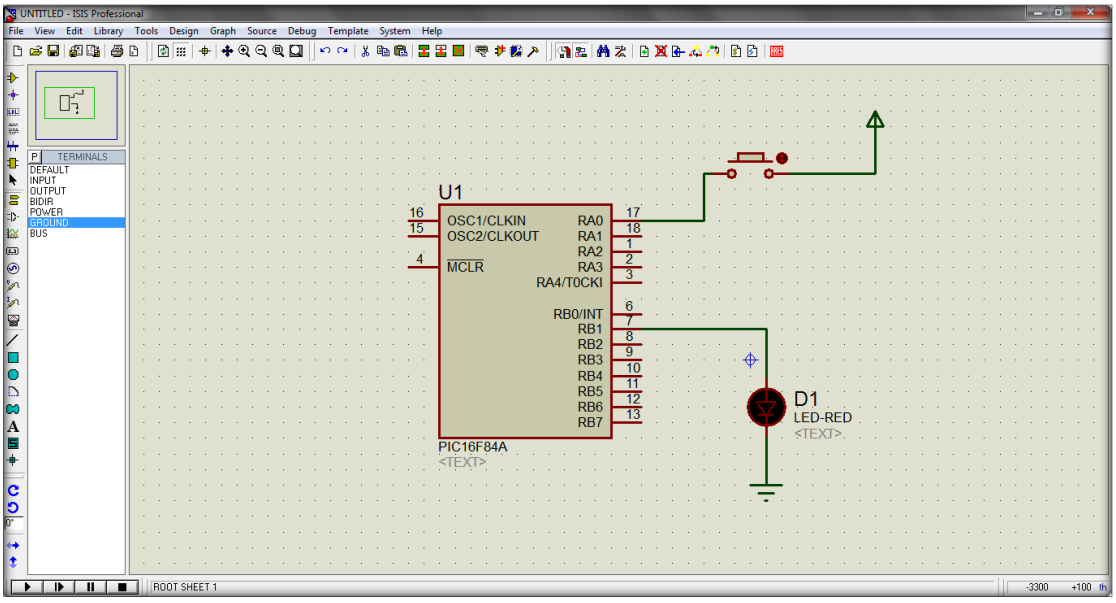


Figure 7.18

- أنقر نفراً مزدوجا على المسيطر الصغري microcontroller ستنفتح نافذة Figure 10.19 تطلب من تحديد الملف Hex file الذي قمت بتوليدة من برنامج mplab

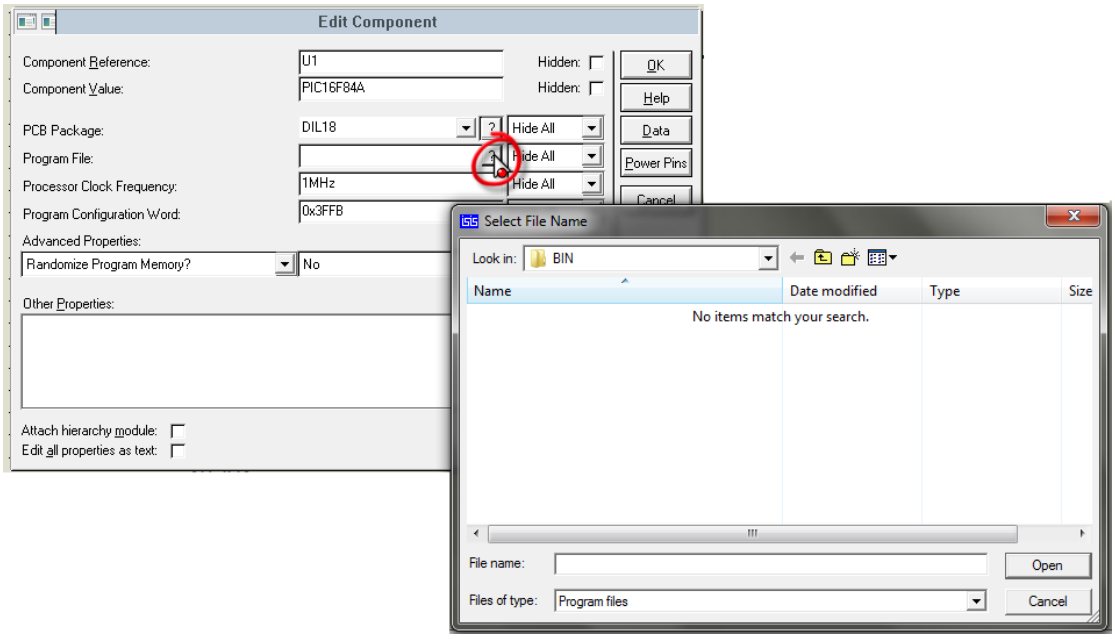


Figure 7.19

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

تم بحمد الله