# Microcontrollers

# المتحكمات الصغرية

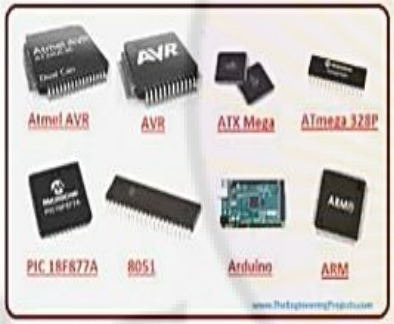
microcontroller

Google

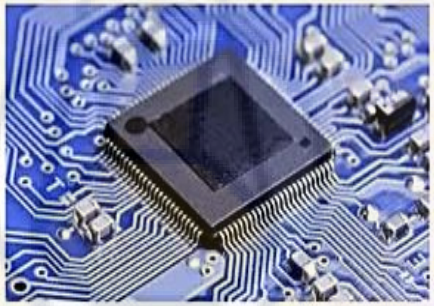
الكل | صور | فيديو | التسوّق | الكتب | المزيد | الإعدادات | أدوات | المجموعات | البحث الآمن

ıs instruments | arm cortex | pic microcontroller | microcontroller 8051 | micro controller | pic | microchip | atmega328p | block diagram | arduino uno

PIC16F887-I/PT Microcontroller, पी...
indiamart.com



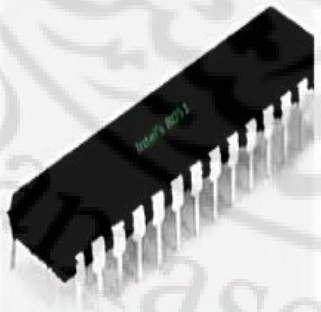

Introduction to Microcontrollers - The Engineering ...
theengineeringprojects.com



What is a Microcontroller? A Look Inside a Microcontr...
arrow.com

How Microcontrollers Work - IntervalZero
intervalzero.com

Amazon.in: Buy Embeddi...
amazon.in



Activate Windows
Go to Settings to activate Windows.

3:12 PM

# What is Microcontroller ?

| | | |
|---|---|---|
| TWI | USART | SPI |
| Hardware Multiplier | Flash | EEPROM |
| Analog Comparator | CPU CORE | SRAM |
| A/D Converter | Register File | I/O pins |
| Brown Out Detector | Analog Reference | Pull-Ups On Demand |
| Reset Circuitry | AVR | High Current Outputs |
| Programmable Watchdog | | Calibrated Oscillator |
| On-Chip Debug | | In System Programming |
| JTAG | Boundary Scan | LCD Interface |

# المتحكم المصغر

يعرف المتحكم المصغر بأنه "حاسب موضوع على شريحة "A Computer on a chip" وهو فعلاً دارة متكاملة Integrated Circuit تشكل حاسباً رقمياً بسيطاً كاملاً يحوي معظم العناصر الأساسية التي توجد في أي حاسب ويمكن أن يبدأ بتنفيذ التعليمات البرمجية المعدّة له دون الاعتماد إلا على عدد قليل من العناصر الخارجية المساندة وأحياناً دون أي عنصر آخر سواه على الإطلاق.

تتميز المتحكمات – باعتبارها حواسب صغيرة جداً – عادة ببنيتها البسيطة ومواردها التي تبدو محدودة مقارنة مع الحواسب الرقمية الشائعة لكنها غالباً ما تكون أكثر من كافية لتأدية المهمة الموكلة إليها، بالإضافة إلى أن بنيتها المبسطة تعود بالعديد من الفوائد: أهمها أن بساطة المتحكمات تمكن من وضعها بالكامل على شريحة واحدة وبتكلفة منخفضة جداً وهذان العاملان (صغر الحجم والتكلفة) من العوامل التي تجعلها مثالية لإدراجها في العديد من الأجهزة من حولنا: كالأجهزة الكهربائية المنزلية وأجهزة القياسات الرقمية وأنظمة التحكم والربط مع الحواسيب وفي المركبات وبعض ألعاب الأطفال وبعض أجزاء الحواسب الشخصية و محيطياته كالفأرة Mouse والعديد من التطبيقات المتنوعة.

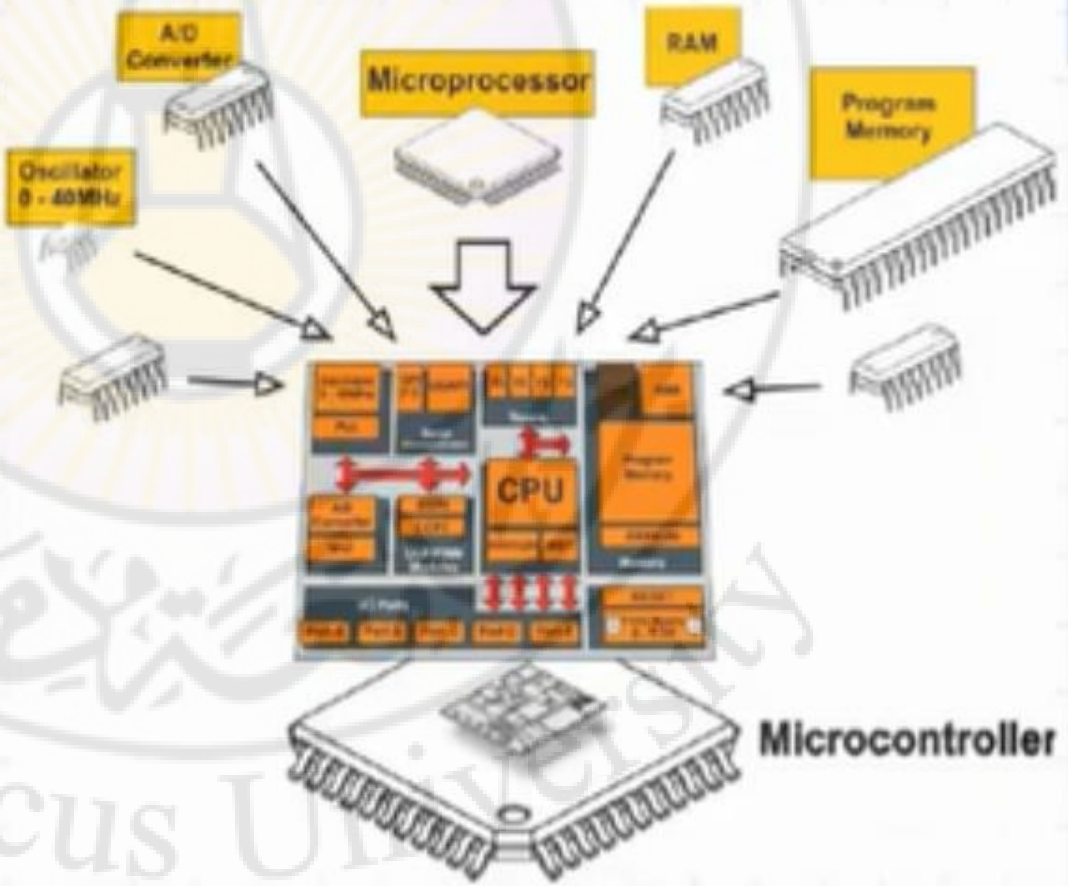
يتألف المتحكم المصغر من معالج مصغر Microprocessor بالإضافة إلى ذاكرة وبعض المحيطيات المساندة وهذا التقسيم منطقي أو وظيفي فالمتحكم فعلياً ليس سوى قطعة واحدة (شريحة دارة متكاملة) لكن هذا التقسيم يعطي رؤية أفضل عنه.

و لجعل المتحكم يقوم بإنجاز وظيفة محددة يجب أولاً شحنه ببرنامج ليقوم بتنفيذه، وهذا البرنامج يوضع في ذاكرة غير متطايرة (أي تحتفظ بمحتوياتها بشكل دائم حتى دون وجود تغذية كهربائية) وتكون هذه الذاكرة عادة أحد أنواع الذاكرة ROM (PROM, EPROM, EEPROM, Flash)، ومن هنا نجد أن المتحكم هو شريحة قابلة للبرمجة, وعادة يكتب البرنامج على حاسب شخصي PC بلغة التجميع الخاصة بالمتحكم بواسطة محرر نصوص ثم يجري تجميعه و تحويله إلى رموز تمثل تعليمات البرنامج بلغة الآلة لذلك المتحكم، كما يمكن كتابة البرنامج بلغة عالية المستوى (كـ C أو BASIC) ومن ثم استخدام مترجم خاص بذلك المتحكم يحول البرنامج من اللغة عالية المستوى المستخدمة إلى لغة الآلة الخاصة بالمتحكم.

Unit 03.pdf - Adobe Acrobat Reader DC
File Edit View Sign Window Help
Home Tools Unit 03.pdf MC Experiment 1.1... Unit 04.pdf PIC16F687 Datash...

## What is a microcontroller (µC)?

- It is a small computer on an IC containing a **processor, memory, & programmable I/O peripherals.**
- Microcontrollers are designed for **embedded applications**, in contrast to the microprocessors used in personal computers or other general purpose applications.



و بعد الحصول على ملف يمثل البرنامج الذي نود من المتحكم أن يقوم بتنفيذه ممثلاً بلغة الآلة الخاصة به يتم استخدام جهاز يدعى بالمبرمجة Programmer لنقل البرنامج من الحاسب وتخزينه داخل المتحكم، والمبرمجة توصل مع الحاسب عبر أحد منافذه كالمنفذ التسلسلي RS-232(والمسمى عادة COM) أو منفذ الطابعة LPT أو منفذ الـ USB ويرفق مع المبرمجة برنامج Softwareخاص بها يقوم بالتخاطب معها وقيادتها ويتيح إجراء العديد من المهام على ذاكرة المتحكم منها شحنها ببرامج معين أو عرض محتوياتها أو مسحها أو غير ذلك.

تعتبر المتحكمات ذات شهرة وانتشار كبيرين الآن، فتقدر بعض الإحصائيات أن مبيعات المتحكمات المصغرة تشكل معظم سوق المعالجات وأن أعداد مبيعاتها تعد أضعاف ما هي عليه معالجات الحواسيب على تنوعها (لكن بالمقابل فإن أرباح سوق معالجات الحواسيب هي الأعلى لارتفاع أسعارها)، وتتوفر الآن متحكمات مصغرة شديدة الصغر يتم وضعها مثلاً في البطاقات الذكية التي منها بطاقات SIM (المستخدمة في أجهزة الهاتف المحمول) وبعض بطاقات الائتمان.

يتم إنتاج المتحكمات من قبل العديد من الشركات، ولها عدة أصناف منها: المتحكم 8051 وهو أول متحكم مصغر و قد اكتسب شهرة واسعة، طورته وأنتجته شركة Intel ثم توقفت عن إنتاجه، وتنتج الآن العديد من الشركات متحكمات مطورة عنه ومتوافقة معه من أشهرها العائلة AT89xxxx من إنتاج شركة Atmel, كما توجد عائلة متحكمات أخرى اكتسبت شهرة كبيرة هي متحكمات PIC والتي تنتجها شركة Microchip وتحمل التسمية PICxxxx وكذلك توجد عائلة AVR التي تنتجها شركة Atmel وقد اكتسبت هذه العائلة شهرة واسعة جداً و ذلك لسهولة استخدامها و سعرها المنخفض ولا زال هناك الكثير من شركات إنتاج المتحكمات من أشهرها Motorolla, Cypress, Phillips, Ubicom وغيرها.

# الأنظمة المدمجة
# Embedded Systems

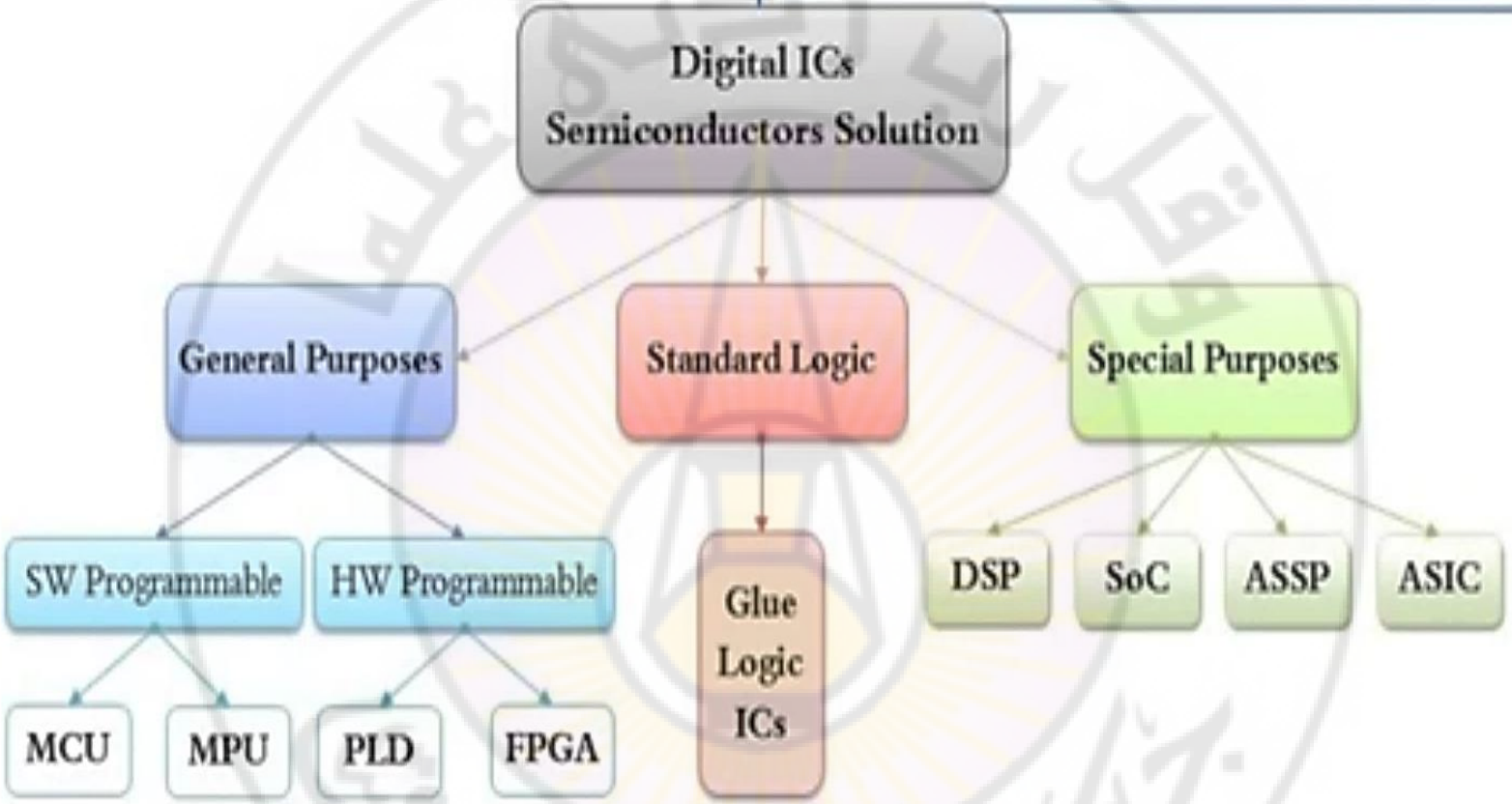
## وعلاقتها بالمتحكمات الصغرية

Damascus University

المتحكمات الصغرية تتفرع من الأنظمة المدمجة

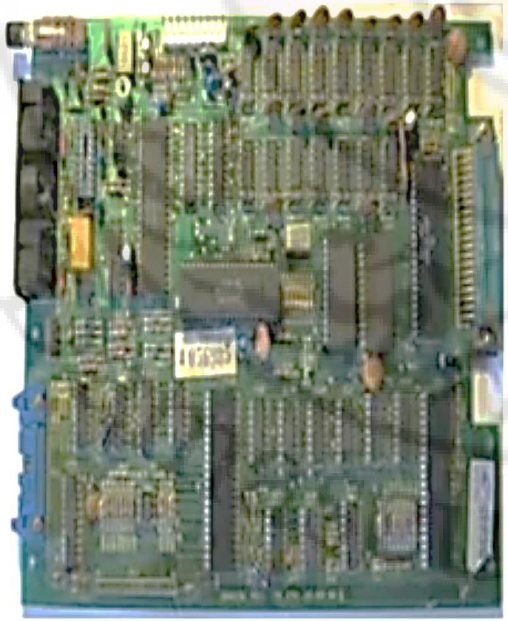
- حيث أن الأنظمة المدمجة : هي تقنية تهدف إلى دمج عدة وظائف سواء كانت مكونات مادية أو برامج . وذلك لدمجها في قطعة واحدة معينة دون الحاجة إلى تعداد الكثير من القطع (أجهزة الاتصال، أقطاب النواقل ، الكترونيات ، حواسب .....) ، وهي تشكل بنيان التطور التكنولوجي ومايسمى بقاعدة bit to bit roll.
- توزع وانتشار الأنظمة المدمجة في عالمنا منتشر في أمور الحياة الحديثة ، وأصناف الدارات المتكاملة كثيرة جدا ، وجزء منها صغير فقط هو المتحكم الصغري.

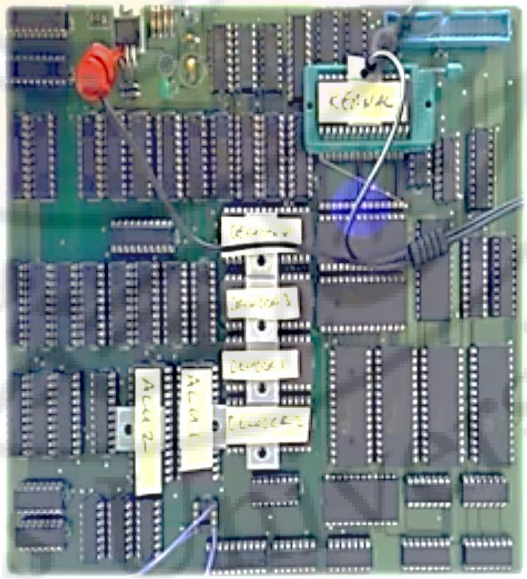
# أصناف الدارات المتكاملة



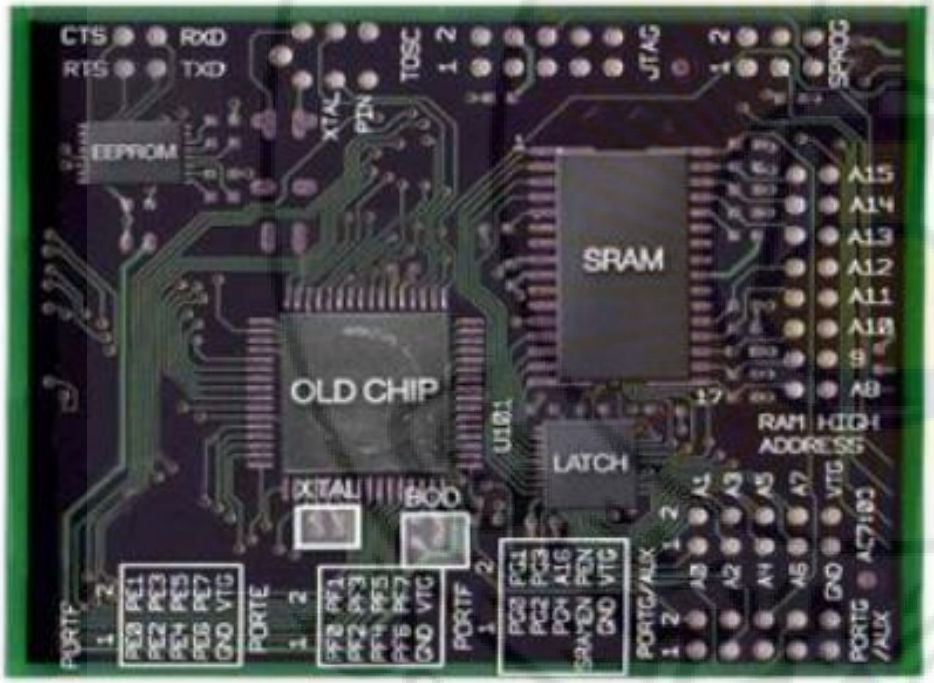
# أهمية المتحكمات الصغرية:

نشأت المتحكمات الصغرية لتجميع الكم الهائل من الدارات الالكترونية ICs ودمجها في شريحة واحدة ، أي توفير العديد من القطع بقطعة صغيرة واحدة .
مثلا المتحكمAVR يدعم خصائص بروتوكولات الاتصال - ذواكر نواة المعالج - محولات من رقمي إلى ثماني والعكس - مؤقتات - شاشات .....إلخ

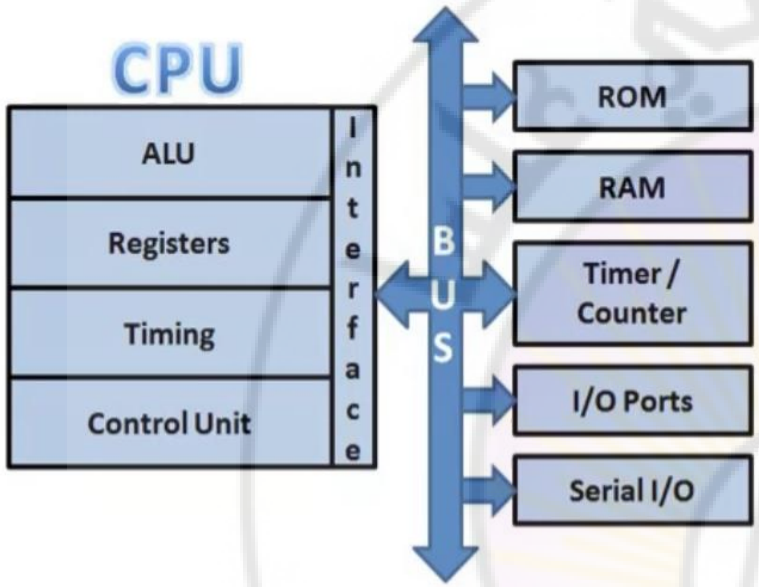


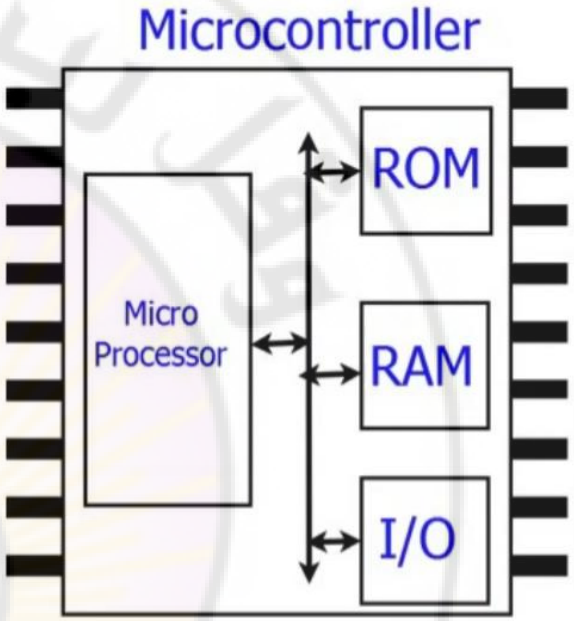


## مالفرق بين المعالج الصغري Microprocessor والمتحكم الصغري Microcontroller؟؟؟

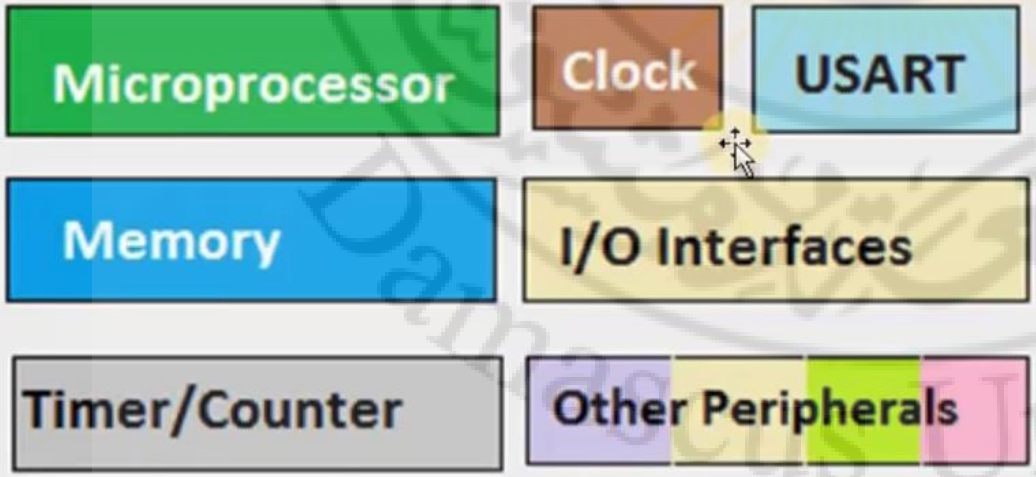


| Microporcessor | VS | Microcontroller |
|---|---|---|

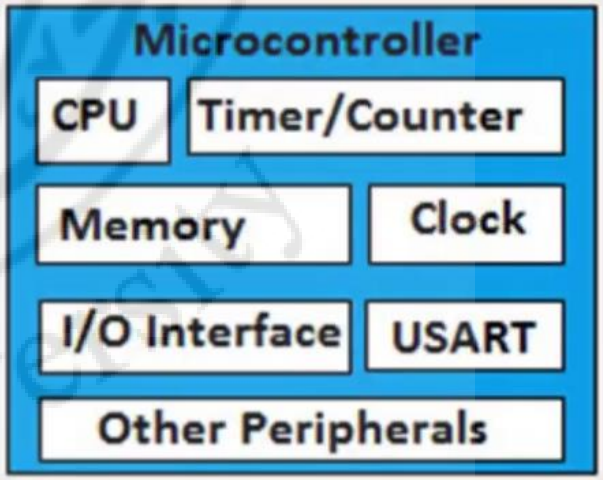



## Many Different Chips

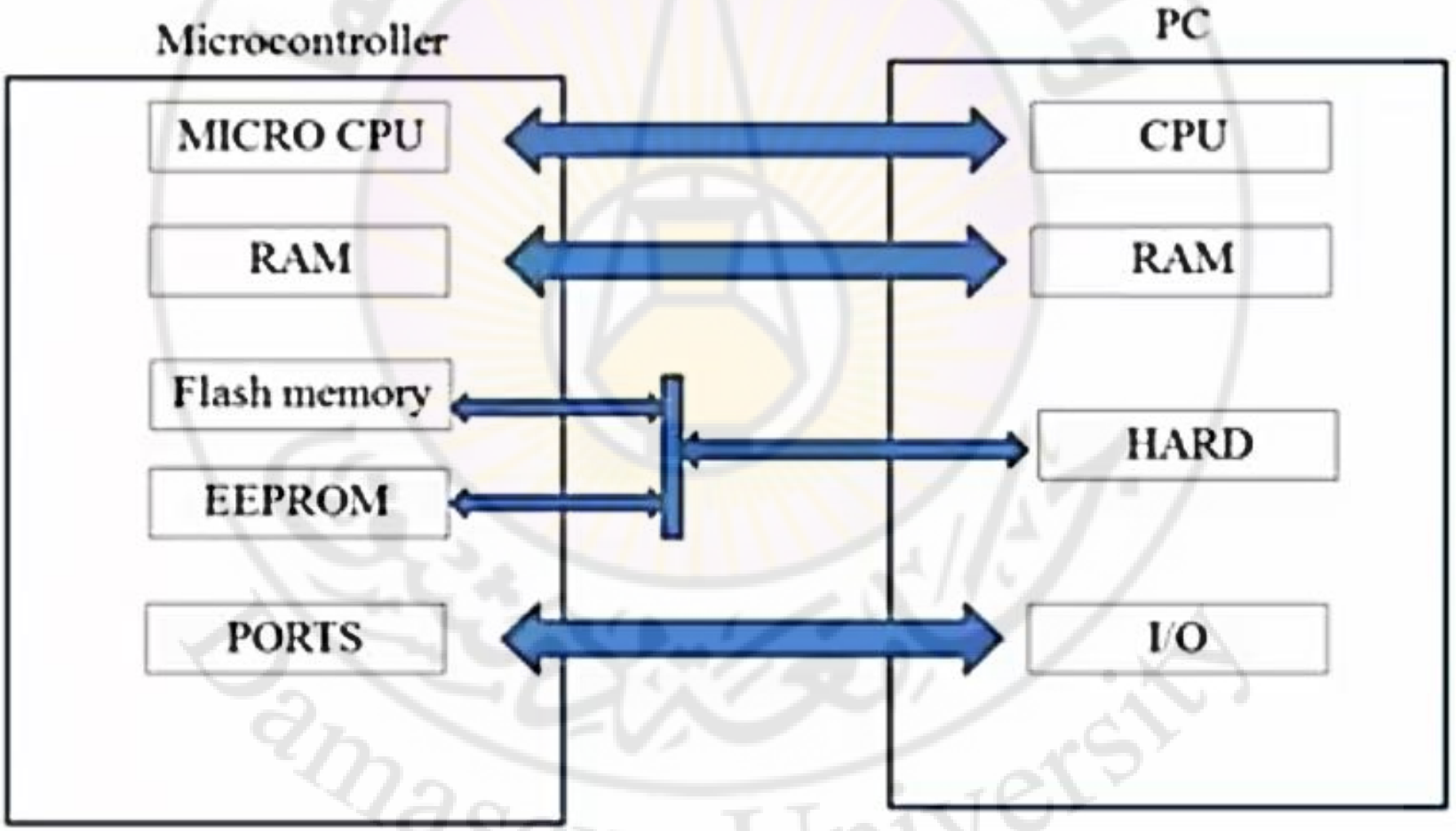



## One Chip



| المعالج الصغري | المتحكم الصغري |
| --- | --- |
| لا يمكن تركيبه في الدارات الالكترونية بدون وحدات محيطية (I/O ، عداد ، مؤقت .....) | يمكن تركيبه في الدارات الالكترونية بشكل مباشر أي لايحتاج وحدات محيطية |
| يتصل مع الوسط الخارجي من خلال الوحدت المحيطية الخاصة به | تم تضمين جميع هذه الواحدات في شريحة واحدة وبحجم شريحة المعالج المصغر |
| مكلف | يستخدم في التطبيقات التي تعتبر فيها التكلفة والطاقة والمساحة بالغة في الأهمية |
| قوة معالجة عالية | قوة معالجة منخفضة |
| استهلاك طاقة كبير | استهلاك منخفض للطاقة |
| تركز مجموعة التعليمات على العمليات كثيفة المعالجة | عمليات على مستوى الـ bit |

# المقارنة مابين الحاسوب الشخصي والمتحكم المصغر

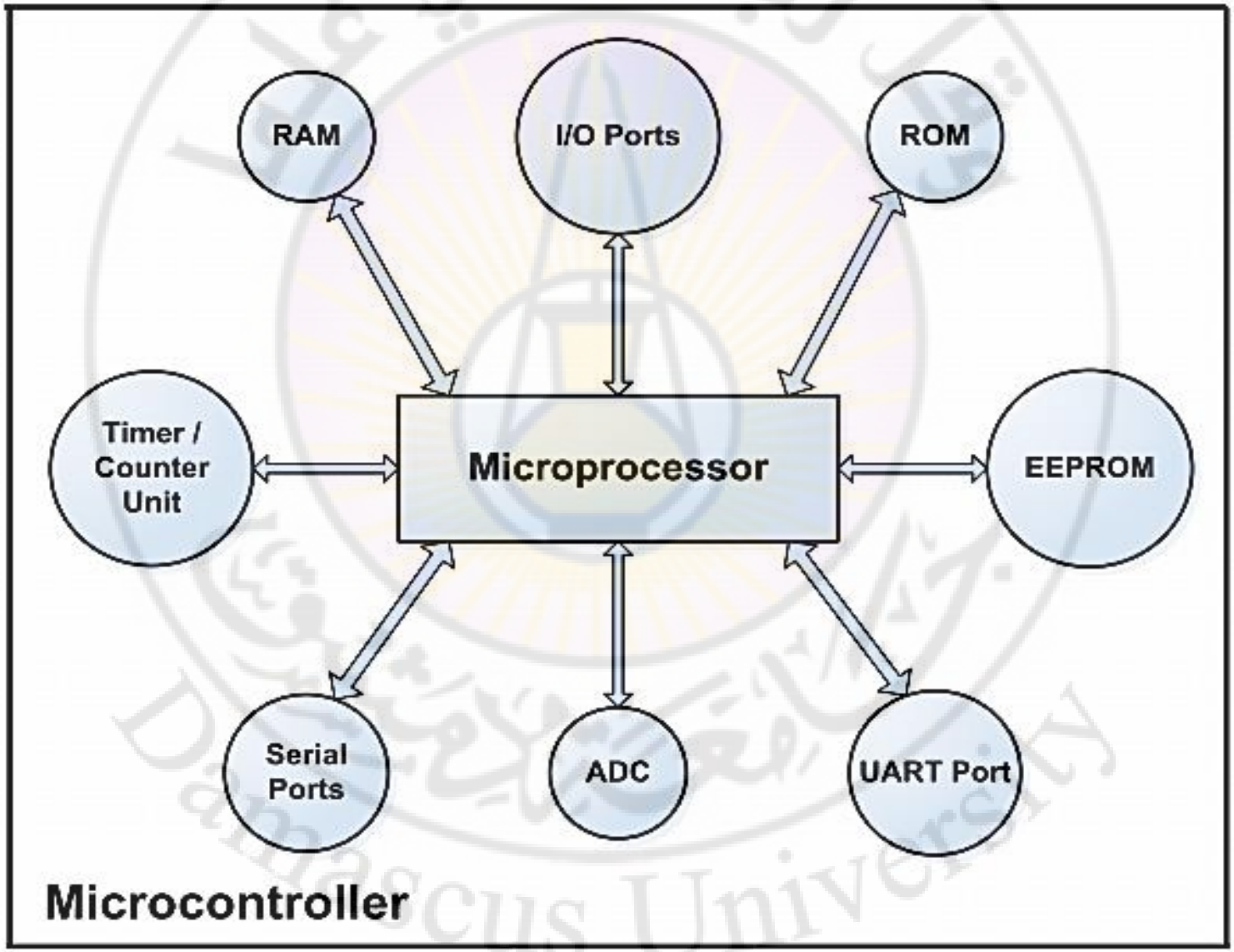


# أنواع المتحكمات المصغرة:

1. حسب طول المعطيات التي يتعامل معها معالج المتحكم المصغر:
   - bit 8
   - bit16
   - bit 32

2. من حيث التطبيقات :
   - متحكمات مصغرة ذات أغراض عامة وذلك لتوظيفها في معظم التطبيقات .
   - متحكمات مصغرة ذات أغراض خاصة والتي هي نفسها ذات الأغراض العامة ولكن تحتوي على طرفيات إضافية لتخدم مجال تطبيق معين

# بنيان المتحكمات الصغرية

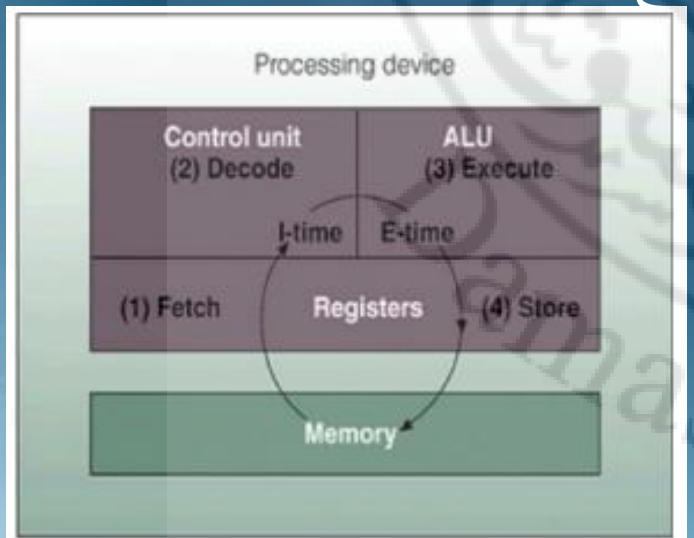

RAM
I/O Ports
ROM
Timer / Counter Unit
Microprocessor
EEPROM
Serial Ports
ADC
UART Port
Microcontroller

## المكونات الأساسية للمتحكم المصغر ذو الأغراض العامة :

1) المعالج CPU: يقوم بالعمليات الحسابية والمنطقية والنقل والتحكم . وملخص عملها كما يلي : قراءة (جلب Fetching) التعليمة من ذاكرة البرنامج - وتفسيرها Decoding- وتنفيذها Excuting ، ومن ثم كتابة النتائج في الذاكرة او بقائها مرحليا فيها

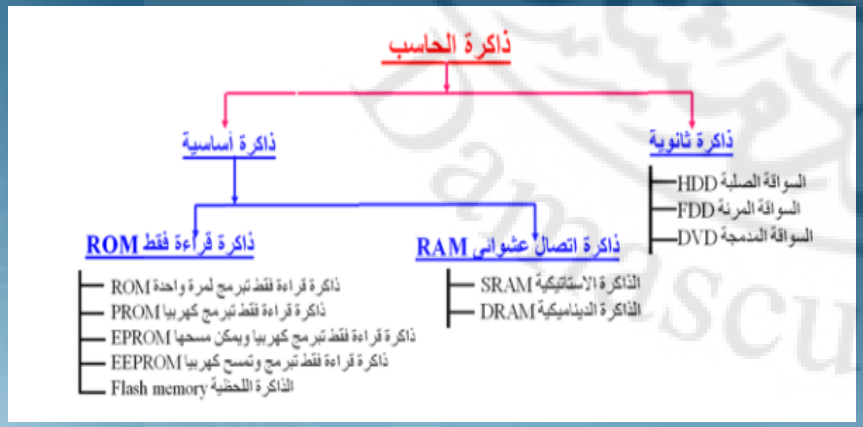
2) مسجلات Registers : وهي عبارة عن ذاكرة مؤقتة يعتمد حجمها على نوع معالج CPU ولها عدة أنواع :

1. مسجلات ذات أغراض عامة GBR.
2. مسجلات خاصة SFR:
   - مسجلات التحكم .
   - مسجلات الحالة .
   - مسجلات المعطيات .



3) الذواكر وهي :

1. ذاكرة البرنامج program memory : ROM تحتوي تعليمات دائمة يتم تكرارها بشكل دائم حسب الكود المكتوب ، بحجم وعرض تخزين معين .
2. ذاكرة البيانات data memory : RAM او ذاكرة المعطيات وفي معظم المتحكمات تسمى Static RAM وتكون عبارة عن مجموعة بنوك ، وذلك لكي يسهل التعامل معها ، حيث أن كل بنك (جزء) يمكن التعامل معه على أنه block واحد ..... وهي ذاكرة متطايرة
3. ذاكرة flash



4 **الذاكرة القابلة للبرمجة PROM :**

الذاكرة القابلة للبرمجة Programmable ROM (PROM) أو الذاكرة القابلة للبرمجة مرةً واحدةً OTP (One-Time Programmable Non-Volatile Memory) NVM هي ذاكرةٌ رقميةٌ يكون فيها كل بت مقفلًا بعنصر إلكتروني لا يمكن التعديل عليه. تُكتب البيانات على هذه الذاكرة بعد تصنيعها وتبقى فيها على الدوام دون إمكانية تعديلها أو إزالتها، ولكتابة محتواها توضع في جهاز يدعى "مبرمج "PROM يُستخدم هذا النوع في الأجهزة المحمولة، والمتحكمات المصغَّرةmicrocontrollers ، وغيرها من الأجهزة الإلكترونية.

5 **الذاكرة القابلة لإعادة المسح والبرمجة EPROM**

يمكن مسح محتوى هذه الذاكرة Erasable Programmable ROM (EPROM) خلافًا للنوع السابق بتعريض الخلايا للأشعة فوق البنفسجية ثمَّ إعادة الكتابة عليها. تستهلك هذه العملية (المسح وإعادة البرمجة) من عمر الذاكرة وتصل عدد مرات المسح وإعادة البرمجة إلى ١٠٠٠ مرة. يوجد أعلى الذاكرة فتحةٌ تسمح بمرور الأشعة فوق البنفسجة إلى الخلايا لمسح محتواها دفعةً واحدةً، لذا يجب نزع هذه الذاكرة من اللوحة ووضعها على جهاز البرمجة لبرمجتها مجدَّدًا وغالبا توضع على مقبس لتسهيل نزعها واعادة تركيبها.

6 **الذاكرة القابلة لإعادة المسح والبرمجة إلكترونيًا EEPROM**

تشبه بنية هذه الذاكرة (EEPROM) (Electrically Erasable Programmable ROM النوع EPROMباستثناء أنَّ عملية المسح والبرمجة تُجرى إلكترونيًّا أي لا داعي لإزالتها من اللوحة أو الدارة.

# المقارنة بين ذواكر المتحكم المصغر

| النوع الذاكرة | الاستخدام | سرعة الكتابة | عدد مرات الكتابة والمسح | ديمومة المعطيات |
|---|---|---|---|---|
| RAM | مكان معالجة المعطيات | سرعة جداً من مرتبة ns | غير محدود | نفقد المعطيات بانقطاع التغذية |
| FLASH | مكان تخزين البرنامج | بطيئة من مرتبة ms | قد يتجاوز 100000 مرة | لا نفقد المعطيات بانقطاع التغذية |
| EEPROM | مكان تخزين معطيات المستخدم | بطيئة من مرتبة ms ولكنها اسرع من FLASH | قد يتجاوز 100000 مرة | لا نفقد المعطيات بانقطاع التغذية |

Damascus University

4) منافذ رقمية Ports : تستخدم لتبادل المعطيات الرقمية مع العالم الخارجي .

5) مؤقتات وعدادات Timers & counters

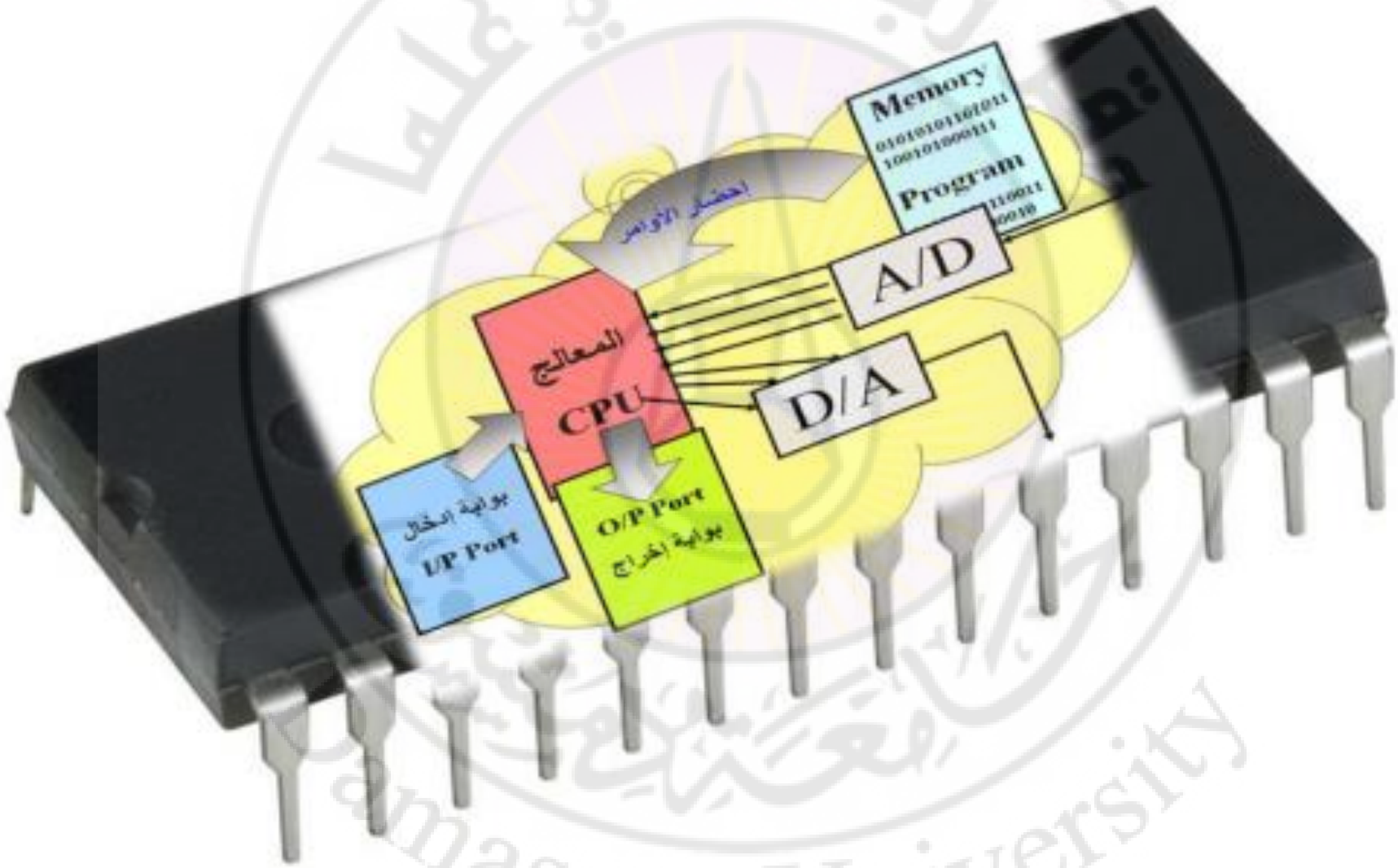
6) نواقل Bus

وقد تحتوي أيضا :

- محولات تشابهية رقمية ADC
- طرفيات اتصال تسلسلي
- طرفيات أخرى

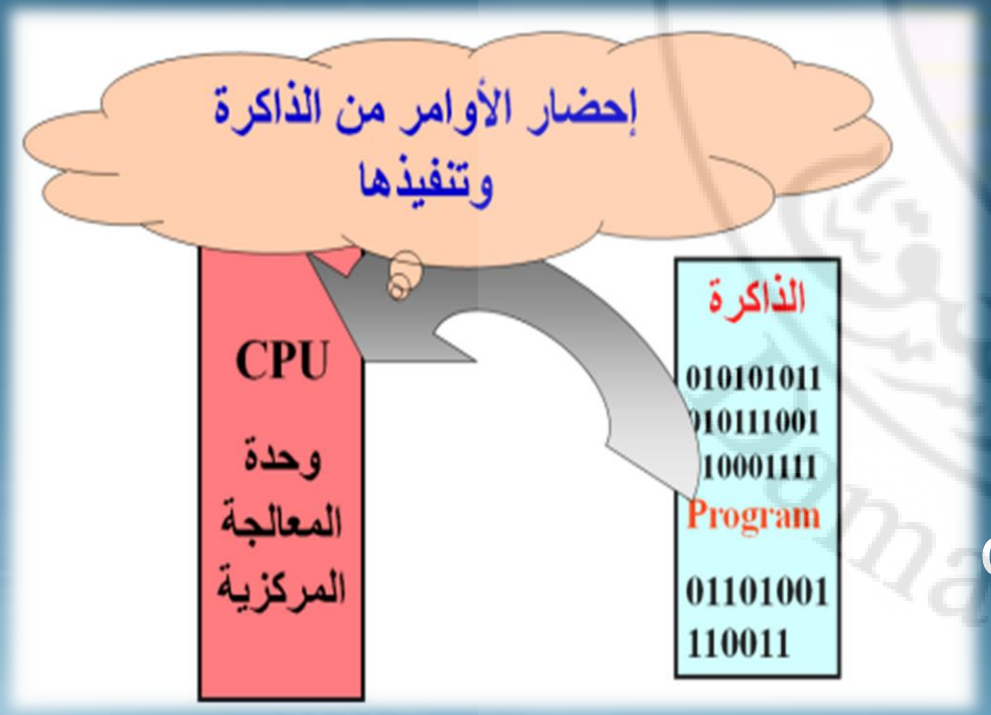
# بنيان المتحكمات الصغرية

# MICROCONTROLLERS ARCHITECTURE


Memory
Program
A/D
D/A
المعالج
CPU
إحضار الأوامر
بوابة الدخال
UP Port
O/P Port
بوابة إخراج

# وظيفة وحدة المعالجة المركزية

تقوم وحدة المعالجة المركزية بجلب وتنفيذ البرنامج المخزن في الذاكرة memoryأو بصيغة أخرى وضع التعليمة المخزنة في الذاكرة قيد التنفيذ ، وذلك لتأدية المهمة المطلوبة من قبل شريحة المتحكم الصغري ، ولكي ينفذ المعالج التعليمة المخزنة في الذاكرة فإنه يمر بدورتين (مرحلتين) :

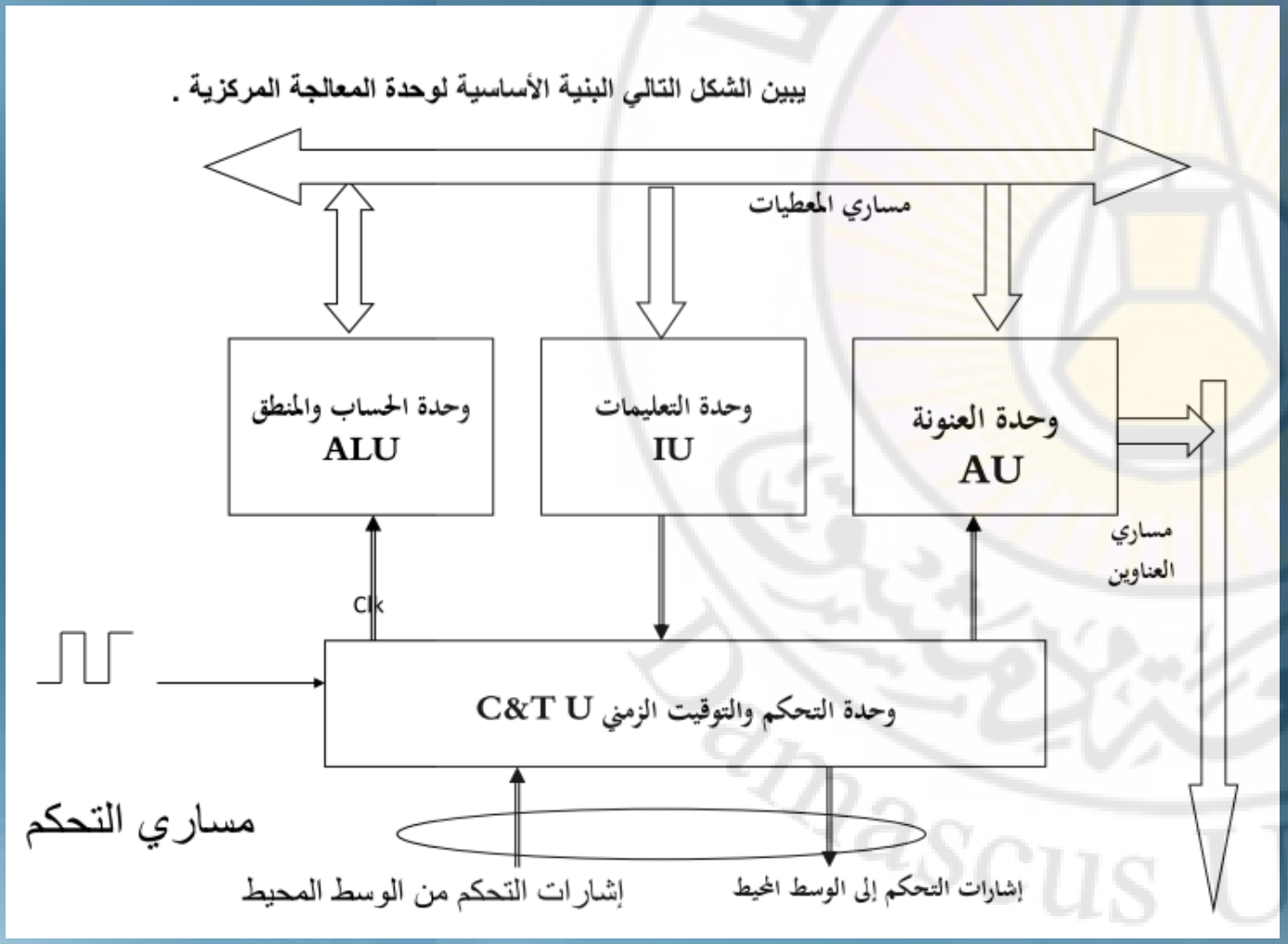



دورة الجلب Fetching Cycle

دورة التنفيذ Execution Cycle

وسوف نتحدث عنها بالتفصيل بعد معرفة البنية الداخلية للـ cpu

# تصميم البنية الداخلية لوحدة المعالجة المركزية CPU




مكونات الـ CPU:

- وحدة الحساب والمنطق ALU.
- وحدة التحكم CU .
- المسجلات Registers .

# وحدة الحساب والمنطق ALU:

وهي المكون الأساسي في وحدة المعالجة المركزية ، وهي عبارة عن دارة منطقية تزودنا بالعمليات الحسابية والمنطقية ، حيث تتكون من وحدتين فرعيتين :

وحدة الحساب Arithmetic Unit : لتنفيذ العمليات الحسابية

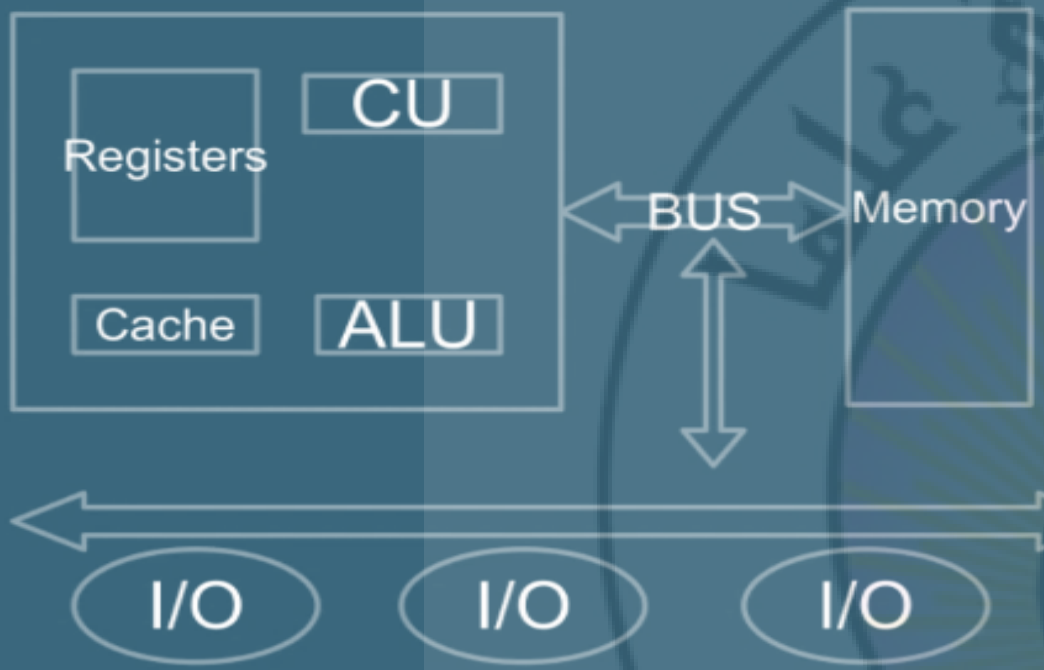
وحدة المنطق Logic unit: لتنفيذ العمليات المنطقية

أي أنها عبارة عن دارة تركيبية تقوم بمجموعة من العمليات الحسابية والمنطقية بناء على مايتم انتخابه من مداخل ، والخرج الخاص بها يكون على مراكم يتم الاحتفاظ به حتى يتم استدعاؤه

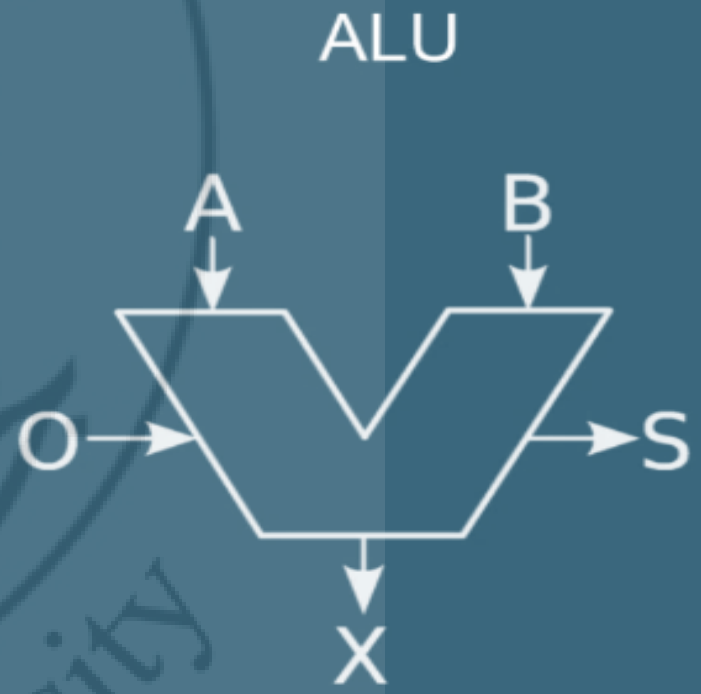
في العادة تقوم وحدة التحكم بإرسال نوع العملية إلى وحدة الحساب والمنطق التي تقوم بتنفيذ العملية وإرسال النتيجة إلى مسجل التراكم .

معظم المتحكمات تكون 8 bit ، أي A=8 bit، B= 8 bit ، X= 8bit أوالمراكم (Acc)

# Computer Architecture




لها مدخلين input1=A و input1=B في الشكل
ومخرج output =O
ومخرج آخر status =S والذي هو الحالة
ولها مدخل ثالث operation=O العمليات والمراكم للنتيجة X




The **ALU** is a digital circuit that provides **arithmetic** and **logic** operation. It is the fundamental building block of central processing unit of a computer.

# ماهي مكونات ALU:

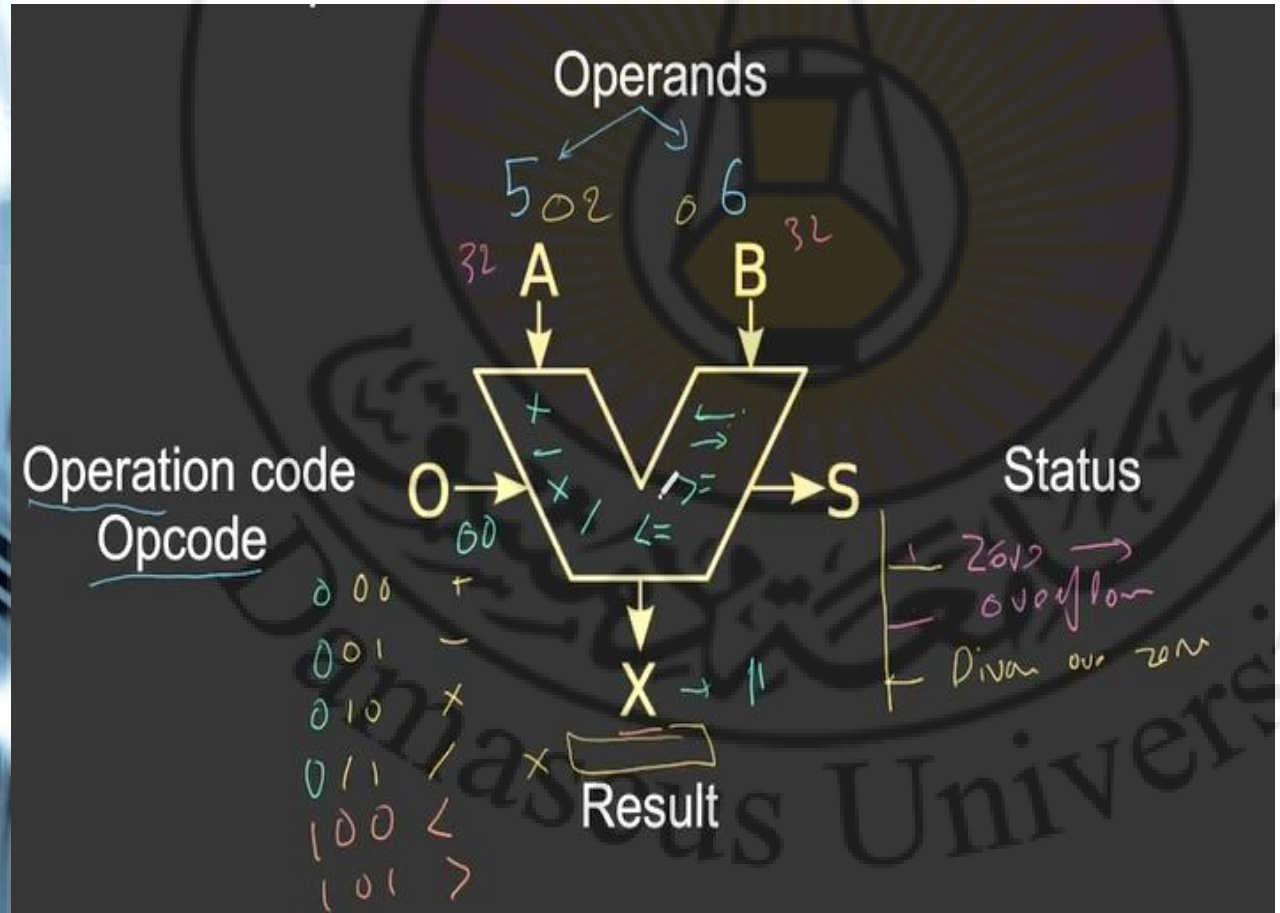
1. **Operands:** هي الأشياء التي سوف تعمل عليها الآلة ، مثلا حالة جمع عددين AوB ، يكون العدد الأول للمدخل A والثاني للمدخل B ، ثم تقوم عملية الجمع ضمن ALU وإخراج القيمة عن طريق X ...
2. **Result :** والتي من خلالها يتم إخراج القيمة المطلوبة .
3. **Operation unit :** وحدة الحساب والمنطق لاتقوم بعملية حسابية واحدة فقط ، أي أنها تدعم مجموعة من العمليات الحسابية (+،-،*،<،=.....إلخ) ، والعمليات المنطقية مثل (shift إزاحة لليمين اواليسار .....إلخ). ويتم استنتاجها كجزء من الـ opcode الخاص بالتعليمة

عندما نقوم بإعطاء المدخلات فإن وحدة الحساب والمنطق لاتكتفي بذلك وإنما تحتاج إلى تحديد نوعية وطبيعة العملية التي سوف تقوم بإجرائها على المدخلات ، وذلك عن طريق مدخل ثالث وهو opcode .

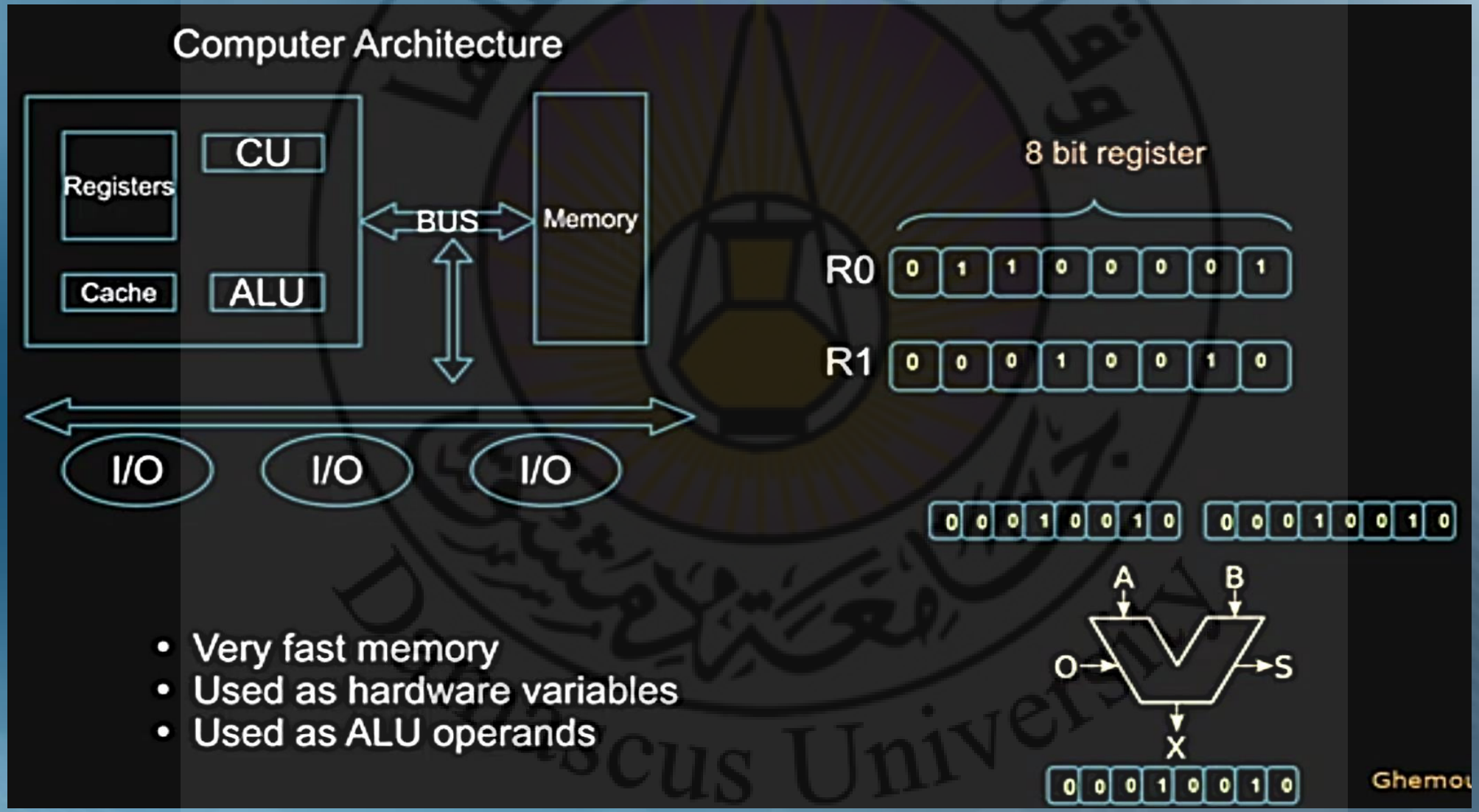
لأن وحدة الحساب والمنطق عبارة عن تركيبة من الدوائر الالكترونية التي تقوم بعمليات حسابية ومنطقية مختلفة لذلك تحتاج إلى اختيار وتحديد لهذه العمليات عن طريق المدخل O

4. **المخرج Status :** وهومسجل الحالة يمثل حالة الخرج للعملية التي قامت بها وحدة الحساب والمنطق .

وهو يعطي معلومة عن سير العملية الحسابية او المنطقية



# REGISTERS المسجلات



- المسجلات هي عباره حجرات موجودة ضمن الذاكرة تستخدم للتخزين المؤقت للمعلومات في صورة خانات ثنائية داخل شريحة المعالج لحين الحاجة إليها، وكل حجرة لها عنوان ومحتوى .... وغالبا ماتكون بحجم bit8 وممكن ان تصل إلى (16-32-64) للأحدث منها

- إن وحدة المعالجة المركزية عبارة عن مجموعة من المسجلات المتجاورة التي تستخدم في تخزين البيانات المدخلة إلى الدارات الالكترونية لإجراء العمليات الحسابية والمنطقية ، ومايتم تخزينه في المسجلات هو تخزين مؤقت حتى تتم معالجة البيانات وتنقل النتائج إلى الذاكرة الرئيسية ، فالتخزين في الذاكرة هوتخزين مؤجل طويل الأمد نسبيا حتى تستدعي بياناتها ومحتوياتها للمعالجة، بينما التخزين على سجلات وحدة المعالجة المركزية تخزين عاجل للمعالجة فقط .

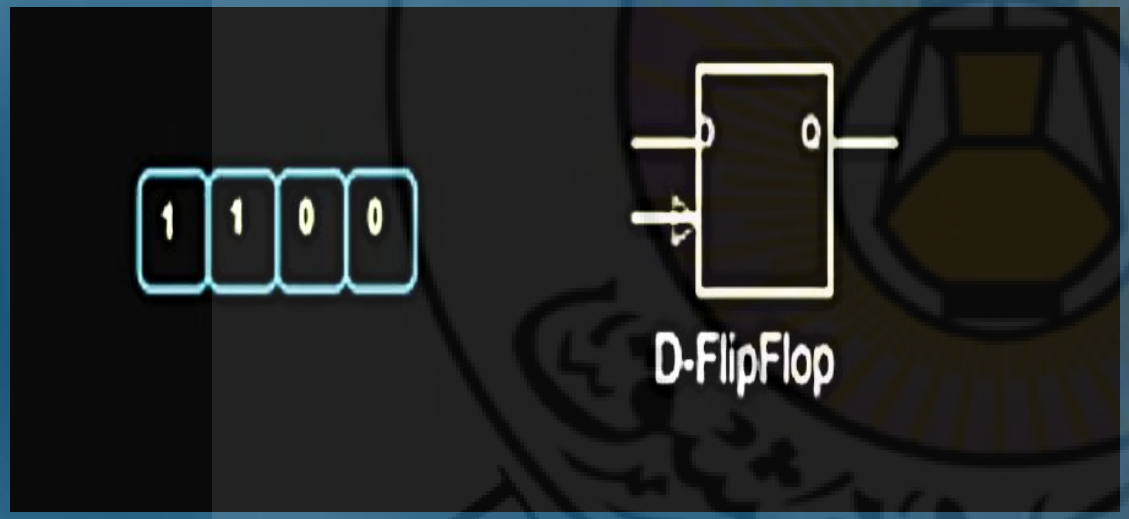
- وتخضع المسجلات للسيطرة المباشرة من قبل وحدة التحكم المركزية cu التي تراقب وصول البيانات إلى المسجلات ، بعدها تعطي وحدة التحكم أوامرها إلى الدارات الالكترونية الحسابية والمنطقية للعمل ثم تراقب حركة وتخزين البيانات في مسجلات أخرى مخصصة للمخرجات .

- كل مسجل نريد القيام بعملية عليه ـ نقوم باستدعائه إلى وحدة الحساب والمنطق ، ثم تتم العملية المطلوبة فيه ومن بعدها إلى المراكم ACC ، وتعاد إلى الذاكرة

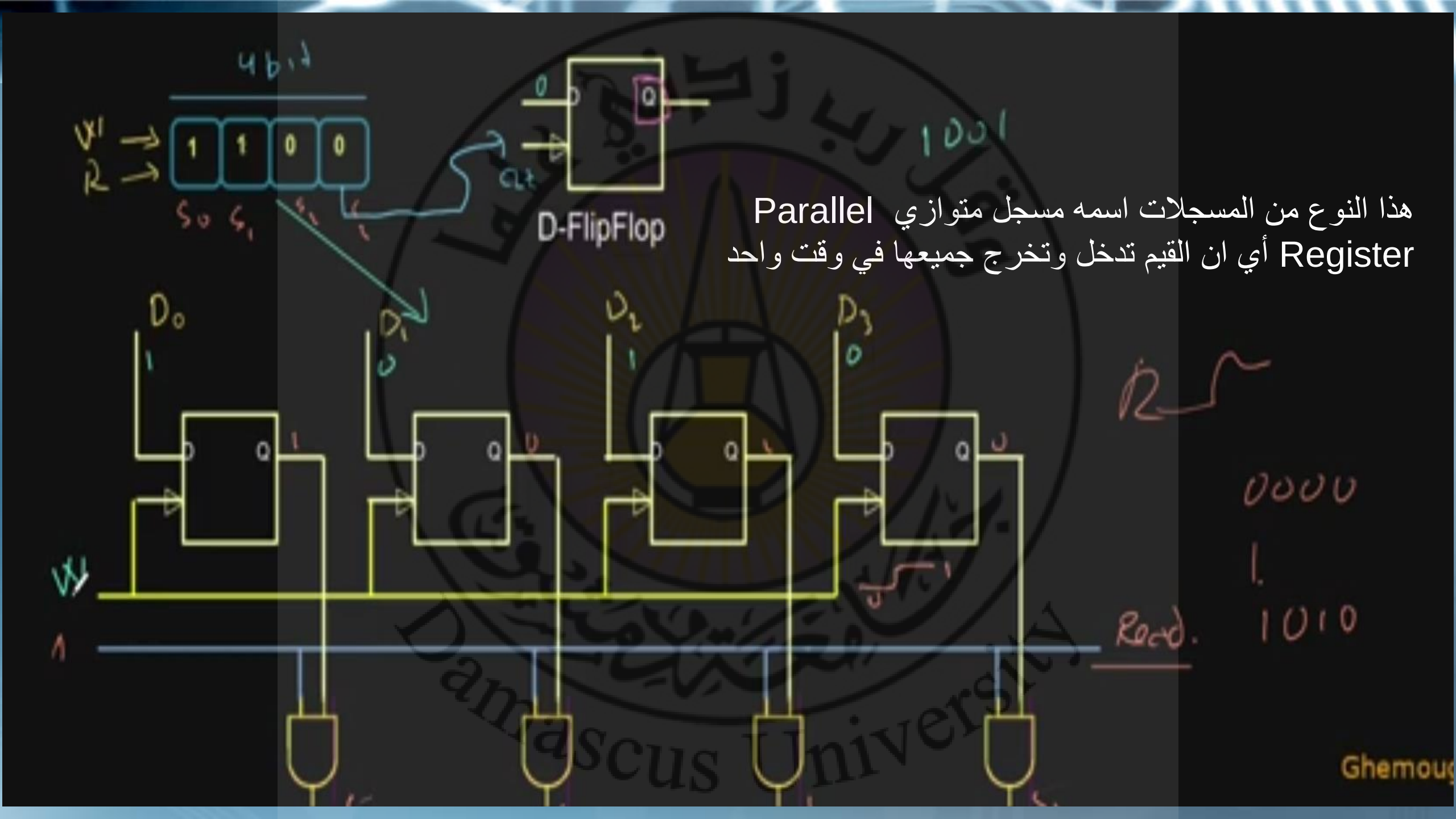
- تتميز بسرعتها الفائقة ، سريعة القراءة والكتابة وهي أسرع ذاكرة موجودة داخل المعالج cpu

# تصميم المسجلات :

يصنع بواسطة مايسمى بوابات Flip Flops ، وهي نوع من الدارات المنطقية فكرتها الأساسية تخزين قيم معينة في المدخل D لإخراجها من Q بعد مرورها من المؤقت الذي يتحكم في تغيير القيمة أو مرورها كما هي.




وكل بت يحتاج إلى واحدة Flip Flop وهو المكون الأساسي لصناعة المسجل (مكون مادي)



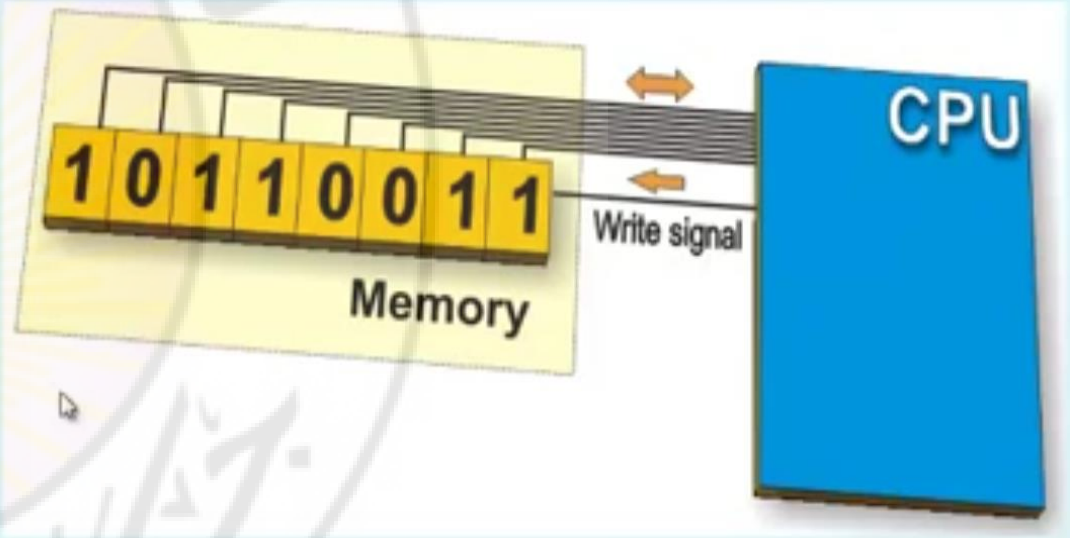

هذا النوع من المسجلات اسمه مسجل متوازي Parallel Register أي ان القيم تدخل وتخرج جميعها في وقت واحد

# أنواع المسجلات

- مسجلات للاستعمال العام : لتخزين بيانات ومعلومات عامة General purpose registers
- مسجل مؤشر التعليمات IP : يؤشر إلى التعليمة الموافقة أي يقوم بتخزين عنوان التعليمة في الذاكرة ، وقيمته تزداد بشكل تلقائي ، أي يقوم بزيادة (عداد) في كل مرة بحيث تلقائيا ينتقل إلى التعليمة التالية وهكذا..... وهو يقوم بتخزين العنوان ولايخزن القيمة بحد ذاتها (المحتوى) لأنه موجود أساسا في الذاكرة .
- مسجل الراية flag : وهو غير موجود في كل المعالجات بشكل مستقل ، عبارة عن مسجل حالة بمقدار 1bit ويعطي معلومة عن حالة المعالج في كل لحظة ، مثلا CF (carry fly) حالة فائض العملية الحسابية ، PF حالة العدد فردي ام زوجي ، ZF حالة الصفر ...
- مسجل IR (instruction register) :يحتوي على قيمة التعليمة المسجلة وهو مقسوم إلى ثلاث أجزاء( op (opcode كود التعليمة ، والمعامل الأول والمعامل الثاني.
- Address register : لتخزين العناوين فقط قراءة منها أو كتابة فيها ونقلها إلى ممر المعطيات
- مسجلات الأدلة Index Registers
- مسجلات القطاعات Segment Registers

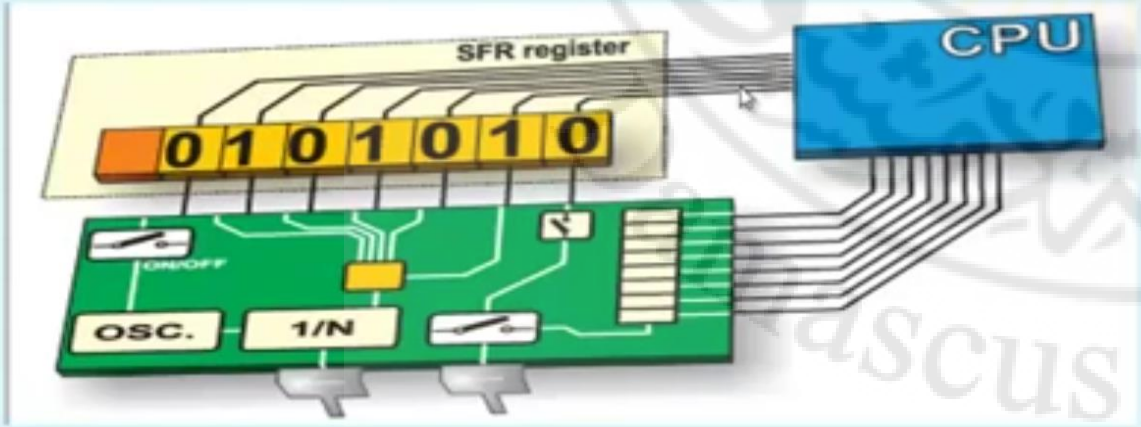
# (2) REGISTER

- a register or a memory cell is an electronic circuit which can memorize the state of one byte.




# (3) SFR REGISTERS

- Special function registers .
- every microcontroller has a number of registers (SFR) whose function is predetermined by the manufacturer .
- such as timers, A/D converter, oscillators and others .

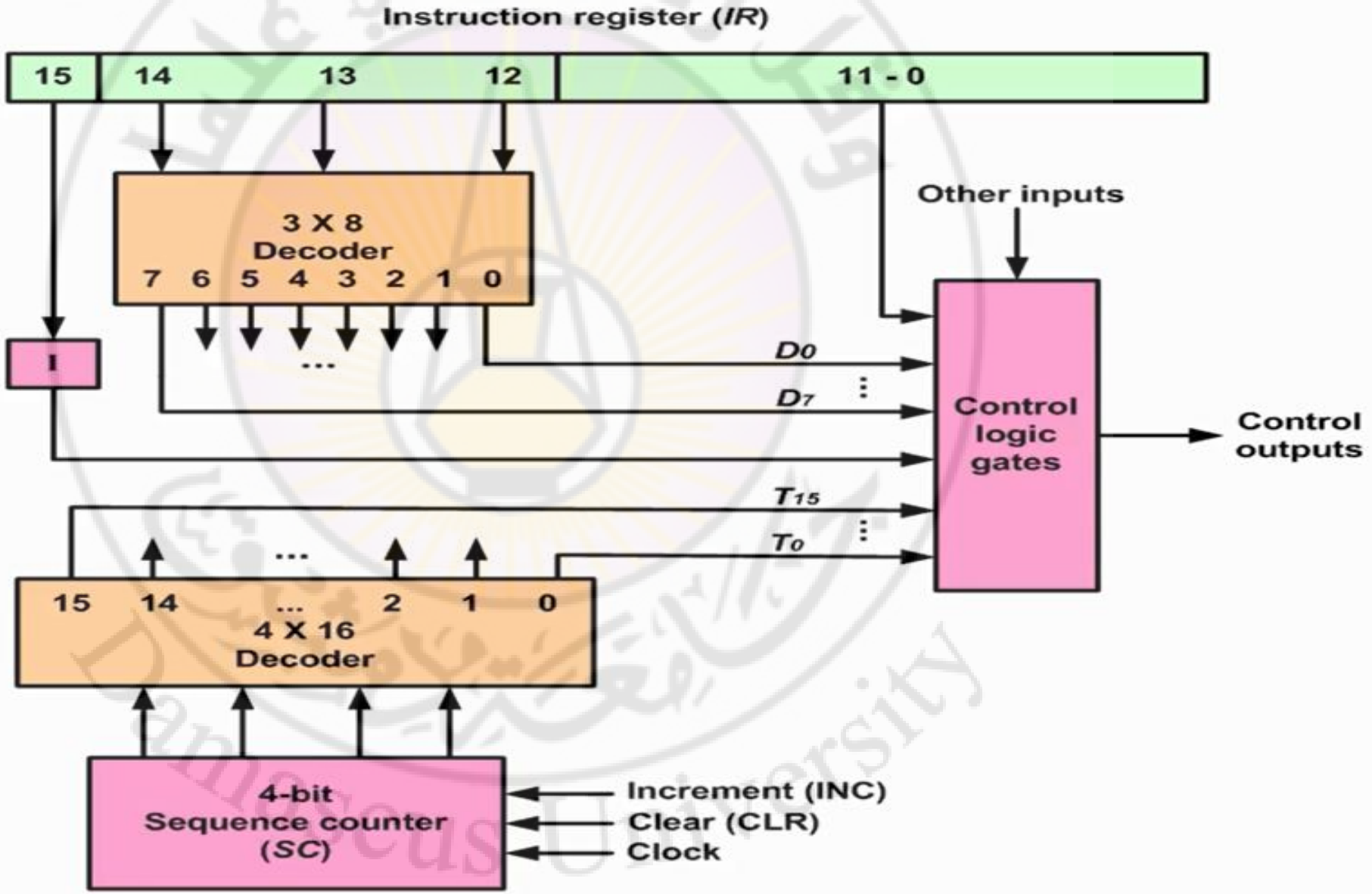


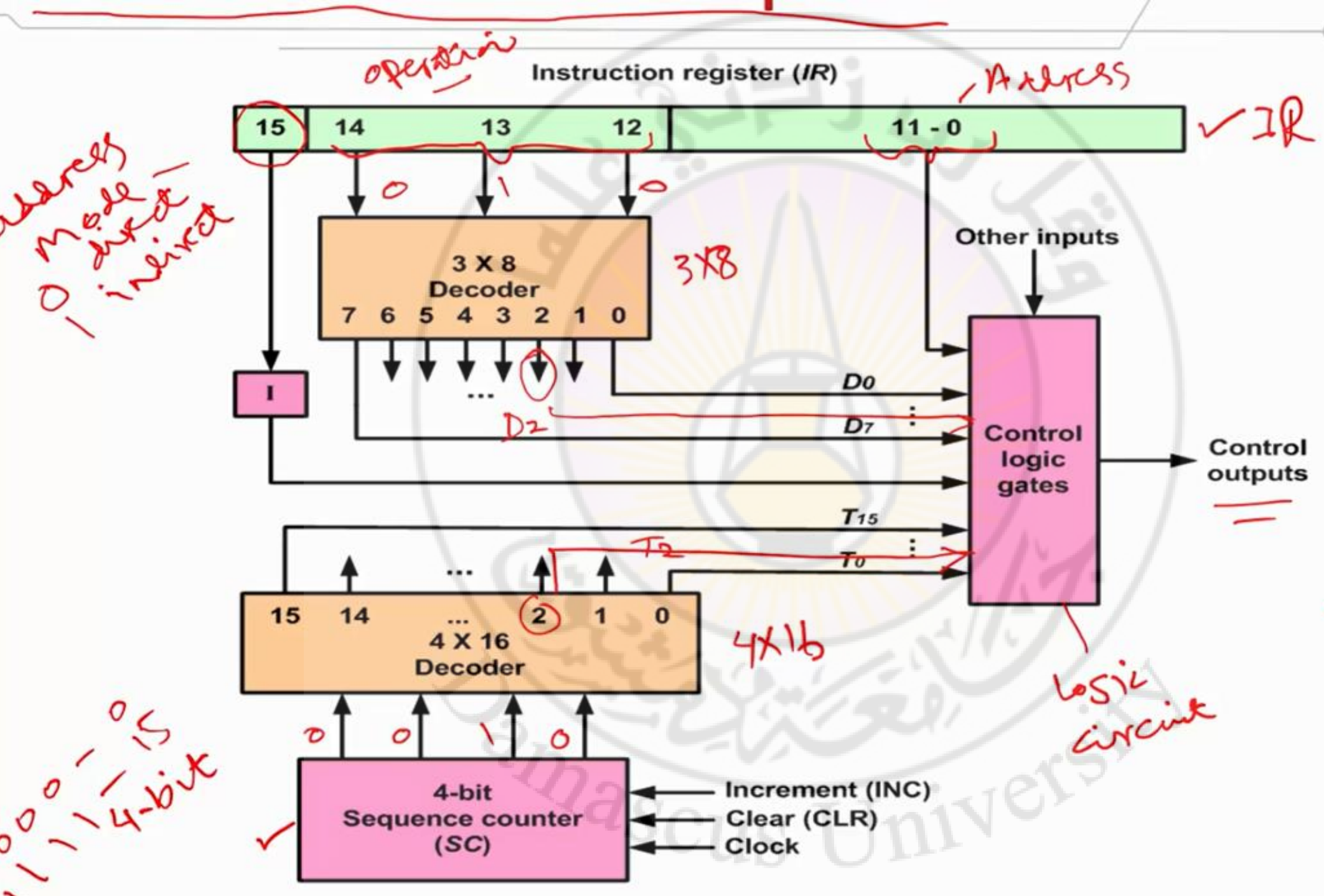
# وحدة التحكم CU( control unit)

وهي الوحدة التي تقوم بقيادة وتوجيه جميع العمليات التي تحدث في المعالج ، وهي التي توجه تشغيل المعالج ،وذلك بالتزامن مع نبضة من نبضات الساعة الخاصة بالمعالج ، حيث تقوم بالتحكم بكافة العمليات المنفذة وتشرف على تسلسل تنفيذ العمليات ، وهي تستخدم وحدة فك تشفير ثنائية لتحويل التعليمات المشفرة إلى إشارات توقيت وتحكم توجه تشغيل المكونات الأخرى .

تتمثل وحدة التحكم بمجموعة الأسلاك التي تتحكم بتدفق المعلومات عبر المعالج وتنظم عمل الوحدات الأخرى الموجودة داخله ، إنوظيفة وحدة التحكم تتغير بتغير البنى الداخلية للمعالج حيث أن وحدة التحكم هي التي تحقق البنيان الداخلي للمعالج بشكل عملي ، ففي المعالجات التي تنفذ x86 فإن وحدة التحكم تنجز المهام التالية : جلب التعليمة وفك شيفرتها وإدارة تنفيذها وتخزين النتيجة .

# Control unit of a basic computer

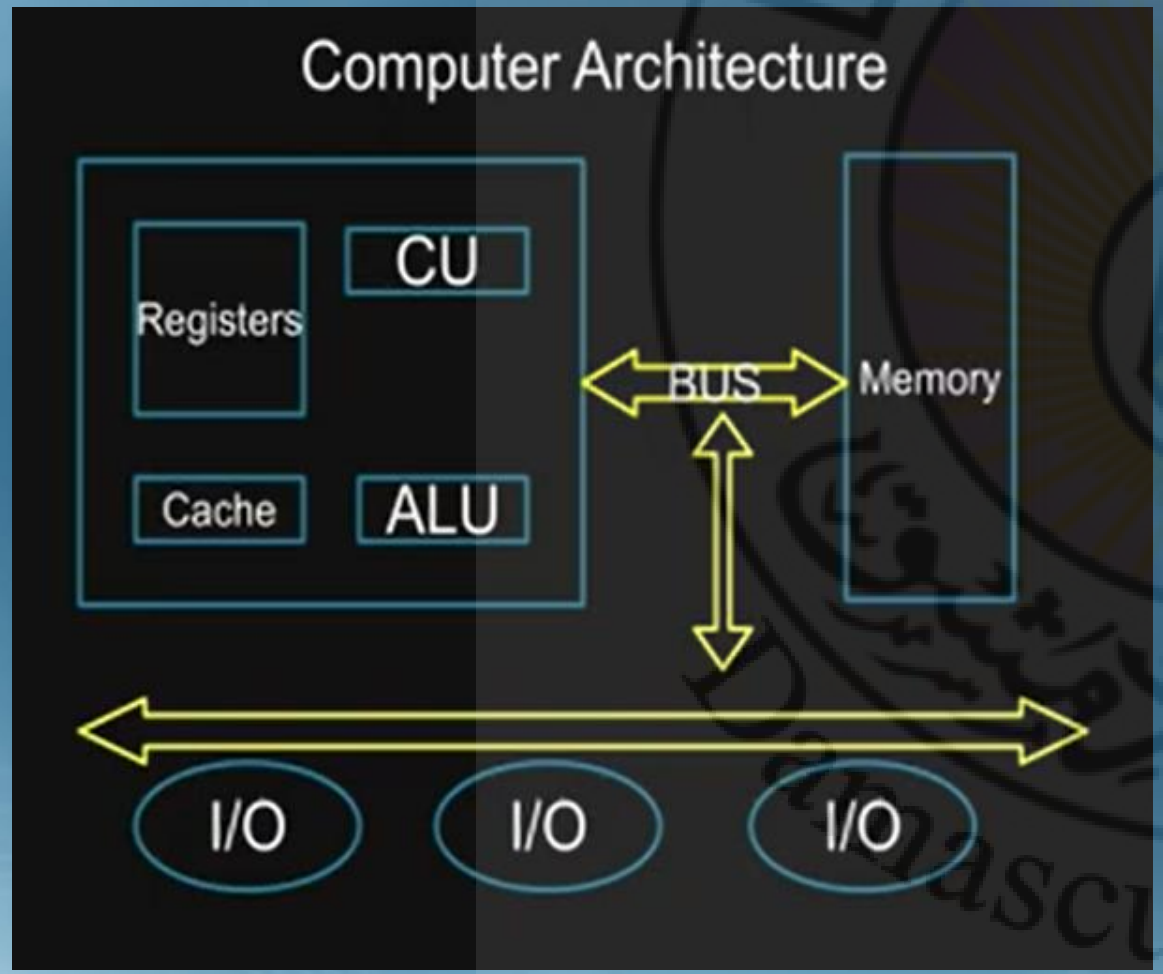



Instruction register (IR)
15
14
13
12
11 - 0
3 X 8
Decoder
7 6 5 4 3 2 1 0
I
Other inputs
D0
D7
D2
Control
logic
gates
Control
outputs
T15
T0
T2
15 14 ... 2 1 0
4 X 16
Decoder
4-bit
Sequence counter
(SC)
Increment (INC)
Clear (CLR)
Clock

الأجزاء التي تشرف عليها وحدة التحكم :

1. MAR (memory address register) وحدة العنونة : وهو الجزء الذي يقوم بأخذ المولدة من قبل العداد pc ونقله إلى ممر المعطيات لإرساله إلى الذاكرة.
2. PC( program counter) وهو يقوم بتوليد عنوان الحجرة الذاكرية التي تحتوي على التعليمة التالية التي سوف يتم تنفيذها.
3. MBR(memory buffer register) وهو عبارة عن مسجل يقوم بتخزين شيفرة التعليمة التي تم إحضارها من الذاكرة.
4. IR(instruction register) وحدة التعليمات : وهو مسجل يحتوي على التعليمة الحالية التي سوف تنفذ في وحدة الحساب والمنطق
5. Timer وحدة التحكم والتوقيت الزمني : وهو دارة توقيت تقوم بتوليد الفترات الزمنية لتنفيذ التعليمات.

# نواقل البيانات BUSES



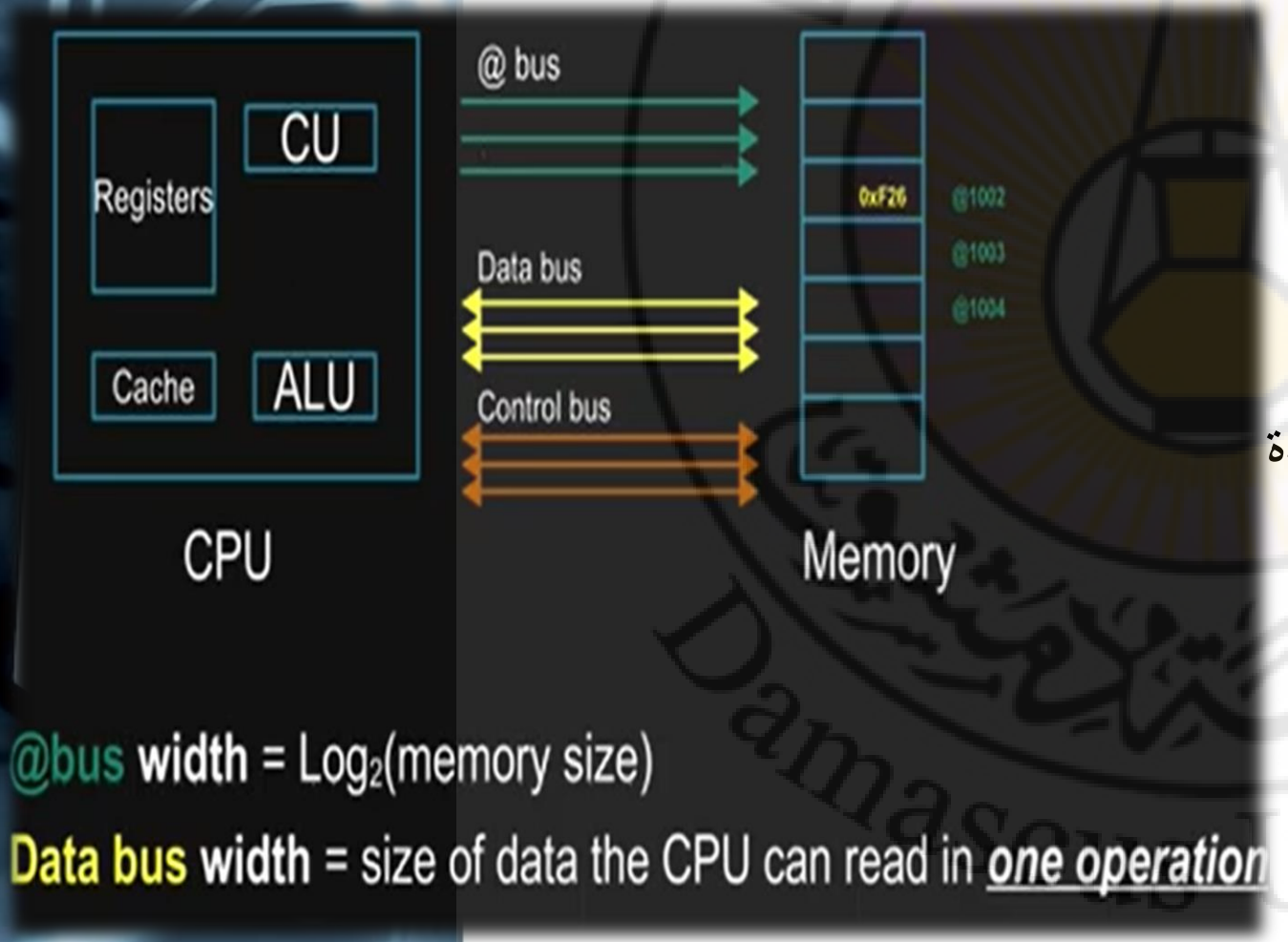

هو وسيلة نقل البيانات بين مختلف العناصر

عند القيام بتنفيذ تعليمة فإن التعليمات تنتقل من الذاكرة إلى المعالج عبر خط البيانات Bus

وأيضا عند الكتابة من المعالج إلى الذاكرة تنتقل هذه البيانات عبر خط البيانات وأيضا إلى المخارج والمداخل عندما نريد الطباعة مثلا فيكون عن طريق خط البيانات (فهو يربط بين المعالج والأجزاء الأخرى )

نأخذ كمثال النواقل بين وحدة المعالجة المركزية والذاكرة .(الذاكرة منقسمة إلى عدة خانات وكل خانة تحمل قيمة معينة ولها عنوان) ، ينقسم الباصbus إلى ثلاث أقسام

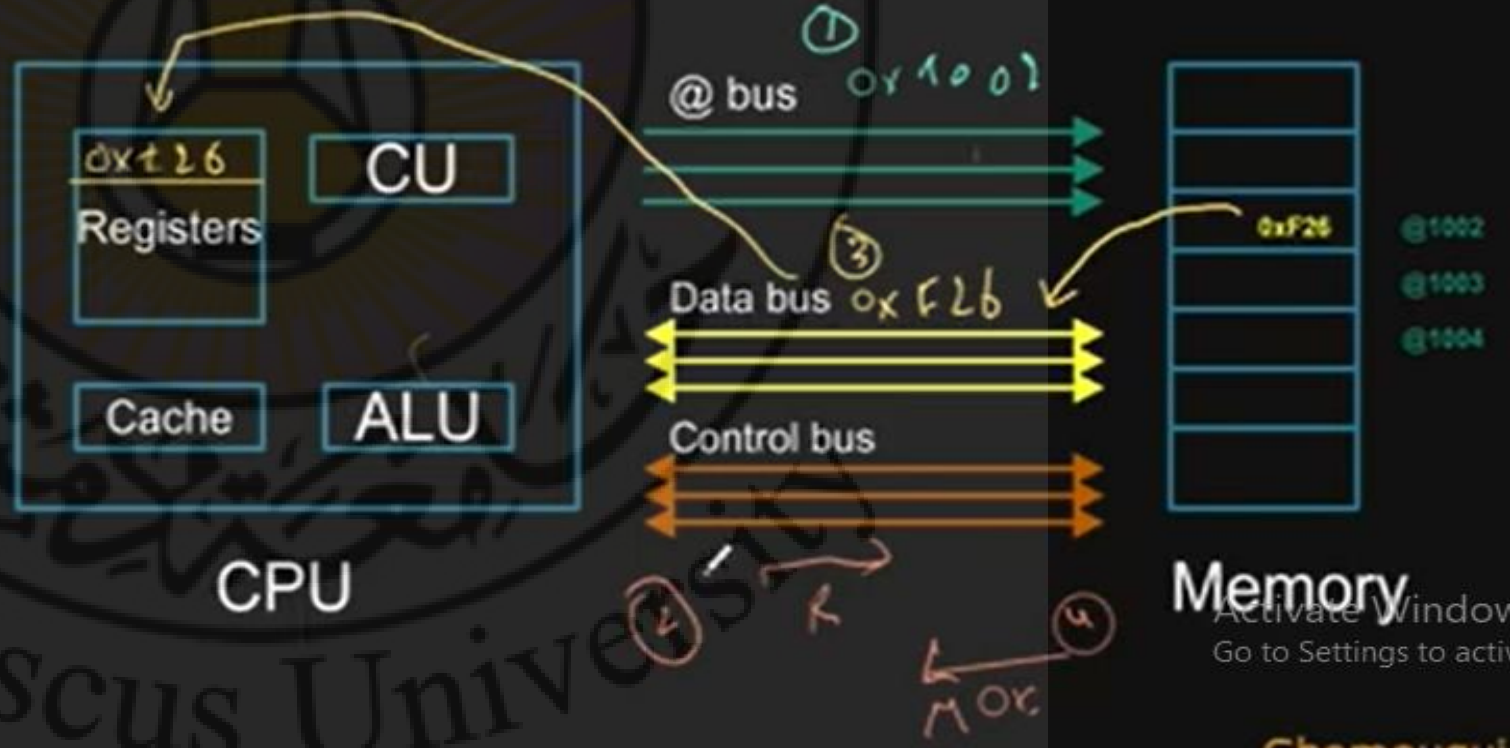
1. **Address bus** : مهمته نقل عنوان المكان المطلوب من الذاكرة للعمل عليه (قراءة أو كتابة) وتكون باتجاه واحد من المعالج إلى الذاكرة ولاتعود عليه
2. **Data bus**: مهمته نقل البيانات (القيم) في الاتجاهين ، في حالة الكتابة من يكون الانتقال من المعالج إلى الذاكرة ، اما في حالة القراءة بالعكس ويعطي عدد البيانات الممكن قراءتها بشكل متزامن
3. **Control bus**: ينقل إشارات معينة خاصة بالقراءة أو الكتابة حسب المطلوب ، أي نقل تعليمات معينة.




عرض الباص :عدد الخطوط الموجودة في الباص ثلاث خطوط
ينقل ثماني قيم مختلفة 000- 001-...
وهي مساحة العنونة ، أي ليس له علاقة بحجم الذاكرة
وعرض الباص يسمح بمعرفة عدد البيانات التي نستطيع نقلها وبالتالي عدد الخانات التي نقرأها

# Example read data from memory

- CPU puts the @ 0x1002 in the @bus
- CPU sends the read signal via the control bus
- Memory puts the content of the @01x002 (0xF26) in the data bus
- Memory sends **data-ready** signal to the cpu via the control bus
- CPU reads the **data** from the data bus and puts it in one of its registers




Activate Windows
Go to Settings to activa

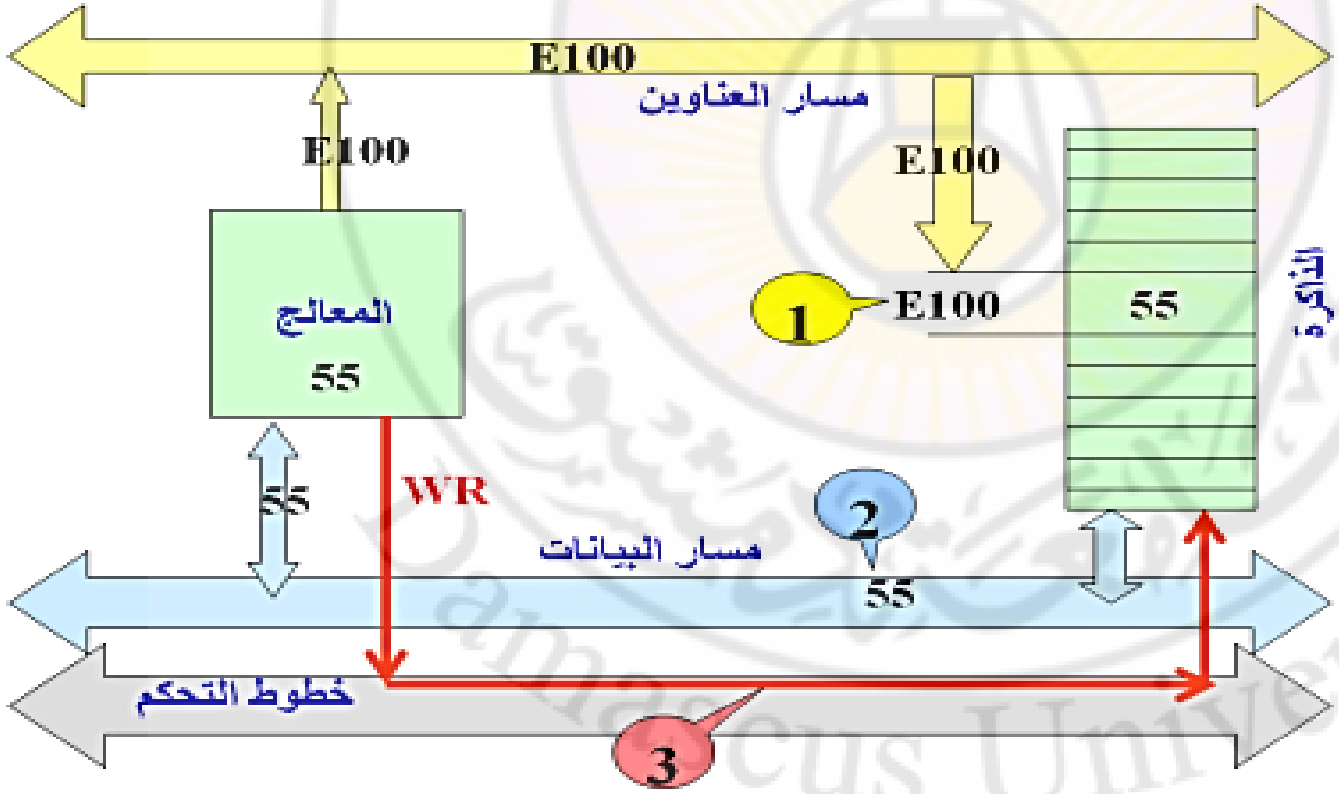
# عملية التسجيل على الذاكرة (الكتابة)

١- يضع المعالج العنوان (١٦ بت) الذى يريد التعامل معه على مسار البيانات (١٦ طرف).

٢- يضع المعالج المعلومة 55H الموجودة بداخله على مسار البيانات.

٣- يقوم المعالج بتنشيط خط التحكم WR بإشارة تخرج منه إلى الذاكرة تخبرها أن الغرض من هذا التعامل هو الكتابة أو التسجيل فى الذاكرة، لذلك يتم تنشيط هذا الخط WR.

لاحظ أن الخطوات الثلاث متعلقة بالمسارات الثلاثة (العناوين، والبيانات، والتحكم)، وبعد تنفيذ هذه الخطوات يتم نقل نسخة من المعلومة 55H وتسجيلها فى العنوان المحدد E100H.

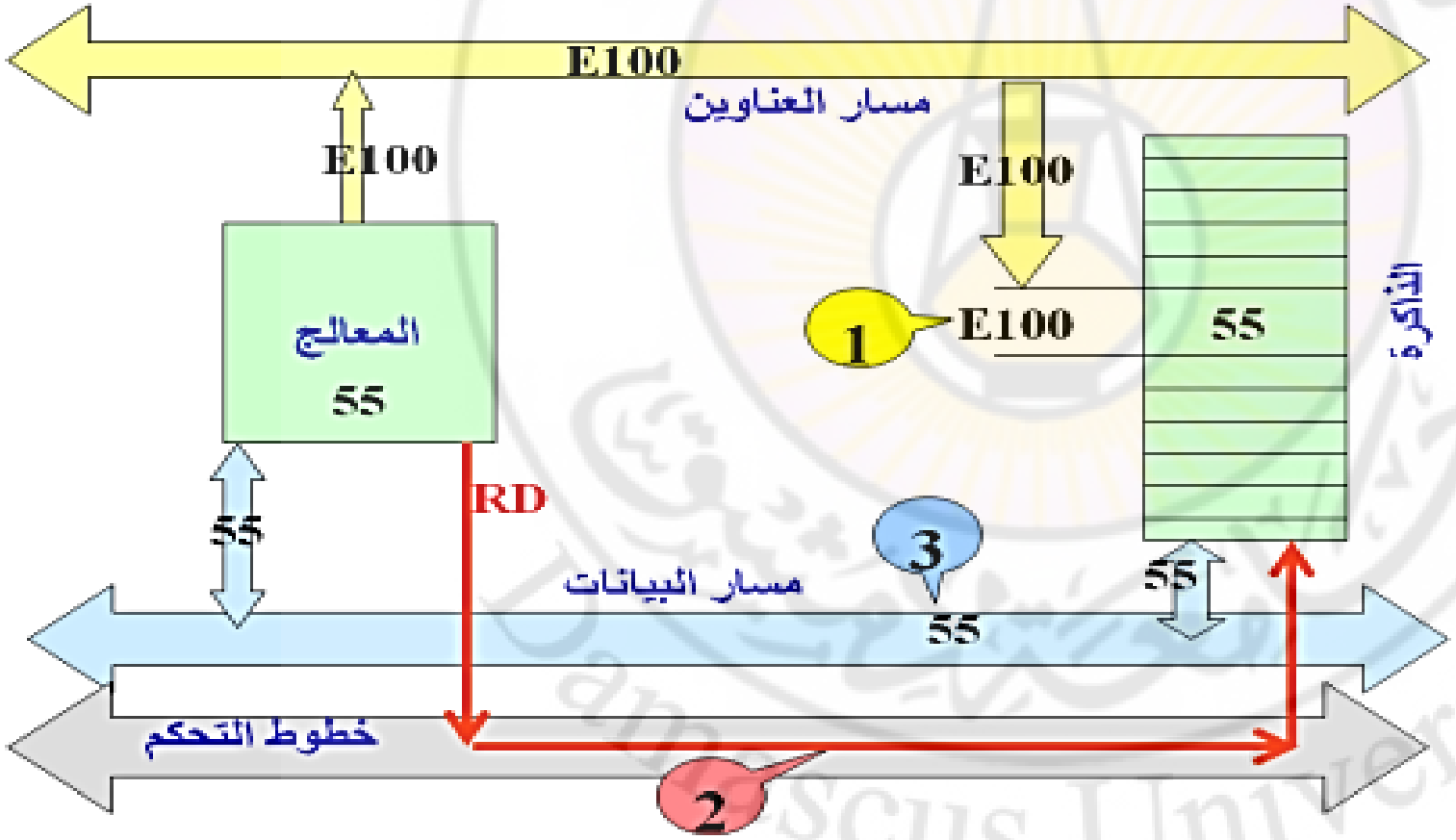


# عملية القراءة من الذاكرة

١- يقوم المعالج بوضع عنوان البايت E100H على مسار العناوين.

٢- هذه المرة يقوم المعالج بتنشيط خط التحكم RD ليخبر الذاكرة بأنه يريد القراءة من هذا العنوان.

٣- على الفور تقوم الذاكرة بوضع نسخة من المعلومة الموجودة في هذه البايت وهي 55H على مسار البيانات، حيث يقوم المعالج بالتقاطها من على مسار البيانات وتسجيلها في المكان المناسب بداخله.



# المتحكمات الصغرية

## بنيان المتحكمات - بنية المتحكم AVR

# نظام توصيل المسارات بين المعالج والذاكرة

# BUSSING SYSTEM

وهما عبارة عن نموذجين تعتمد عليهما أنظمة الكمبيوتر ، والاختلاف الكبير بين هاتين البنيتين ينشأ وفقا للطريقة التي يتم فيها فصل وحدة المعالجة المركزية عن الذاكرة.

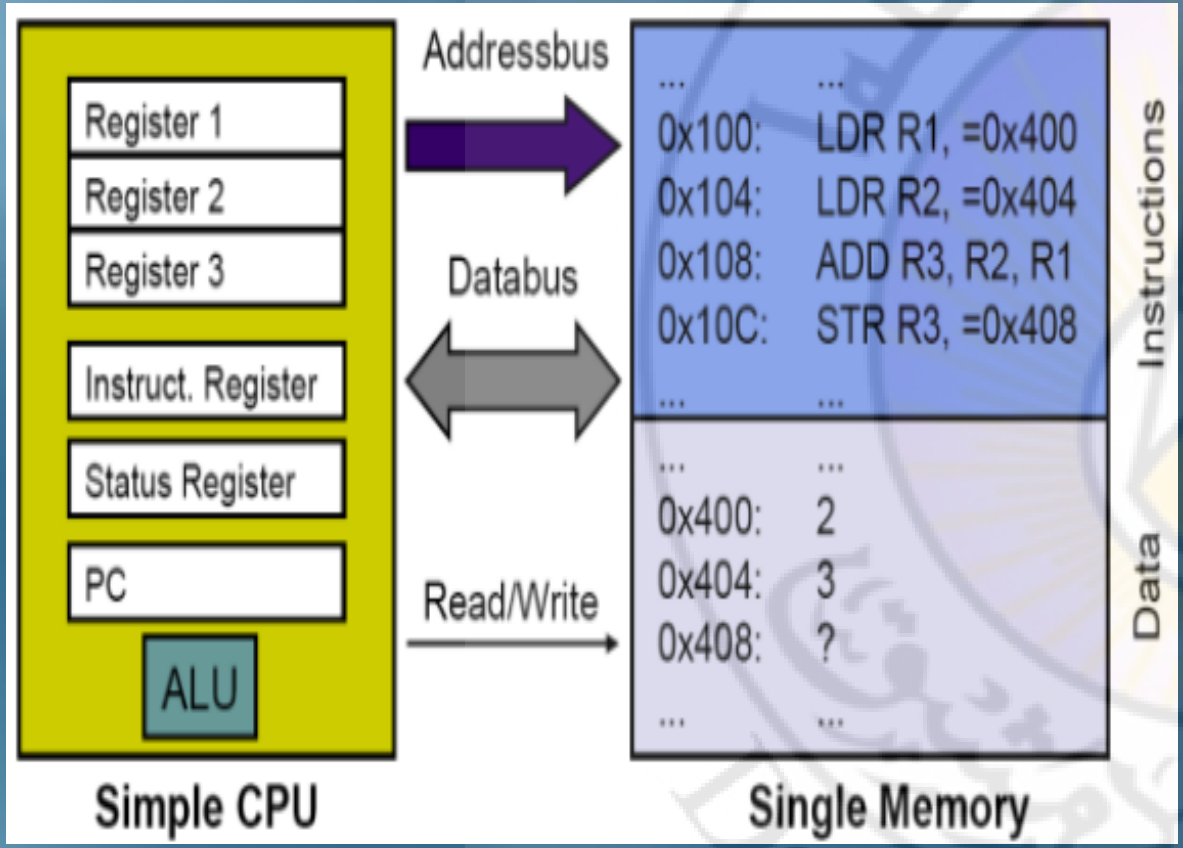
وهي قائمة على تقسيم المسارات بين المعالج والأجهزة المحيطة إلى ثلاث مسارات (العناوين والبيانات والتحكم)

Damascus University

# معمارية VON-NEUMANN

في هذه البنية البرنامج يتم وضعه في الذاكرة وتقوم وحدة المعالجة المركزية باستدعاء أوامر البرنامج واحدا تلو الآخر حتى يتم الانتهاء من البرنامج ، لذلك تمتاز هذه الطريقة بوجود ذاكرة واحدة تتعامل معها وحدة المعالجة المركزية ، تحتوي البرامج والبيانات معا ، من خلال المسارات الثلاثة.

**معمارية Von-Neumann:** تعتمد على ناقل وحيد لنقل البيانات والتعليمات بين الذاكرة ووحدة المعالجة المركزية بحيث :

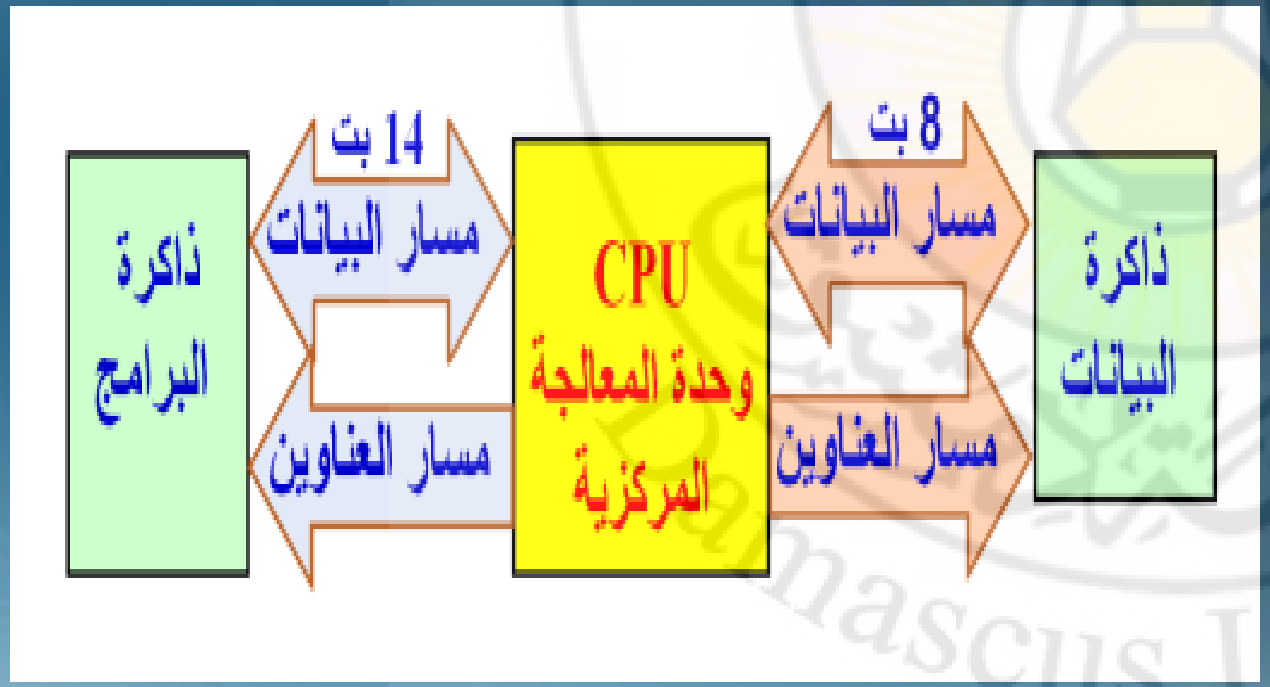


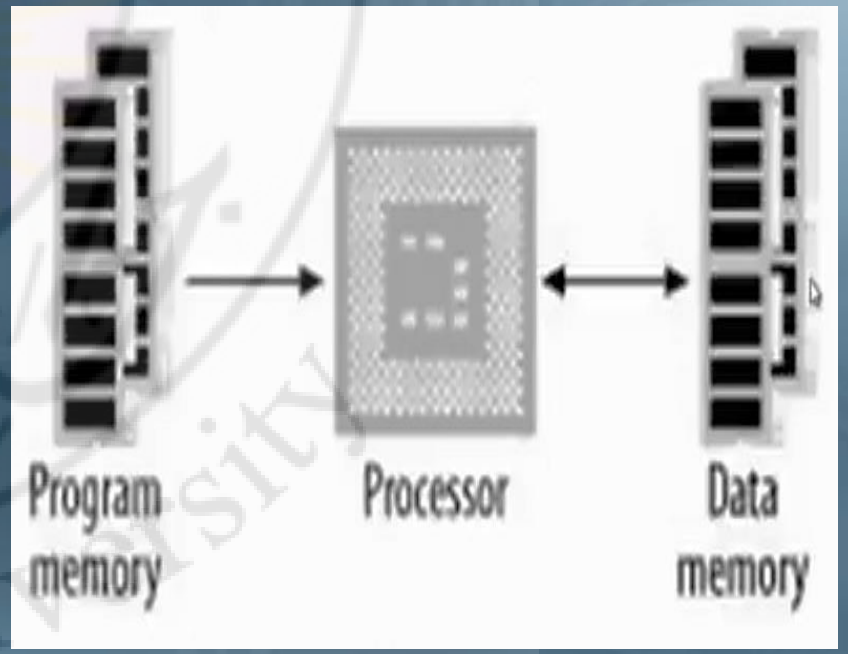

1. يقوم المعالج بجلب كود التعليمات من الذاكرة .
2. يقوم بقراءة البيانات من الذاكرة.
3. إجراء العمليات على البيانات.
4. إعادة كتابة تلك البيانات على الذاكرة.
5. وحدة التعامل مع الذاكرة هي البايت byte (8bit) والكثير من أوامر وحدة المعالجة تتكون شيفراتها من أكثر من

بايت في الذاكرة ، ولذلك فإن مثل هذه الأوامر ستقوم وحدة المعالجة المركزية بإحضارها في أكثر من عملية قراة من الذاكرة ، وبذلك تأخذ وقت كبير وتكون عملية المعالجة بطيئة.

## معمارية هارفارد HARVARD:

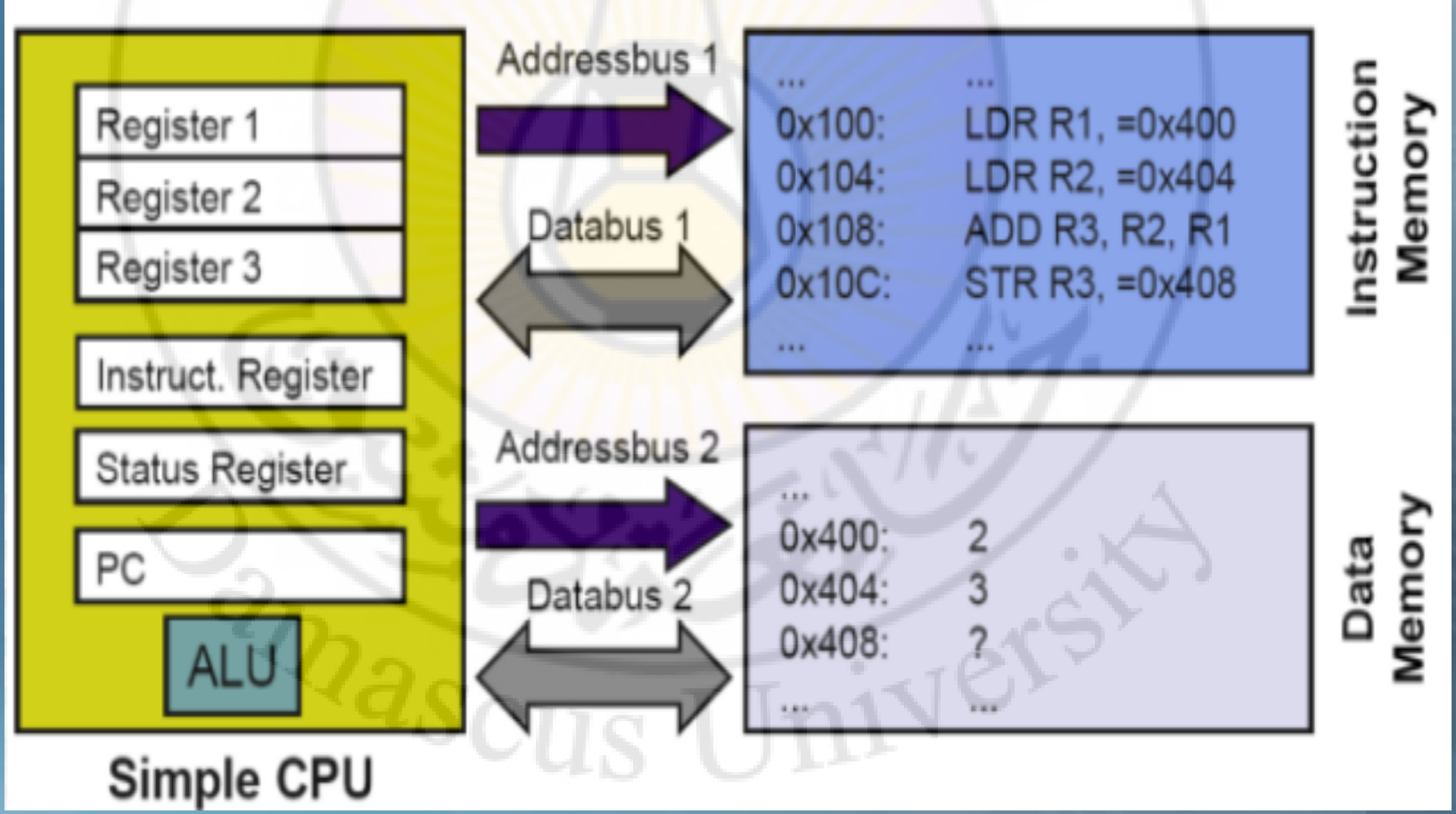
تختلف ذاكرة البيانات عن ذاكرة التعليمات ولكل واحدة خطوط عنونة وتحكم وممر معطيات مختلف عن الأخرى ، أي تم فصل الذاكرة على جزأين جزء خاص بالبيانات يتم التعامل معه بوحدة البايت ، ويتصل مع وحدة المعالجة المركزية من خلال مسار بيانات ومسار عناوين خاص خاص بهذا الجزء من الذاكرة ، اما الجزء الثاني هو ذاكرة خاصة بالبرنامج فقط تكون وحدة التعامل معه هي عدد من البتات يساوي أكبر عدد من بتات أي أمر من الأوامر.



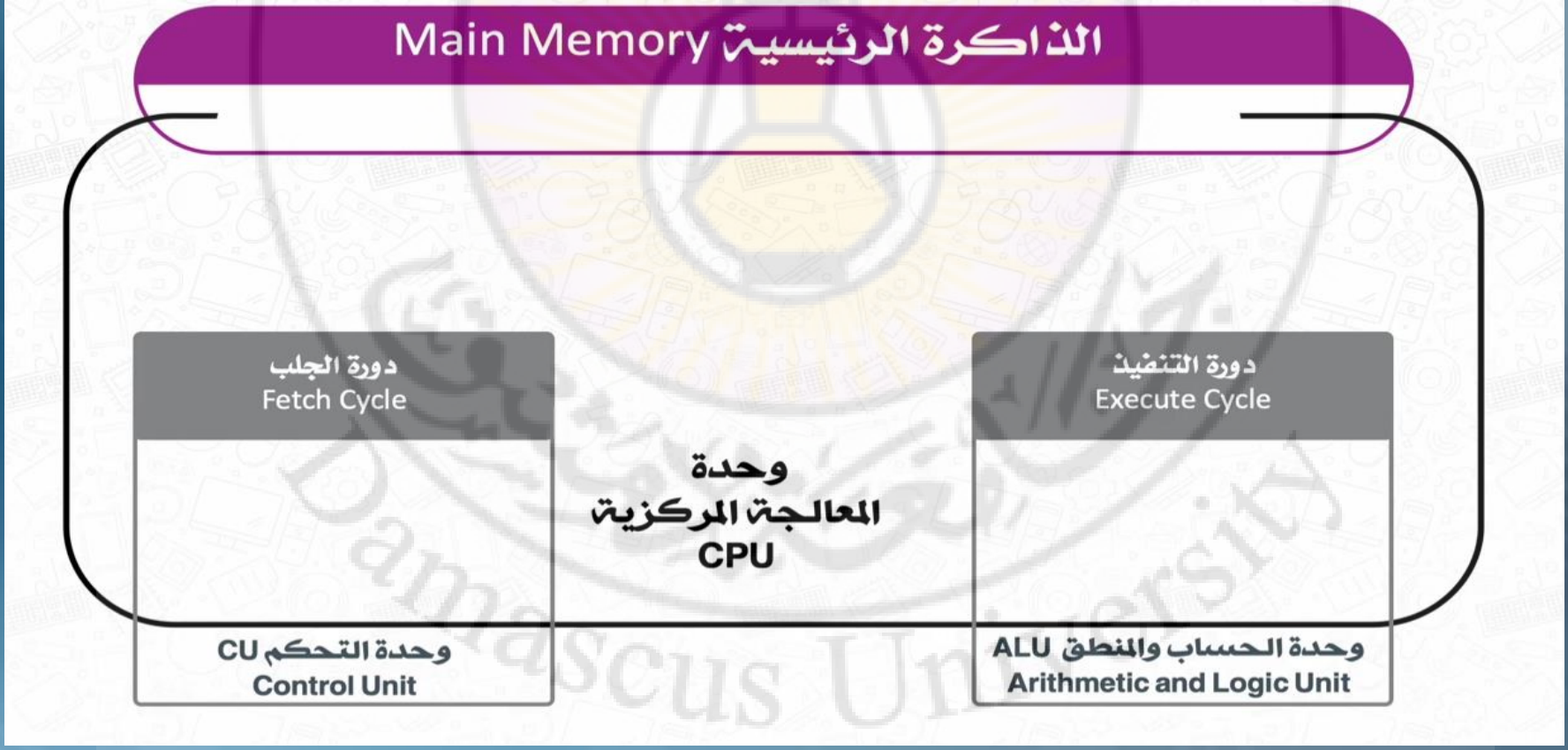
## معمارية Harvard:

ناقلين منفصلين أحدهما لنقل التعليمات والآخر لنقل البيانات ، وتختلف ذاكرة البيانات عن ذاكرة التعليمات حيث أن لكل ذاكرة خطوط عنونة وتحكم وممر معطيات مختلفة ، وبالتالي تتم عملية قراءة التعليمات والبيانات في نفس الوقت



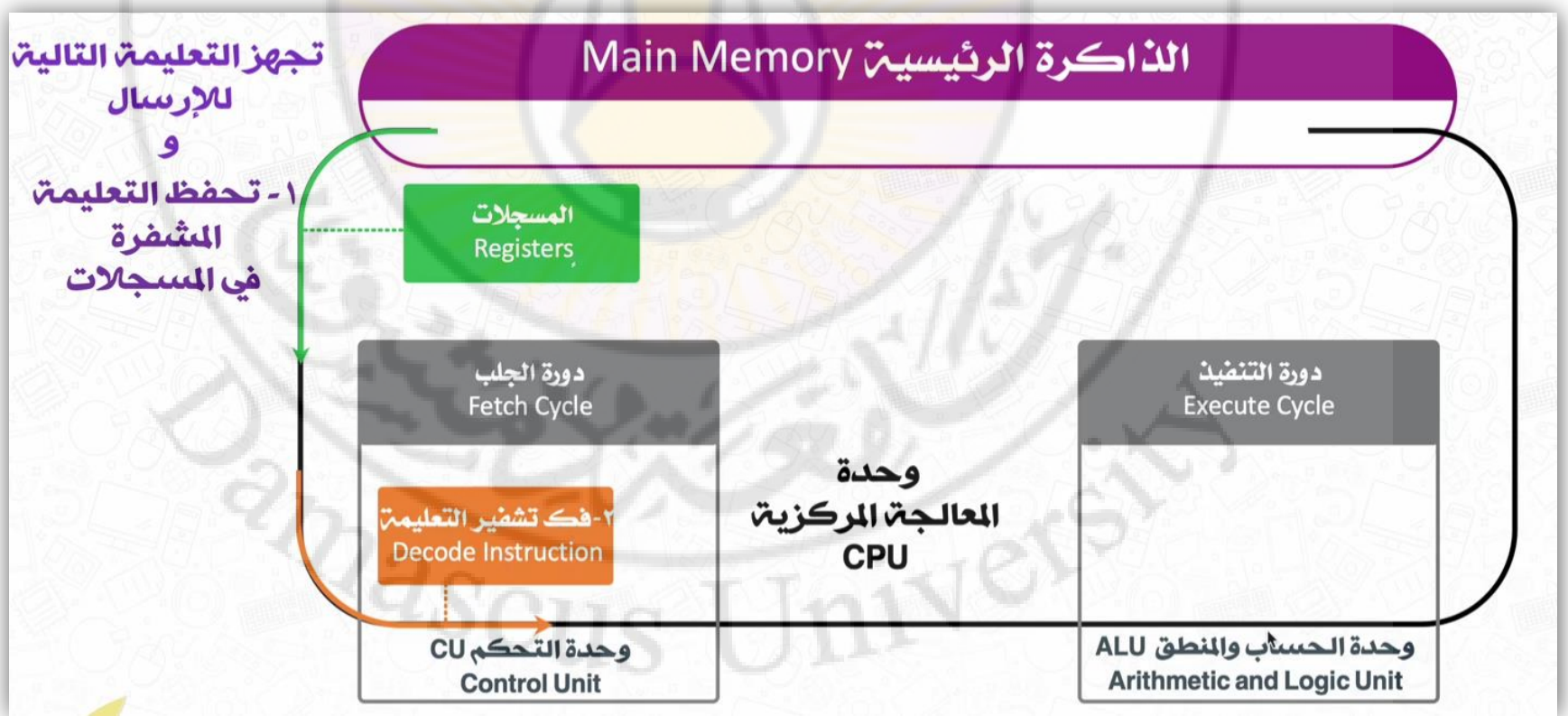
# دورة الجلب والتنفيذ
# FETCH – EXECUTE CYCLE

تحصل مابين وحدة المعالجة المركزية CPU والذاكرة الرئيسية RAM



حيث أن وحدة التحكم المركزية تحتوي وحدتين أساسيتين وهما وحدة التحكم ووحدة الحساب والمنطق .

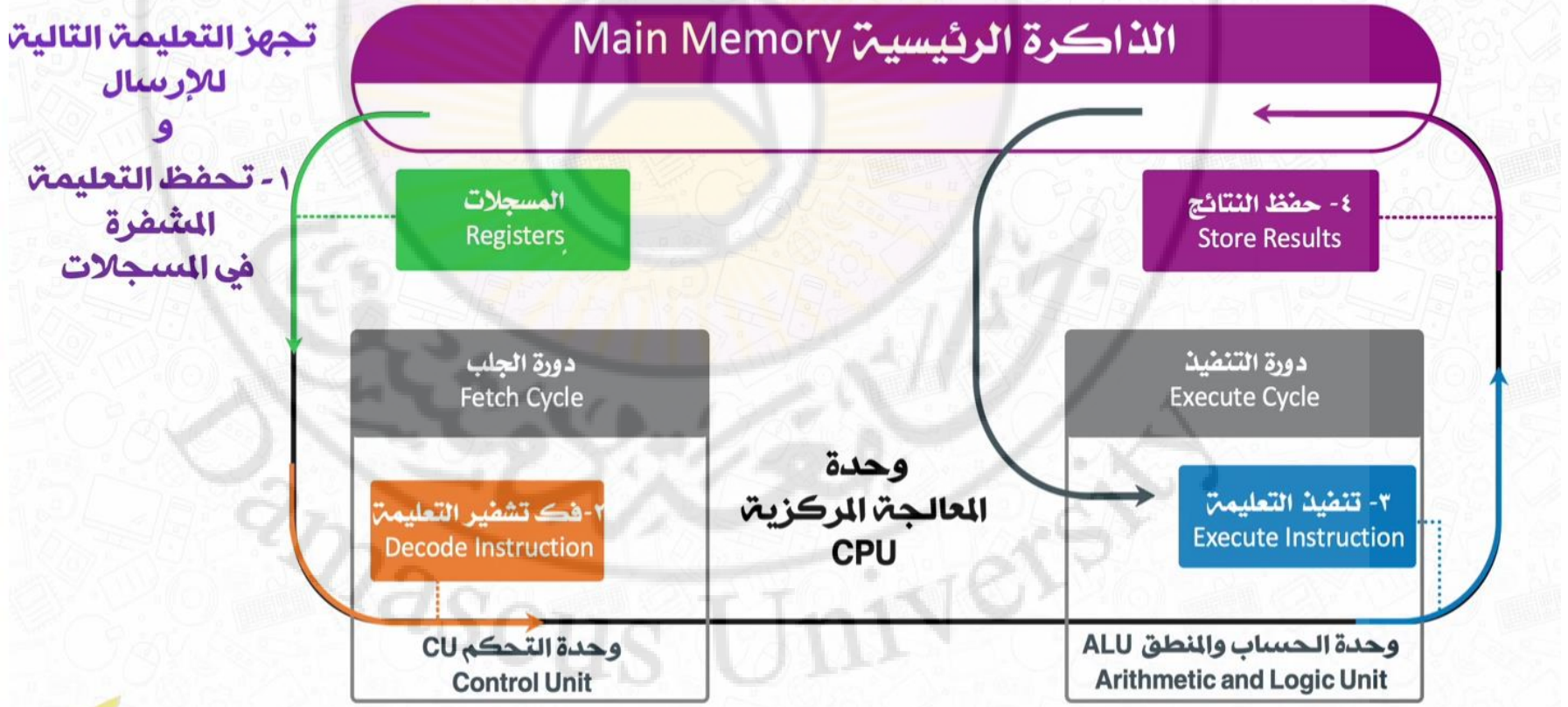
**دورة الجلب :** تحدث في وحدة التحكم حيث يقوم المعالج بجلب التعليمة من الذاكرة الرئيسية ثم يقوم بحفظها داخل المسجلات (حيث ان المسجلات عبارةعن ذواكر موجودة داخل المعالج cpu ) يتم حفظ التعليمة المشفرة داخل المسجلات ومن ثم يتم تجهيز التعليمة التالية للإرسال .



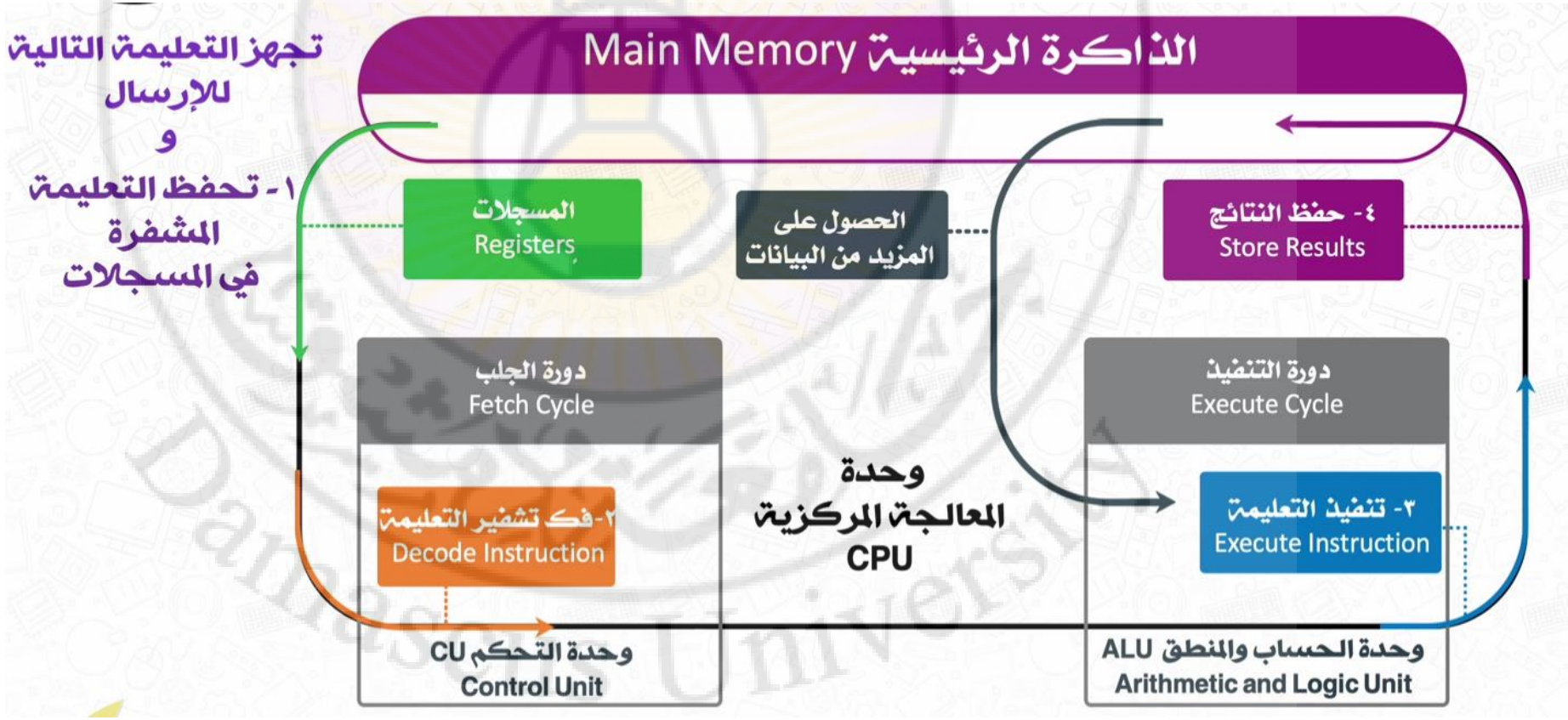
وبعدها يتم فك تشفير التعليمة داخل وحدة التحكم cu لتدخل إلى وحدة الحساب والمنطق ALU حيث تبدأ دورة تنفيذ التعليمة ..

**دورة التنفيذ :** يتم تنفيذ التعليمة داخل وحدة الحساب والمنطق ، وبعد ذلك تحفظ النتائج داخل الذاكرة الرئيسية .

إذا كان هناك أوامر أخرى مثل ( عمليات حسابية +،-،*،/ أو منطقية ...OR,XOR)سوف تقوم الذاكرة الرئيسية بالحصول على المزيد من البيانات من دورة التنفيذ حتى تقوم بتنفيذ هذه العملية ، ثم يتم حفظها ...



ثم تنتقل إلى العملية التي تم تجيزها سابقا ليتم تكرار جميع المراحل من جديد
(جلب التعليمة المجهزة إلى المسجلات – فك شيفرة التنفيذ في وحدة التحكم – تنفيذ العملية في وحدة الحساب والمنطق – الحصول على المزيد من البيانات من الذاكرة والرجوع إلى دورة التنفيذ- حفظ النتائج في الذاكرة ) وهكذا.....
أي أنها تعمل بشكل دوري بدون توقف وهذا كله يحصل خلال أجزاء من الثانية ...
يقوم المعالج بمليار عملية في الثانية الواحدة وذلك بالنسبة ل(1 غيغاهيرتز ) .



# تنفيذ الأوامر في المتحكمات

- بما أن وحدة المعالجة المركزية عندما تقوم بتنفيذ أي أمر على دورتي الجلب والتنفيذ ، في هذا النظام عندما تكون الوحدة مشغولة في دورة الجلب فإنها لاتفعل أي شيء آخر ، وعندما تكون مشغولة في دورة التنفيذ فإنها لاتفعل أي شيء آخر ، ولذلك فإن الزمن الكلي لتنفيذ أي أمر يكون مجموع دورتي الجلب والنفيذ معاً. أي انه بفرض أن المعالج يقوم بجلب الأمر في نبضة تزامن واحدة فإن الزمن الكلي لتنفيذ الأمر في هذا المعالج سيكون اثنين من نبضات التزامن ، لقد كانت هذه التقنية موجودة في الأجيال الأولى من المعالجات (bit8) ، ولكن تم استبداله فيما بعد بفكرة خطوط الإنتاج او خطوط الأنابيب pipelines

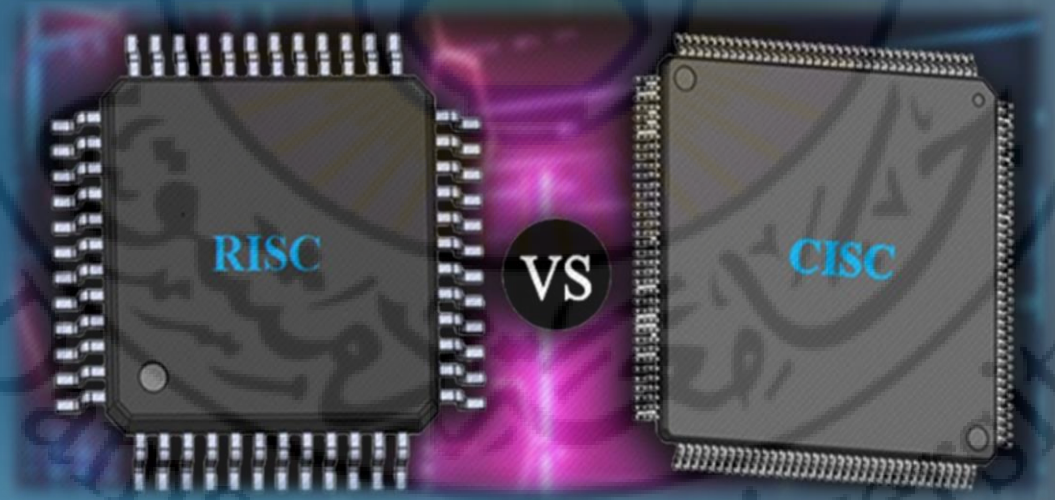
- المتحكمات التي تعمل بهذا النظام ومنها المتحكمات AVR تقسم عملية تنفيذ أي أمر إلى مرحلتين منفصلتين ، كلاهما تعملان في نفس الوقت وباستمرار ، واللتين هما مرحلة الجلب والتنفيذ ....

- الجديد هنا أنه في أثناء انشغال وحدة الجلب في إحضار الأمر رقم n ، فإن وحدة التنفيذ تكون مشغولة في نفس الوقت في تنفيذ الأمر السابق وهو الأمر n-1 بعد الانتهاء فإن الأمر رقم n ينتقل إلى مرحلة التنفيذ ويدخل في نفس الوقت الأمر رقم n+1 في مرحلة الجلب (كما راينا في مرحلة الجلب والتنفيذ) ....

# بنى مسجلات التحكم

## معمارية مجموعة التعليمات

## CISC & RISC



- كما ذكرنا سابقا تقوم وحدة المعالجة المركزية بتنفيذ عدد من العمليات المعقدة باستخدام التعليمات المرمزة ، وخلال التنفيذ تقوم وحدة التحكم بجلب التعليمة من الذاكرة وتفسيرها في وحدة التعليمات IU قبل الشروع في تنفيذها ، على هذا فإن زمن التنفيذ يتضمن الزمن اللازم لفك ترميز التعليمات . ويجري تقصير هذا الزمن ببدء جلب التعليمة التالية من الذاكرة خلال طور تنفيذ التعليمة السابقة .

- تسمى وحدات المعالجة المستخدمة لهذا النوع من التعليمات بمعالجات CISC وهي اختصارا لـ Complex Instruction Set Computer يحتوي هذا النوع من المعالجات مابين 100-400 تعليمة تقوم بعمليات معقدة نسبيا (مثلا عنونة غير مباشرة للذاكرة ) وتتضمن عدد محدود من المسجلات المساعدة.
- كما يوجد نوع آخر من وحدات المعالجة المركزية او المعالجات التي تستخدم تعليمات محدودة وبسيطة وتسمى بمعالجات RISC الذي يحتوي على عدد محدود من التعليمات البسيطة أقل من 100 تعليمة تنفذ مباشرة دون الحاجة على تفسير

وتسمى معمارية مجموعة التعليمات ISA(instruction set architecture) وتعرف بأنها الواجهة الفاصلة بين البرامج والمكونات المادية ، وهي تعرف صيغة العمليات وأنواعها التي تقبلها وحدة المعالجة المركزية cpu إضافة إلى أنواع المعاملات وأنماط العنونة والعناوين الذاكرية التي يمكن الوصول إليها.

من وجهة نظرية بنيان الحاسب ، يمكن تصنيف المعالجات المصغرة إلى فئتين هما CICS (Complex Instruction Set Computer) مجموعة التعليمات المعقدة وRISC (Reduced Instruction Set Computer) أي مجموعة التعليمات المخفضة .

**معمارية CISC:** تنفذ التعليمات عن طريق برامج صغيرة تحوي عدد من التعليمات المعقدة ، وتستغرق وقت طويل عند تنفيذ المهمة ، هدفها توفير مجموعة التعليمات التي تدعم اللغات عالية المستوى ومن ثم تسهيل عملية الترجمة compiling.

***معمارية RISC:*** تستخدم مجوعة التعليمات ذات الحجم الصغير والمحسنةعلى نحو كبير ، بالإضافة إلى القدرة على تخصيص مجموعة التعليمات.

يقلل تصميم CISC من عدد التعليمات في البرنامج عن طريق توفير تعليمات خاصة قادرة على أداء عمليات معقدة ، بينما يقلل تصميم RISC من متوسط عدد دورات الساعة المطلوبة لتنفيذ تعليمة واحدة .

**مثلا:**

- تعليمة تقوم بمسح محتويات الحجرة 100h من الذاكرة وذلك باستخدام معالج CISC:

```
clear 0x0100
```

- برنامج مكافئ باستخدام معالج RISC:

```
load R , 0x0000
store R , 0x0100
```

Damascus University

# مجموعة تعليمات وحدة المعالجة المركزية CPU Instruction

- يمتاز كل معالج بمجموعة بمجموعة من التعليمات التي يمكن ان تكون مكونة من ثُمانية واحدة byte إلى ثلاث ثُمانيات او اكثر حسب نوع المعالج .
- فالتعليمة هي سلسلة من الأرقام الثنائية بلغة الآلة ، وتتضمن الثمانية ثمان خانات ثنائية يمكنها تمثيل 256 تعليمة مختلفة ، وتقسم هذه التعليمات إلى مجموعات ذات وظائف متشابهة أهمها :

•التعليمات الحسابية : وهي مجموعة التعليمات التي تحقق عمليات : الجمع ، الطرح ، الضرب ، القسمة ، زيادة القيمة المخزنة في سجل ما بمقدار واحد ، إنقاص القيمة المخزنة في سجل ما بمقدار واحد .

• التعليمات المنطقية : وهي مجموعة من التعليمات التي تحقق العمليات المنطقية AND, NOT, OR, XOR والمقارنة بين محتوى المراكم وسجل آخر . يضاف إلى هذه العمليات عمليات إزاحة نحو اليمين لمحتويات سجل ما والإزاحة نحو اليسار ، وتدوير محتوى السجل.

• تعليمات نقل المعطيات : وهي مجموعة التعليمات التي تحقق عمليات نقل المعطيات من مواقع محددة في الذاكرة ، إلى السجلات ضمن وحدة المعالجة المركزية ، كما تحقق النقل بين السجلات الداخلية في الوحدة . يضاف إليها تعليمات إخراج المعطيات من الوحدة وإخراج المعطيات عبر بوابات الدخل/ الخرج المتصلة بالمعالج .

•تعليمات التفريع أو القفز : وهي مجموعة التعليمات التي تغير من تتابع تنفيذ البرنامج . مثل تعليمات القفز المشروطة بنتيجة عملية سابقة أو بقيمة معينة أو القفز غير المشروط . كما تضم تعليمات طلب البرامج الجزئية والعودة من البرامج الجزئية إلى البرنامج الرئيسي.

لايمكن كتابة البرنامج بلغة الآلة نظرا لصعوبة التعامل مع نظام الترقيم الثنائي (واحد-صفر) ، لذلك تم تعريف لغة برمجة تسمى لغة التجميع Assembly Language ، تعطي رمزا مكون من ثلاثة أو أربعة حروف لكل تعليمة ، وتكتب المعطيات بالشكل الست عشري ، والجدول التالي يبين أمثلة توضيحية عن لغة المجمع

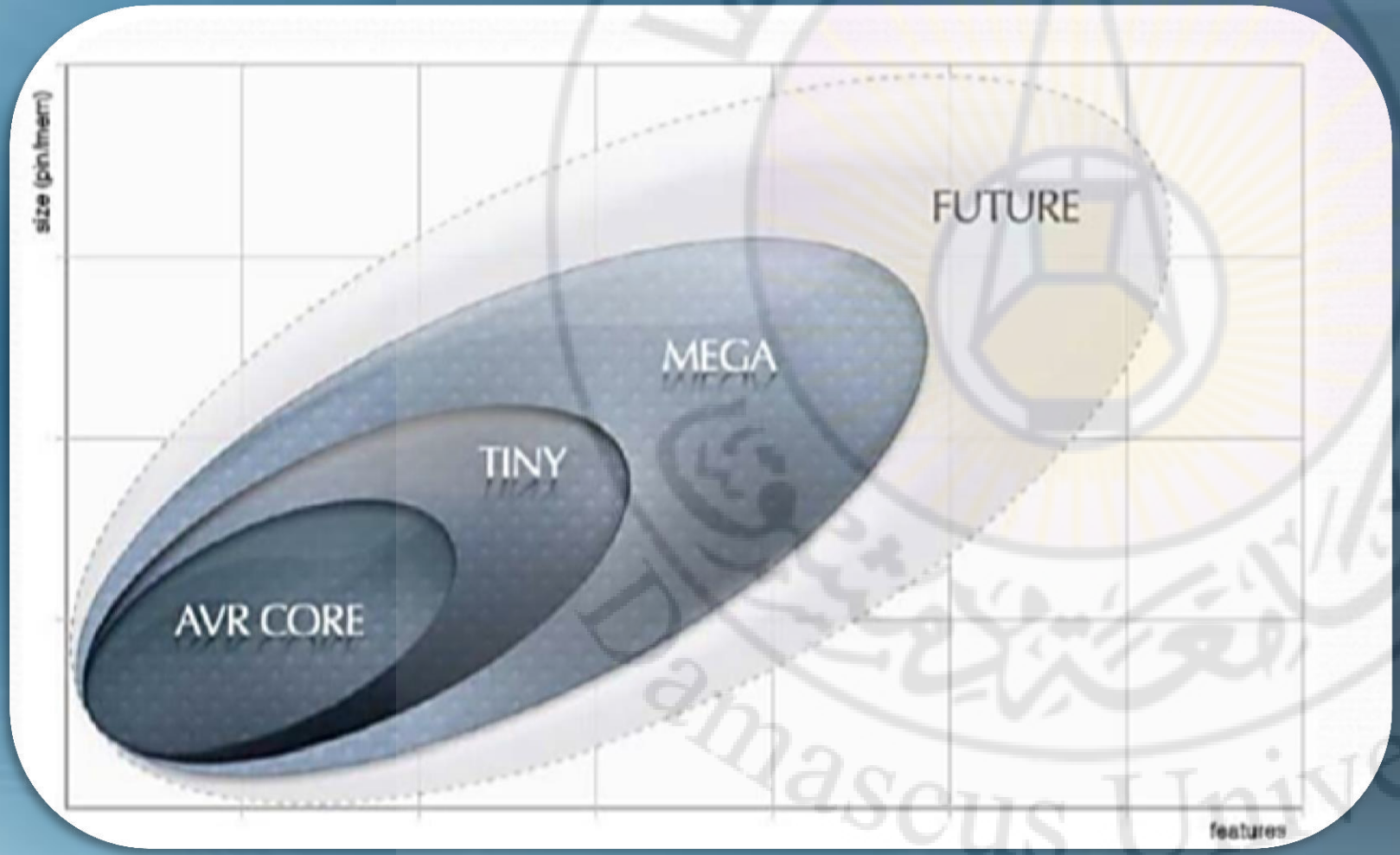
| التعليمة | العمليات التي تنفَّذ |
|---|---|
| ADD AX,BX | جمع محتوى السجل BX إلى سجل المراكم ووضع الناتج في المراكم |
| SUB AL,CL | طرح محتوى السجل CL من محتوى الجزء الأدنى من سجل المراكم |
| XOR CH,24 | تنفيذ العملية المنطقية XOR بين محتوى السجل CH والقيمة 24 |
| XOR CX,CX | تصفير السجل CX (أي تحميله بالقيمة صفر) |
| NOT BX | إجراء عملية عكس منطقية على القيمة المحتواة في السجل BX وتسجيل الناتج في السجل BX |
| MOV BX,AX | نقل محتوى السجل AX (المراكم) إلى السجل BX |
| MOV DX,AX | نقل محتوى سجل المراكم إلى السجل DX |
| DEC DX | إنقاص محتوى السجل DX بمقدار واحد |

# مقارنة بين أشهر عائلات المتحكمات المصغرة 8bit:

Unleash your Creativity!

| الميزة / العائلة | AVR (Atmel) | 8051 (Intel) | PIC (Microchip) | HC11 (Motorola) |
|---|---|---|---|---|
| البنية الأساسية | Harvard | Von-Neumann | Harvard | Von-Neumann |
| تقنية النواة | RISC | CISC | RISC | CISC |
| تردد التشغيل الأعظمي | 20MHz | 24MHz | 20MHz | 8MHz |
| نبضة لكل تعليمة | 1 | 12 | 4 | 8 |
| تعليمة في الثانية | 20MIPS | 2MIPS | 5MIPS | 1MIPS |
| عدد التعليمات | 132 | 215 | 32 | 200 |
| حجم ذاكرة البرنامج | 256KB | 32KB | 64KB | 32KB |
| عرض ناقل التعليمات | 16-bit | 8-bit | 12-bit | 8-bit |

# متحكمات AVR




**عوائل متحكمات AVR:**

1. سلسلة Tiny: صغيرة الحجم والذاكرة ، للتطبيقات الصغيرة .
2. سلسلة Atmega: حجم ذاكرة متوسط للتطبيقات المتوسطة والمعقدة.
3. سلسلة Xtmega: تستخدم للتطبيقات المعقدة التي تحتاج سرعة عالية وحجم ذاكرة كبير

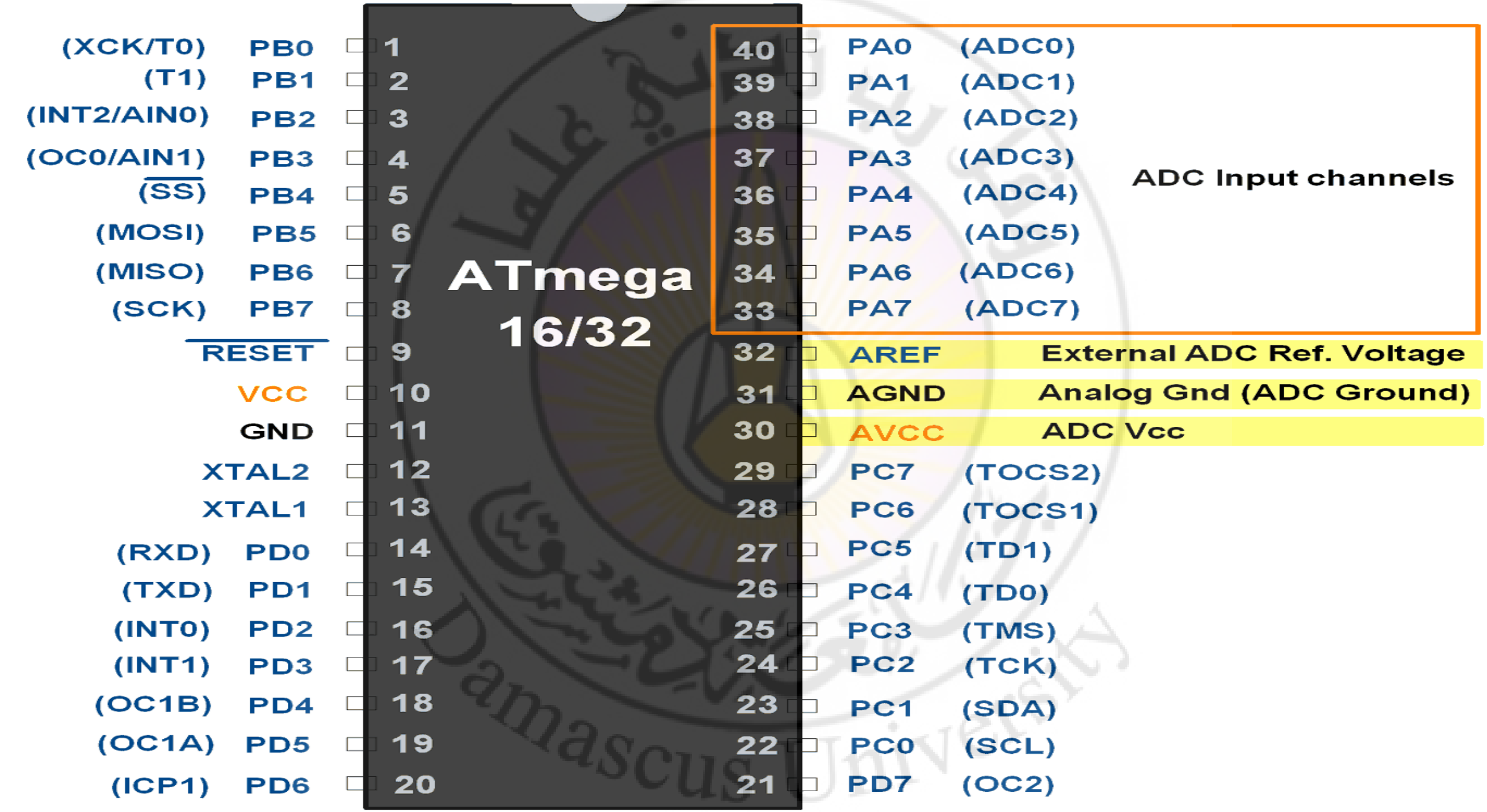
## أهم مميزاتها

- من منتجات شركة ATMEL الأمريكية
- تم تطويرها في مختبرات الشركة الموجودة في النرويج 1990
- تعتبر من أكثر المتحكمات المصغرة انتشاراً
- تفوّقت بشكلٍ كبير على العديد من نظيراتها من متحكمات bit-8
- تستخدم البنية Harvard + RISC التي تتميز بالأداء العالي وبالطاقة المنخفضة
- احتوت قائمة التعليمات في متحكمات AVR على 132 تعليمة
- يُنفذ معظمها خلال دورة آلة واحدة (1-cycle)
- حتى 20 مليون تعليمة في الثانية الواحدة
- إمكانية برمجة المتحكم دون فصلة عن النظام (In-system Programming)

# تكويد المتحكم AVR

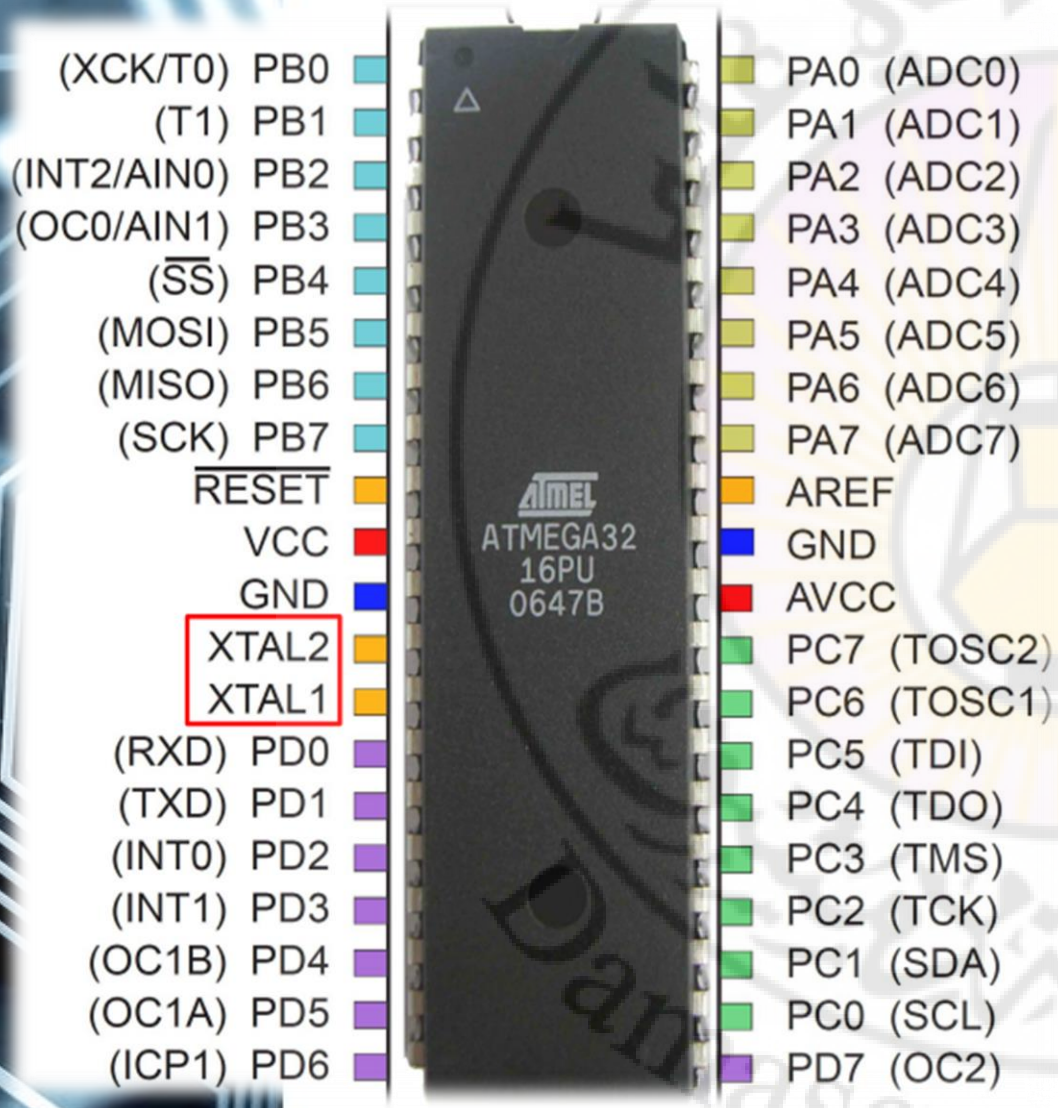
# البنية الداخلية لعائلة المتحكمات AVR

## متحكم AVR Atmega16


(XCK/T0) PB0 1
(T1) PB1 2
(INT2/AIN0) PB2 3
(OC0/AIN1) PB3 4
(SS) PB4 5
(MOSI) PB5 6
(MISO) PB6 7
(SCK) PB7 8
RESET 9
VCC 10
GND 11
XTAL2 12
XTAL1 13
(RXD) PD0 14
(TXD) PD1 15
(INT0) PD2 16
(INT1) PD3 17
(OC1B) PD4 18
(OC1A) PD5 19
(ICP1) PD6 20
ATmega 16/32
40 PA0 (ADC0)
39 PA1 (ADC1)
38 PA2 (ADC2)
37 PA3 (ADC3)
36 PA4 (ADC4)
35 PA5 (ADC5)
34 PA6 (ADC6)
33 PA7 (ADC7)
ADC Input channels
32 AREF External ADC Ref. Voltage
31 AGND Analog Gnd (ADC Ground)
30 AVCC ADC Vcc
29 PC7 (TOCS2)
28 PC6 (TOCS1)
27 PC5 (TD1)
26 PC4 (TD0)
25 PC3 (TMS)
24 PC2 (TCK)
23 PC1 (SDA)
22 PC0 (SCL)
21 PD7 (OC2)

## أقطاب المتحكمات المصغرة :

- أقطاب التغذية الرقمية (VCC, GND).
- أقطاب التغذية التشابهية (AVCC, AGND).
- أقطاب الدخل والخرج (GPIOs).
- أقطاب البرمجة (SPI).
- أقطاب المقاطعات الخارجية (INTx).
- أقطاب مصادر التوقيت (XTAL1, XTAL2).
- أقطاب المؤقتات/العدادات (T0, T1, T2).
- أقطاب المبدلات التشابهية الرقمية (ADC).
- أقطاب النوافذ التسلسلية (UART,SPI,...)
- أقطاب المقارن التشابهي (AIN0,1).
- أقطاب إشارات PWM (OCx).
- أقطاب نافذة المراقبة JTAG (Debug).

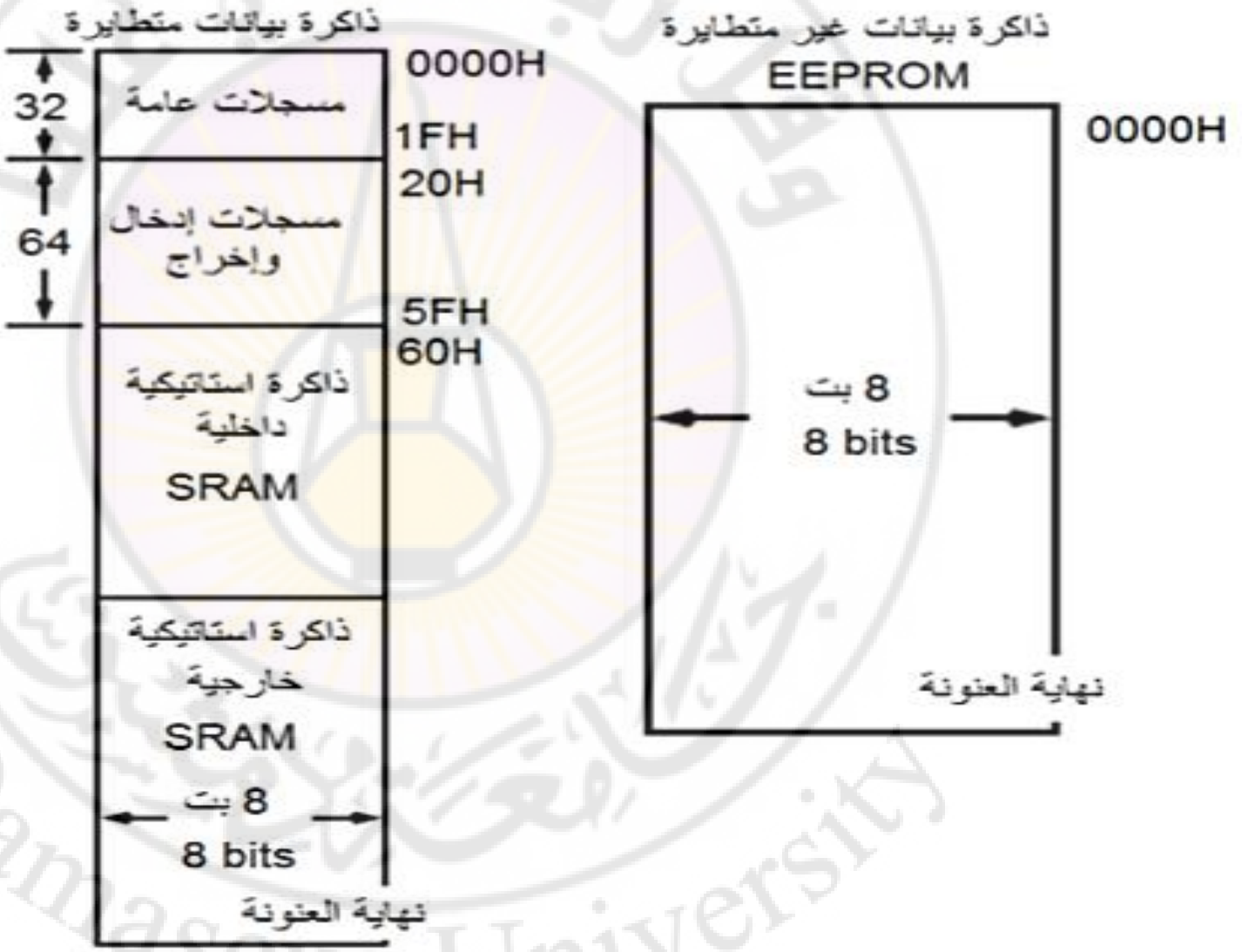


# بنيان و مكونات متحكمات AVR

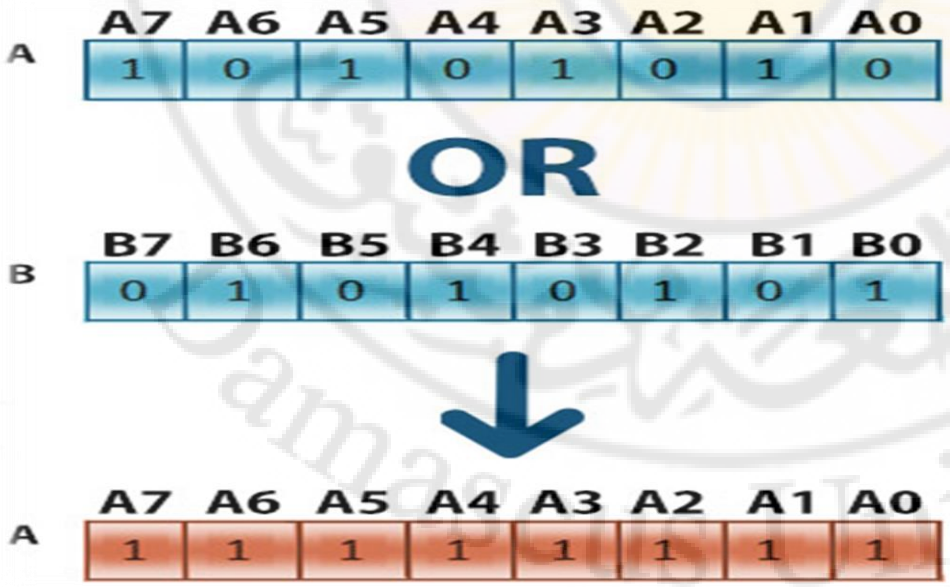
# البنية العامة لمتحكمات AVR

- تعتمد هذه المتحكمات على معمارية هارفارد التي تستخدم ذاكرة خاصة للبرنامج (لكي تخزن التعليمات) من النوع Flash ROM التي لاتمحى بانقطاع التغذية الكهربائية . وذاكرة أخرى من النوع SRAM التي تتلاشى بانقطاع التغذية الكهربائية ، ومن ضمن ذاكرة SRAM يوجد مسجلات الأغراض العامة ومسجلات التحكم ، ويوجد ذاكرة ثالثة EEPROM يمكن استخدامها من قبل المبرمج لتخزين المتحولات الهامة التي يجب ان تبقى مخزنة بعد انقطاع التغذية الكهربائية ...

أقسام ذاكرة البيانات في متحكمات AVR:



- تعتمد معمارية RISC ، حيث تركزعلى مجموعة تعليمات بسيطة ، وبالتالي تكون أسرع في التنفيذ (لكن الكود البرمجي يكون أطول) ، حيث يتم تنفيذ معظم التعليمات في نبضة ساعة واحدة - حيث ان المتحكم الذي يعمل بتردد 1MHZ يستطيع تنفيذ مليون تعليمة تقريبا في الثانية الواحدة -
- وتعتمد على المسجلات في تنفيذ التعليمات حيث تكون معاملات التعليمة (الدخل) مخزن في المسجلات وأيضا يتم تخزين النتيجة (خرجa التعليمة) في مسجل آخر ، حيث يحتوي 32 مسجل أغراض عامة كما في المثال التالي :

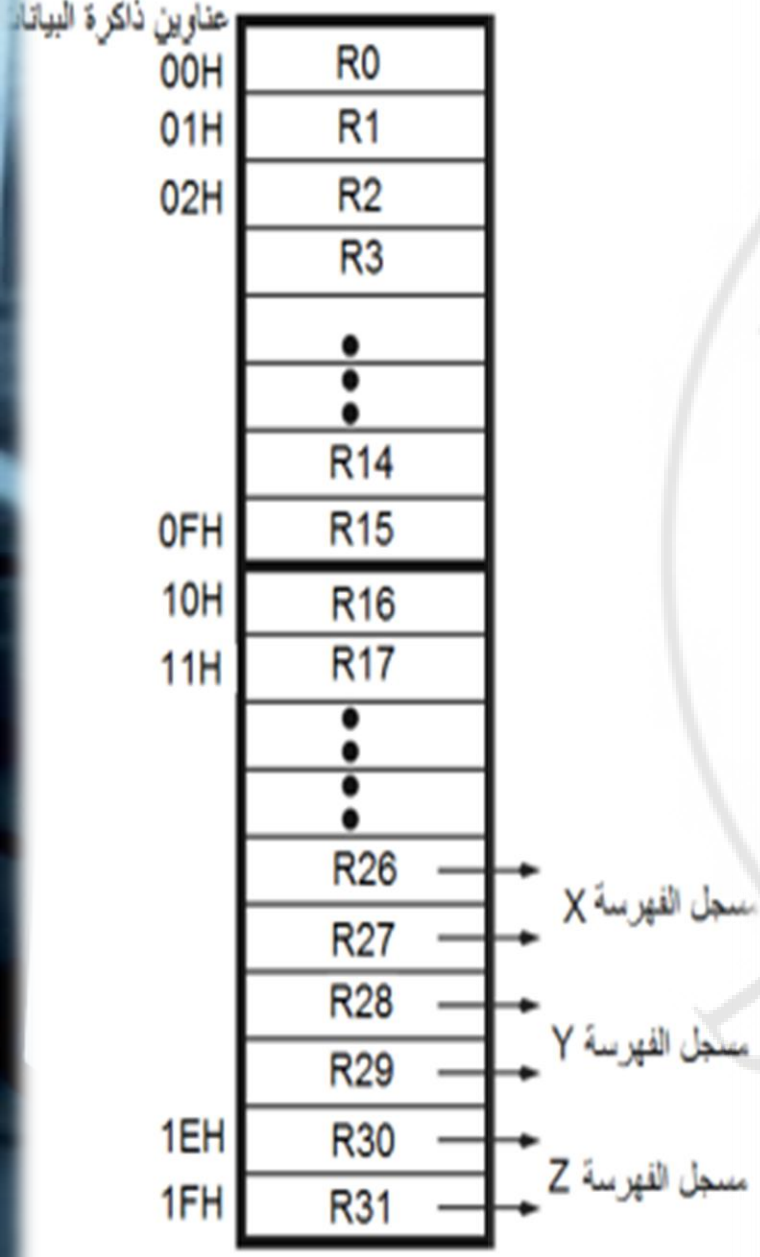


## ❑ المسجلات العامة في متحكمات AVR:

يحوي معالج AVR مجموعة من المسجلات عامة الأغراض تدعى بملف المسجلات Register File. يحوي هذا الملف 32 مسجل من R0 حتى R31 . يمكن إجراء جميع العمليات الحسابية والمنطقية على هذه المسجلات. تشكل هذه المسجلات جزءاً من ذاكرة المعطيات حيث تحتل مجال العناوين من 0x00 حتى 0x1F وذلك لتسهيل الوصول إليها. لكنها لا تعتبر ذاكرة RAM من الناحية الفيزيائية .

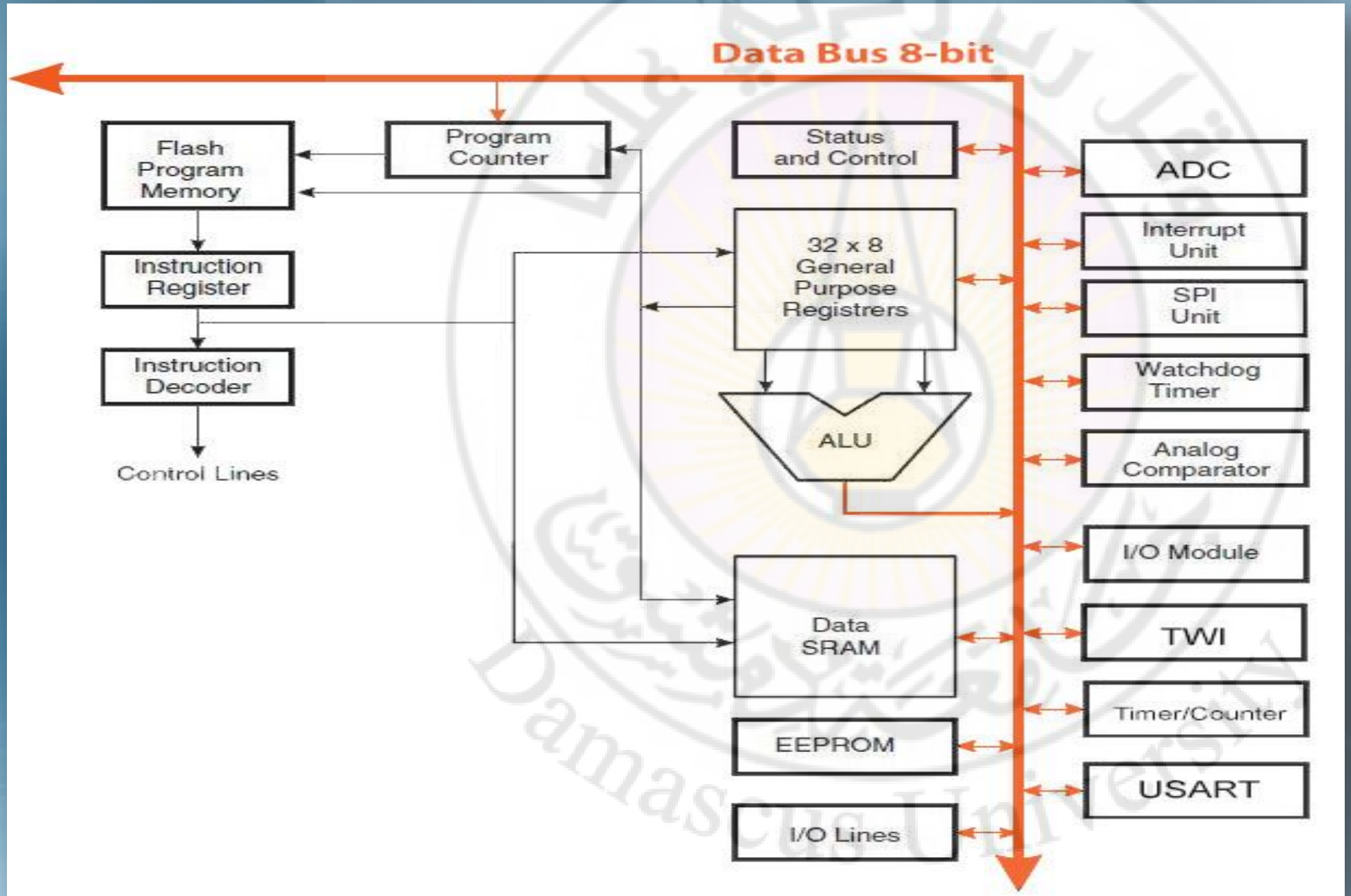
**المؤشرات عامة الأغراض**

يمكن استخدام المسجلات الستة الأخيرة من ملف المسجلات كثلاث مؤشرات إلى الذاكرة . يشكل كل زوج من المسجلات 8 بت مؤشر بطول 16 بت كما في الشكل التالي:




## ❑ المسجلات الخاصة SFR

يتم التحكم بطرفيات المتحكم AVR من خلال مجموعة من مسجلات التحكم الخاصة . تشكل هذه المسجلات فضاء عنونة خاص يسمى فضاء الدخل/الخرج ويمكن الوصول إليه بتعليمات IN و OUT . يمتد مجال عنونة الدخل /الخرج من العنوان 0x00 إلى 0x3F . يمكن الوصول إلى مسجلات التحكم أيضاً من خلال فضاء عنونة ذاكرة المعطيات حيث يحتل المجال من 0x20 إلى 0x5F وتستخدم في هذه الحالة التعليمات LD و ST . أما ذاكرة RAM الحقيقية فتبدأ من العنوان 0x60 .




مخطط صندوقي للمكونات الداخلية للمتحكمات AVR

## ❑ أهم مكونات AVR

1. وحدة الحساب والمنطق.
2. عداد البرنامج pc : ليشير إلى عنوان الأمر عدد بتاته يكون حسب كمية الذاكرة للمتحكم.
3. مسجل الأوامر IR: يحوي شيفرة الأمر الذي يتم تنفيذه الآن
4. محلل شيفرة الأوامر Instruction decoder.
5. مسجل الحالة SR ومنه:
    1. علم الحمل أو الاستلاف carry flag C: البت رقم 0 يساوي صفر في مسجل الحالة وواحد عند حدوث حمل
    2. علم الصفر zero flag Z: البت رقم 1 في مسجل الحالة يساوي واحد إذا كانت النتيجة صفر وصفر فيما عدا ذلك
    3. علم السالبية negative flag N: البت رقم 2 إذا كانت النتيجة موجبة (إذا كان آخر بت صفر) والنتيجة سالبة إذا كان آخر بت يساوي واحد
    4. علم الفيضان overflow flag V: البت رقم 3 يبين الفيضان في قيم الجمع والطرح
    5. علم الإشارة sign flag S: البت رقم 4 هو عملية XOR يبين الإشارة الصحيحة للنتيجة إذا حدث فيضان وتسبب في تدميره.

Damascus University

6. منافذ الدخل والخرج I/O PORTS يختلف عددها حسب المتحكم وتكون بعرض 8 bit.
7. هزاز داخلي Internal Oscillators : ذو بنية مقاومة ومكثف RC .
8. مؤقت / عداد Timer/ Counter : يمكن ان يكوت بعرض 8 او 16 bit.
9. مؤقت مراقبة watchdog Timer: يعيد إقلاع المتحكم إذا تعطل تنفيذ التعليمات عند نقطة واحدة لفترة زمنية معينة ويوقف عملية تنفيذ البرنامج ويحمي البرنامج من أي أخطاء،
10. محول تشابهي رقمي ADC : يمكن أن يكون بدقة (10/ 12) bit
11. واجهات اتصال متنوعة 12C - SPI - USART - TWI.
12. مقارن تشابهي Analog comparator
13. ساعة الزمن الحقيقي RTC clock

كما انها تمتلك عدة أنماط لتوفير الطاقة ووجود حماية من انخفاض جهد التغذية التي تضع المتحكم في وضع إعادة الإقلاع إلى أن يعود جهد التغذية إلى المستوى الطبيعي

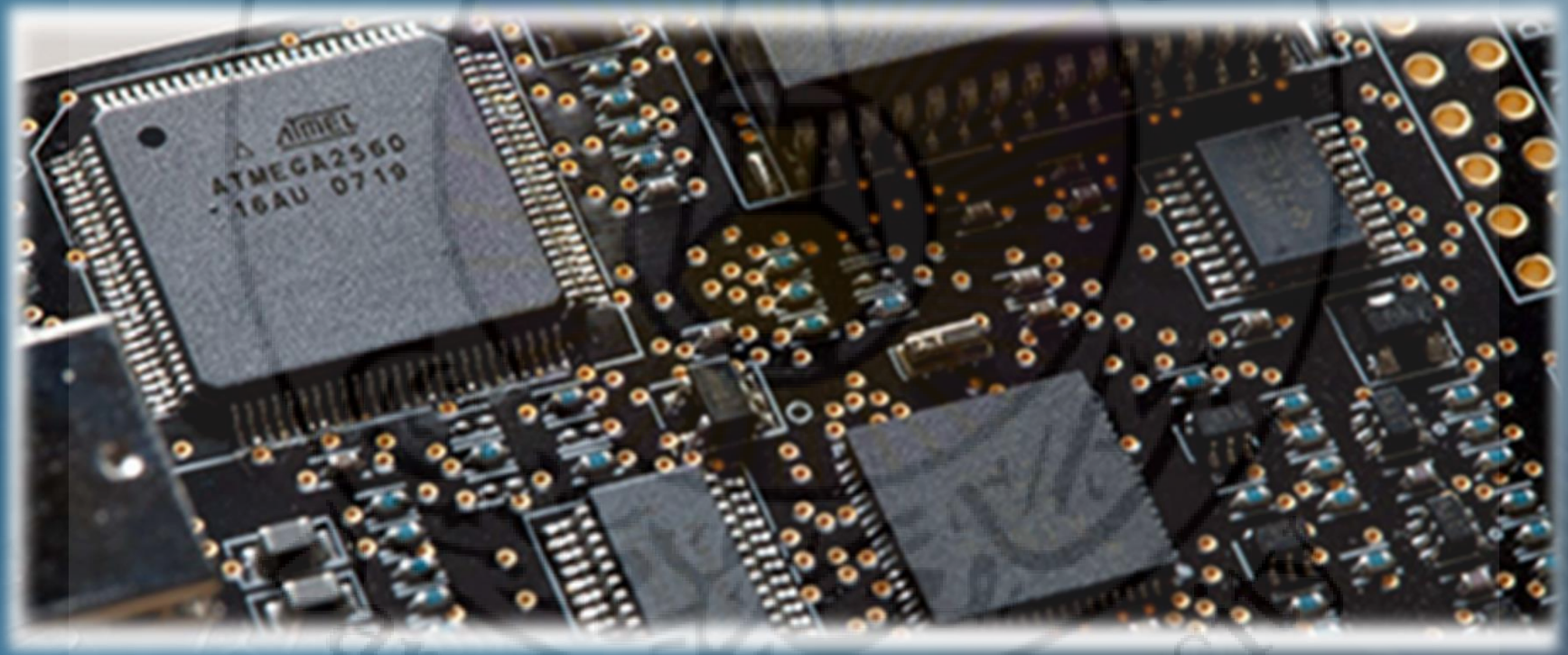
## المواصفات العامة للمتحكم ATMEGA 16:

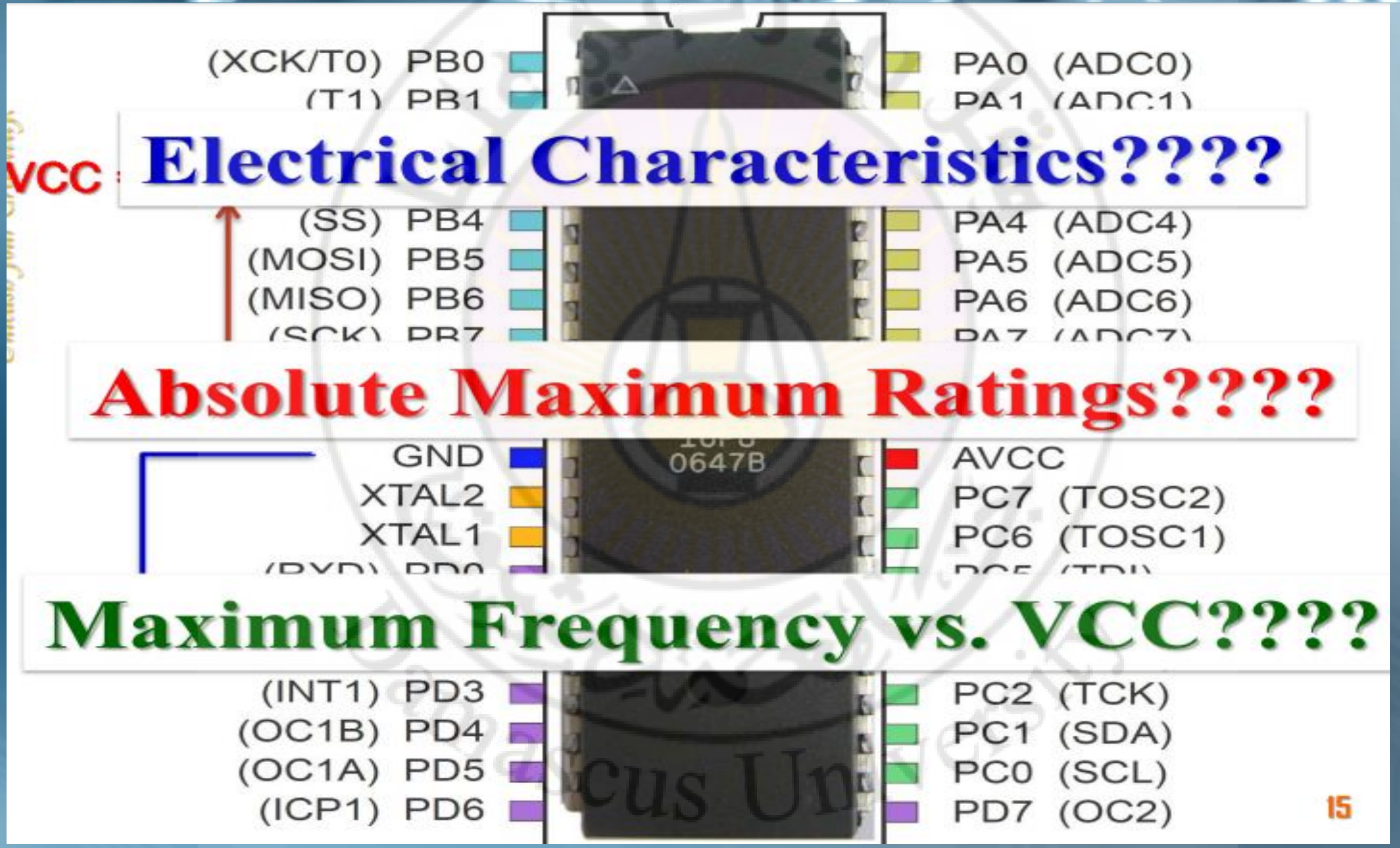
- معالج صغري AVR 8bit ذو 40 قطب ، يعمل بتقنية Risc
- يغذى ضمن المجال (2.7-5)V
- يحتوي أربع منافذ دخل وخرج A, B, C, D
- وحدة الحساب والمنطق متصلة مع 32 مسجل ذو أغراض عامة ، وهذه المسجلات تعطي لوحدة الحساب والمنطق إمكانية كبيرة في استخدام جميع طرق العنونة ، أي تعطي فعالية كبيرة في إنجاز العمليات الحسابية والمنطقية ...
- يوجد خطين للعنونة : خط عنونة مباشرة وخط عنونة غير مباشرة
- تصل سرعة المعالجة إلى 16 مليون تعليمة في الثانية وذلك عند تردد MHZ16
- يحتوي ذاكرة RAM و EEPROM و ,Flash ووحدات دخل وخرج وعداد البرنامج .

عداد البرنامج : يؤشر على التعليمة التي يتم تنفيذها حاليا .

- وحدة SPI : هي طريقة اتصال تسلسلية تستخدم لبرمجة المتحكم.
- وحدة العداد المراقب .
- مقارن تشابهي : وهو عبارة عن مكبر عمليات يعمل في نمط المقارن.
- كما يوجد مؤقتات Timers وطرفيات أخرى.

# النشرة الفنية للمتحكم الصغري DATA SHEET




(XCK/T0) PB0
(T1) PB1
PA0 (ADC0)
PA1 (ADC1)
VCC
Electrical Characteristics????
(SS) PB4
(MOSI) PB5
(MISO) PB6
PA4 (ADC4)
PA5 (ADC5)
PA6 (ADC6)
Absolute Maximum Ratings????
GND
XTAL2
XTAL1
AVCC
PC7 (TOSC2)
PC6 (TOSC1)
Maximum Frequency vs. VCC????
(INT1) PD3
(OC1B) PD4
(OC1A) PD5
(ICP1) PD6
PC2 (TCK)
PC1 (SDA)
PC0 (SCL)
PD7 (OC2)
15

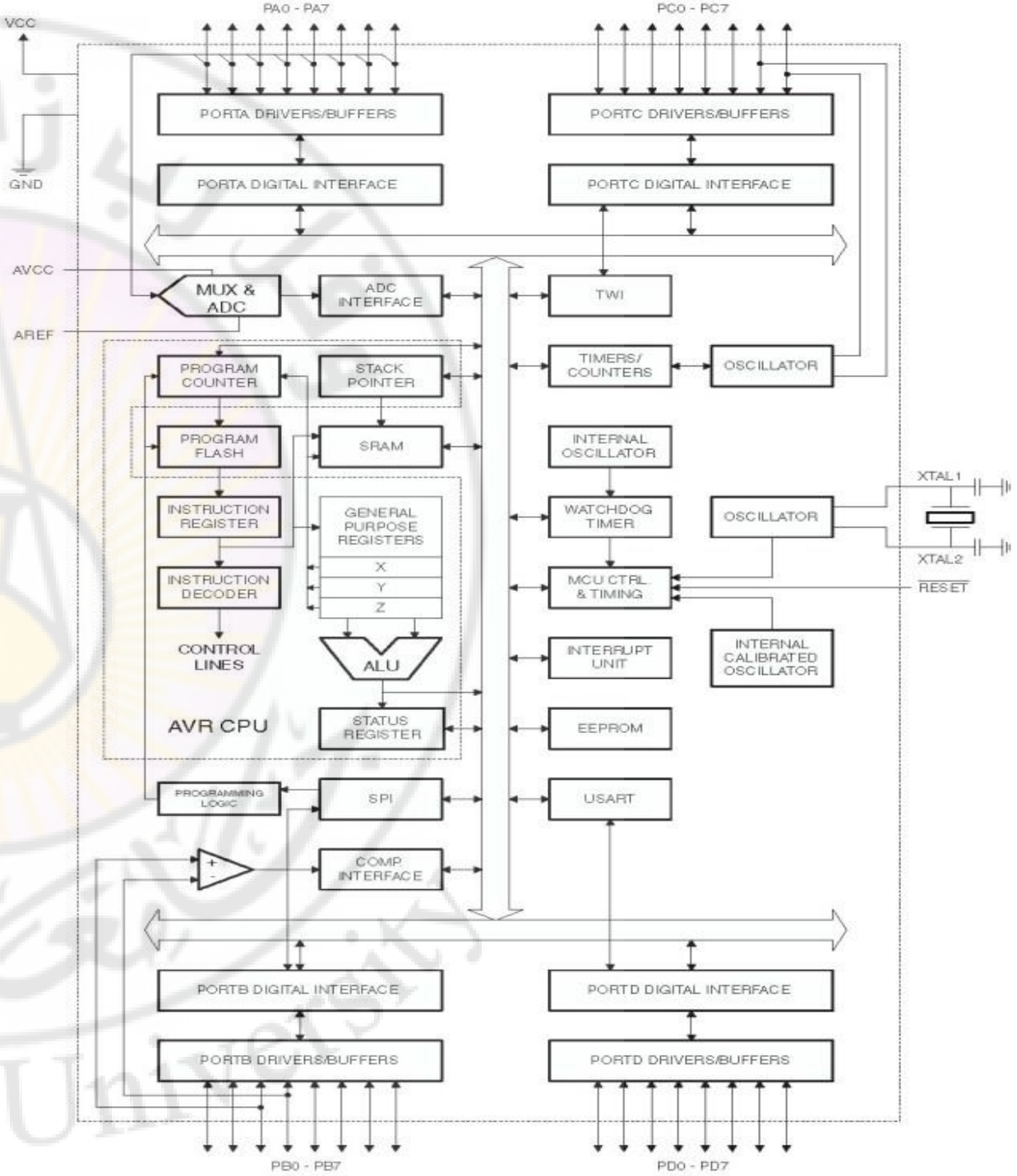
## Features

- High-performance, Low-power Atmel® AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 16 Kbytes of In-System Self-programmable Flash program memory
  - 512 Bytes EEPROM
  - 1 Kbyte Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    - 8 Single-ended Channels
    - 7 Differential Channels in TQFP Package Only
    - 2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7V - 5.5V for ATmega16L
  - 4.5V - 5.5V for ATmega16
- Speed Grades
  - 0 - 8 MHz for ATmega16L
  - 0 - 16 MHz for ATmega16
- Power Consumption @ 1 MHz, 3V, and 25°C for ATmega16L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

# 8-bit AVR® Microcontroller with 16K Bytes In-System Programmable Flash

## ATmega16
## ATmega16L

Rev. 2466T–AVR–07/10



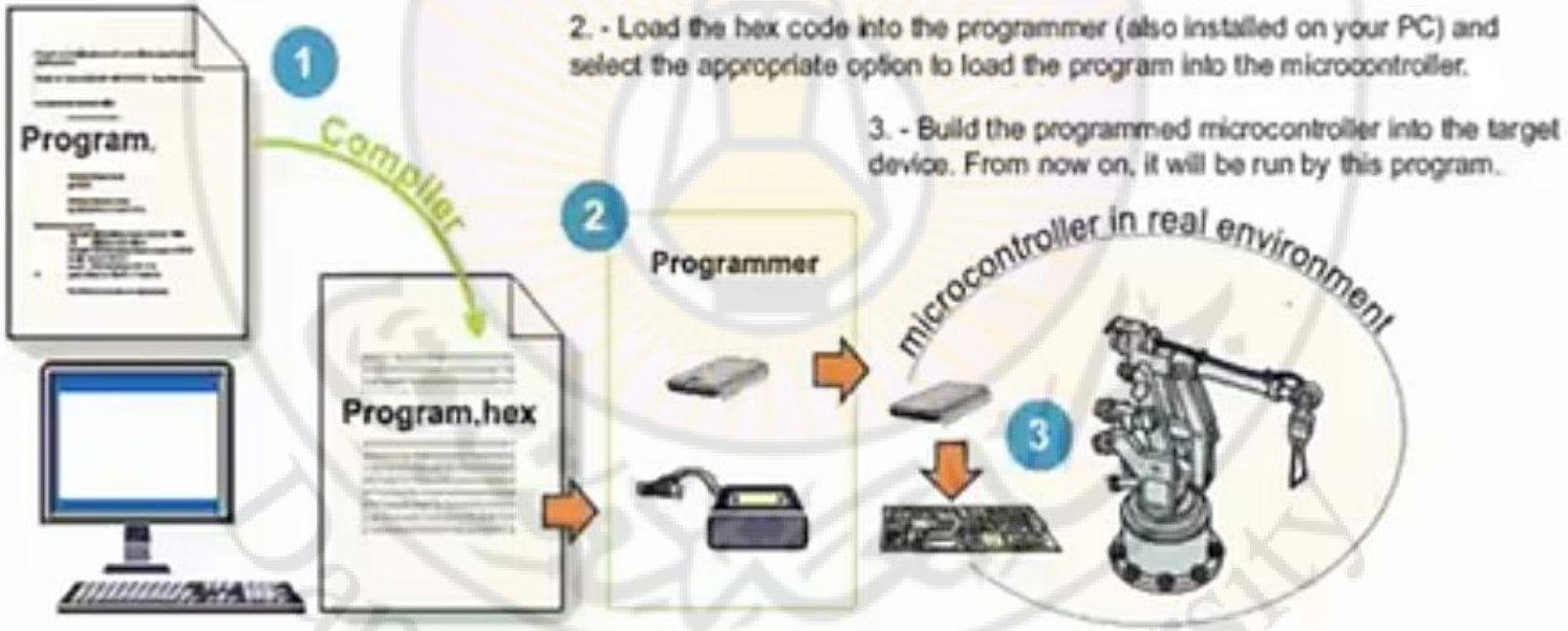

Damascus University

# كيفية برمجة المتحكم المصغر

1. - Use the compiler installed on your PC to write a program in one of the high-level programming languages and select the appropriate option to compile it into a hex code.

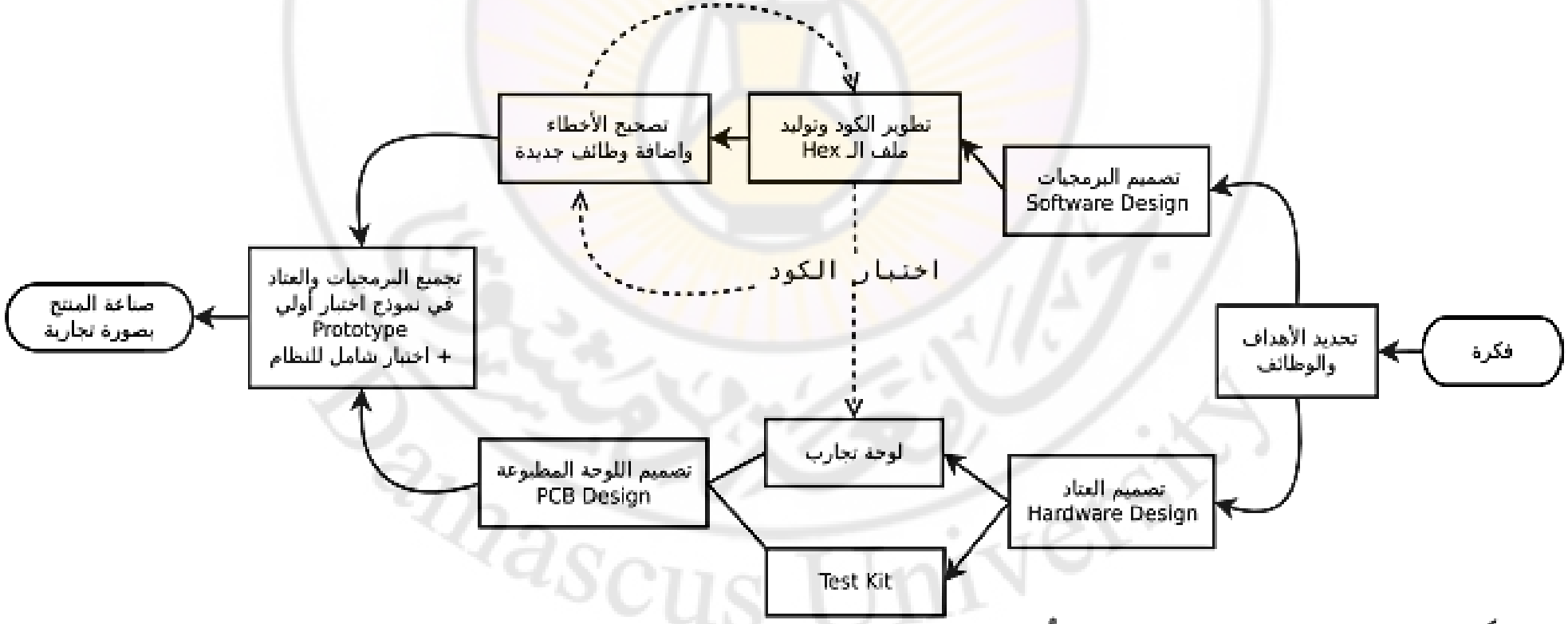
2. - Load the hex code into the programmer (also installed on your PC) and select the appropriate option to load the program into the microcontroller.

3. - Build the programmed microcontroller into the target device. From now on, it will be run by this program.



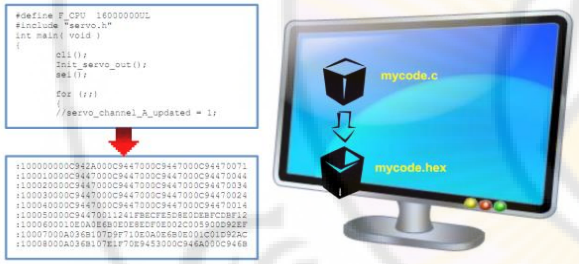

Damascus University

الصورة التالية توضح الخطوات التي يتبعها مصممو الأنظمة المدمجة في تطوير أي منتج بداية من الفكرة حتى صناعة المنتج بصورة تجارية، كما نرى هناك مساران أساسيان وهما تصميم الـ software وتصميم الـ Hardware.

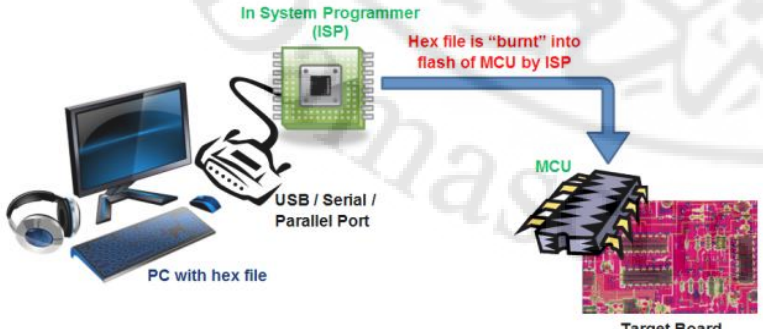


# مراحل تطوير برامج المتحكمات الصغرية:

1. كتابة برنامج المتحكم باللغة التي نختارها ( لغة التجميع أو أي لغة) قمنا باختيار C ، على وسط برمجة تم اختياره مثل (Atmel studio،mikro C ،code vision) او غيرها.
2. توليد ملف Hex والحصول على النسخة الستعشرية للبرنامج : وهو الذي يحتوي على البرنامج الحقيقي الذي سيخزن داخل ذاكرة المتحكم والذي يتم توليده تلقائيا .
3. بناء دائرة المشروع واختبارها ومحاكاتها على برنامج بروتيوس proteus ،للحصول على القيم المناسبة لكل المكونات .




3. رفع البرنامج إلى ذاكرة المتحكم ثم استخدام الكود (النسخة الستعشرية) من برنامج البرمجة، واختبار البرنامج ، بعد قراءة ملف الهيكس وتحويله إلى النظام الثنائي.




4. ثم ننتقل لمرحلة التطبيق على القطع والمكونات الفعلية .

# الحلقة اللانهائية في برامج المتحكمات:

يختلف مفهوم كتابة برنامج لمتحكم يقوم بالتحكم في عملية صناعية معينة ، مثلا التحكم في سرعة محرك ، عن برنامج يحسب رواتب الموظفين ، في المثال الأول يقوم المتحكم يقراءة سرعة المحرك ويقارنها بقيمة مرجعية لهذه السرعة ، فإذا كانت السرعة الحقيقية للمحرك أعلى من السرعة المرجعية فإنه يقوم بعمل إجراء يخفض من السرعة الحقيقية ، اما إذا كانت السرعة الحقيقية أقل من السرعة المرجعية فإنه يقوم برفع السرعة الحقيقية ، وهكذا يظل برنامج المتحكم يدور في حلقة لانهائية في البرنامج ، يقرا السرعة الحقيقية ويقارنها بالسرعة المرجعية ولن يخرج المتحكم من هذه الحلقة إلا إذا تم إيقاف النظام بالكامل ، لذلك يلزم وجود حلقة لانهائية في مثل هذه التطبيقات.

أما برنامج حساب رواتب الموظفين يجب ان يقوم به أحد المعالجات وليس أحد المتحكمات لأن المعالج سيقوم بحساب الرواتب مرة في اول كل شهر ثم يخرج من البرنامج أو يتوقف عن العمل .

ومعنى انه يتوقف عن العمل أي يتم الخروج من وسط البرمجة في لغة C والخروج إلى وسط نظام التشغيل الخاص بالحاسب .

ففي أبسط تطبيق عندما يضيء LED بواسطة متحكم ويتوقف ، فإنه بعد أن يضيء عليه أن يدخل في حلقة لانهائية لايعمل فيها أي شيء ، كما في حلقة{ while (1){ }، حيث ستكون هذه هي طريقة تعطيل المتحكم من عمل أي شيء يؤثر في التطبيق وفي نفس الوقت يحافظ على عمل التطبيق ، حيث لامجال للخروج من الحلقة اللانهائية إلى نظام التشغيل.

# البرنامج الأول

## إنارة وإطفاء دايود ضوئي بمعدل معين

المطلوب وصل ثنائي ضوئي LED على متحكم AVR Atmega 16 على القطب PINC.0 أو الطرف PC0 ، وكتابة البرنامج بحيث يومض الليد كل 500 ميلي ثانية.

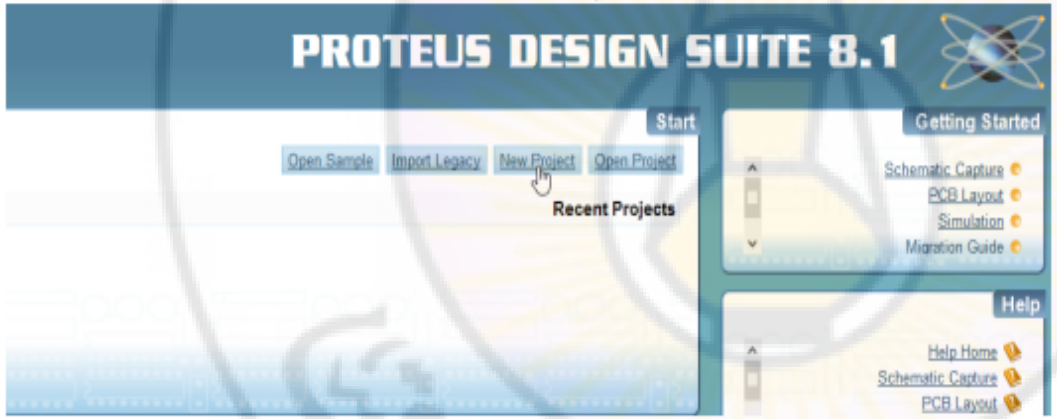
نبدأ برسم الدائرة المطلوبة على برنامج المحاكاة بروتوسProteus:

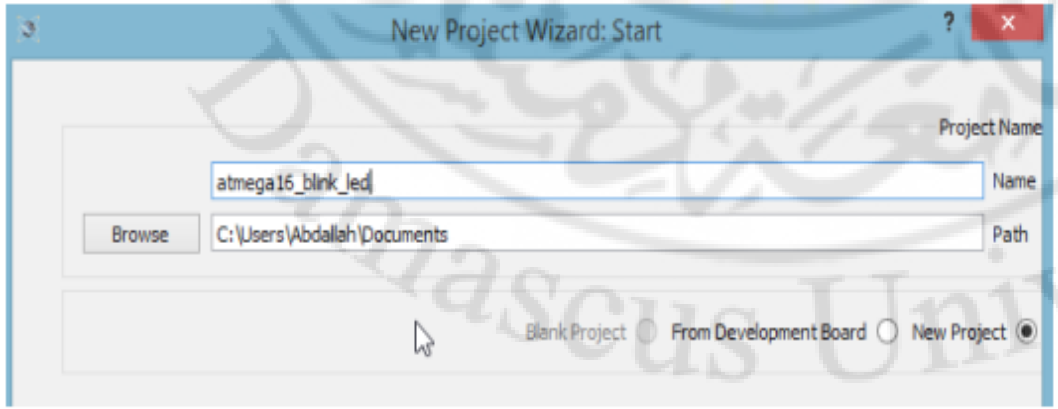
يعتبر من البرامج البسيطة نقوم بالعمل وفق المراحل التالية :

تحديد المستلزمات (ليد + متحكم)

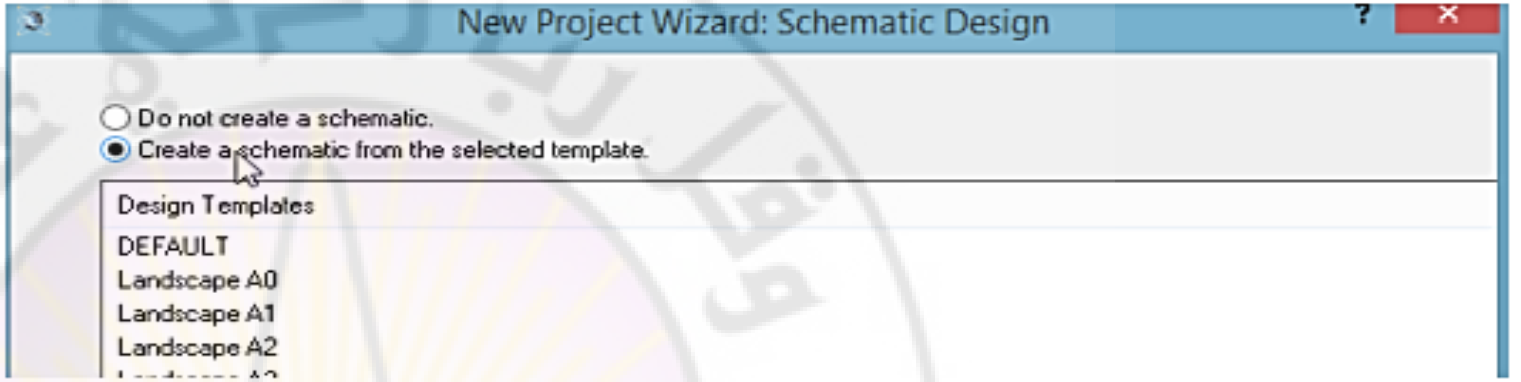
تكامل المستلزمات هي ان يكون المتحكم متوائم مع خرج 5 فولت.....

في البداية قم بعمل مشروع جديد ولنسميه ATmega16_blink_led واختر المكان الذي تريد أن تحفظ به ملفات المشروع كما في الصور التالية:

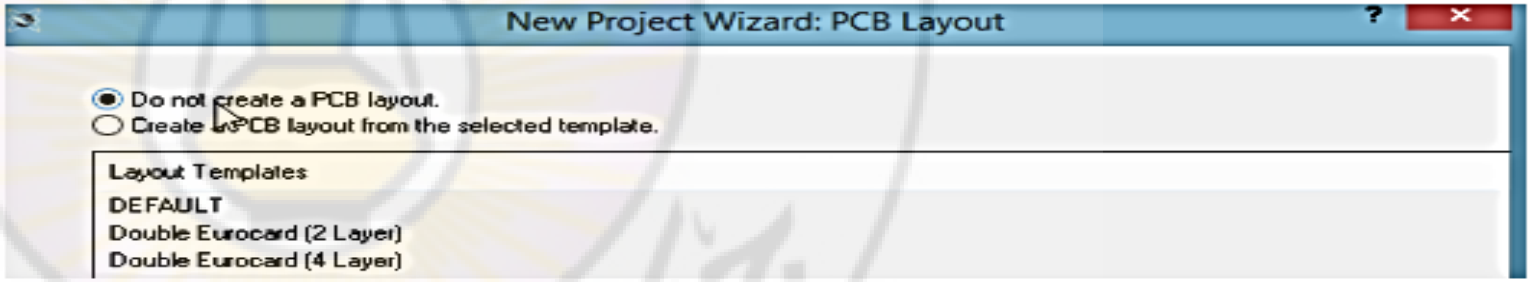


اختر تصميم مخطط جديد Create a new schematic بالمقاس الافتراضي default
ثم نضغط next




لن نحتاج أن نصنع تصميم PCB في الوقت الحالي، لذا قم باختيار Don't Create PCB ثم
اختر في الصفحة التي تليها No Firmware



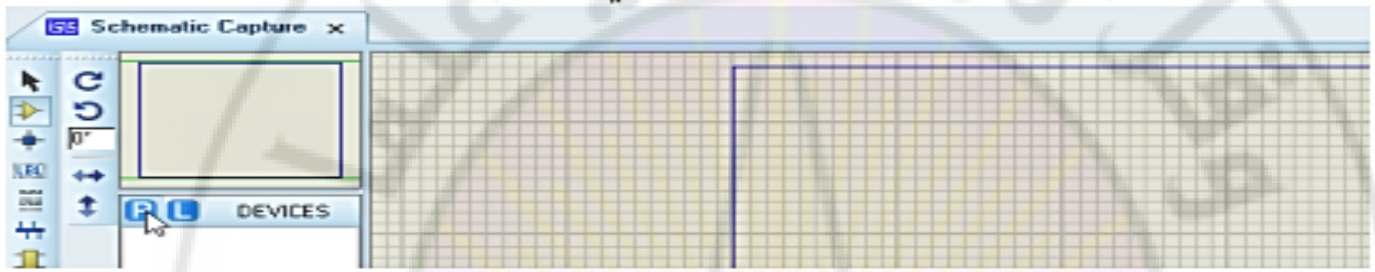
والآن أصبح لدينا ملف المخطط Schematic جاهز لنبدأ بتوصيل المكونات الإلكترونية مع
بعضها البعض، في التجارب القادمة سنقوم باستخدام المكونات التالية (سنقوم بإضافتها في
قائمة المكونات المستخدمة في المحاكاة من خلال الضغط على زر P من قائمة devices

تظهر شاشة تصميم الدارة ...

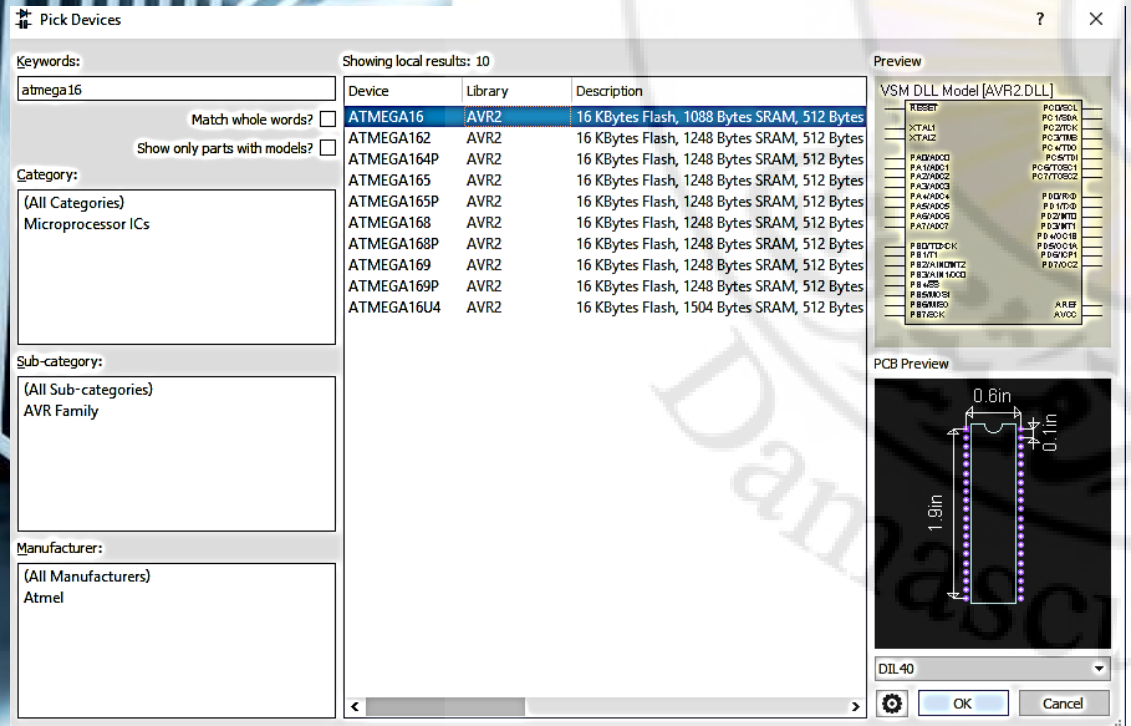
نقوم باختيار القطع المطلوبة للدائرة بالضغط على الزر P (pick device)

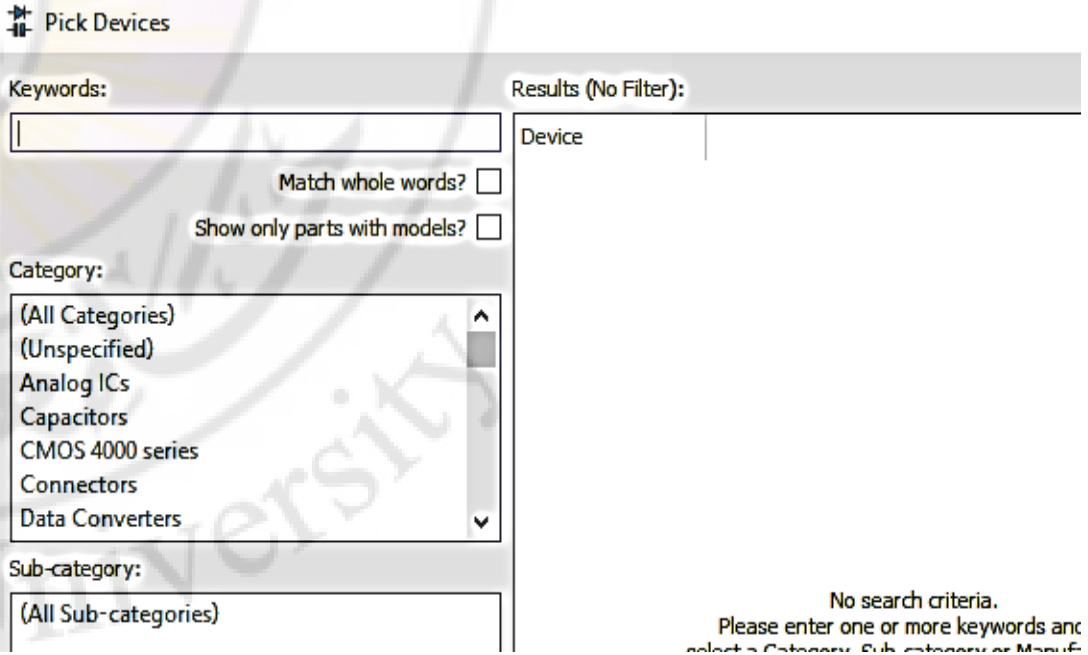
- ATmega16
- LED (yellow)
- Resistor 330
- LED bar

يمكنك استخدام خاصية البحث عن المكونات كما في الصور التالية

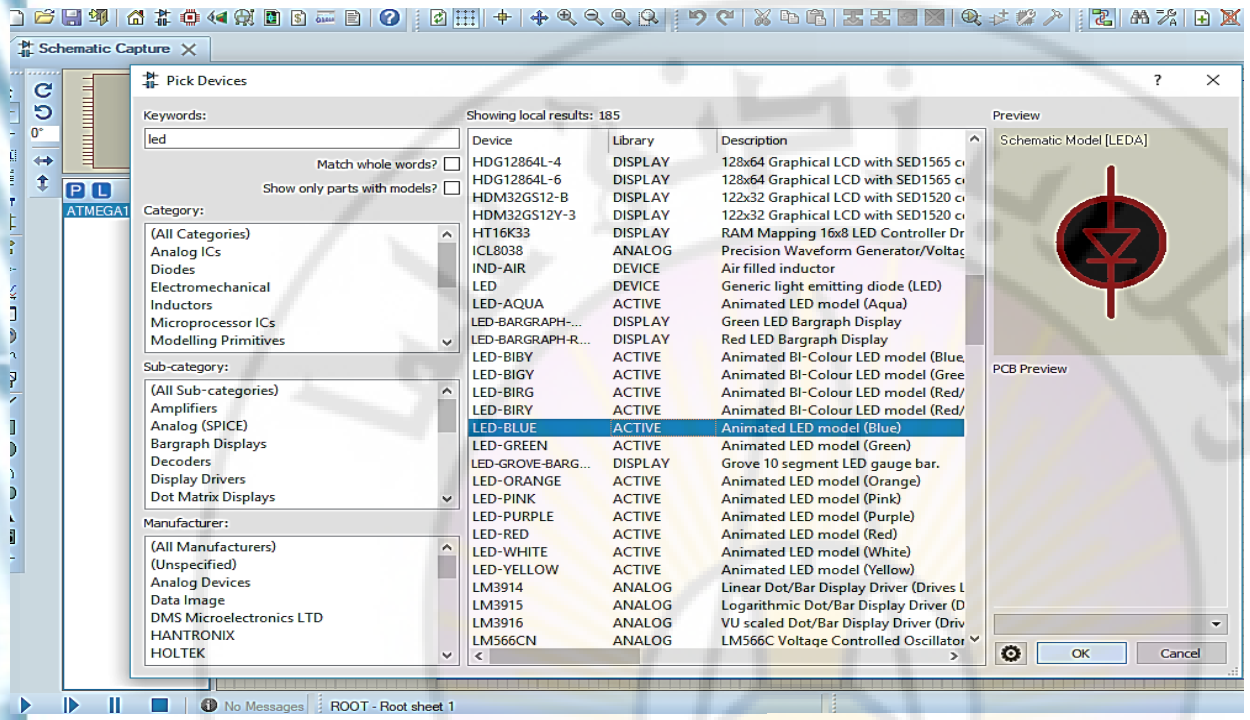



ثم من المربع Keywords نقوم بكتابة أسماء القطع المطلوبة والضغط عليها لاختيارها

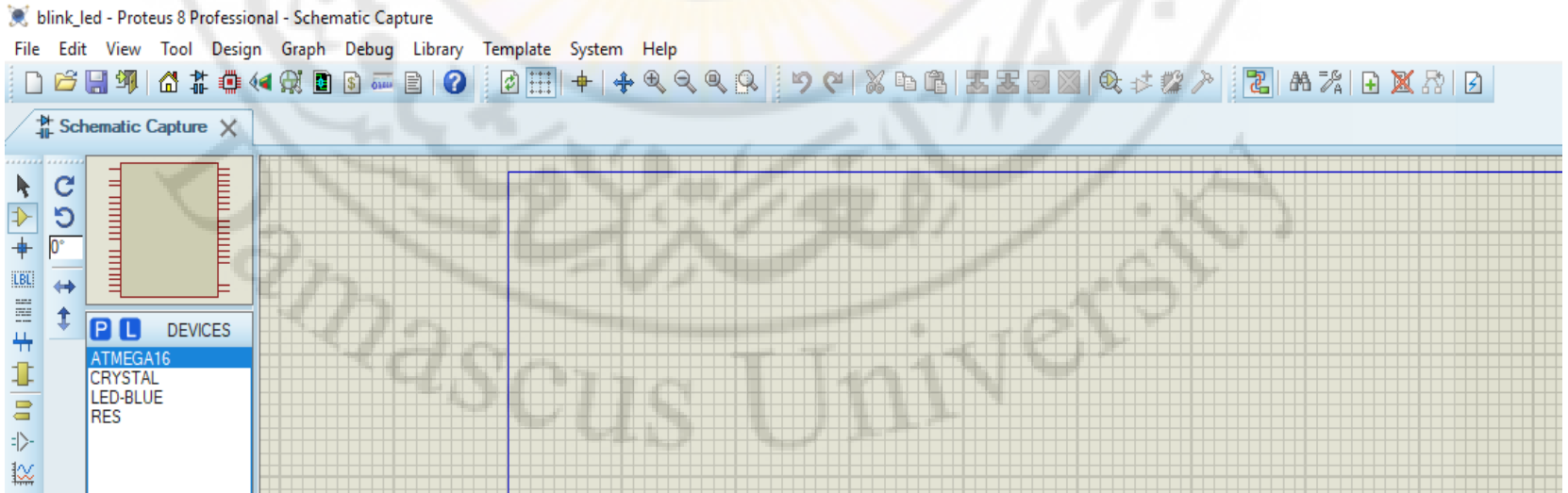


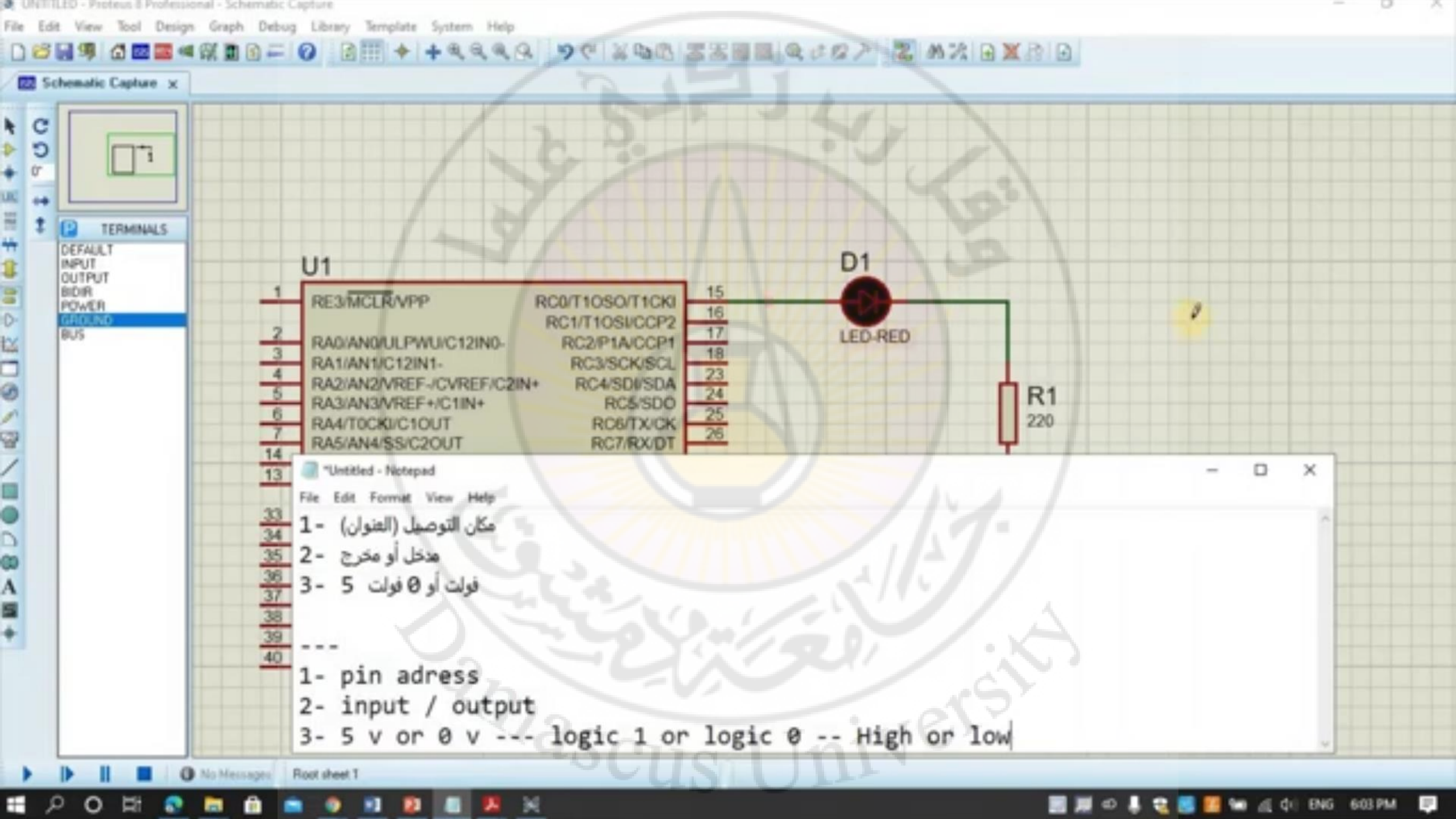



ثم من المربع Keywords نقوم بكتابة أسماء القطع المطلوبة والضغط عليها لاختيارها

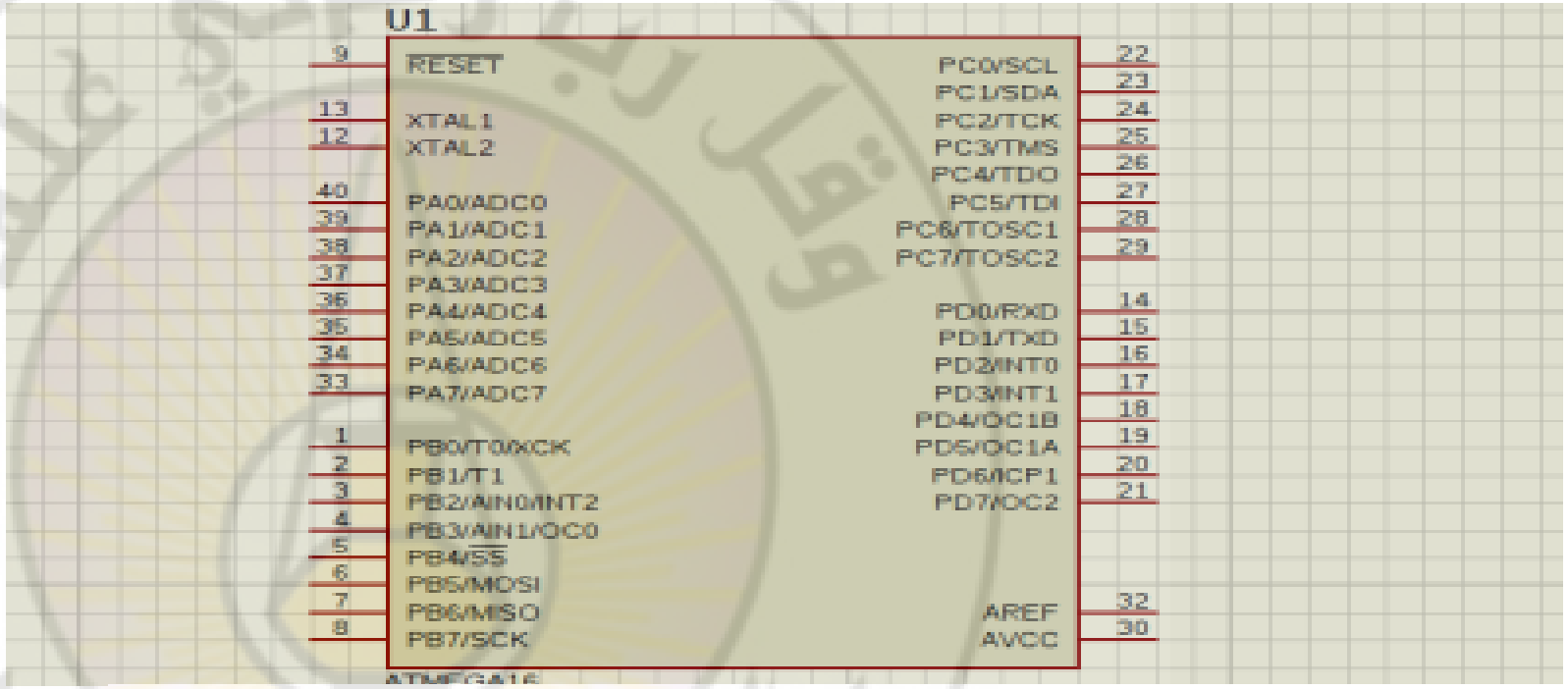



بعد إضافة جميع المكونات سنجد أن المستطيل الجانبي الخاص بالمكونات أصبح يحتوي على معظم المكونات التي نحتاجها للتجارب القادمة كما في الصورة التالية:

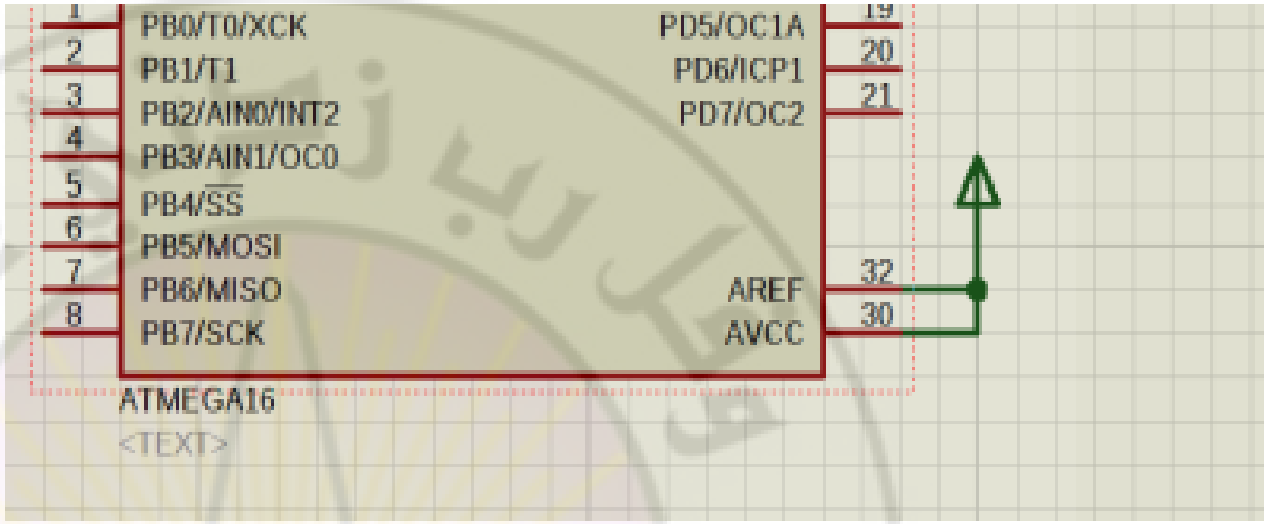



UNTITLED - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
BUS
U1
RE3/MCLR/VPP
RA0/AN0/ULPWU/C12IN0-
RA1/AN1/C12IN1-
RA2/AN2/VREF-/CVREF/C2IN+
RA3/AN3/VREF+/C1IN+
RA4/T0CKI/C1OUT
RA5/AN4/SS/C2OUT
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2
RC2/P1A/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
D1
LED-RED
R1
220
*Untitled - Notepad
File Edit Format View Help
1- مكان التوصيل (العنوان)
2- مدخل أو مخرج
3- 5 فولت أو 0 فولت
---
1- pin adress
2- input / output
3- 5 v or 0 v --- logic 1 or logic 0 -- High or low
No Messages
Root sheet 1
ENG
6:03 PM

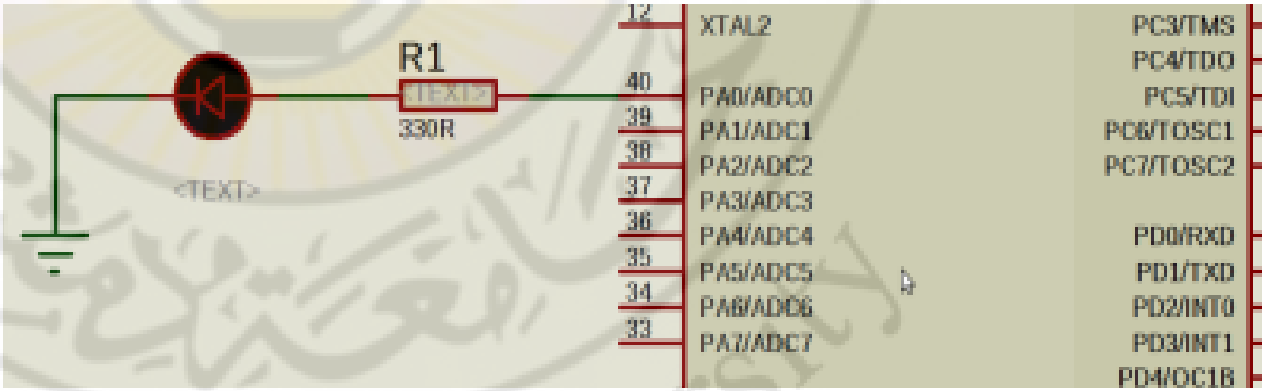
والآن سنبدأ في بناء أول دائرة لتجربة الـ Blinking led، في البداية سنقوم بوضع المتحكم ATmega16 داخل إطار رسم برنامج بروتس كما في الصورة التالية:



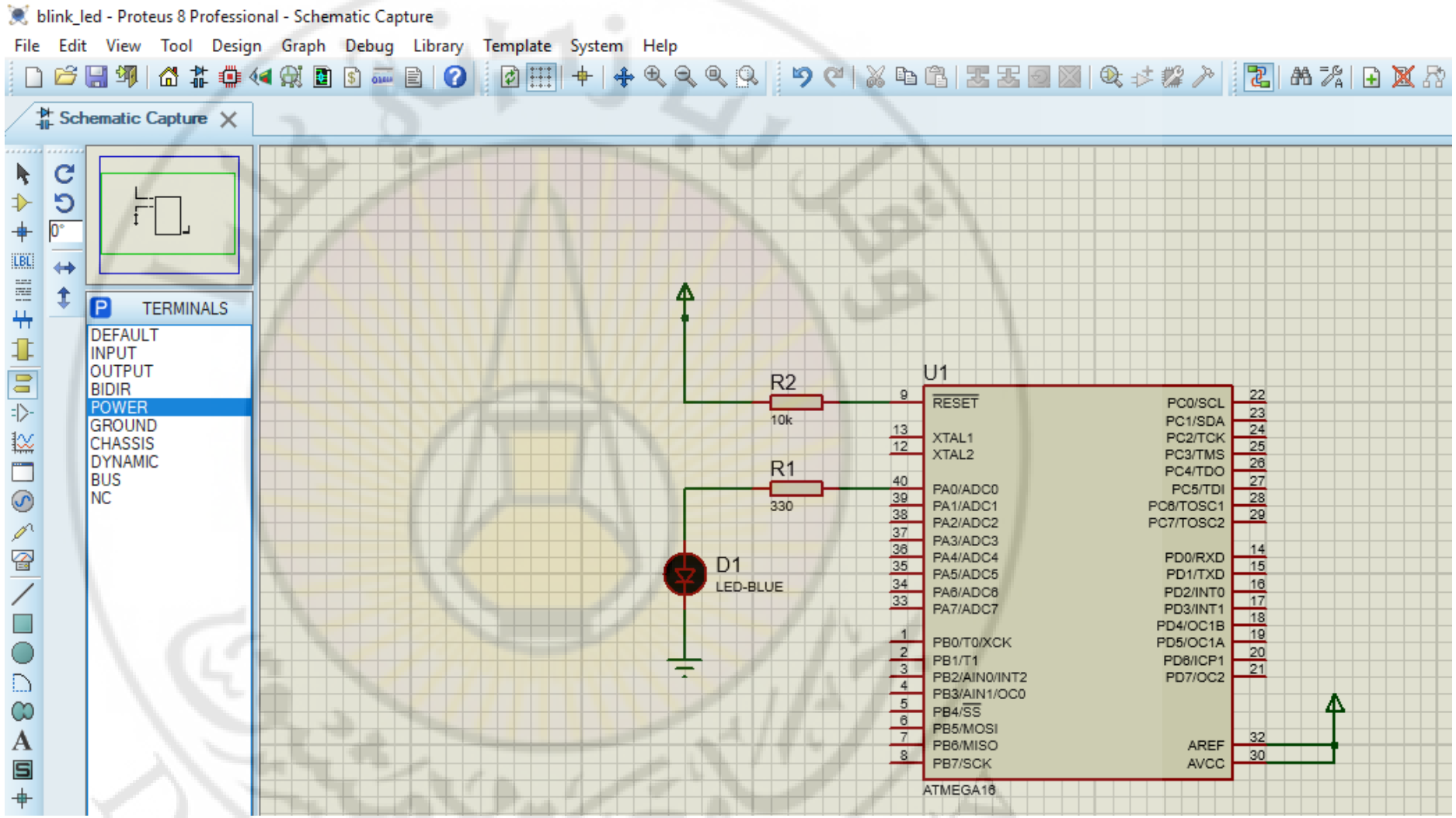

ثم سنختار قائمة terminal من برنامج بروتس (القائمة التي تحتوي على رموز البطارية - الموجب والسالب) ومنها سنضيف power ونقوم بتوصيلها بالمدخل AREF و AVCC (سنتحدث عن كلا المخرجين بالتفصيل في الفصل الخاص بالمحول التناظري\الرقمي ADC).




والآن قم بإضافة الدايود الضوئي والمقاومة على المخرج PA0 ثم وصل الطرف السالب من الدايود على علامة Ground (يمكنك إضافتها من قائمة Terminals أيضاً) كما في الصورة التالية:




قم بإضافة مقاومة أخرى ووصلها بالمدخل RESET في الطرف الأيسر من المتحكم ثم وصل الطرف الآخر بعلامة Power - وقم بتعديل قيمة المقاومة لتصبح 10 كيلو (تكتب 10k) كما في الصورة التالية:

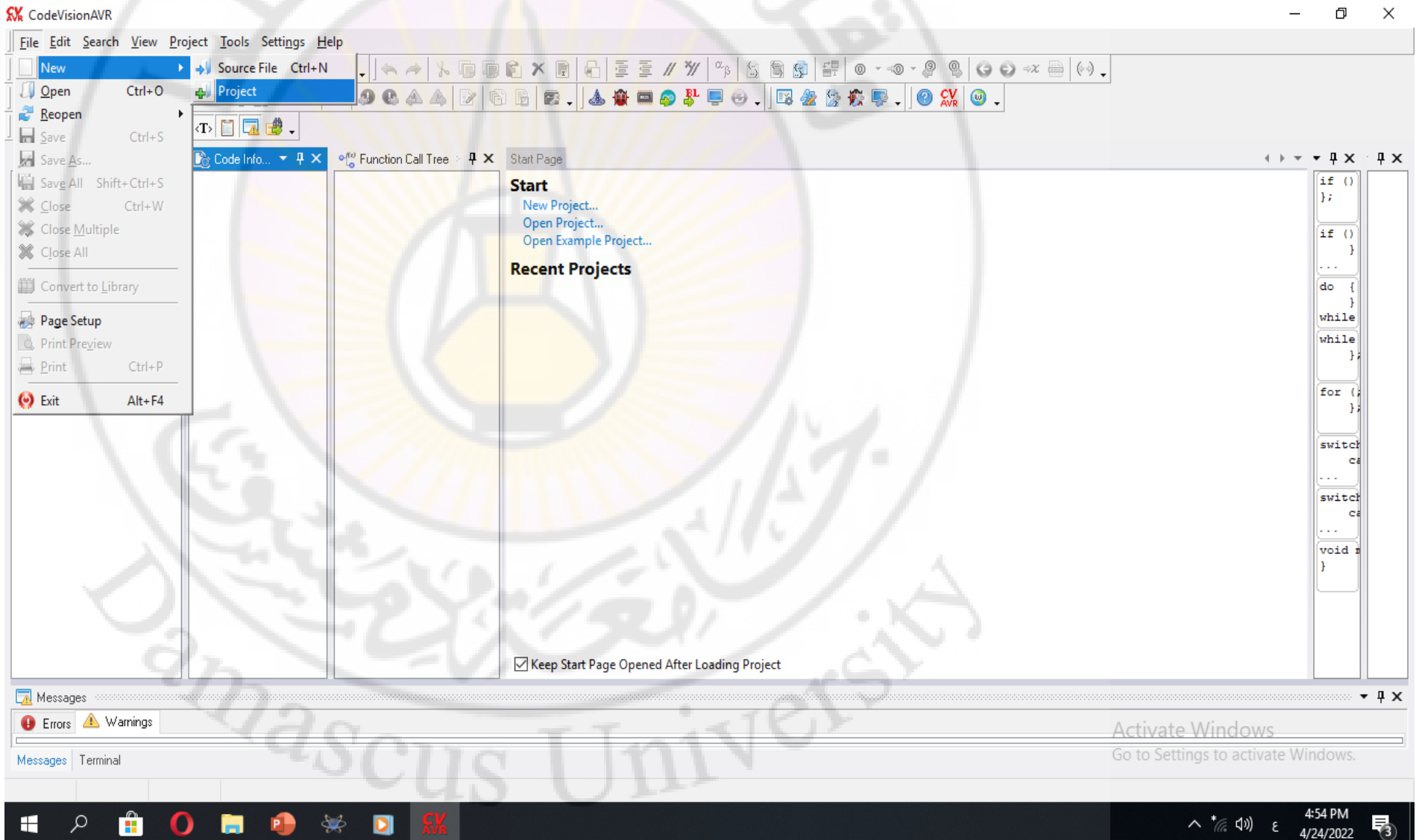



وبذلك نكون قد انتهينا من توصيل المكونات الأساسية ويبقى إضافة ملف الهيكسا Hex وذلك بعد كتابة الكود المطلوب ... وقد تم اختيار برنامج Code Vision AVR

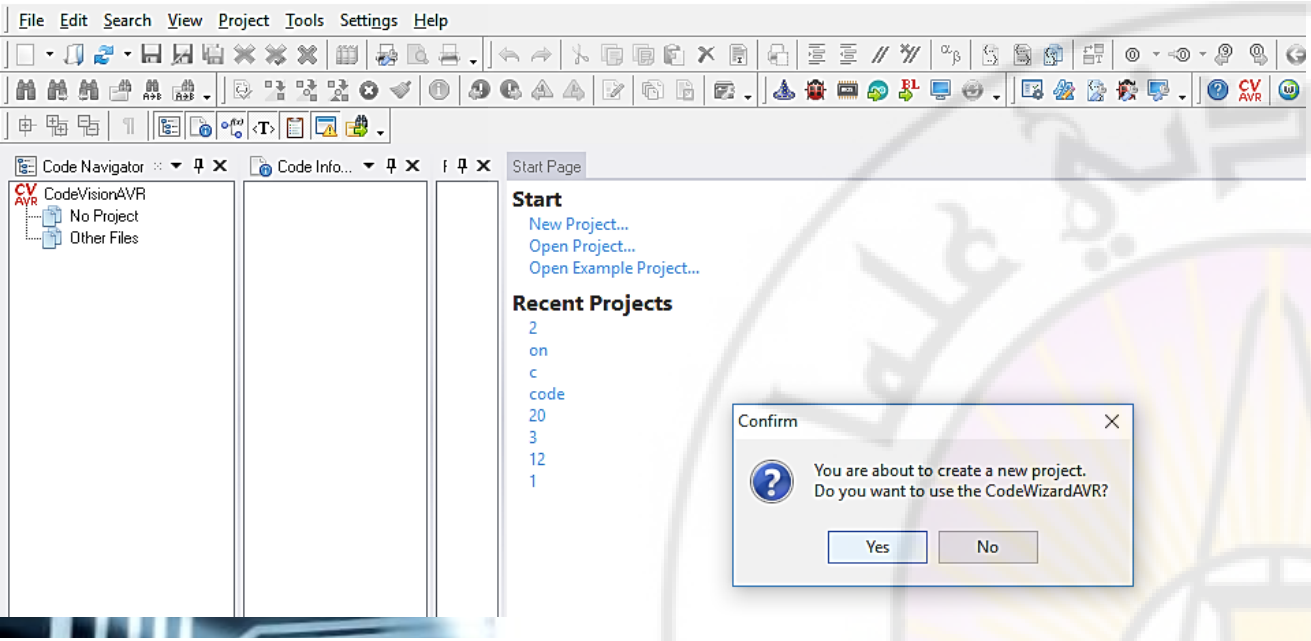
يتوفر في هذا البرنامج طريقتين لكتابة الكود إما كتابته بشكل يدوي Source File أومن خلال تعديل المداخل والمخارج بالنسبة للمسجلات project

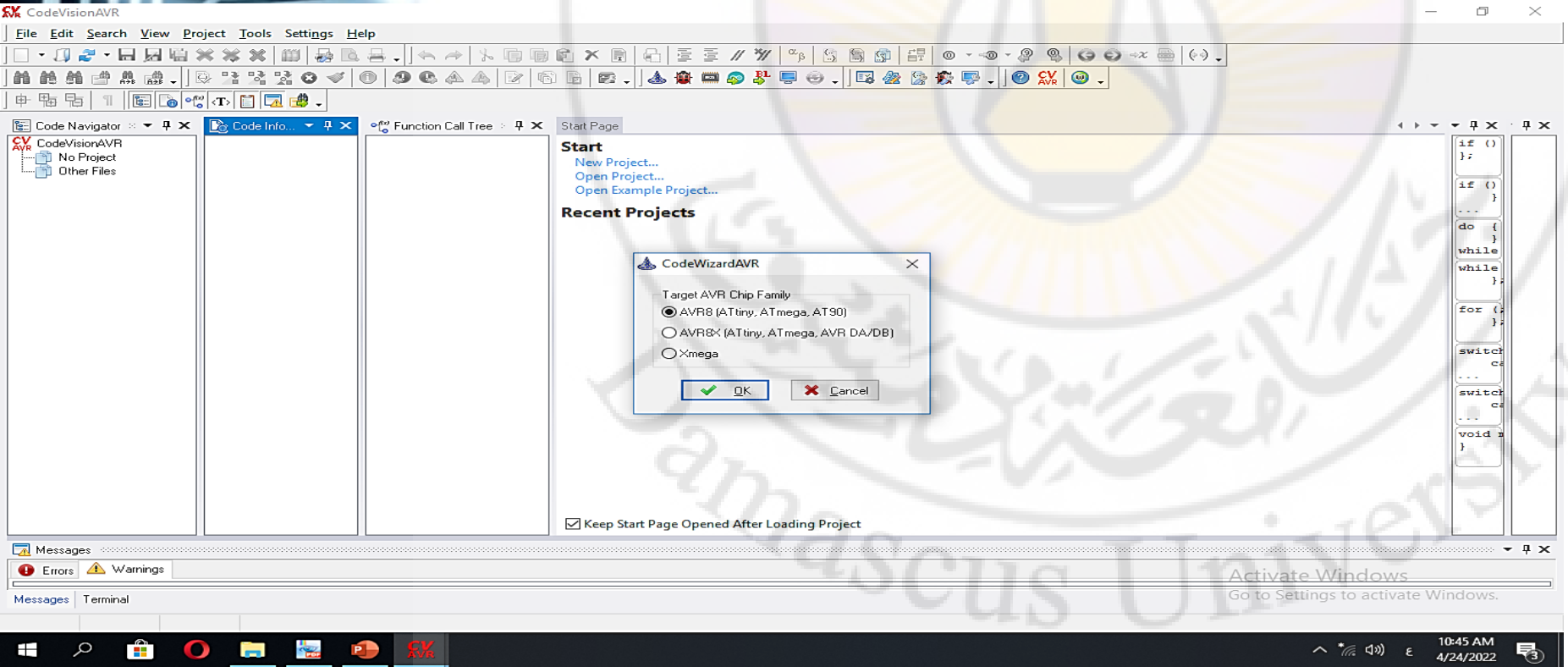
الأول نقوم بكتابته لمرة واحدة وهو عمل اعتيادي لأي برنامج يتطلب استدعاءه عند الحاجة وهو ملف رأسي

اما النوع الثاني فهو من الملفات الجاهزة والتي تعطي جميع الأسطر البرمجية ونقوم بالتعديل حسب الدارة المطلوبة



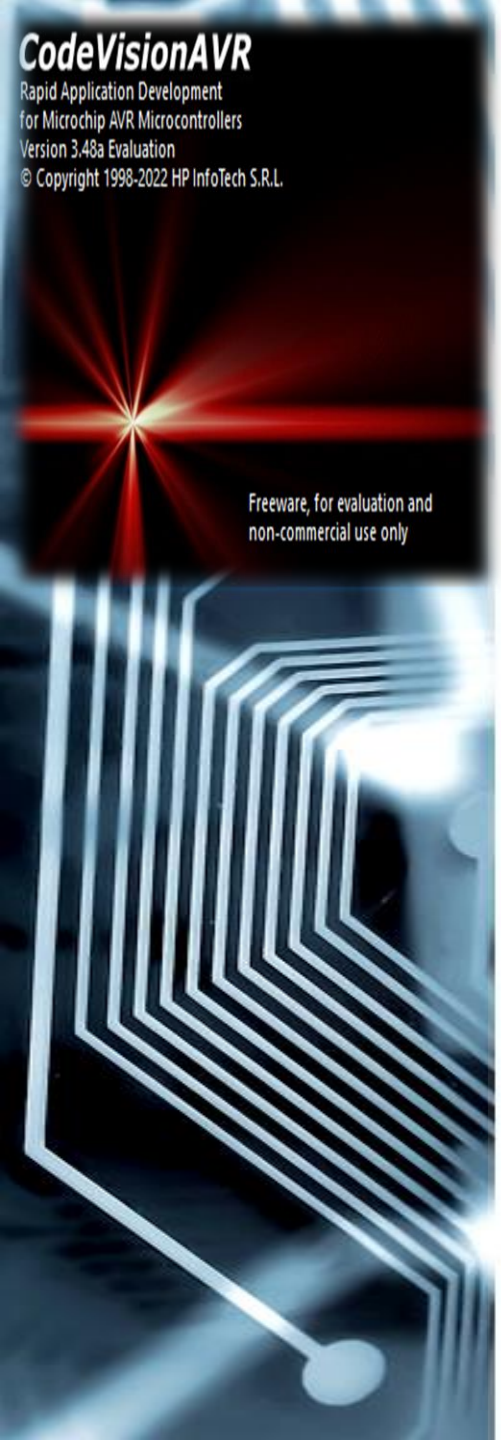
سنبدأ باستخدام الأكواد الجاهزة Project file كما في الصور التالية

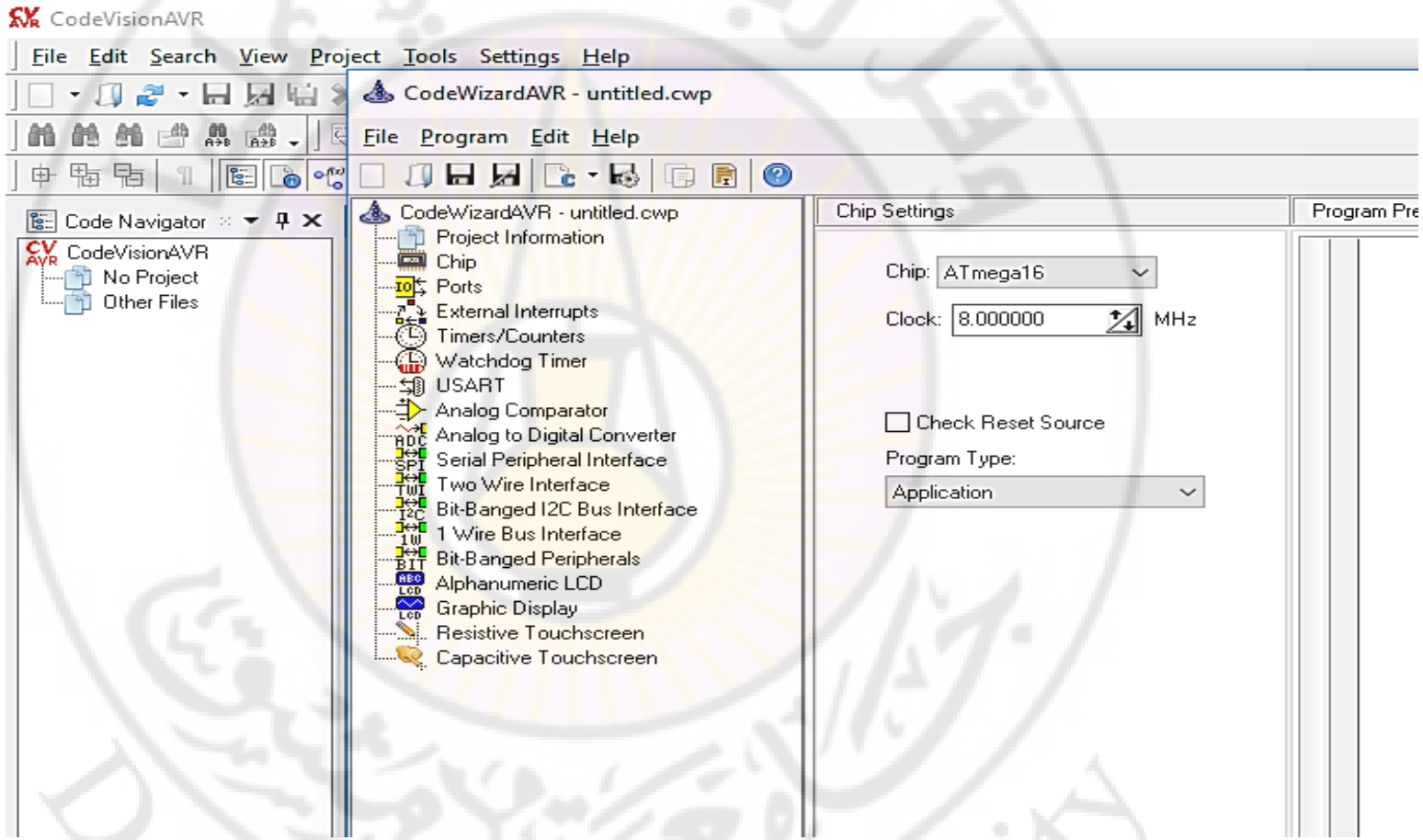



ثم نختار نوع المتحكم
Atmega 16 بعد الضغط على
الخيار الأول

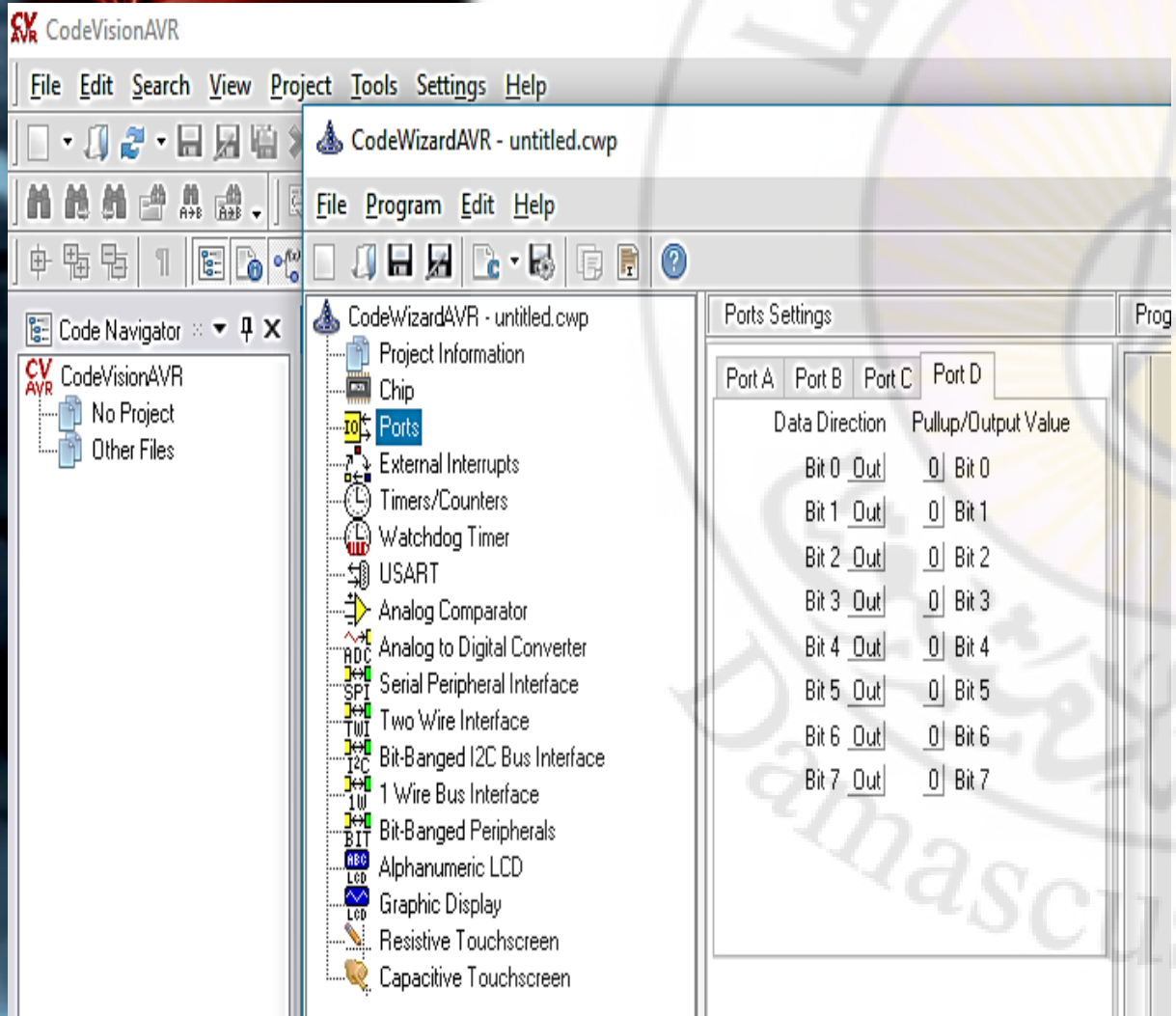
وتحديد التردد 8MHZ وهو نفسه الذي تم تحديده سابقا من خلال المؤقتات

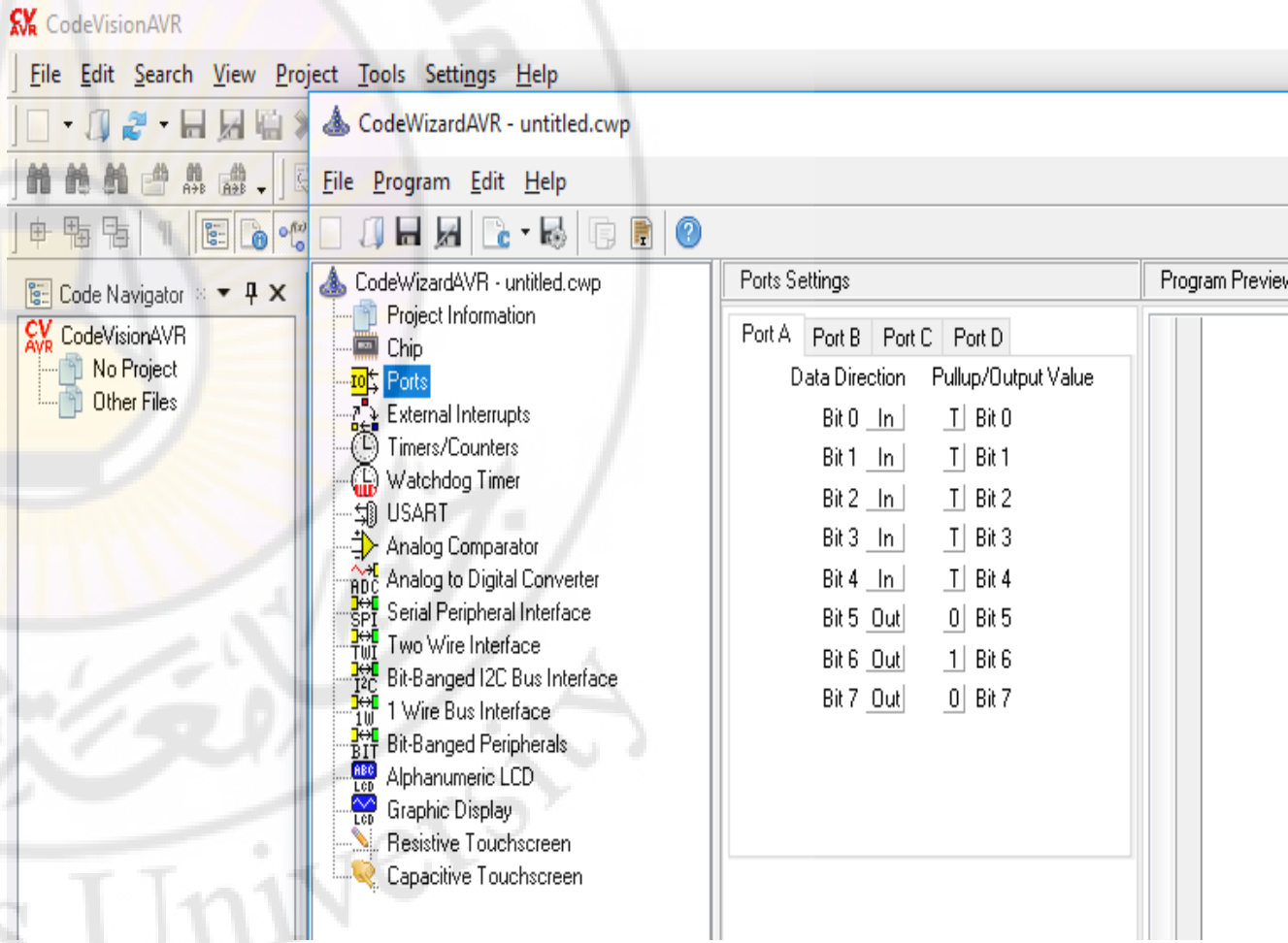


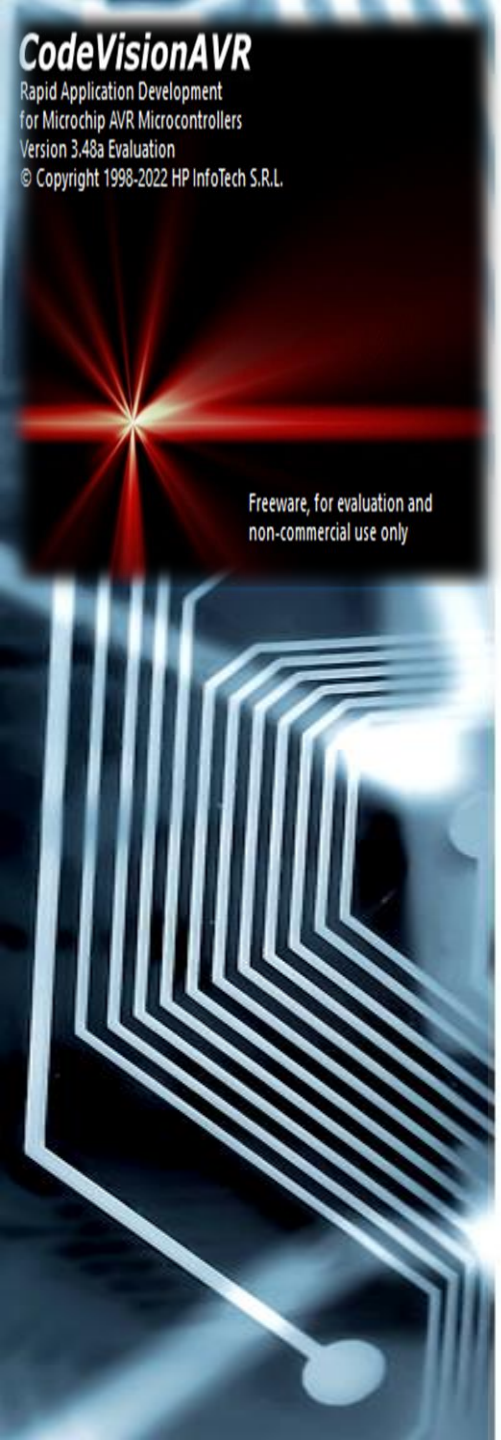




Damascus University

ثم نقوم بتحديد الPORTS على انها دخل أو خرج
دون كتابة أي تعليمة نذهب مباشرة إلى PORTB مثلا ونقوم بتغيير البنات من دخل إلى خرج كما نحتاج ،
وبفرض مثلا نريد الPORTD على انه خرج بشكل كامل
وبعد الانتهاء من تحديد حالات المداخل والمخارج نقوم بحفظ البرنامج





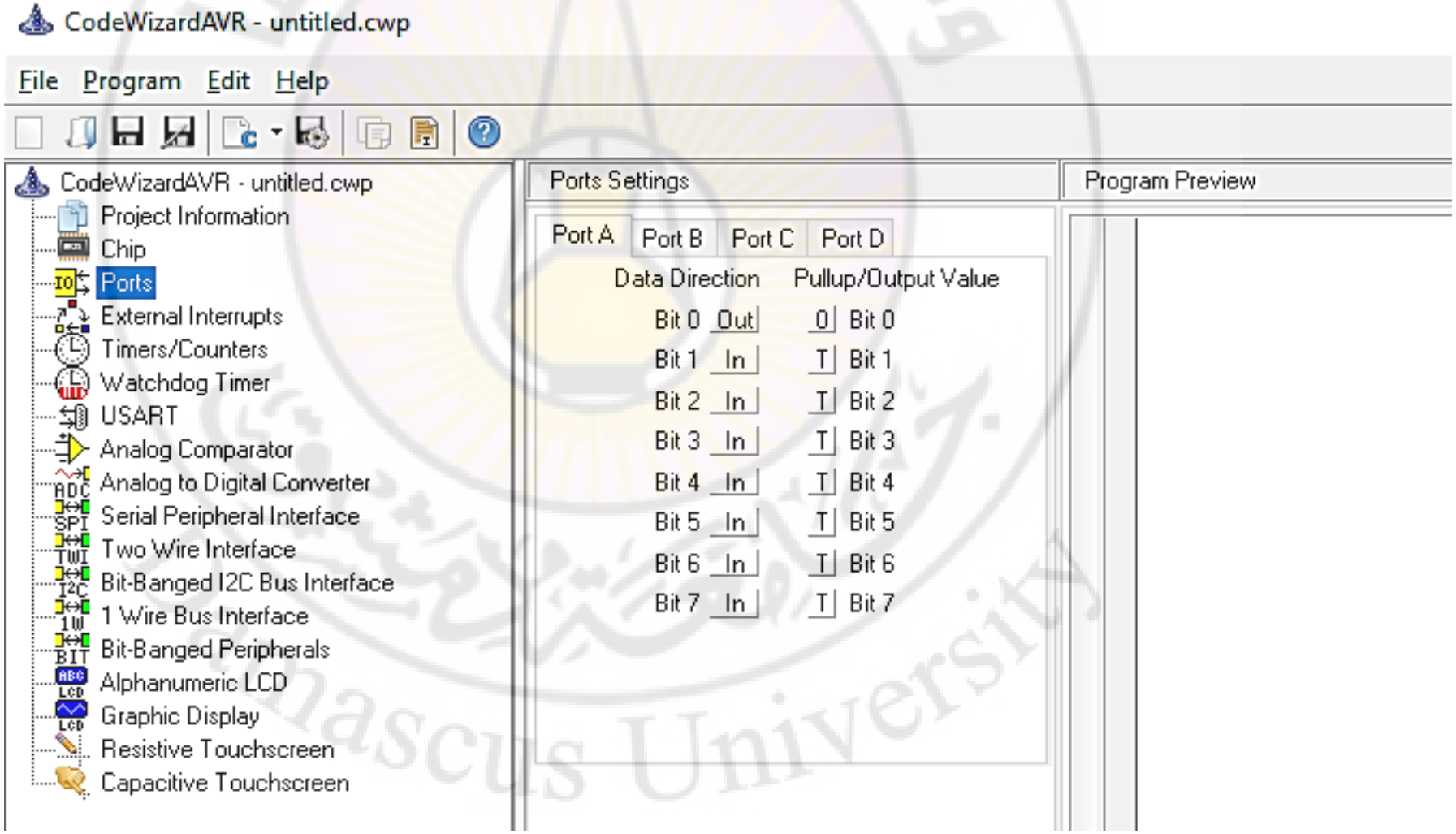

ونقوم بتحديد PORT A0 على انه خرج ، والحالة صفر ، أي free statue . .

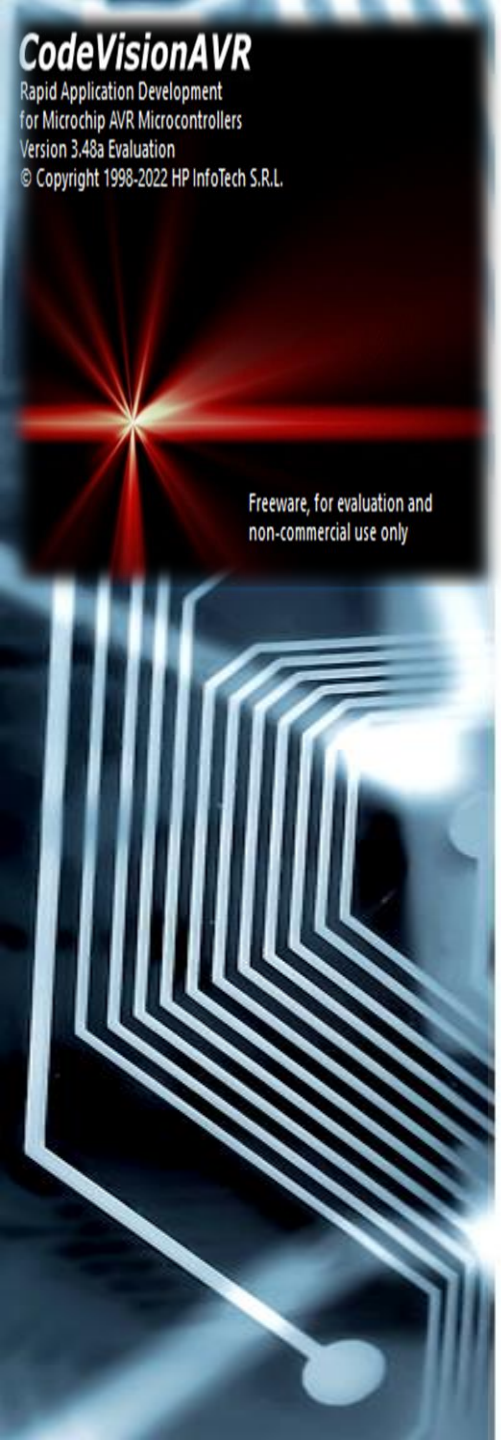
حيث كما يظهر يوجد حالتان :

free statue لايملك أي قيمة ابتدائية ويعمل حسب مايتم إدخاله.

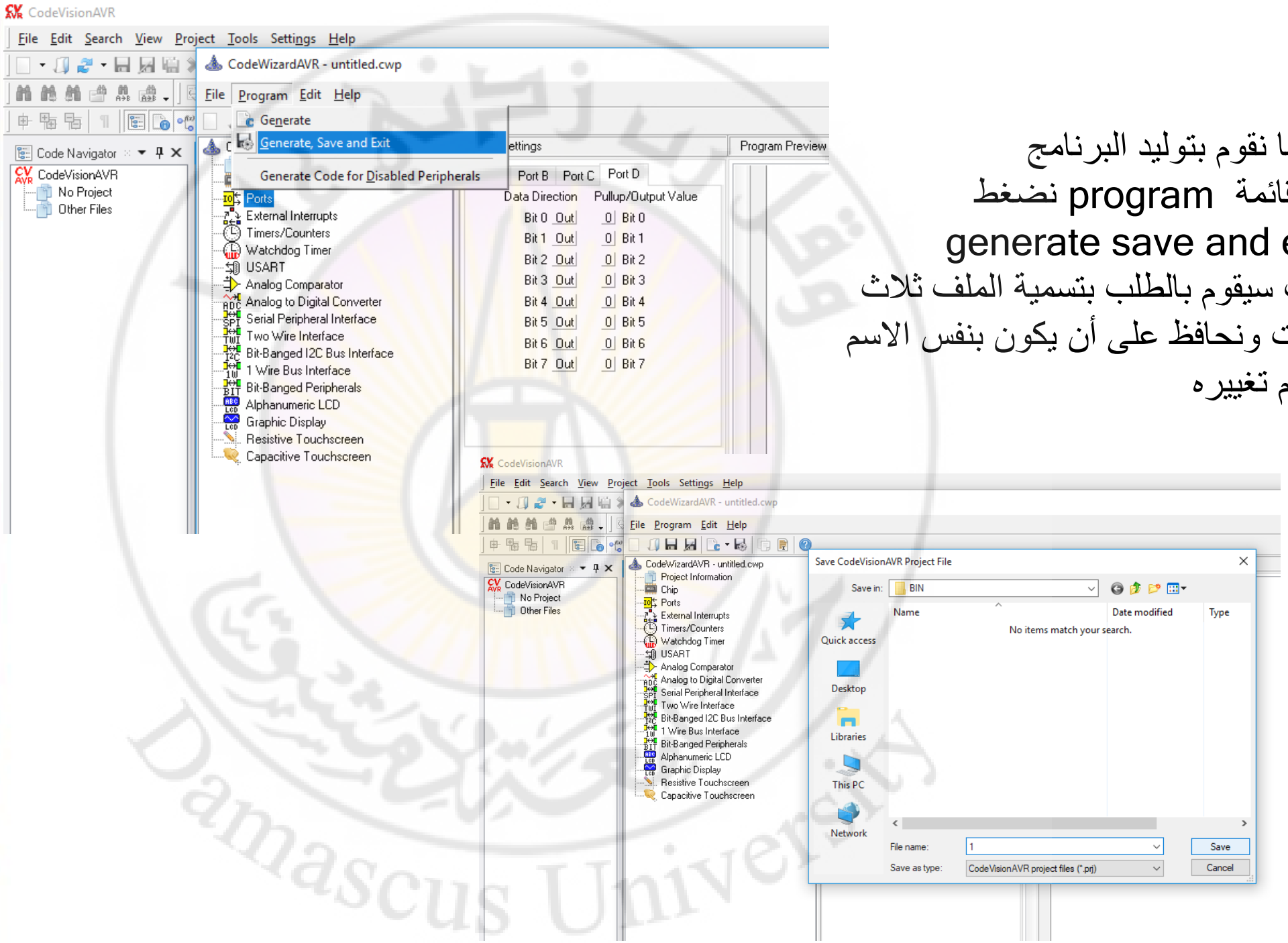
Pull up في هذه الحالة يتم تفعيل مقاومة الرفع فيزيائيا لتعطي قيمة 1 منطقي أي أن pin في حالة التفعيل.

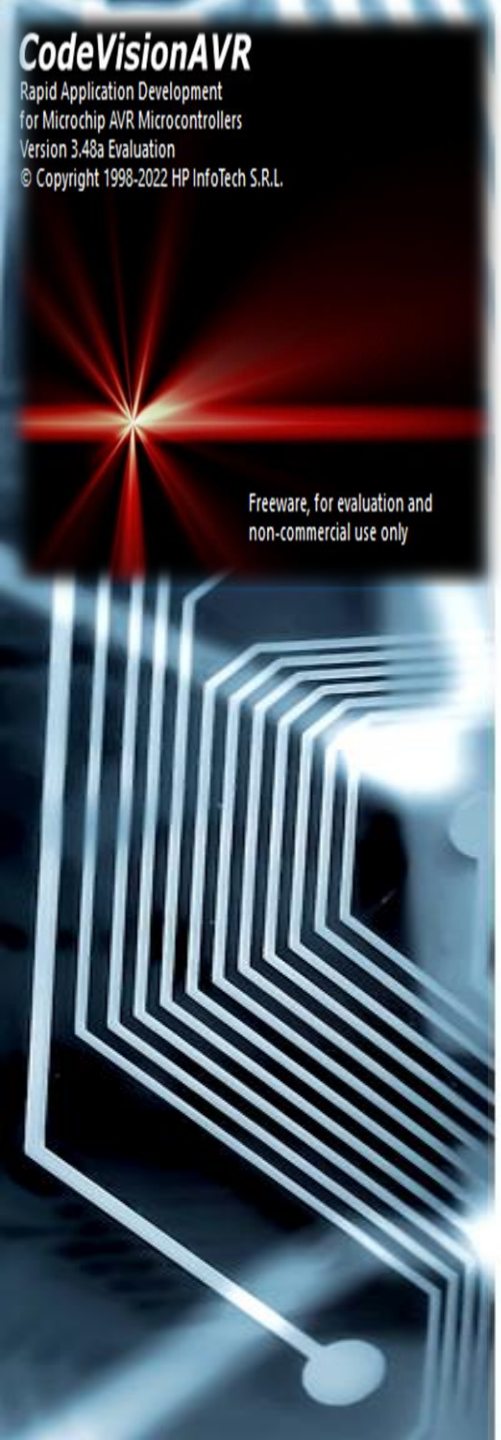
وبهذا نكون قمنا برمجة كل PIN على حدا





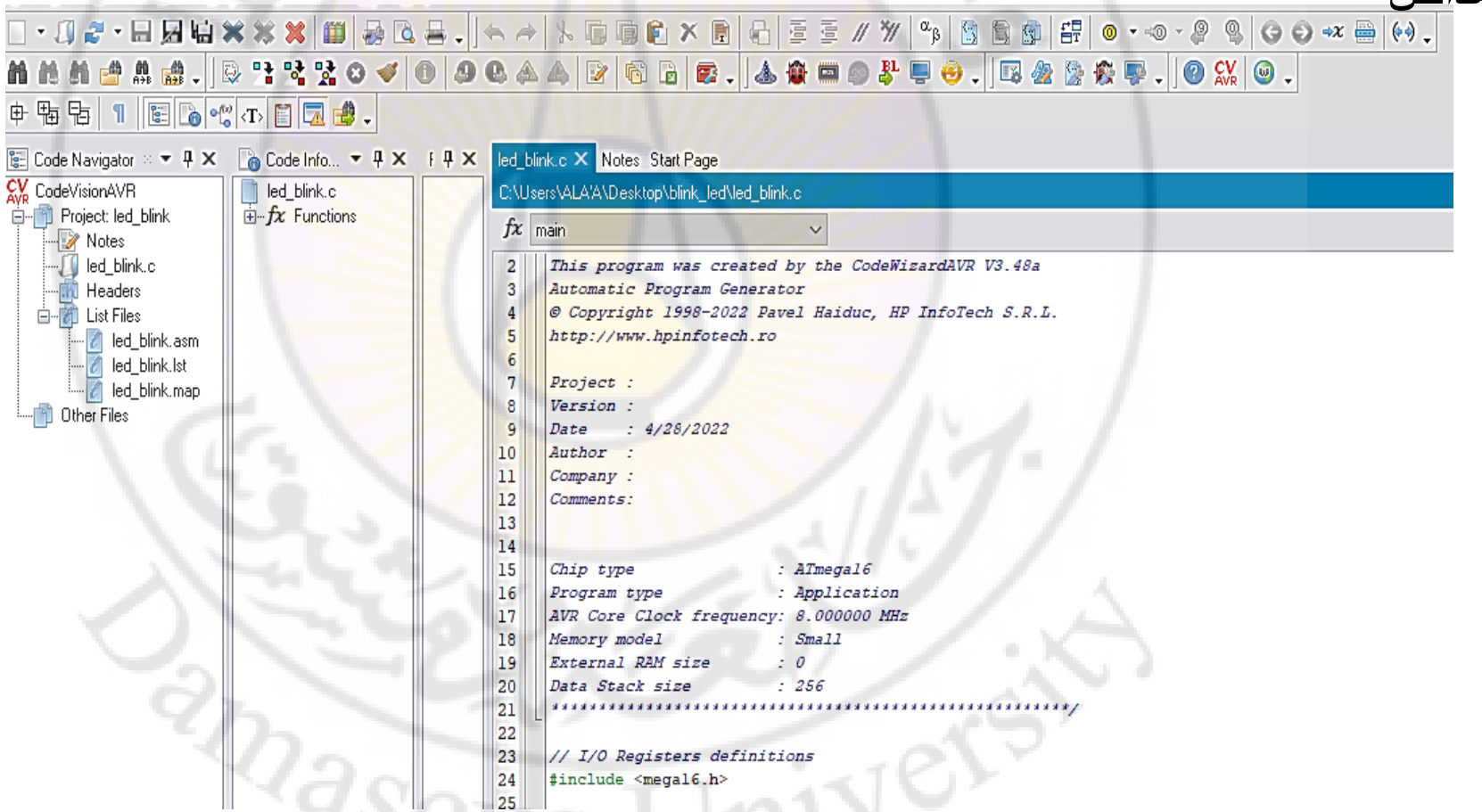

بعدها نقوم بتوليد البرنامج
من قائمة program نضغط
generate save and exit
حيث سيقوم بالطلب بتسمية الملف ثلاث
مرات ونحافظ على أن يكون بنفس الاسم
وعدم تغييره






وهكذا قام ببناء الشكل العام للبرنامج ، (تظهر حالة البرنامج – التاريخ – نوع المتحكم – التردد .....)

نتوجه الآن إلى التابع الأساسي void main حيث قام باستدعاء مكتبة mega.16 ونرى ان المداخل قد تم تعينها كما قمنا بضبطها سابقا وكما قام بكتابتها وفق نظام Hexa مباشرة مع ملاحظات عن كافة حالات المداخل



CodeVisionAVR - C:\Users\ALA'A\Desktop\blink_led\led_blink.prj

File Edit Search View Project Tools Settings Help

Code Navigator | Code Info... | led_blink.c | Notes | Start Page

CodeVisionAVR
- Project: led_blink
  - Notes
  - led_blink.c
  - Headers
  - List Files
    - led_blink.asm
    - led_blink.lst
    - led_blink.map
- Other Files

led_blink.c
- Functions

C:\Users\ALA'A\Desktop\blink_led\led_blink.c

main

```
27
28  void main(void)
29  {
30  // Declare your local variables here
31
32  // Input/Output Ports initialization
33  // Port A initialization
34  // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=Out
35  DDRA=(0<<DDA7) | (0<<DDA6) | (0<<DDA5) | (0<<DDA4) | (0<<DDA3) | (0<<DDA2) | (0<<DDA1) | (1<<DDA0);
36  // State: Bit7=P Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=0
37  PORTA=(1<<PORTA7) | (0<<PORTA6) | (0<<PORTA5) | (0<<PORTA4) | (0<<PORTA3) | (0<<PORTA2) | (0<<PORTA1) | (0<<PORTA0);
38
39  // Port B initialization
40  // Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
41  DDRB=(1<<DDB7) | (1<<DDB6) | (1<<DDB5) | (1<<DDB4) | (1<<DDB3) | (1<<DDB2) | (1<<DDB1) | (1<<DDB0);
42  // State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
43  PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);
44
45  // Port C initialization
46  // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
47  DDRC=(0<<DDC7) | (0<<DDC6) | (0<<DDC5) | (0<<DDC4) | (0<<DDC3) | (0<<DDC2) | (0<<DDC1) | (0<<DDC0);
48  // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
49  PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
50
```

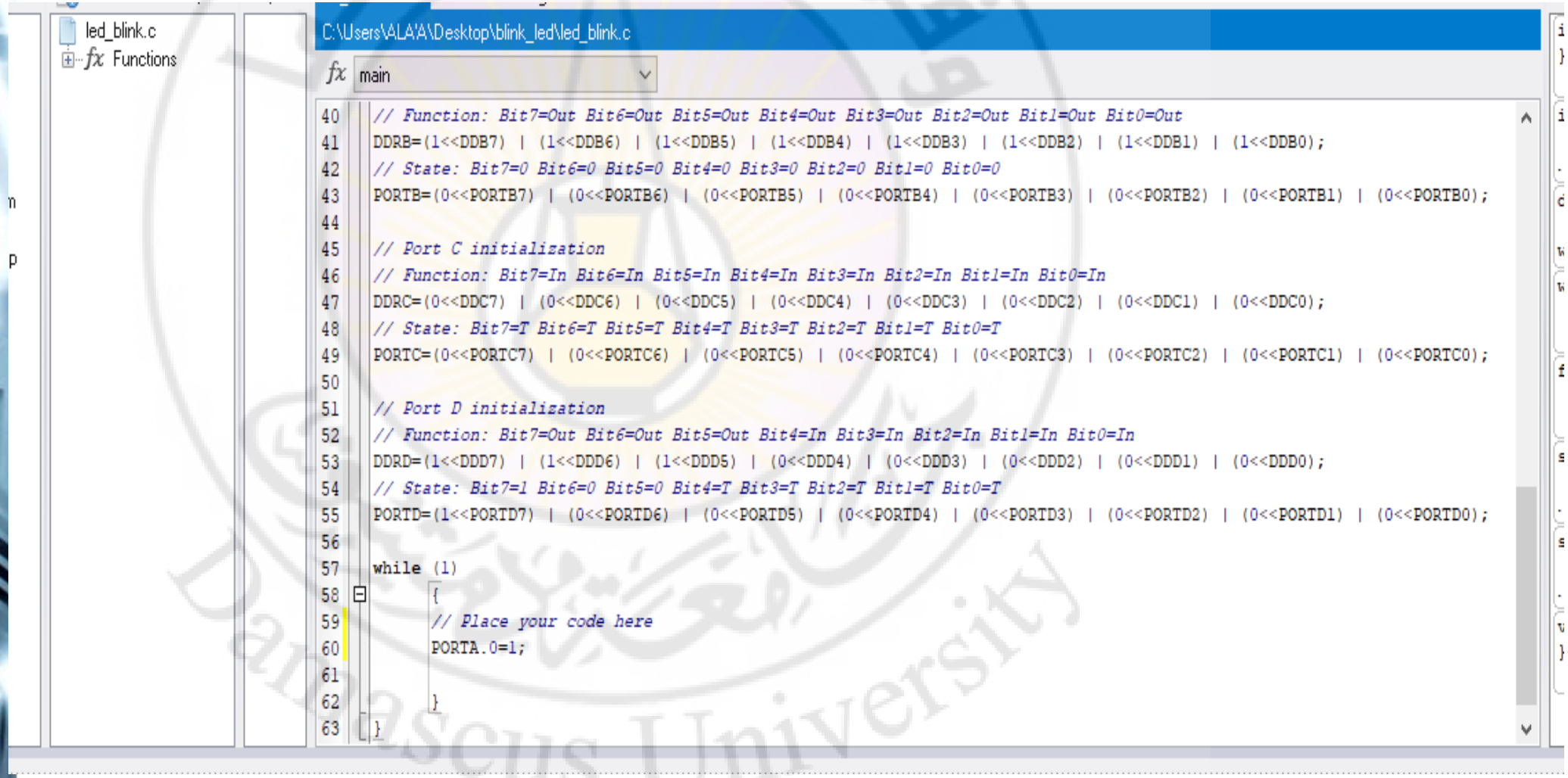
```
if ()
};

if ()
    }
...

do  {
    }
while

while
    };

for (;
    };

switch
    ca
...

switch
    ca
...

void m
}
```

Messages

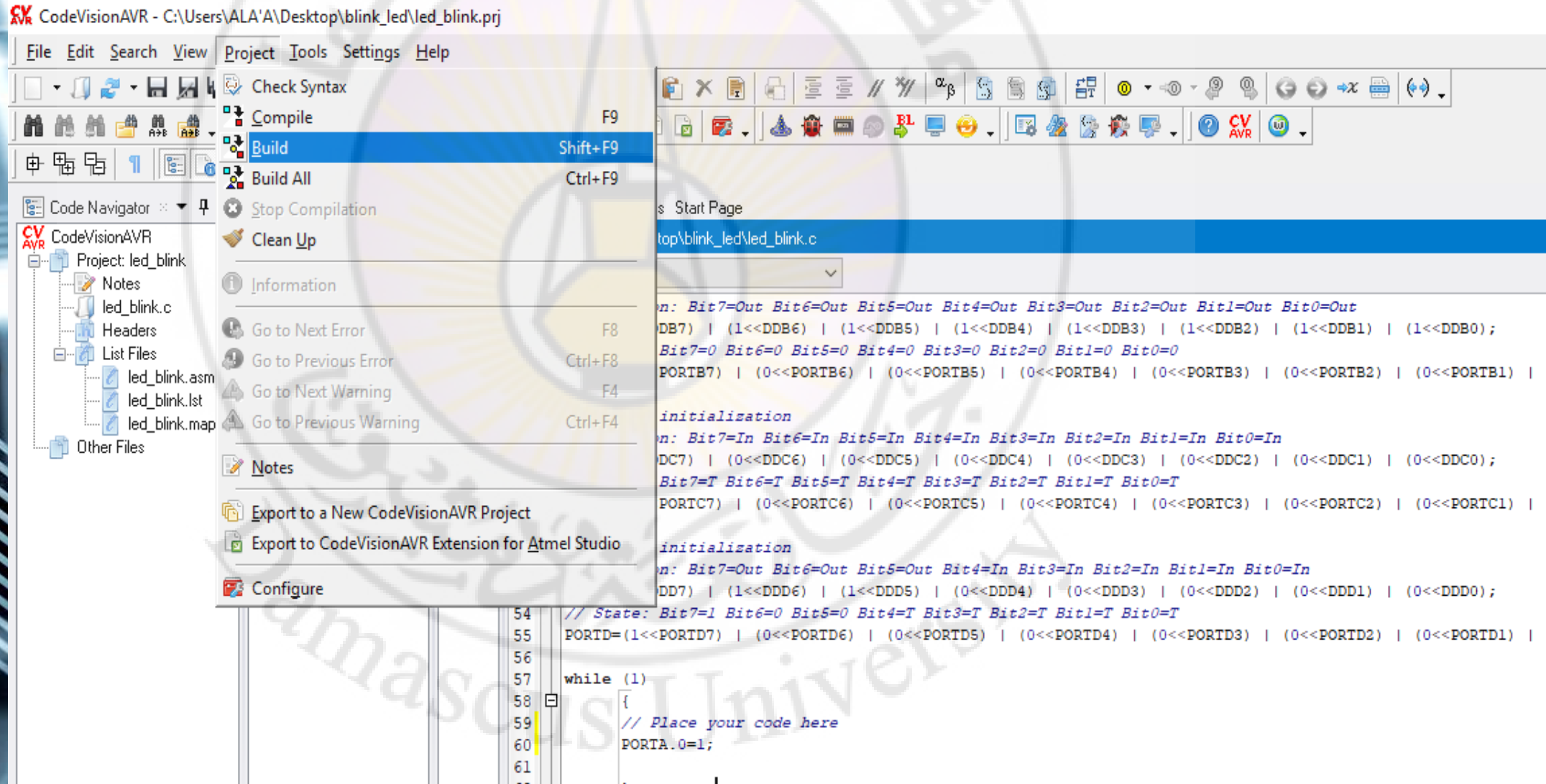
Errors | Warnings

القسم الأهم هو حلقة while اللانهائي والتي تقوم بالعمل بشكل مستمر طالما التغذية الكهربئيى موجودة .

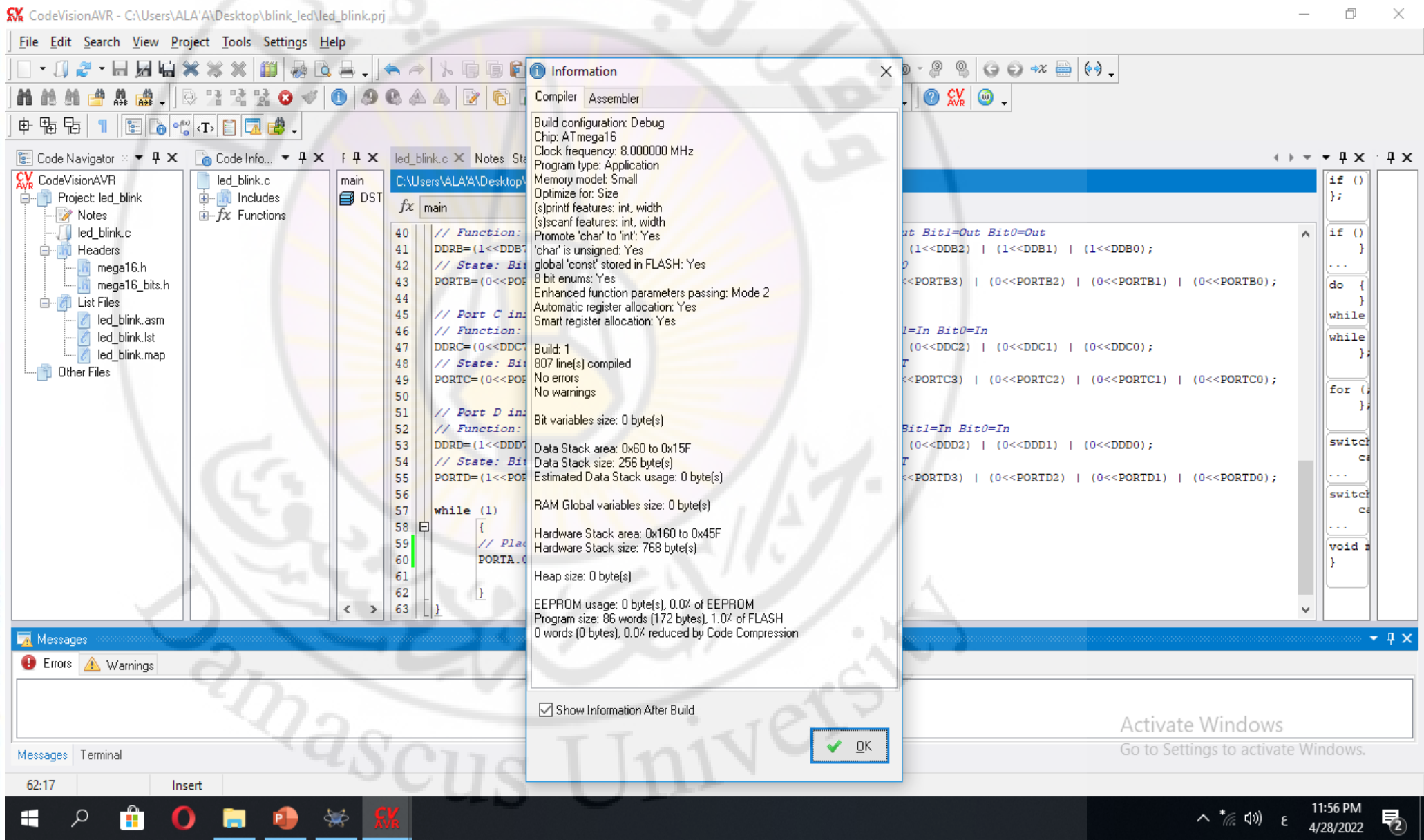
نقوم بكتابة الكود المطلوب كما في الشكل :



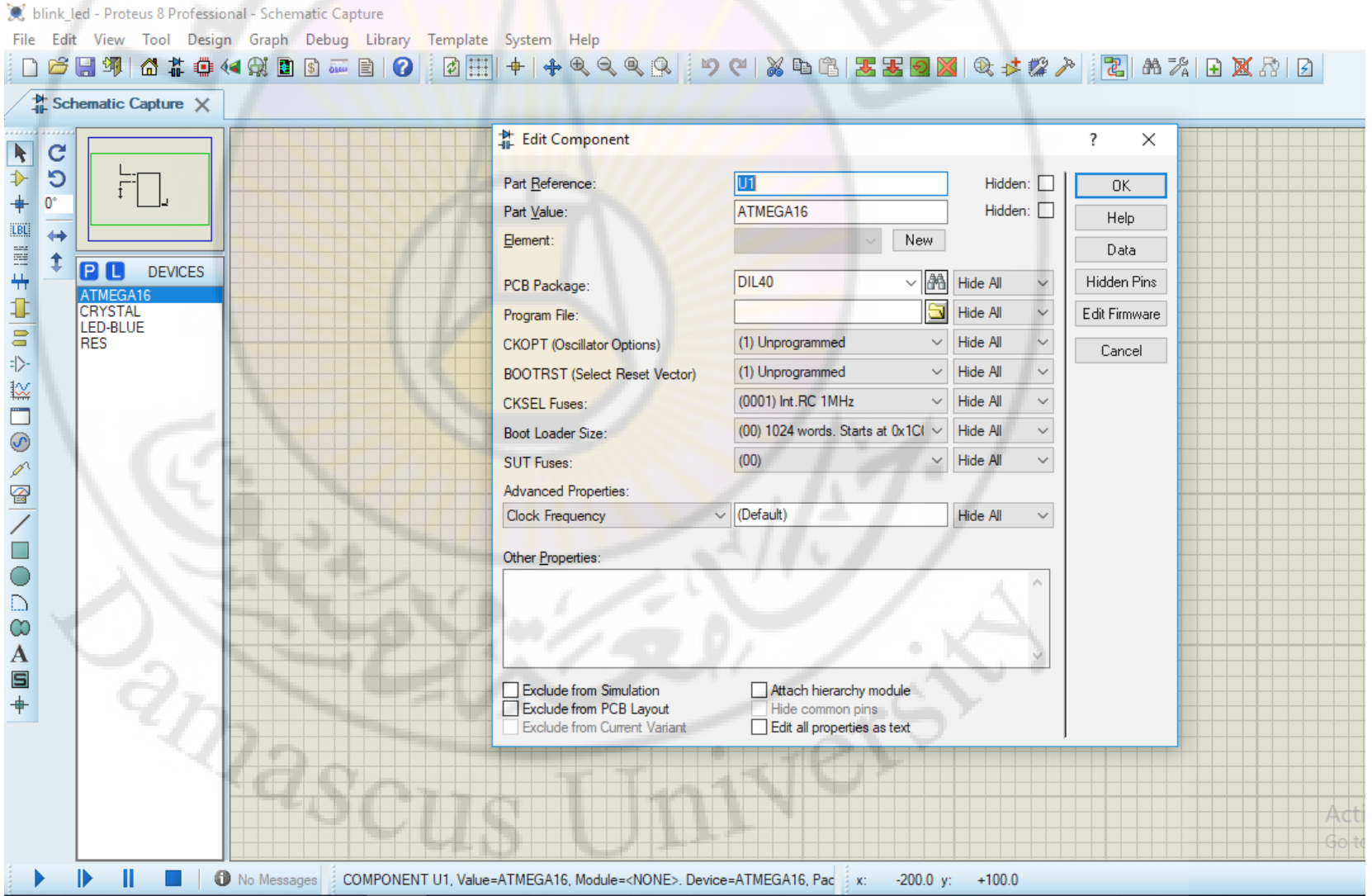
ثم ننتقل إلى التبويب project ونختار Build (ليتم بناء البرنامج بلغة الحاسب 1و0) Hexa ، ومن ثم يتم نقله إلى المتحكم

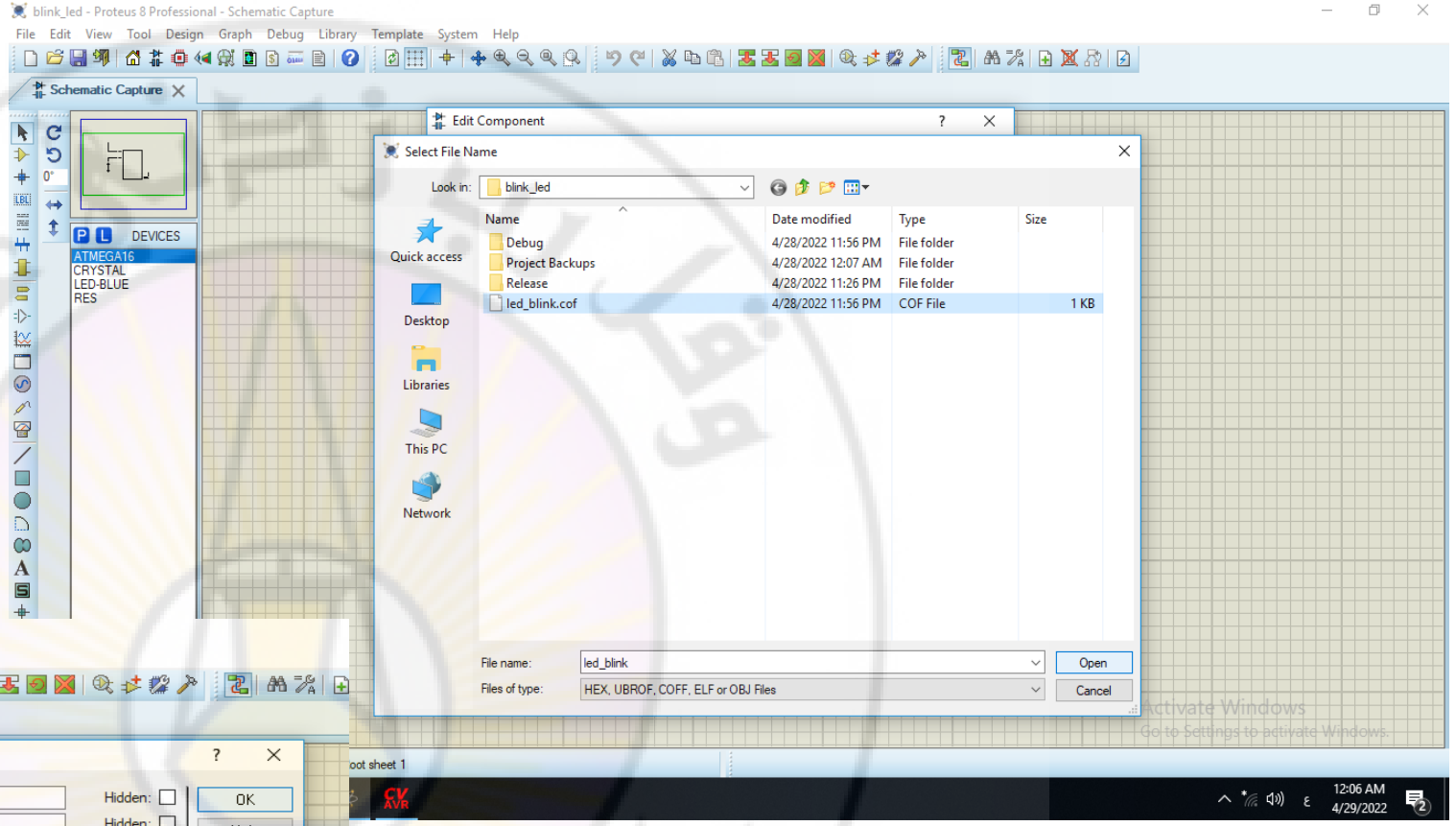


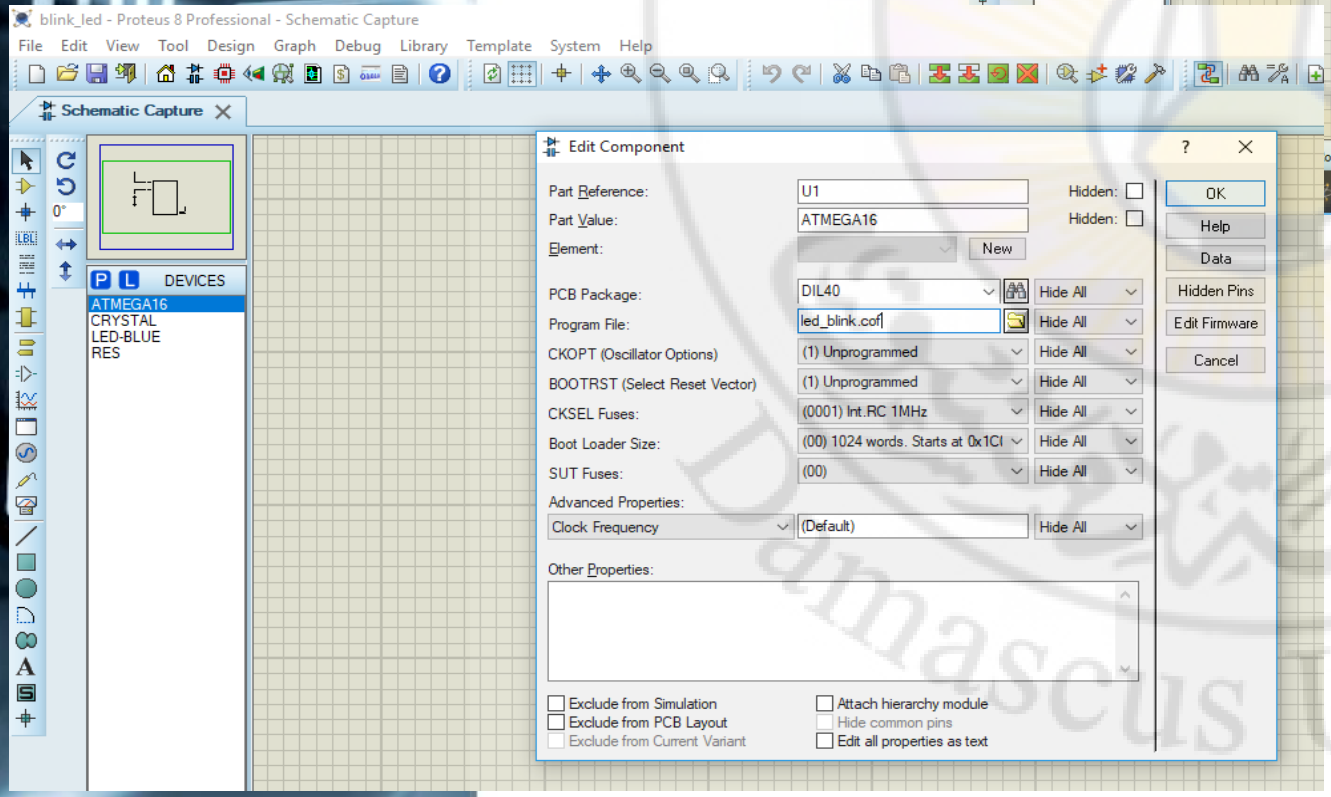
حيث تظهر شاشة تبين معلومات عن المتحكم - الذاكرة - التوقيت - الأخطاء.



يتم نقل هذا الملف إلى المتحكم من خلال برنامج المحاكاة protues ...
نضغط مرتين على المتحكم حتى تظهر شاشة نختار منها file ، ونختار اسم الملف




blink_led - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
ATMEGA16
CRYSTAL
LED-BLUE
RES
Edit Component
Select File Name
Look in: blink_led
Name
Date modified
Type
Size
Debug
4/28/2022 11:56 PM
File folder
Project Backups
4/28/2022 12:07 AM
File folder
Release
4/28/2022 11:26 PM
File folder
led_blink.cof
4/28/2022 11:56 PM
COF File
1 KB
Quick access
Desktop
Libraries
This PC
Network
File name: led_blink
Files of type: HEX, UBROF, COFF, ELF or OBJ Files
Open
Cancel
Activate Windows
Go to Settings to activate Windows.
12:06 AM
4/29/2022

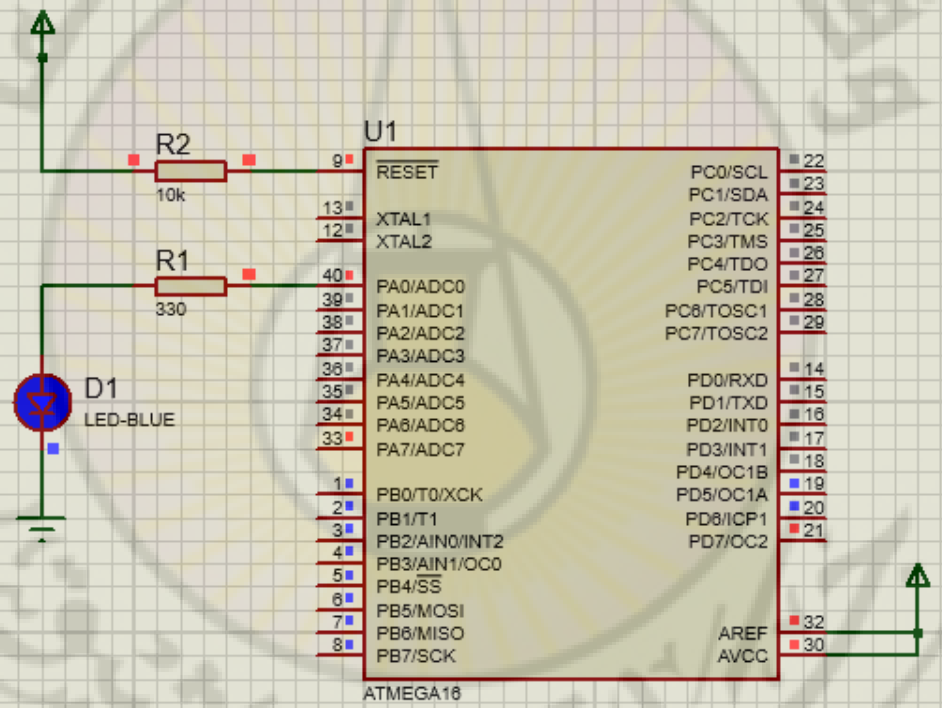

blink_led - Proteus 8 Professional - Schematic Capture
File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
DEVICES
ATMEGA16
CRYSTAL
LED-BLUE
RES
Edit Component
Part Reference: U1
Hidden:
Part Value: ATMEGA16
Hidden:
Element:
New
PCB Package: DIL40
Hide All
Program File: led_blink.cof
Hide All
CKOPT (Oscillator Options): (1) Unprogrammed
Hide All
BOOTRST (Select Reset Vector): (1) Unprogrammed
Hide All
CKSEL Fuses: (0001) Int.RC 1MHz
Hide All
Boot Loader Size: (00) 1024 words. Starts at 0x1C
Hide All
SUT Fuses: (00)
Hide All
Advanced Properties:
Clock Frequency
(Default)
Hide All
Other Properties:
Exclude from Simulation
Exclude from PCB Layout
Exclude from Current Variant
Attach hierarchy module
Hide common pins
Edit all properties as text
OK
Help
Data
Hidden Pins
Edit Firmware
Cancel

blink_led - Proteus 8 Professional - Schematic Capture

File Edit View Tool Design Graph Debug Library Template System Help

Schematic Capture

نشغل البرنامج ليضيء الليد




DEVICES
ATMEGA16
CRYSTAL
LED-BLUE
RES

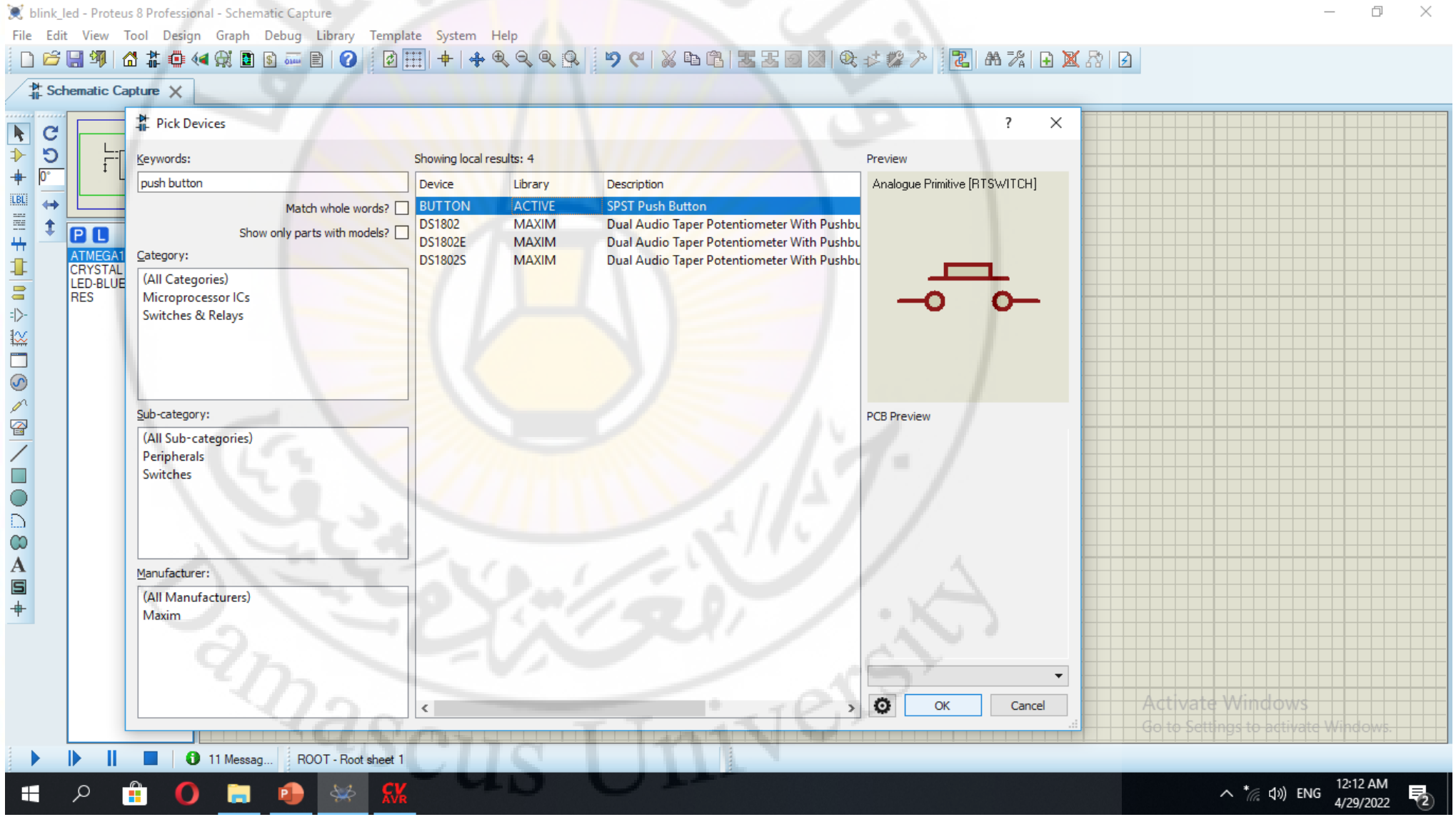
5 Message(s) ANIMATING: 00:00:04.800000 (CPU load 10%) x: -4700.0 y: -2300.0

Activate Windows
Go to Settings to activate Windows.

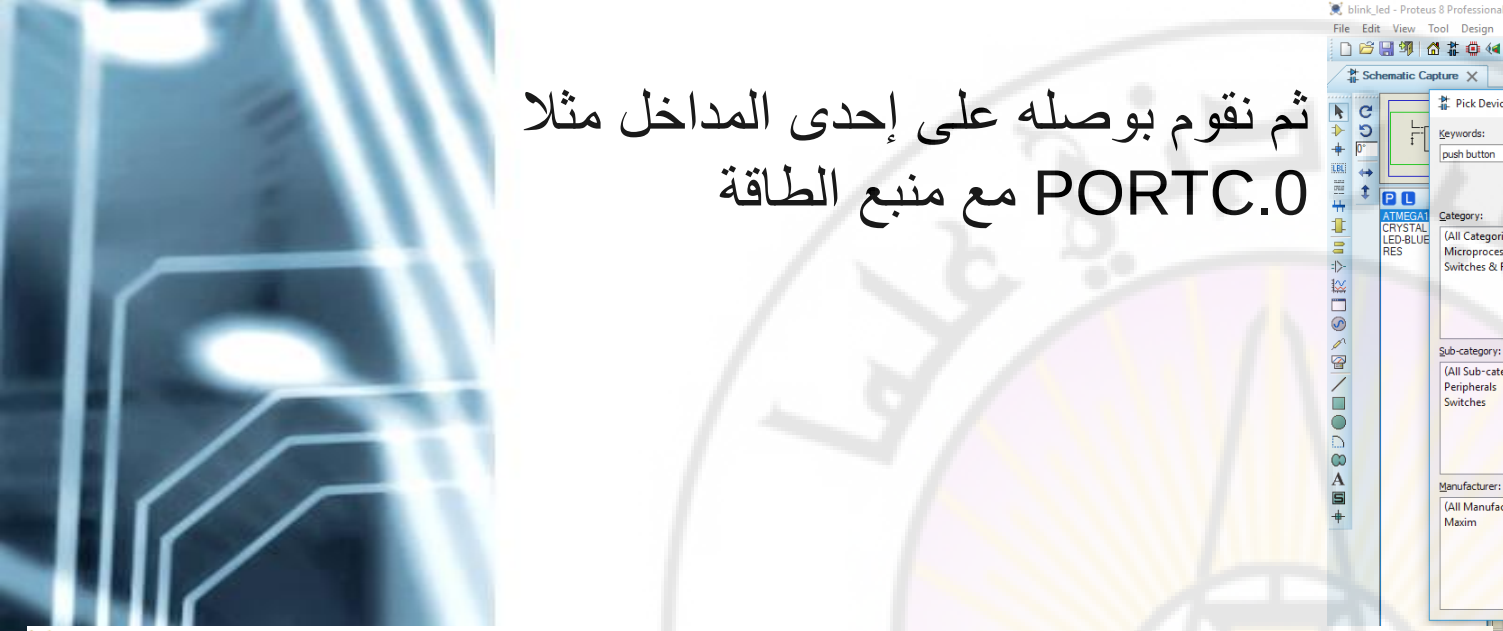
12:09 AM
4/29/2022

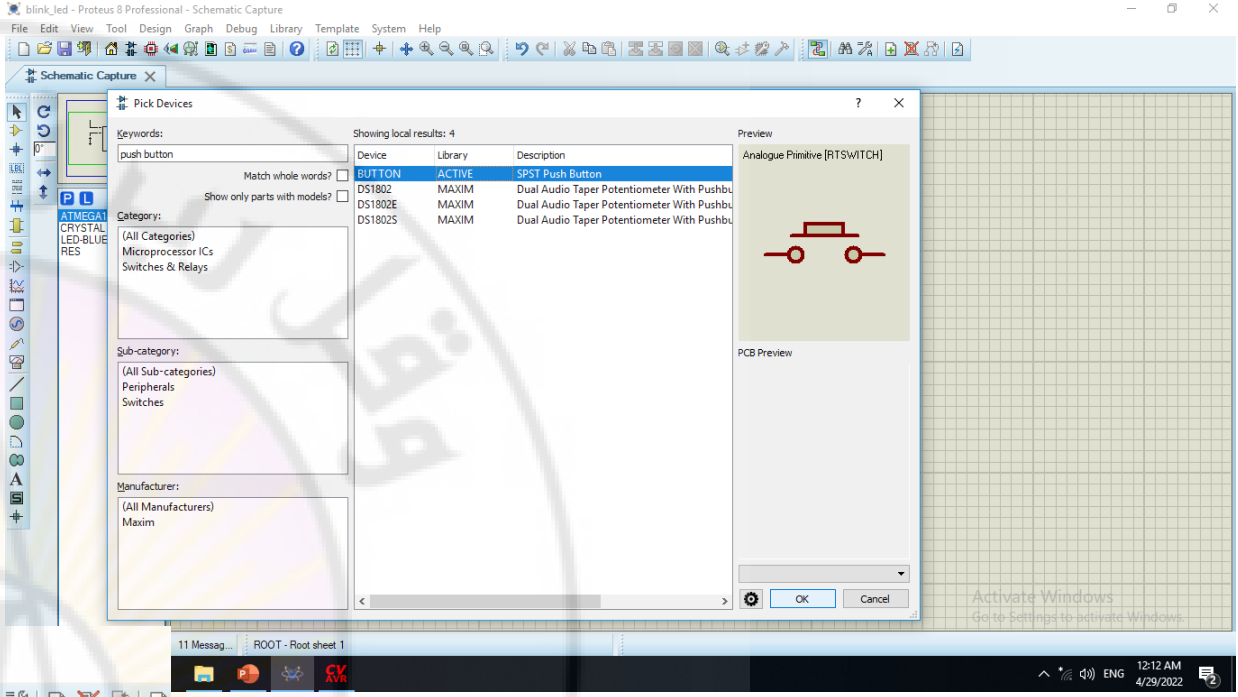
نقوم بتعديل على الدارة بإضافة زر تشغيل كما يلي :

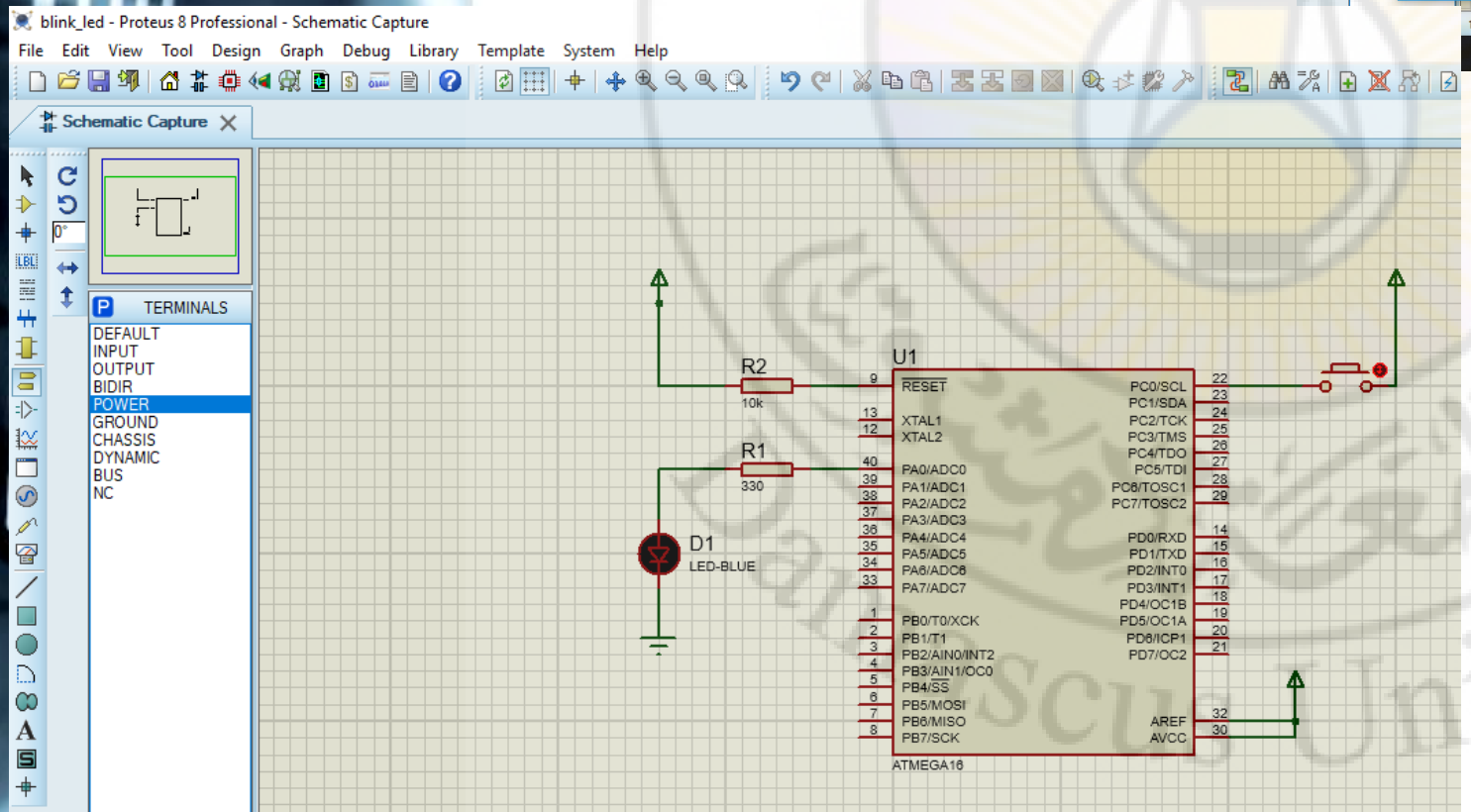
نختار push button من البروتيوس



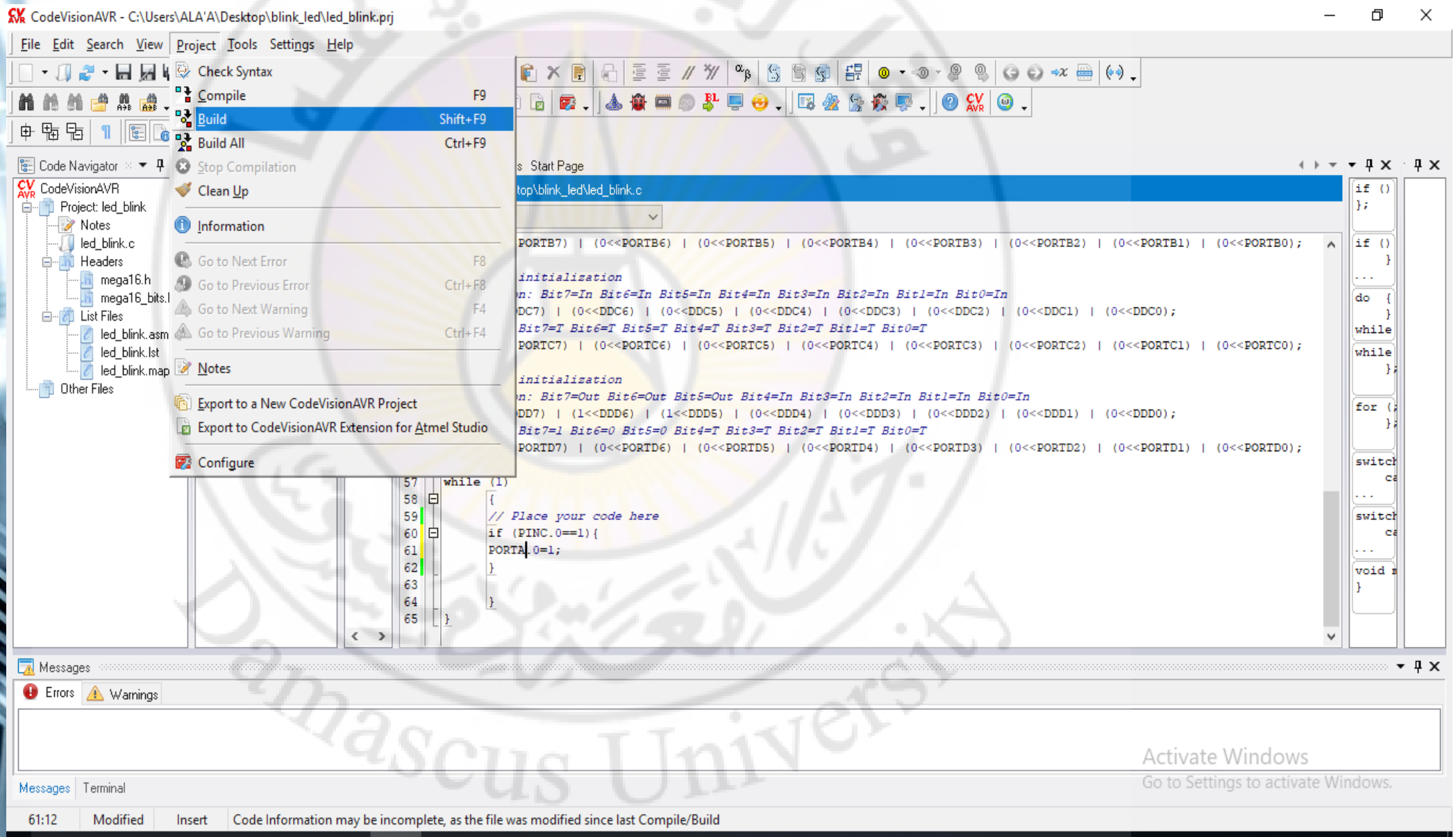
ثم نقوم بوصله على إحدى المداخل مثلا PORTC.0 مع منبع الطاقة





ثم نقوم بالتعديل على الكود بإضافة حلقة if الشرطية ، مع الانتباه لكتابة السجل PIN لأننا نتعامل مع منفذ دخل ....



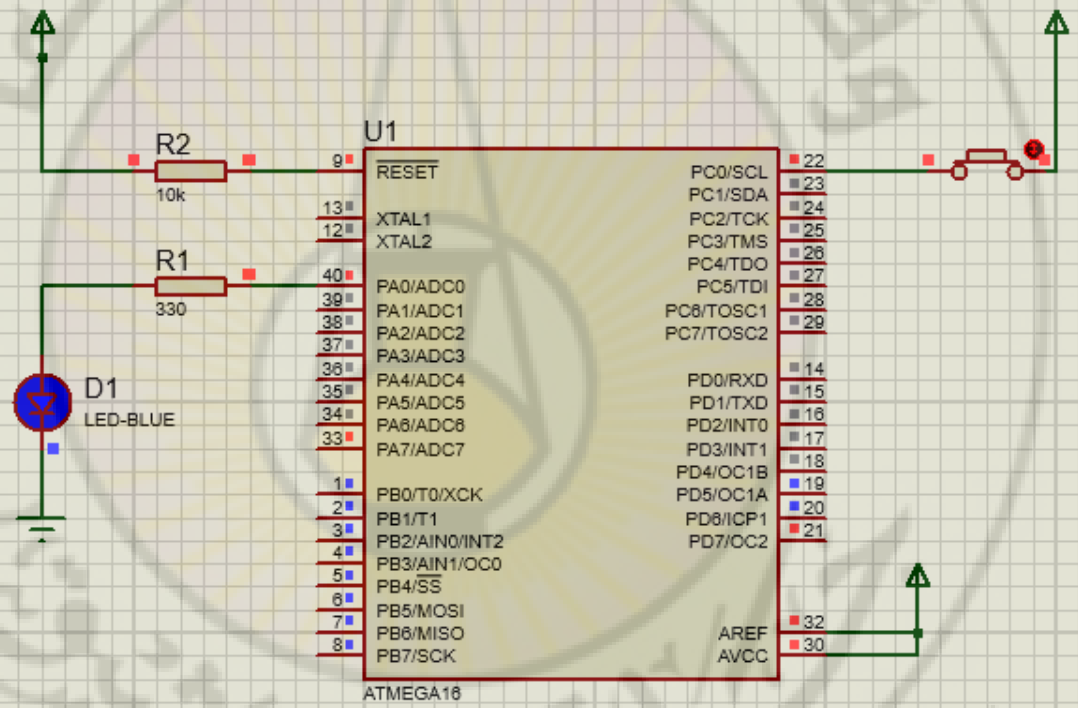
blink_led - Proteus 8 Professional - Schematic Capture

File Edit View Tool Design Graph Debug Library Template System Help

Schematic Capture

TERMINALS

DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
CHASSIS
DYNAMIC
BUS
NC



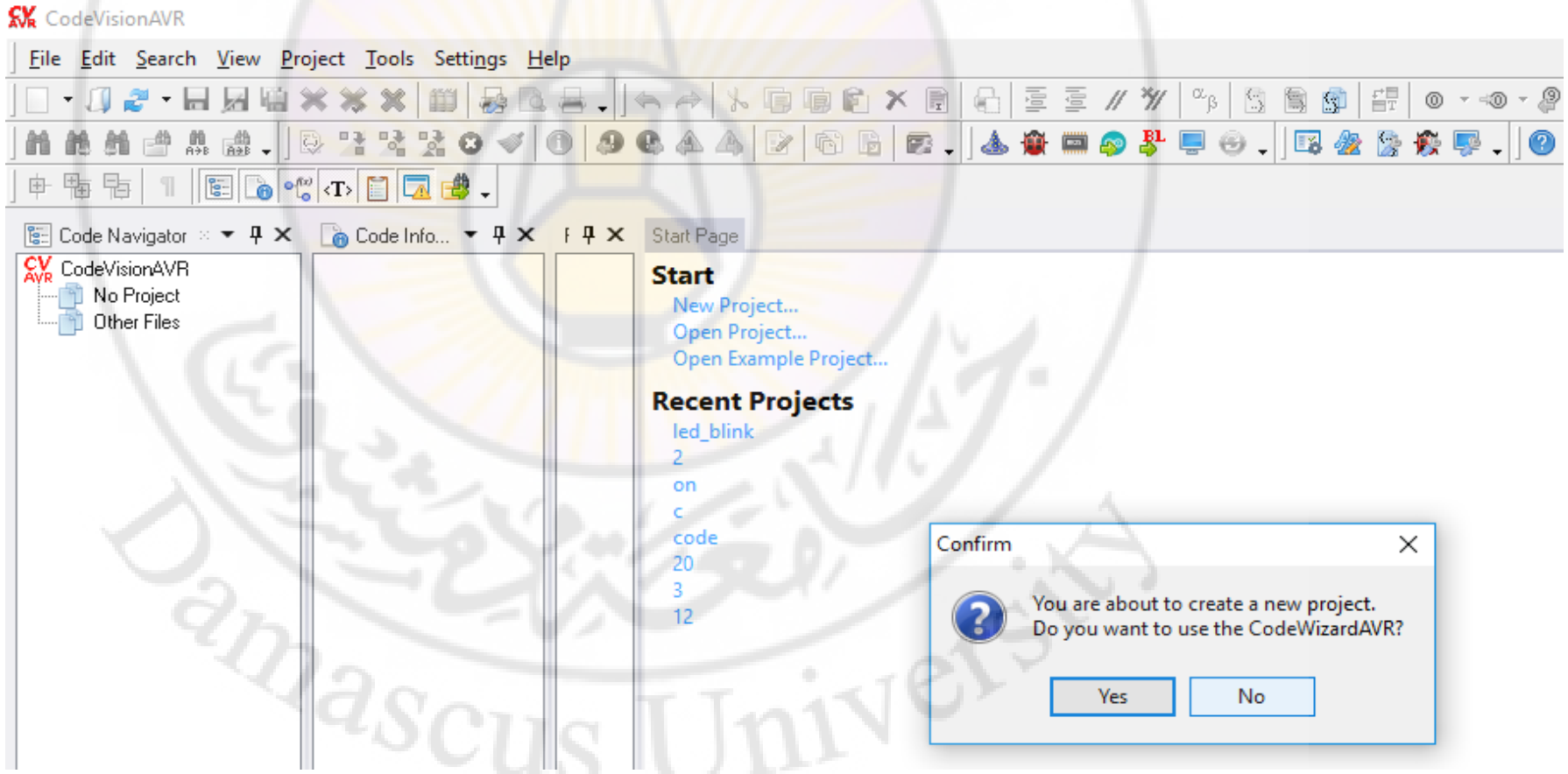

5 Message(s) ANIMATING: 00:00:04.550000 (CPU load 19%) x: +1700.0 y: +1100.0

Activate Windows
Go to Settings to activate Windows.

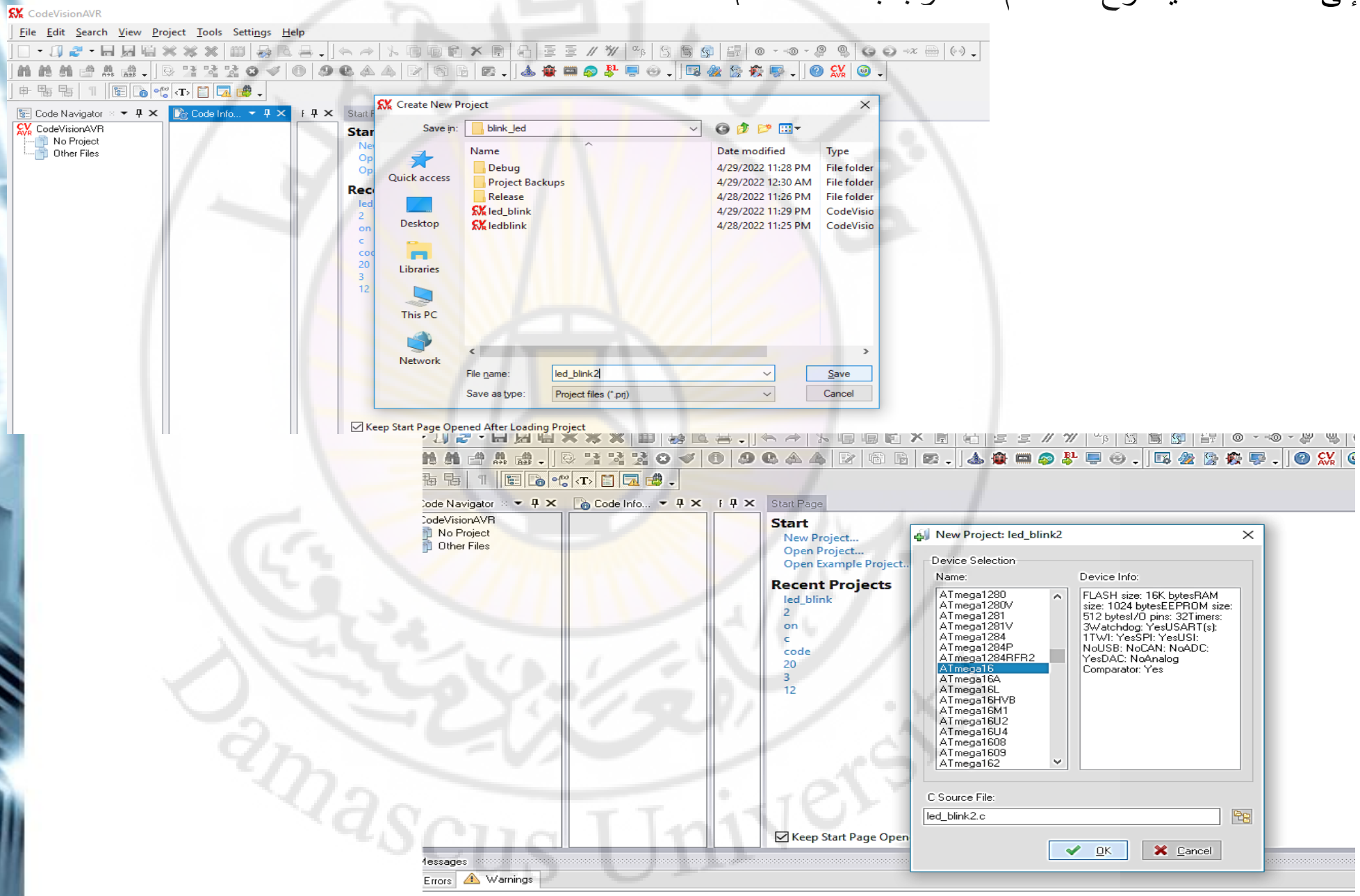
ENG 12:29 AM 4/29/2022

الان سوف نقوم بتعديل الكود لجعل الليد يومض flasher لمدة 1 ثانية ويطفئ لمدة 1 ثانية وهكذا ... تكون عند طريق إضافة مكتبة delay من اجل ضبط أزمنة الوصل والقطع لليد .. ونعدل الكود بإضافة عبارة ()delay_ms

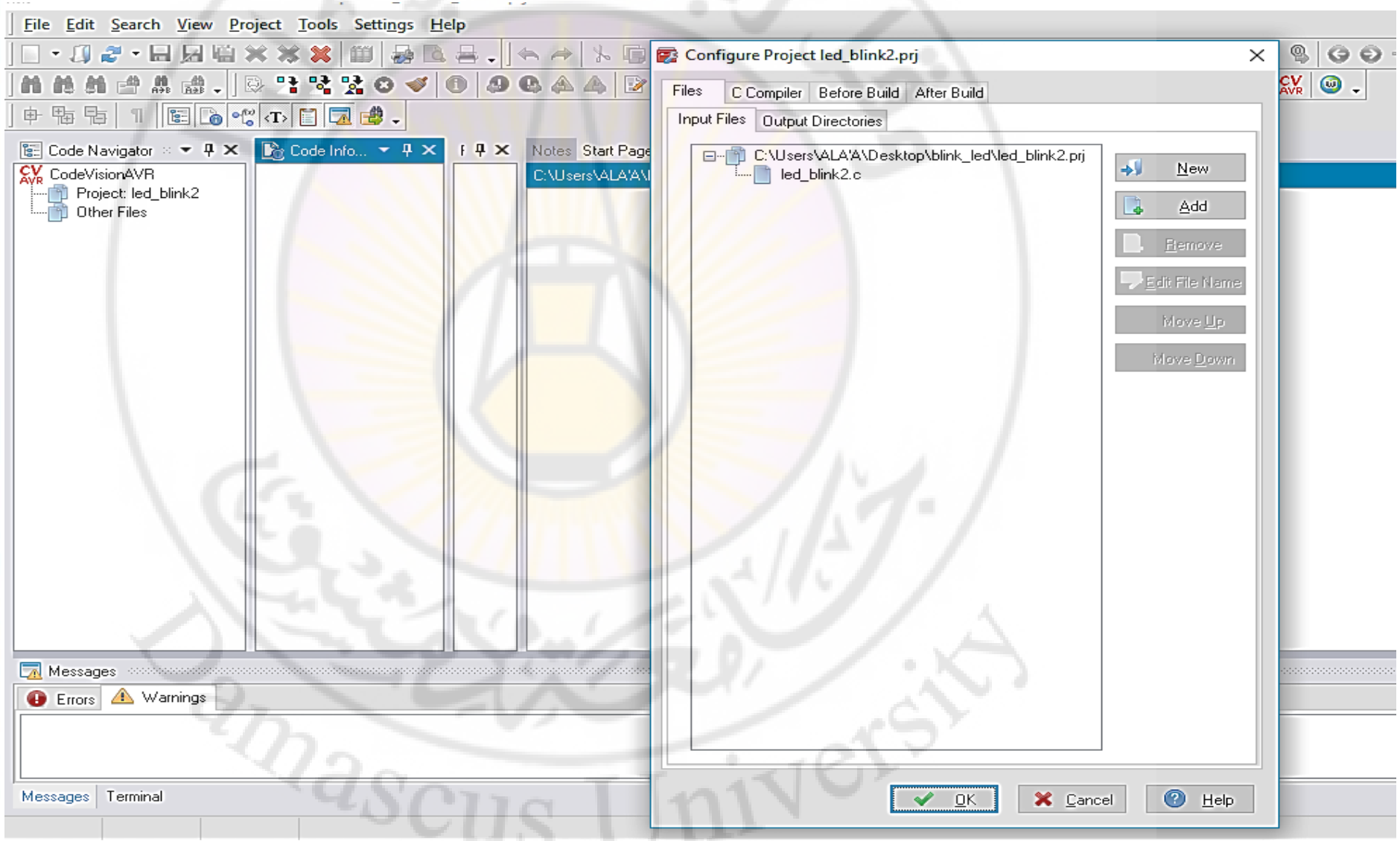
ولكننا سوف نقوم بكتابتها عن طريق برمجة كل الـ PORT وباستخدام source file كما يلي:
من قائمة new نختار source file
لكن أولا نقوم باختيار project file من قائمة file ولكن لانأخذ خيار المساعدة ، نقوم بالضغط على no



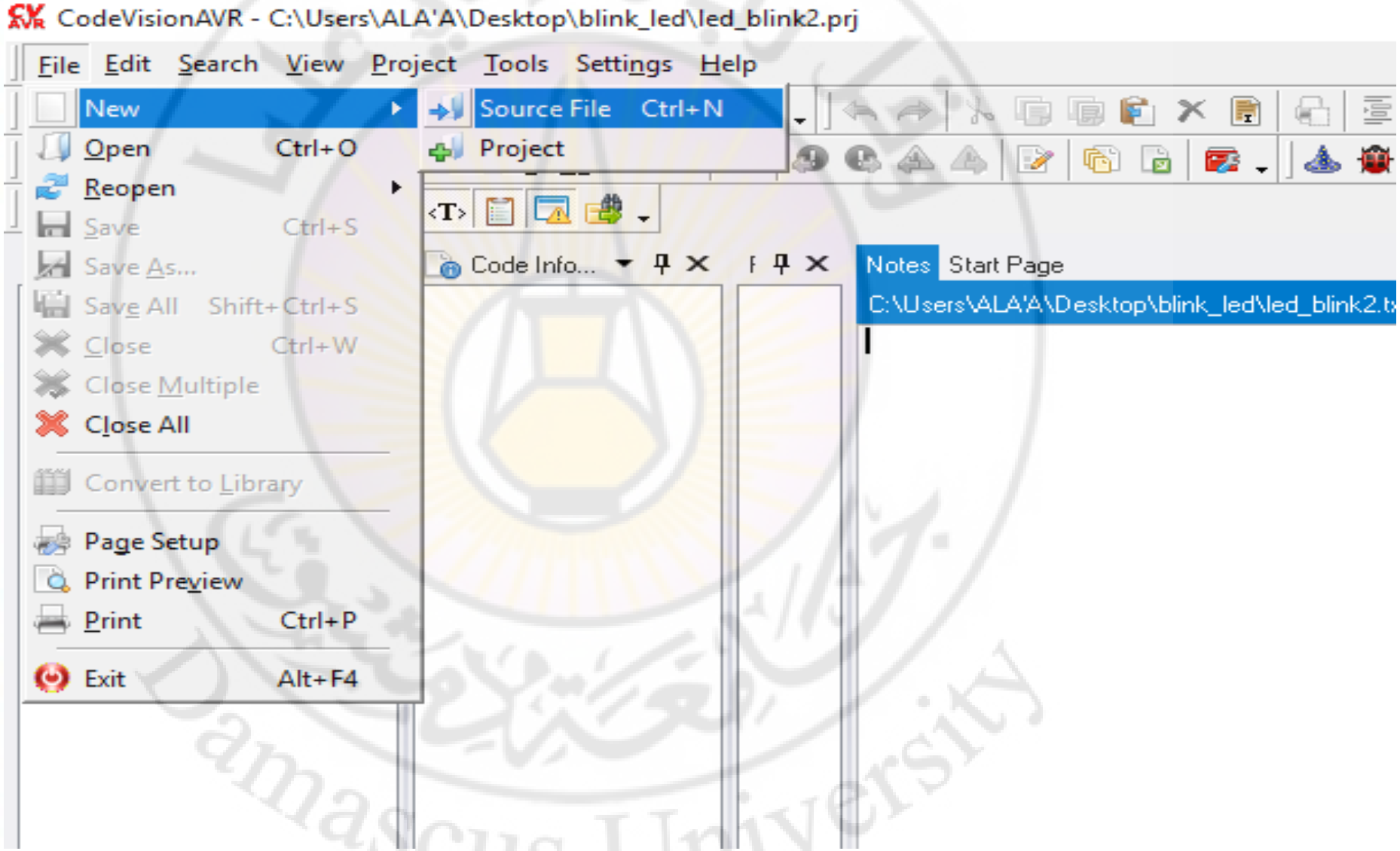
ينتقل إلى صفحة لتحديد نوع المتحكم المطلوب بعد حفظ اسم الملف



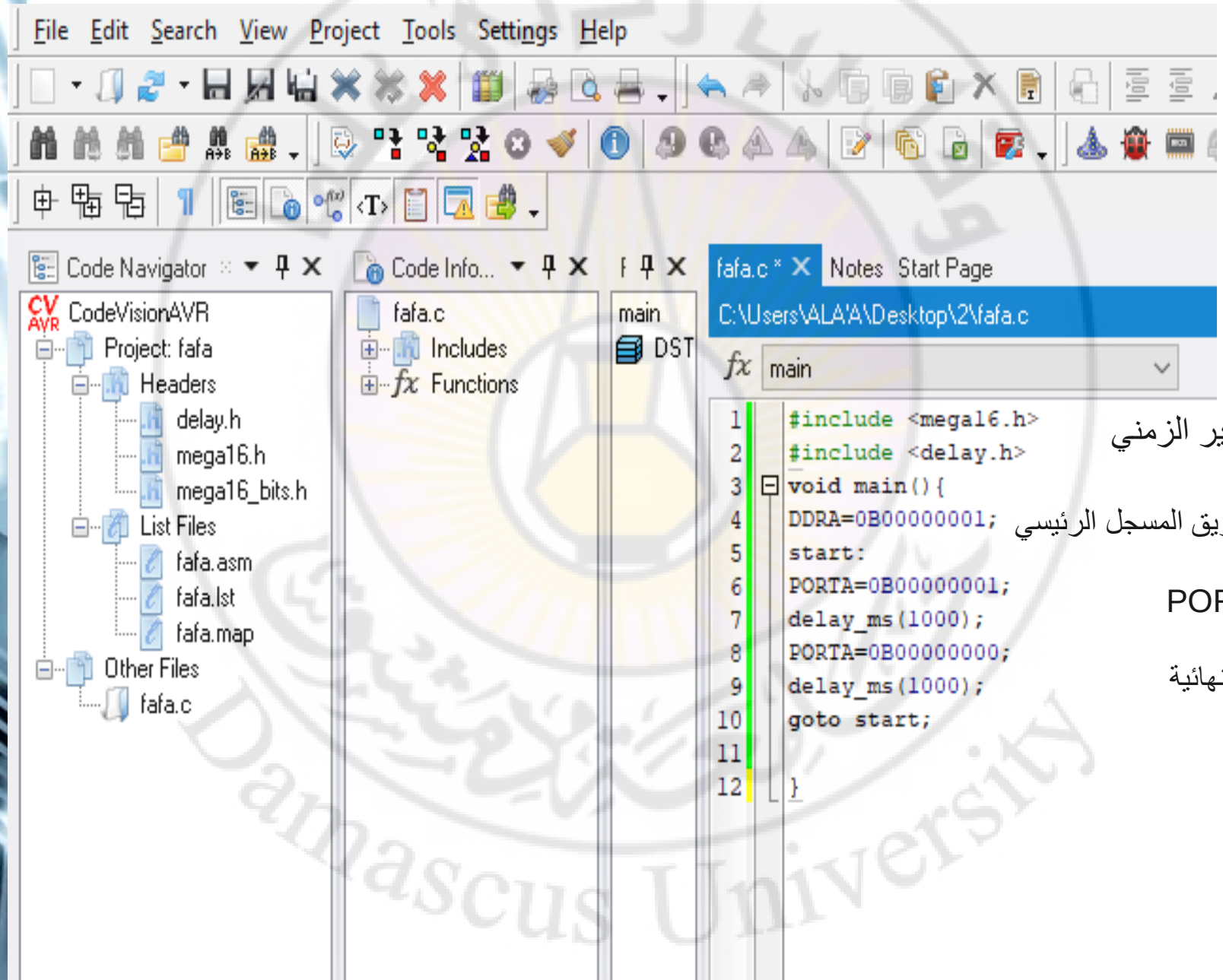
تظهر قائمة configuration لإضافة الملف



ثم نقوم بفتح ملف Source file لتظهر شاشة لكتابة الكود



تظهر لنا صفحة فارغة لكتابة الكود يدويا ،، نقوم بالكتابة كما في الشكل التالي :



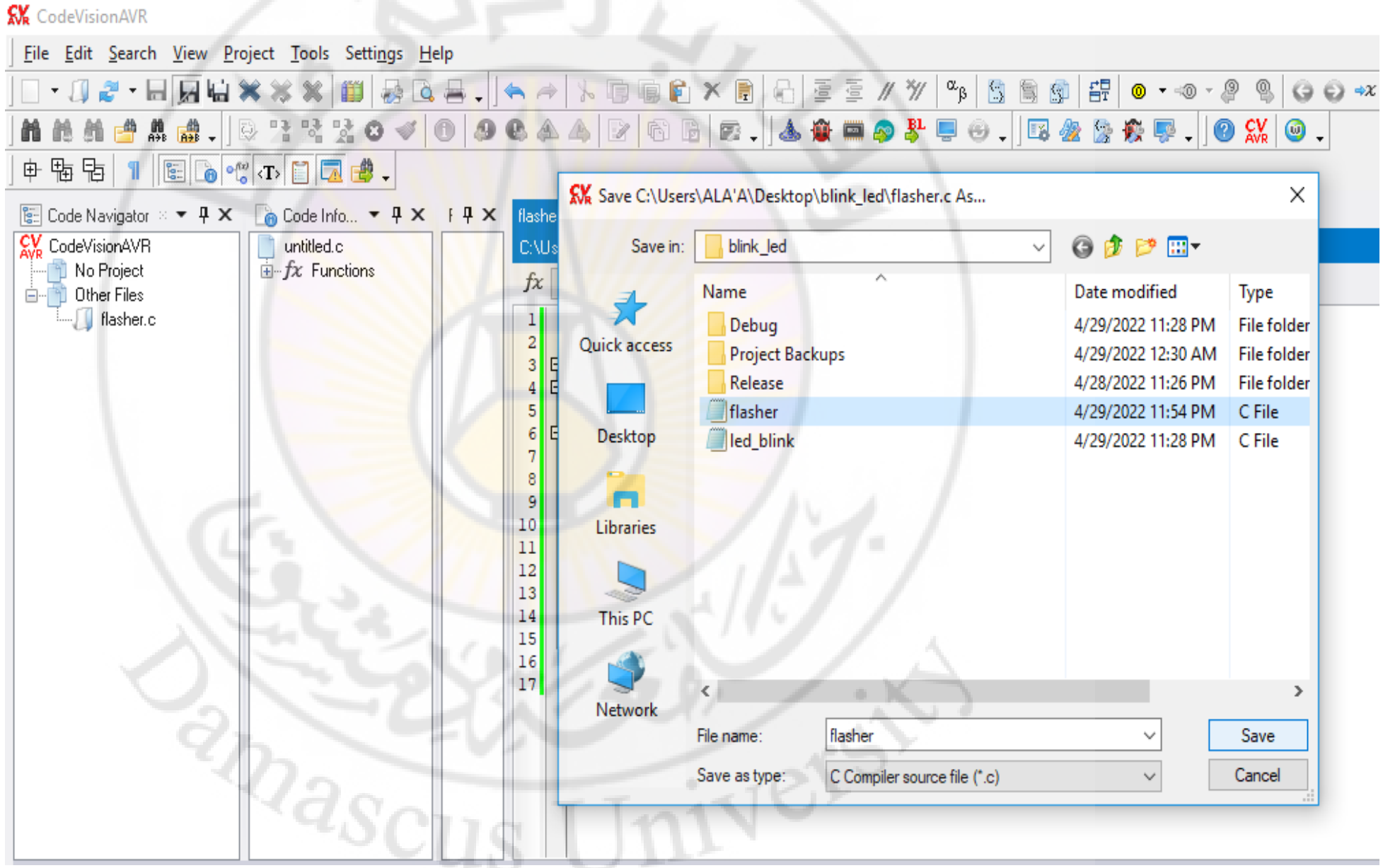

استدعاء مكتبة المتحكم والتاخير الزمني

استدعاء التابع الرئيسي

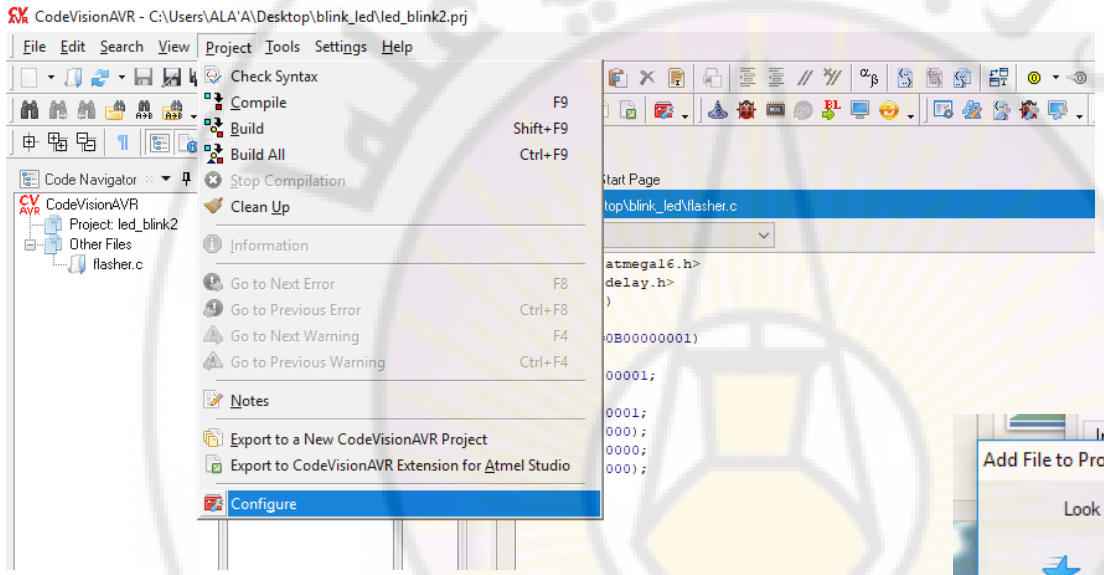
تحديد وظيفة الـ PORT عن طريق المسجل الرئيسي

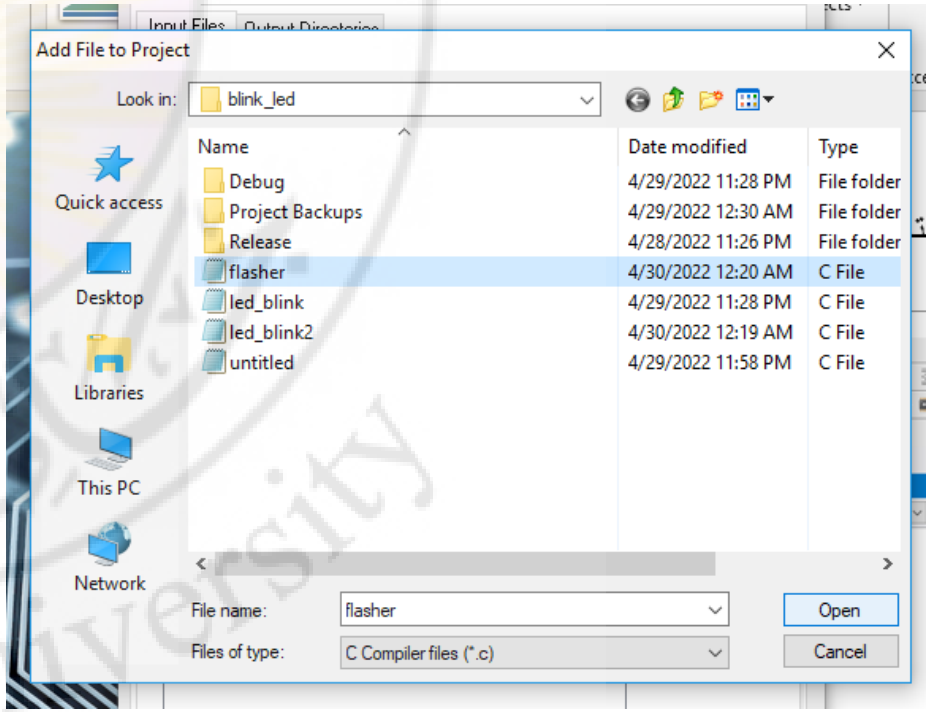
إضاءة اول خانة من خانات PORTA
وإطفاؤه باستخدام التأخير الزمني
وقمنا باستخدام حلقة go to اللانهائية

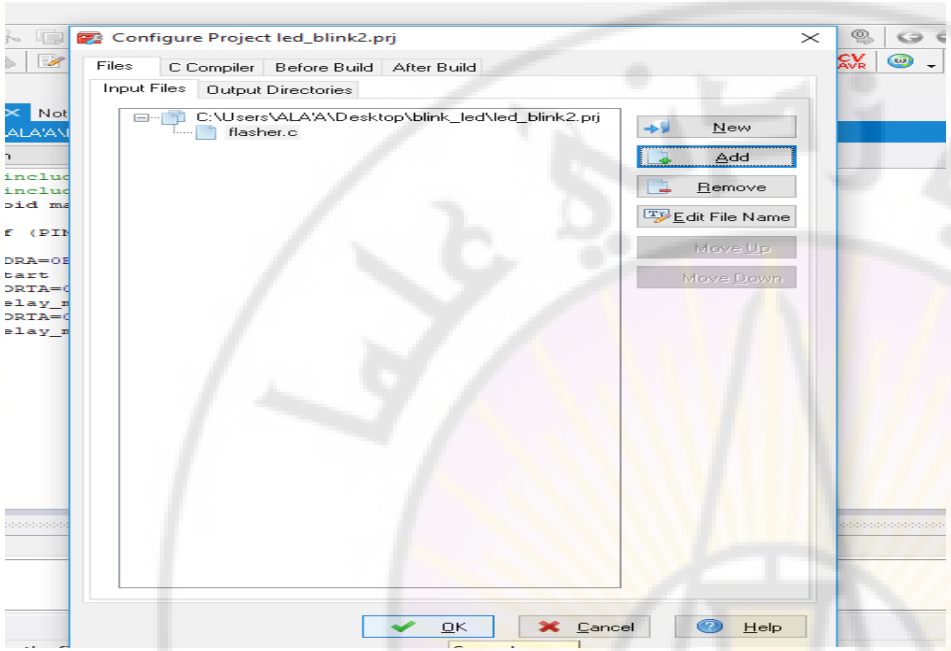
ثم نقوم بحفظ الملف باسم مناسب من قائمة save as



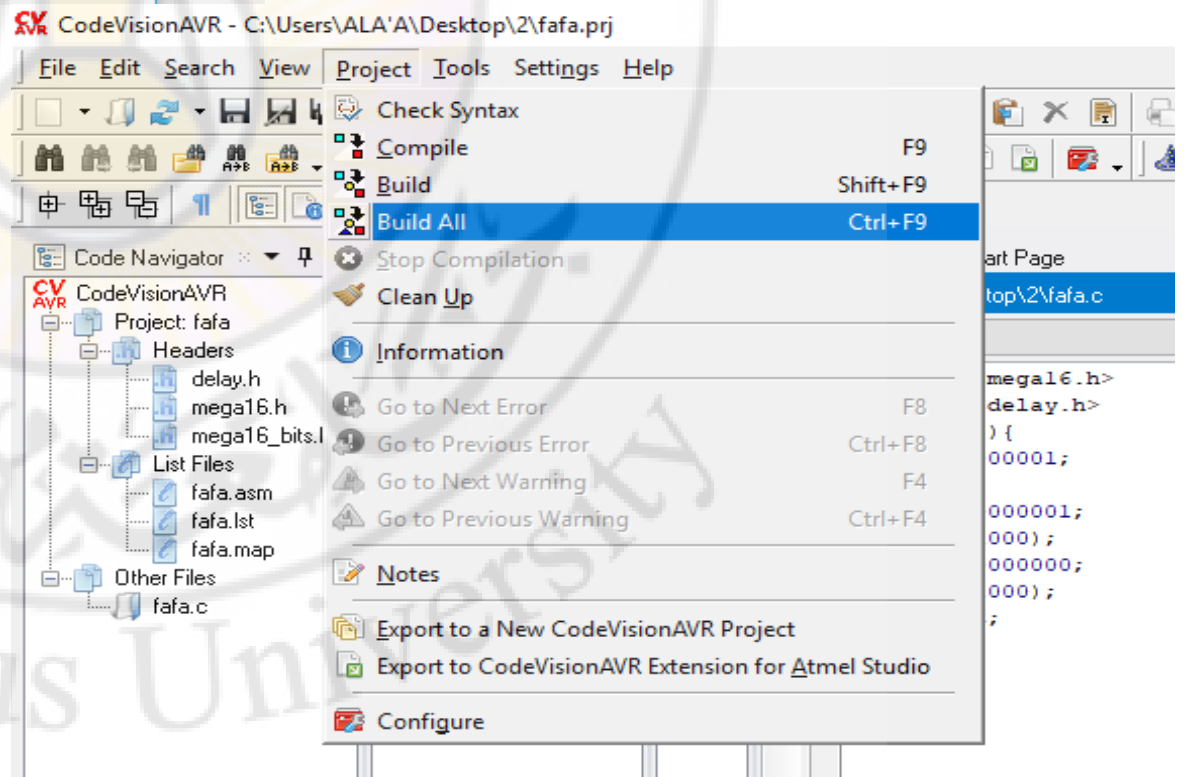
من قائمة project نختار configure



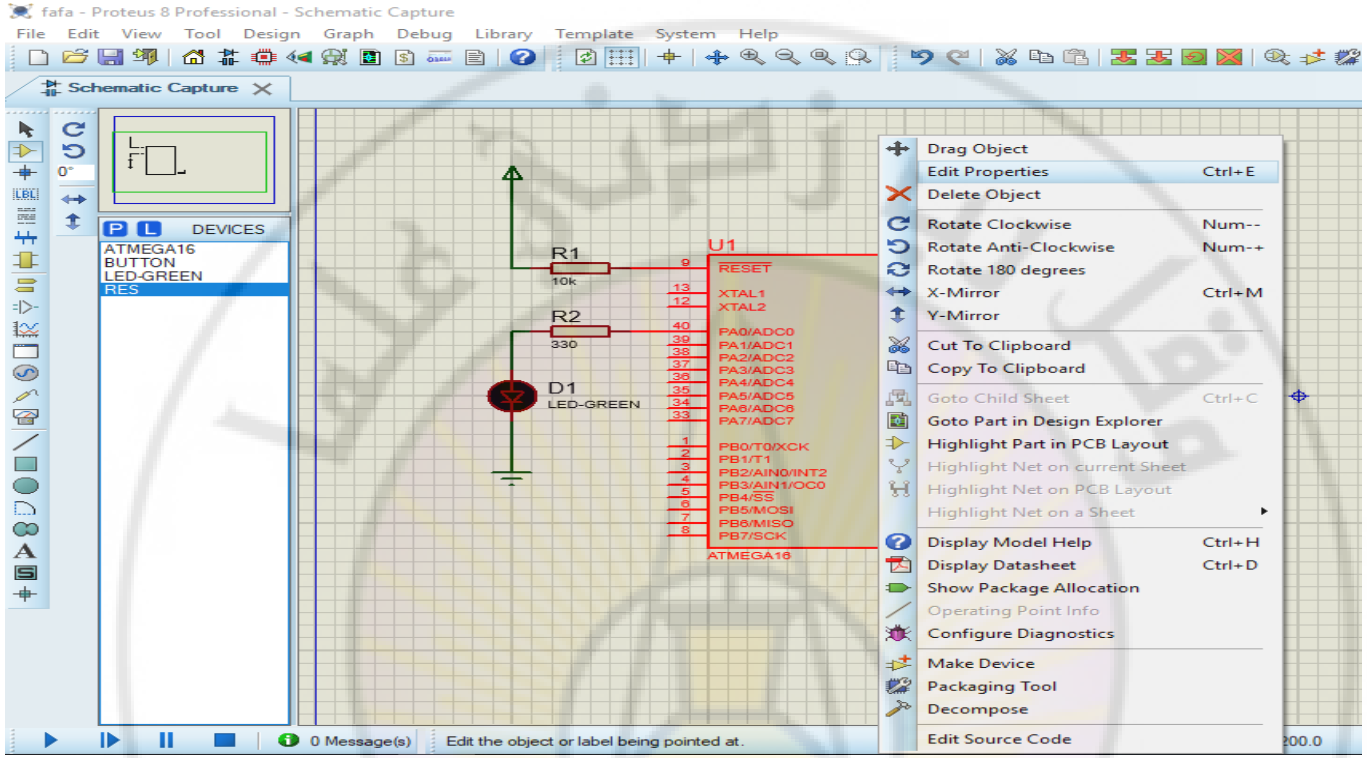



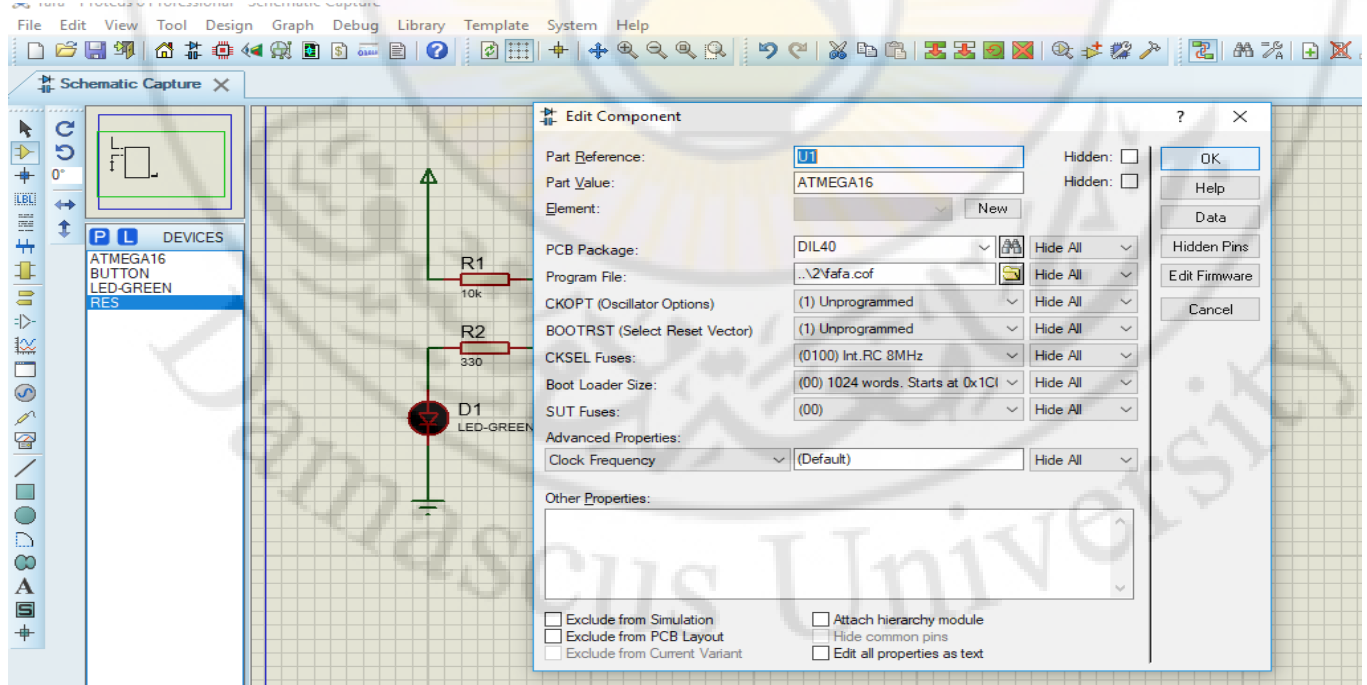



نضيف الملف المطلوب (ممكن ان يطلب استبداله بملف ال project الأول) .. وذلك ليتم ربط الملف مع دارة البروتيوس




ثم من نفس القائمة project نضغط على Build all حتى يتم توليد ملف الهيكسا



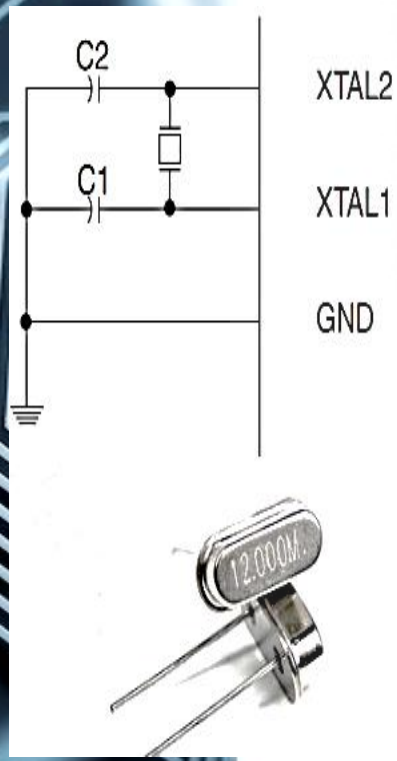

ننتقل بعدها إلى البروتيوس
ونضغط بالزر اليميني على
المتحكم ونضغط على Edit
project لتظهر قائمة تحميل ملف
الهيكسا

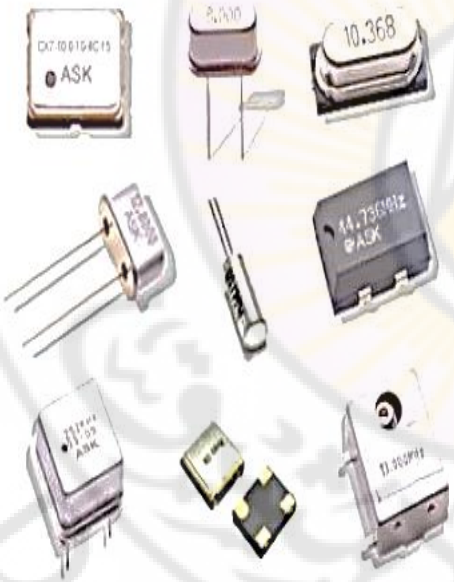
# ماهية المتحكمات الصغرية

# مصادر التغذية أو توزيع مصادر التوقيت للعائلة AVR

لتنظيم الوقت في برامج المتحكمات نحتاج إلى عدة مؤقتات (ساعات) ( مؤقت للمعالج، لل RAM، للFlash، للADC ، وللمداخل والمخارج .....إلخ) وغيرها من العناصر الأخرى التي نستخدمها في الدارة.

تسمى اوسوليتر الذي يقوم بعمل النبضات والتي تسبه نبضات القلب لتنظيم وتنفيذ عمل المتحكم وإعطاء قيم دقيقة ، وقد لايستخدم في كل الدوائر ، ومنها مايكون داخلي ومنها خارجي وهو الأفضل ، وسندرس مصادر التوقيت في المتحكم AVR:



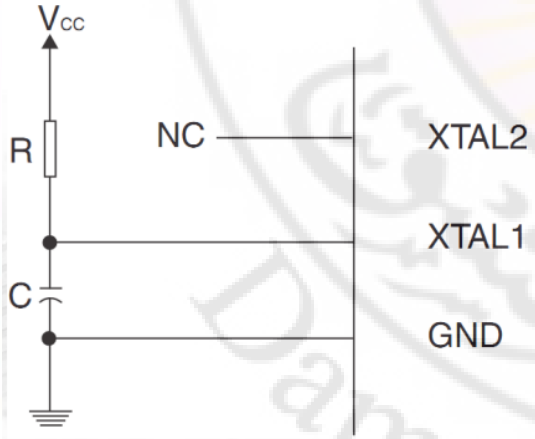

1. المهتز الكريستالي Crystal Oscillator :له أشكال عديدة نلاحظ وجود قيمة المهتز عليه ويتم وصله حسب الدارة في القطبين XTAL1 (input) و XTAL2 (output) ، هذا المهتز عبارة عن مادة كريستالية طبيعية لديها القدرة على أن تعطي نبضات (اهتزازات) عندما نعطيها جهد فولت (توتر)، وأيضا العكس تعطي توتر(فولت) عندما نقوم بالضغط عليها ، وهي دائرة خارجية ذات دقة أعلى وافضل من غيرها بكثير، لأن المادرة المكونة منها دقيقة جدا ، ولها سرعات مختلفة (8-16-.....)وتحديد السرع من خلال حجم المهتز الكريستالي(مكعب بحجم معين) ومن خلال data sheet .

2. المهتز الداخلي RC : وهي دائرة داخل المتحكم تقوم بإعطاء نبضات ولكن بدقة اقل من الكريستالي الخارجي ، فمثلا عند درجة حرارة 25 وجهد v5 يعطي المهتز الداخلي قيم.

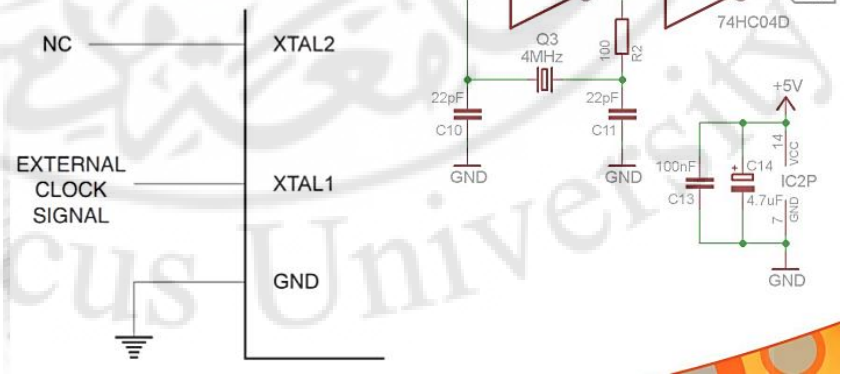
**1.0MHz, 2.0MHz,**
**4.0MHz, 8.0MHz**

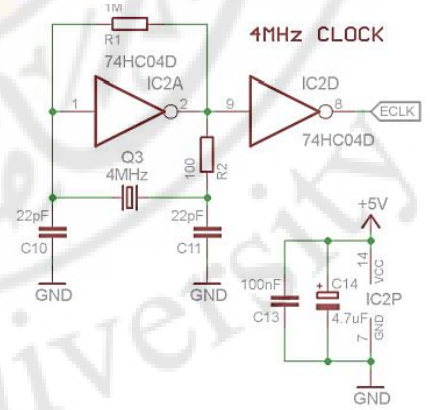
**at 5V and 25°C > ±3%**

3. المهتز RTC (External RC): وهو النوع الأكثر دقة Real Time Clock ويوصل على القطبين TOSC1 و TOSC2




4. External Clock

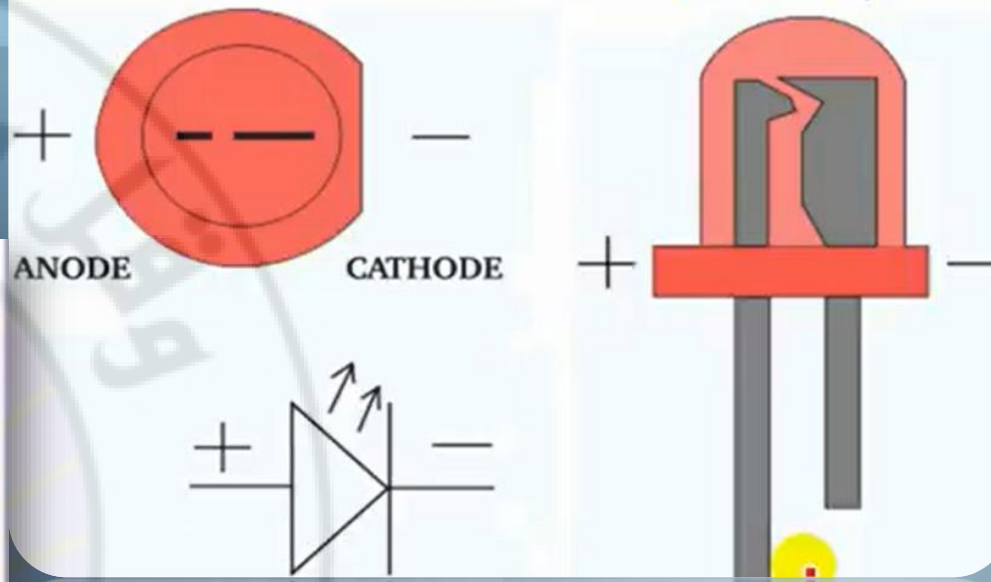



4. External Clock

# التجربة الثانية
# Flasher لأكثر من ليد ضوئي

## اهم الخصائص و المميزات :




## خصائص الليد :

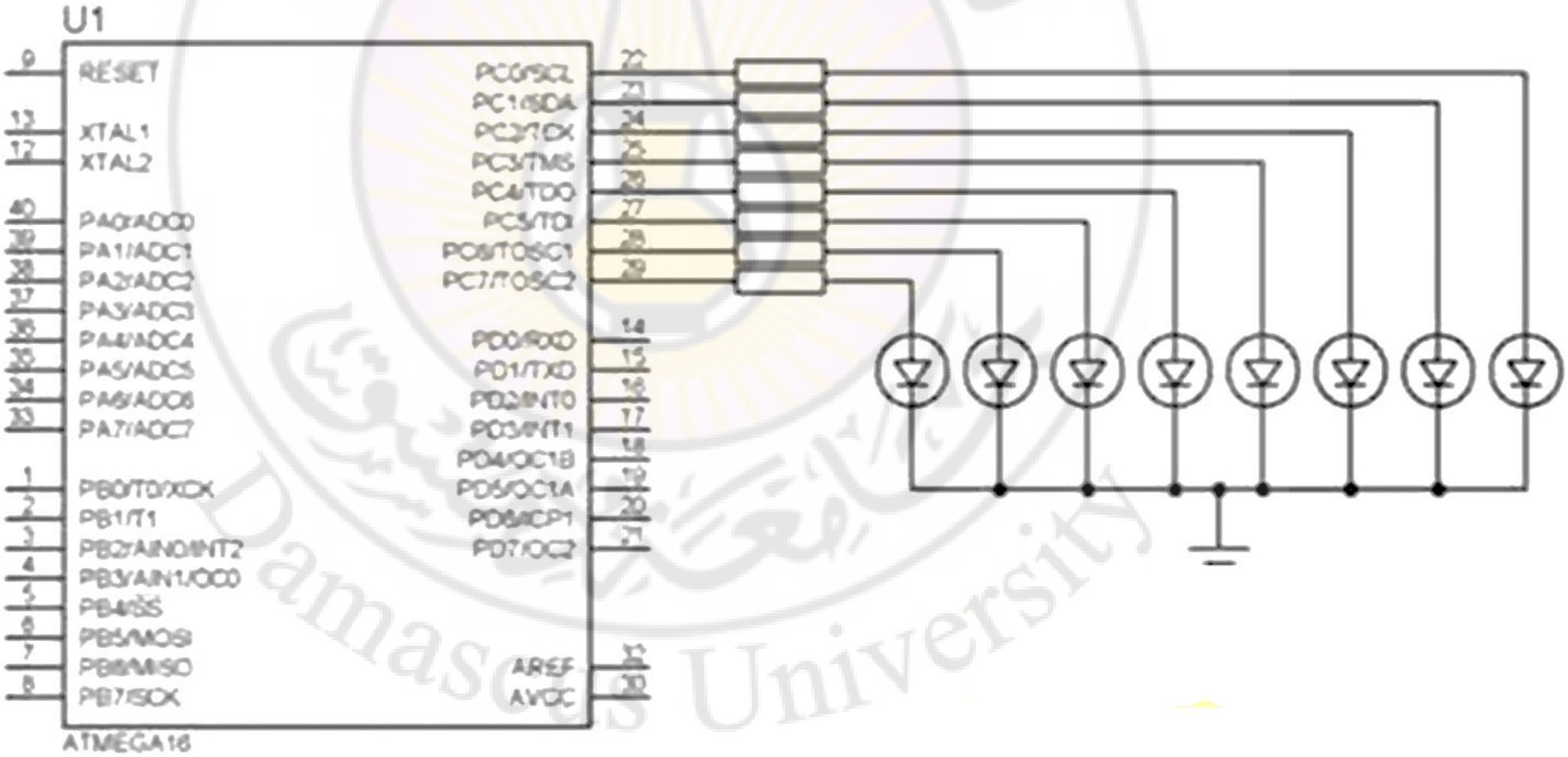
| Type | Colour | $I_F$ max. | $V_F$ typ. | $V_F$ max. | $V_R$ max. | Luminous intensity | Viewing angle | Wavelength |
|---|---|---|---|---|---|---|---|---|
| Standard | Red | 30mA | 1.7V | 2.1V | 5V | 5mcd @ 10mA | 60° | 660nm |
| Standard | Bright red | 30mA | 2.0V | 2.5V | 5V | 80mcd @ 10mA | 60° | 625nm |
| Standard | Yellow | 30mA | 2.1V | 2.5V | 5V | 32mcd @ 10mA | 60° | 590nm |
| Standard | Green | 25mA | 2.2V | 2.5V | 5V | 32mcd @ 10mA | 60° | 565nm |
| High intensity | Blue | 30mA | 4.5V | 5.5V | 5V | 60mcd @ 20mA | 50° | 430nm |
| Super bright | Red | 30mA | 1.85V | 2.5V | 5V | 500mcd @ 20mA | 60° | 660nm |
| Low current | Red | 30mA | 1.7V | 2.0V | 5V | 5mcd @ 2mA | 60° | 625nm |

**$I_F$ max** : التيار الأعظمي الأمامي المار في الثنائي ..
**$V_F$ typ** : الجهد الأمامي النموذجي من اجل تشغيل الثنائي ..
**$V_F$ max** : الجهد الأمامي الأعظمي الذي يمكن للثنائي أن يتحمله ..
**$V_F$ max** : الجهد العكسي الأعظمي الذي يمكن للثنائي أن يتحمله ..
**Luminous intensity** : شدة السطوع للثنائي .. mcd = millicandela
**Viewing angle** : زاوية انعكاس الرؤية للإضاءة ..
**Wavelength** : طول موجة الضوء الصادر .. nm = nanometer

# تطبيق (الأضواء المتحركة)

برمج المتحكم ليتم أضاءة LEDs بخمسة طرق مختلفة وفق اللدارة التالي:

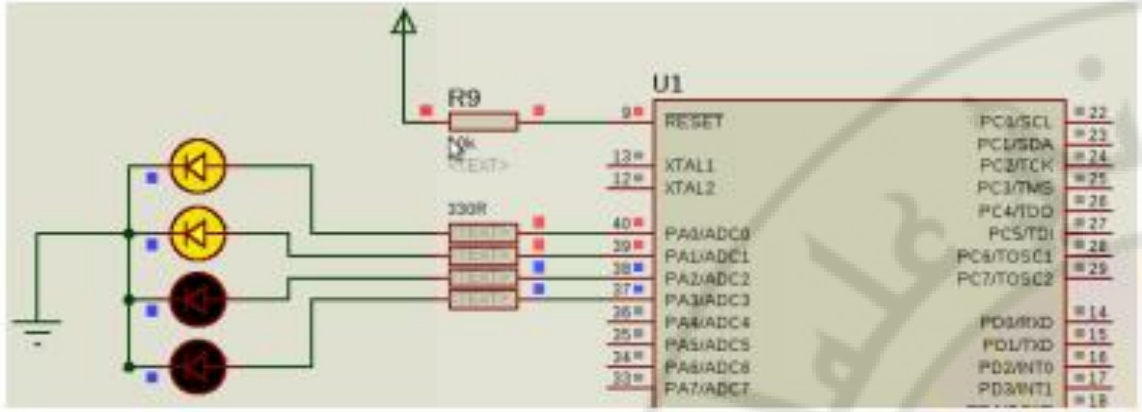
حسب الشكل يتم تعريف بنات ال PORTC جميعها على أنها خرج .. أي PORTC خرج

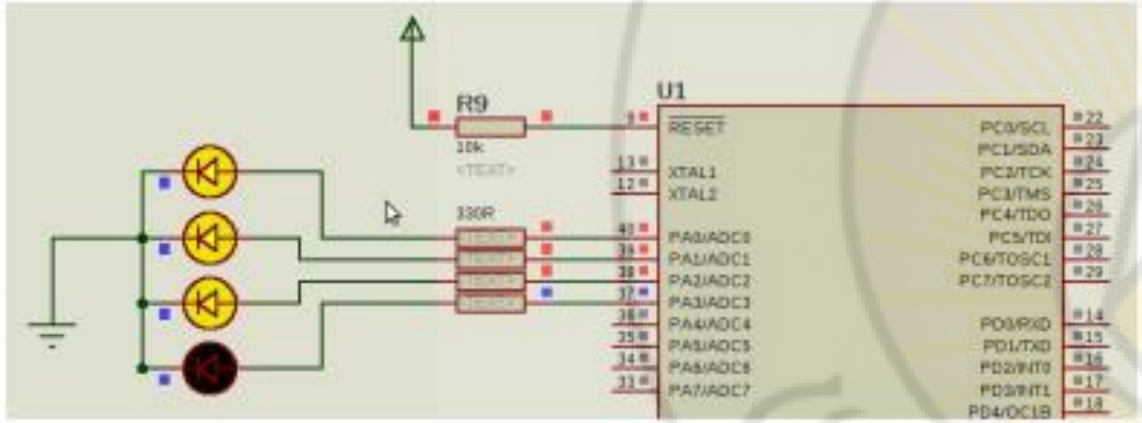


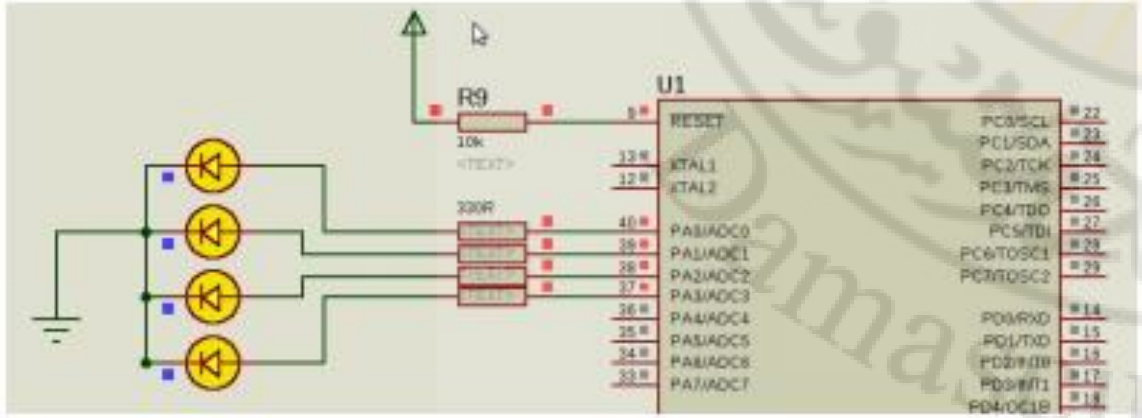
## الكود البرمجي :

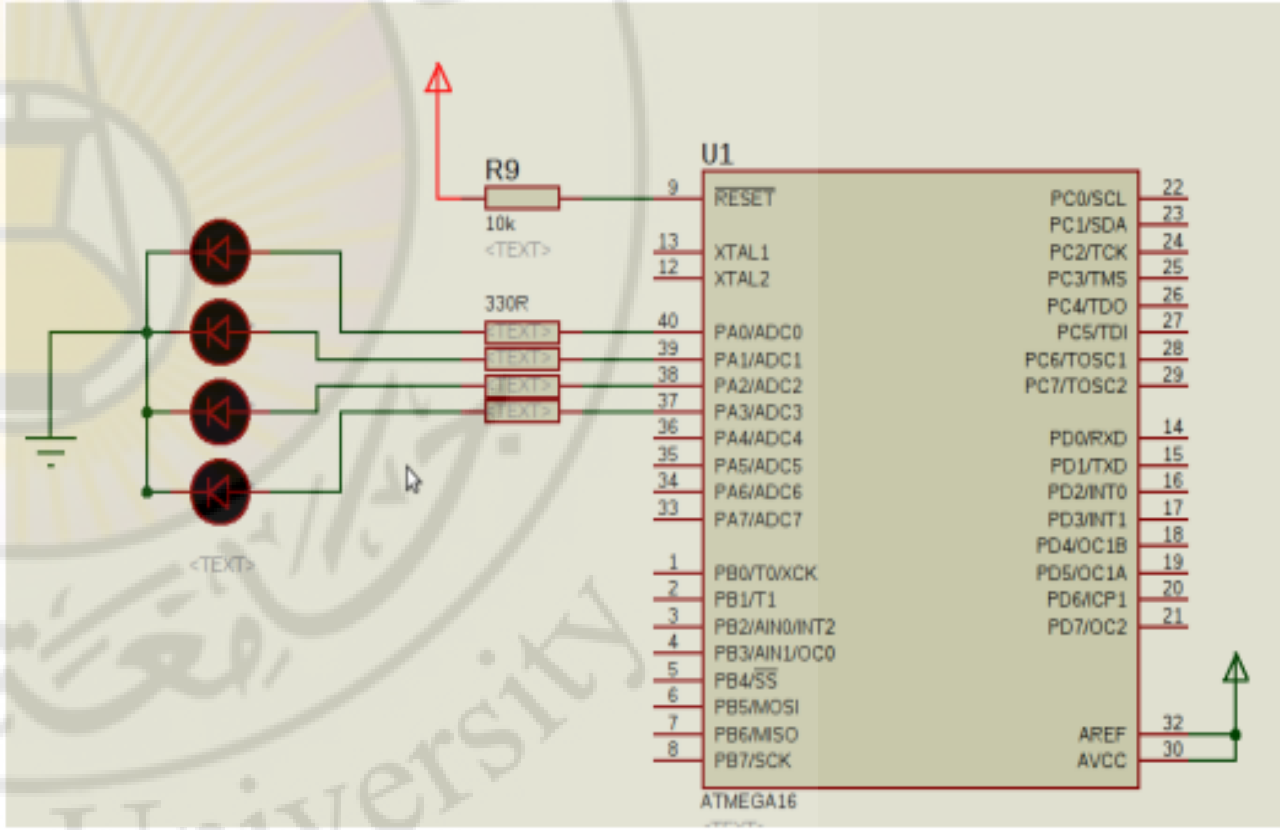
```
#include <mega16.h>
#include <delay.h>
void main(){
DDRC=0B11111111;
while (1)
    {
   PORTC=0B00011000;
   delay_ms(200);
   PORTC=0B00100100;
   delay_ms(200);
   PORTC=0B01000010;
   delay_ms(200);
  PORTC=0B10000001;
   delay_ms(200);
  PORTC=0B01000010;
   delay_ms(200);
  PORTC=0B00100100;
   delay_ms(200);
    }
}
```

في هذا المثال سنستخدم 4 دايودات ضوئية حيث سنقوم بتطوير الكود المستخدم في المثال الأول ليعمل بعدد **4 من الدايودات ضوئية**. وكما هو موضح في الصورة التالية نجد الدايودات متصلة على الأطراف من PA0 إلى PA3 سواء كنت تستخدم ATmega16 أو ATtiny84.

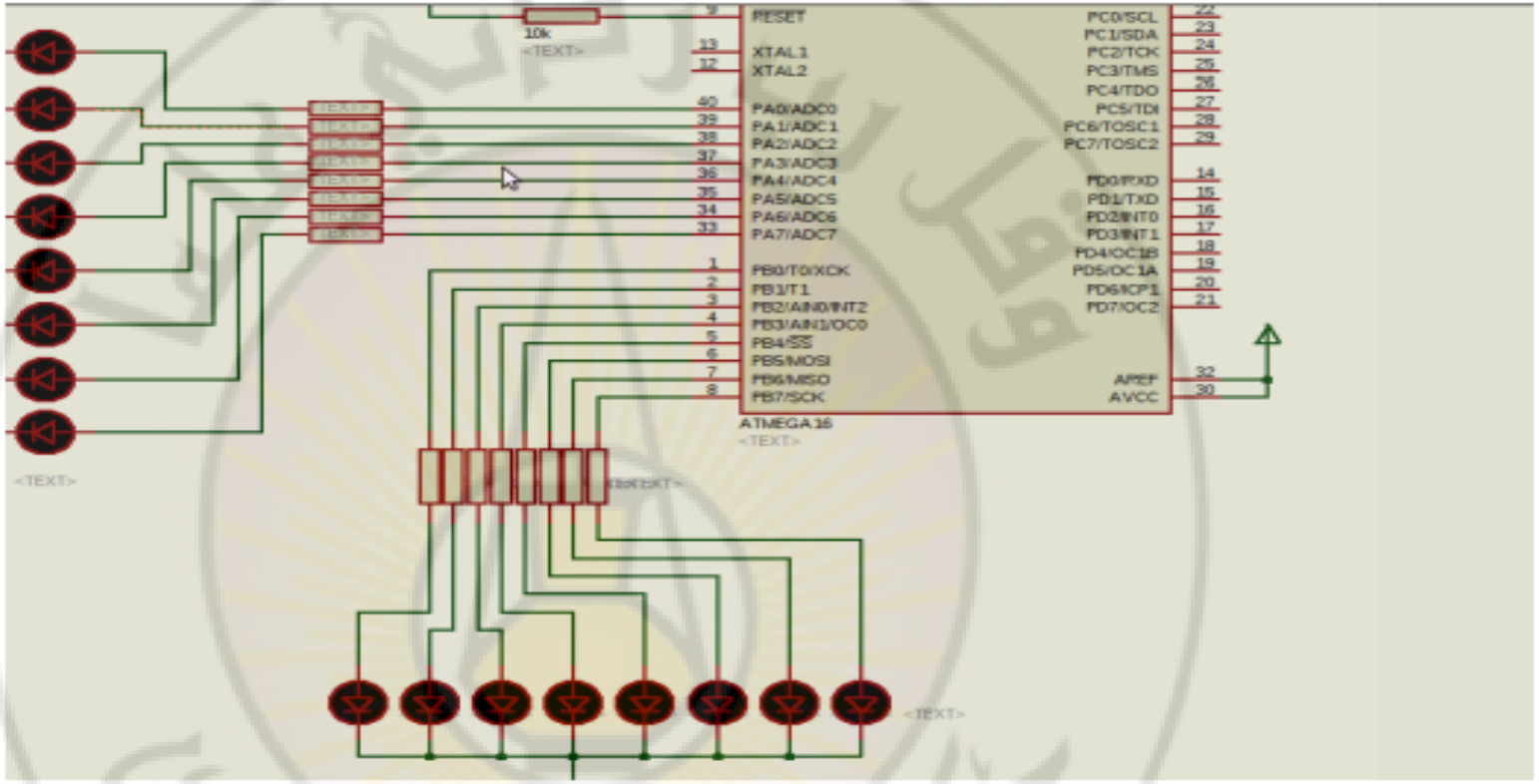


في هذا المثال سنقوم بتوصيل 16 دايود على جميع أطراف البورت A و B بحيث يتصل 8 دايودات ضوئية لكل بورت كما هو موضح بالصورة التالية



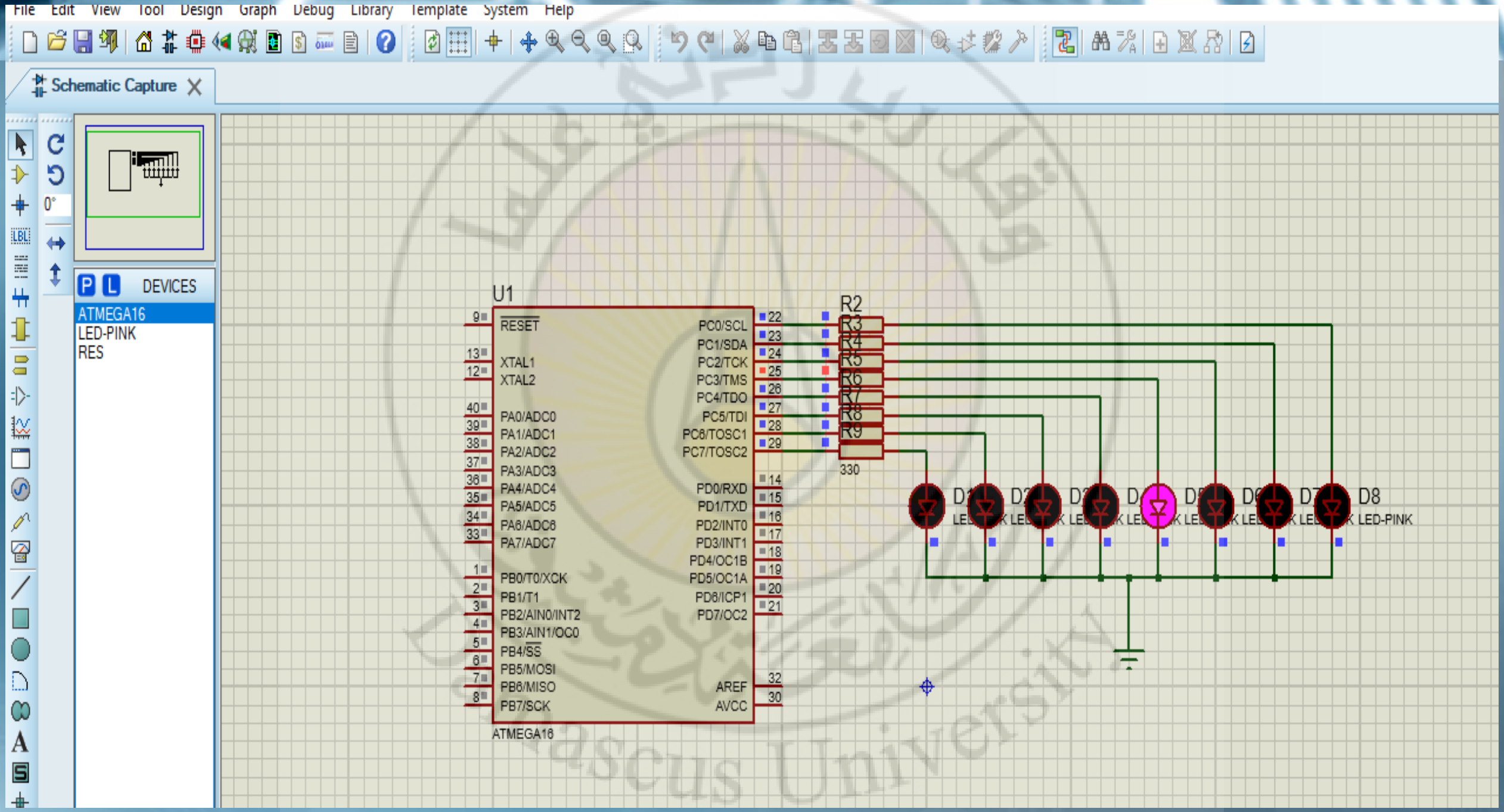

قم بتشغل 8 دايودات ضوئية على جميع أطراف البورت A واجعلها تضيء بالترتيب التالي مع تأخير ربع ثانية فقط بين كل أمر. (لا تنسى أن جميع أطراف البورت ستعمل كخرج)

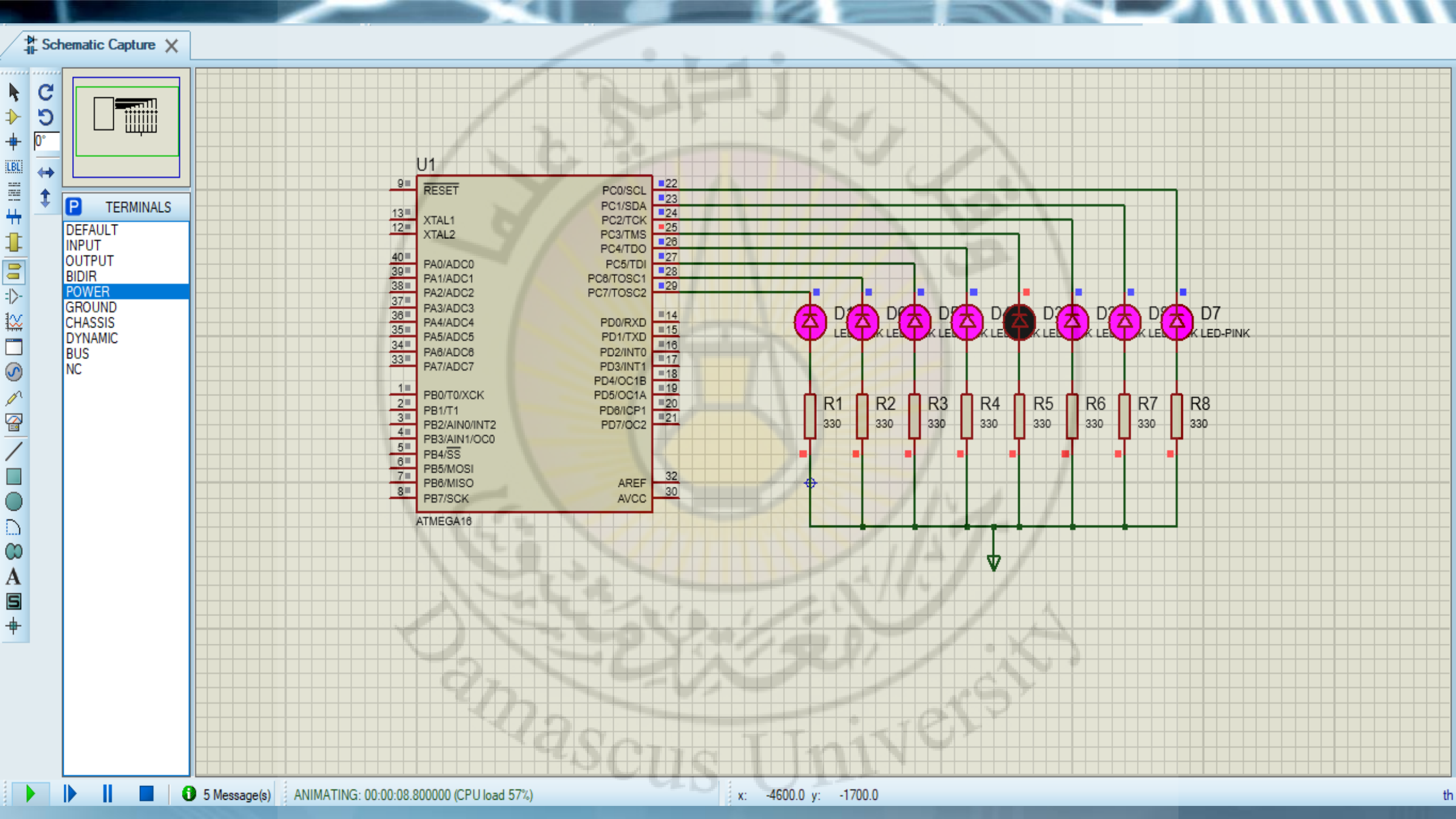
00000001
00000011
00000111
00001111
10000000
11000000
11100000
11110000

وصل ثمانية ثنائيات ضوئية (LEDs) إلى متحكم AVR على البوابة PORTC، وكتابة برنامج بحيث تومض الثنائيات بالتتابع بالاتجاهين..

1- التنفيذ في حالة المتحكم هو منبـــع للتيار

2- التنفيذ في حالة المتحكم هو مصرف للتيار

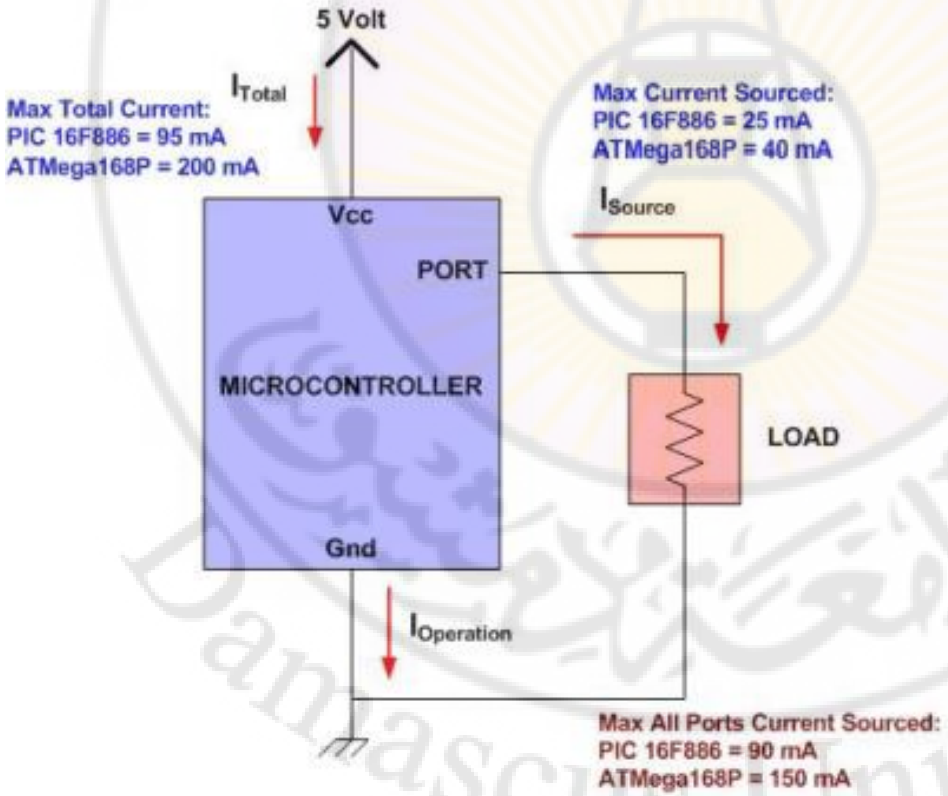

File Edit View Tool Design Graph Debug Library Template System Help
Schematic Capture
0°
DEVICES
ATMEGA16
LED-PINK
RES
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC6/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
ATMEGA16
R2
R3
R4
R5
R6
R7
R8
R9
330
D8
LED-PINK


Schematic Capture
TERMINALS
DEFAULT
INPUT
OUTPUT
BIDIR
POWER
GROUND
CHASSIS
DYNAMIC
BUS
NC
U1
RESET
XTAL1
XTAL2
PA0/ADC0
PA1/ADC1
PA2/ADC2
PA3/ADC3
PA4/ADC4
PA5/ADC5
PA6/ADC6
PA7/ADC7
PB0/T0/XCK
PB1/T1
PB2/AIN0/INT2
PB3/AIN1/OC0
PB4/SS
PB5/MOSI
PB6/MISO
PB7/SCK
ATMEGA16
PC0/SCL
PC1/SDA
PC2/TCK
PC3/TMS
PC4/TDO
PC5/TDI
PC8/TOSC1
PC7/TOSC2
PD0/RXD
PD1/TXD
PD2/INT0
PD3/INT1
PD4/OC1B
PD5/OC1A
PD6/ICP1
PD7/OC2
AREF
AVCC
D7
LED-PINK
R1
R2
R3
R4
R5
R6
R7
R8
330
5 Message(s)
ANIMATING: 00:00:08.800000 (CPU load 57%)
x: -4600.0 y: -1700.0

```
fx main

while (1)
      {
      // Place your code here
         PORTC = 0B00000001;
   delay_ms(50);
   PORTC = 0B00000010;
   delay_ms(50);
   PORTC=0B00000100;
   delay_ms(50);
   PORTC=0B00001000;
   delay_ms(50);
   PORTC=0B00010000;
   delay_ms(100);
   PORTC=0B00100000;
   delay_ms(50);
   PORTC=0B01000000;
   delay_ms(50);
   PORTC=0B10000000;
   delay_ms(50);
   PORTC=0B01000000;
   delay_ms(50);
   PORTC=0B00100000;
   delay_ms(50);
   PORTC=0B00010000;
   PORTC=0B00010000;
   delay_ms(100);
   PORTC=0B00100000;
   delay_ms(50);
   PORTC=0B01000000;
   delay_ms(50);
   PORTC=0B10000000;
   delay_ms(50);
   PORTC=0B01000000;
   delay_ms(50);
   PORTC=0B00100000;
   delay_ms(50);
   PORTC=0B00010000;
   delay_ms(50);
   PORTC=0B00001000;
   delay_ms(50);
   PORTC=0B00000100;
   delay_ms(50);
   PORTC=0B00000010;
   delay_ms(50);
   PORTC=0B00000001;
   delay_ms(50);

      }
```

1. هناك طريقتان لوصل الأحمال مع مخارج المتحكم :

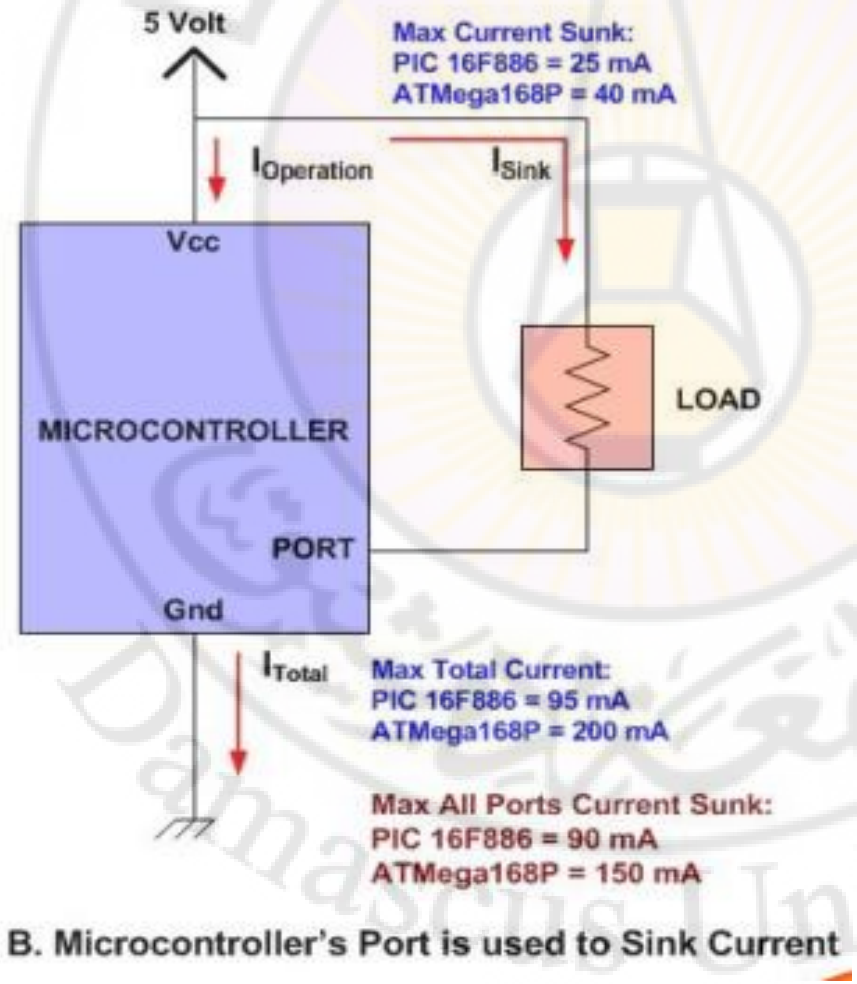
الـPORT أو PIN أي القطب يعمل كمنبع للتيار أي أحمال القطب (ليد مثلا) تكون موصولة من طرف على القطب (PORT) والطرف الآخر على الأرضي ، نلاحظ أن القطب سيصبح منبع للتيار




A. Microcontroller's Port is used as a Current Source

2. الـPORT أو PIN أي القطب يعمل كمصرف للتيار أي أحمال القطب موصولة من طرف على القطب (PORT) والطرف الآخر على التيار (الموجب) ، أي أن التيار يتحرك من الطرف الموجب ليدخل إلى الحمل .

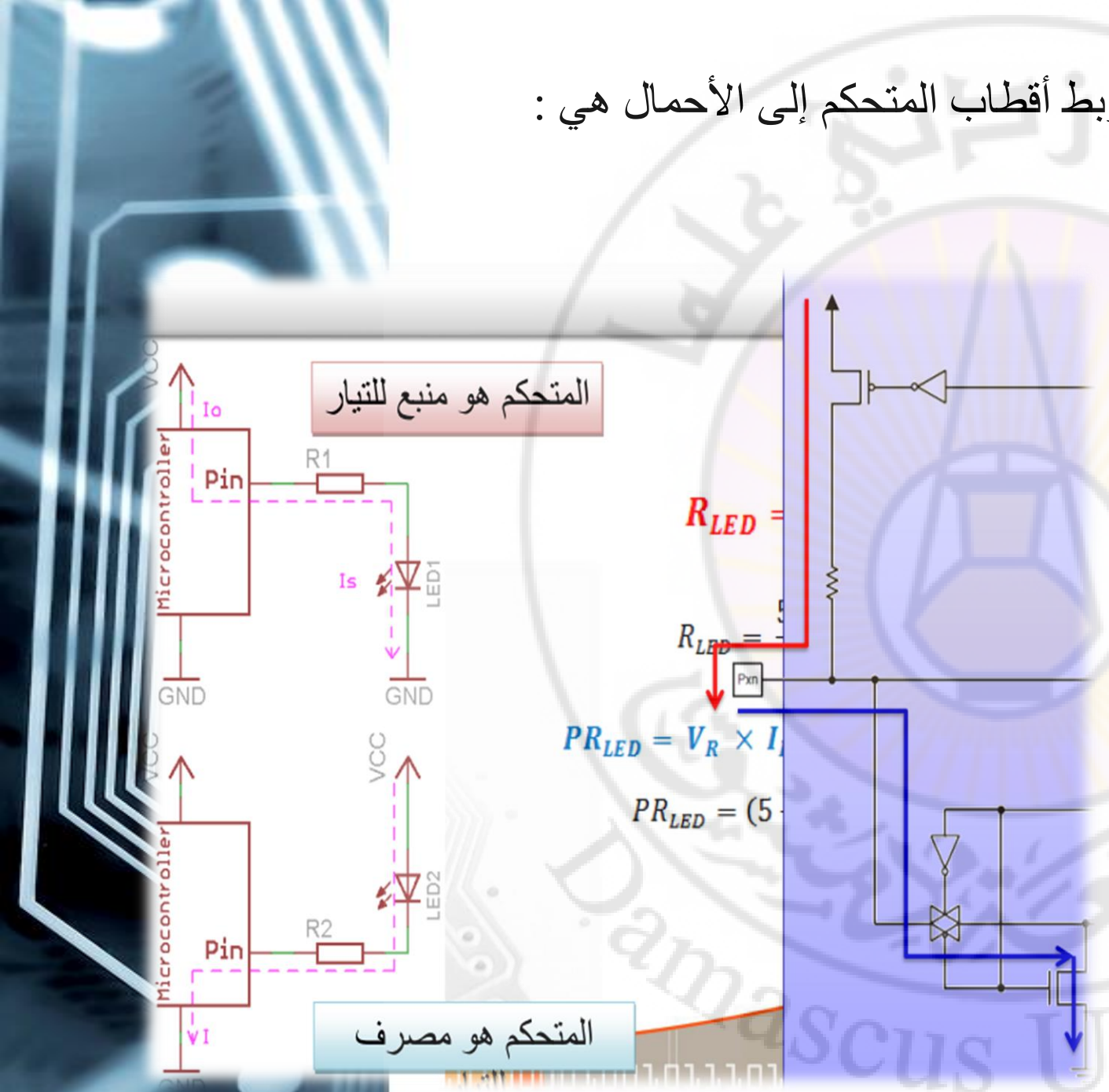
ومتحكمات الAVR تدعم طريقتي التوصيل




B. Microcontroller's Port is used to Sink Current

أهم الاعتبارات التي يجب أن تؤخذ بعين الاعتبار عند ربط أقطاب المتحكم إلى الأحمال هي :

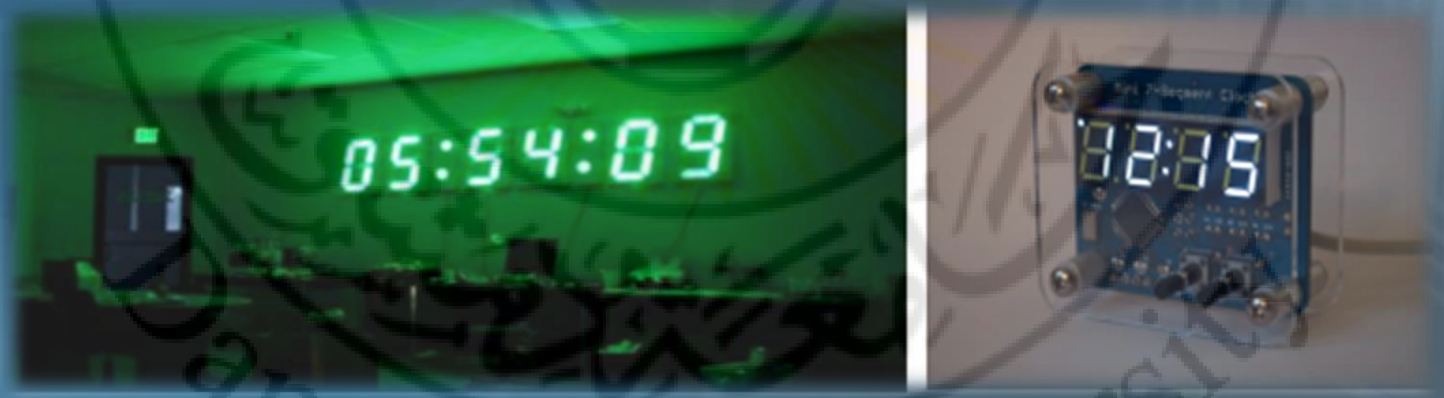
1. التيار الأعظمي المستهلك من قطب المتحكم (Vcc to Gnd) الذي يمكن سحبه أو تصريفه عن طريق المتحكم بشكل كلي هو 200 ميلي امبير وفق مواصفات عائلة المتحكمات AVR.
2. قيمة التيار الممكن سحبه أو تصريفه لقطب خرج من أقطاب المتحكم تتراوح عادة من 20 ~ 40 mA حسب المواصفات الكهربائية للمتحكم المصغر AVR

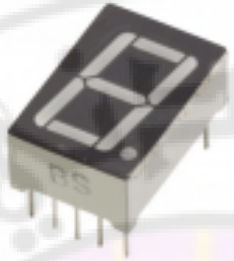



والأحمال ذات الاستطاعات العالية تحتاج وسائل إضافية سنتعرف عليها لاحقا

# المقاطعة السباعية

# Seven Segment

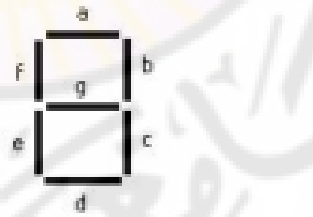




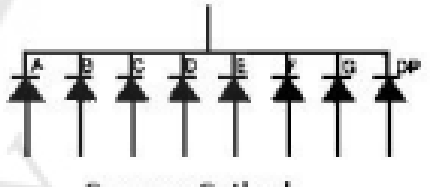
"المقاطعة السباعية - تنطق سيفين سيجمنت" **7segment** وهي عبارة عن مستطيل صغير يحتوي على 7 مقاطع مضيئة باستخدام دايودات ضوئية (متوفرة باللون الأحمر والأخضر والأزرق).وتستخدم في عرض الأرقام وبعض حروف اللغة الإنجليزية.

تتوفر هذه القطعة الإلكترونية في الأسواق بمختلف الأحجام فمنها ماهو صغير جداً مثل المستخدمة في الساعات الرقمية الرخيصة ومنها ماهو كبير الحجم مثل المستخدمة في إشارات المرور (اللوحة المضيئة التي تعرض الوقت المتبقي لتفتح إشارة المرور).

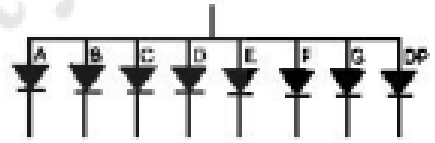
تتكون السيفين سيجمنت من 7 دايودات ضوئية متصلة ببعضها البعض إما عن طريق توصيل الطرف الموجب وتسمى common anode أو عن طريق توصيل الطرف السالب وتسمى common cathode **(سنستخدم في التجارب التالية النوع common cathode)**.ويسمى كل دايود ضوئي بأحد حروف الأبجدية الإنجليزية A,B,C,D,E,F,G كما هو موضح في الصورة التالية:




7-segment display with segment identification.




Common Cathode




Common Anode

Equivalent circuit, with decimal point.

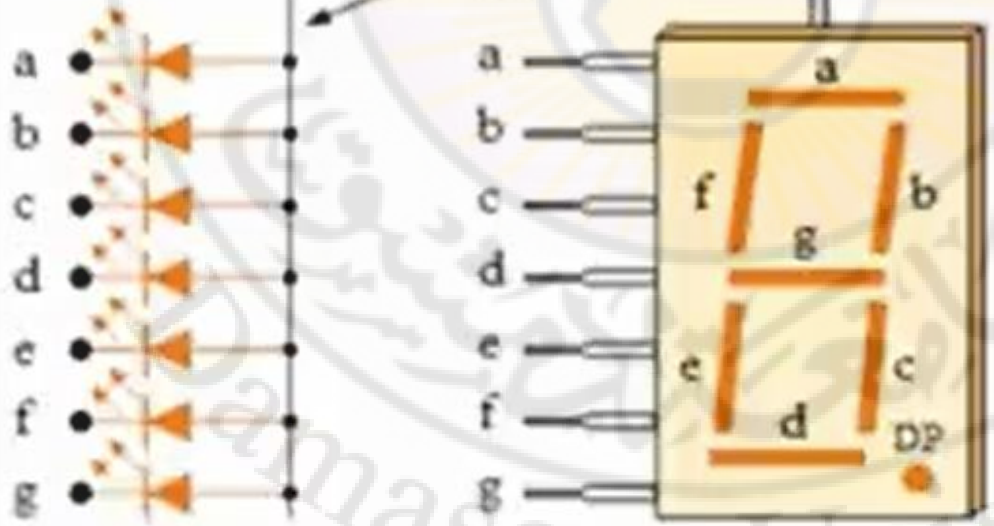
لعرض أي رقم من الأرقام العشرية سنستخدم الجدول التالي والذي يوضح الحالة التي يجب أن يكون عليها كل دايود ضوئي حتى يتم عرض رقم معين.

| DIGIT | LEDs TO GLOW | | | | | | |
|---|---|---|---|---|---|---|---|
| | a | b | c | d | e | f | g |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 2 | 1 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| 5 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 6 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 7 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 8 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 9 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |

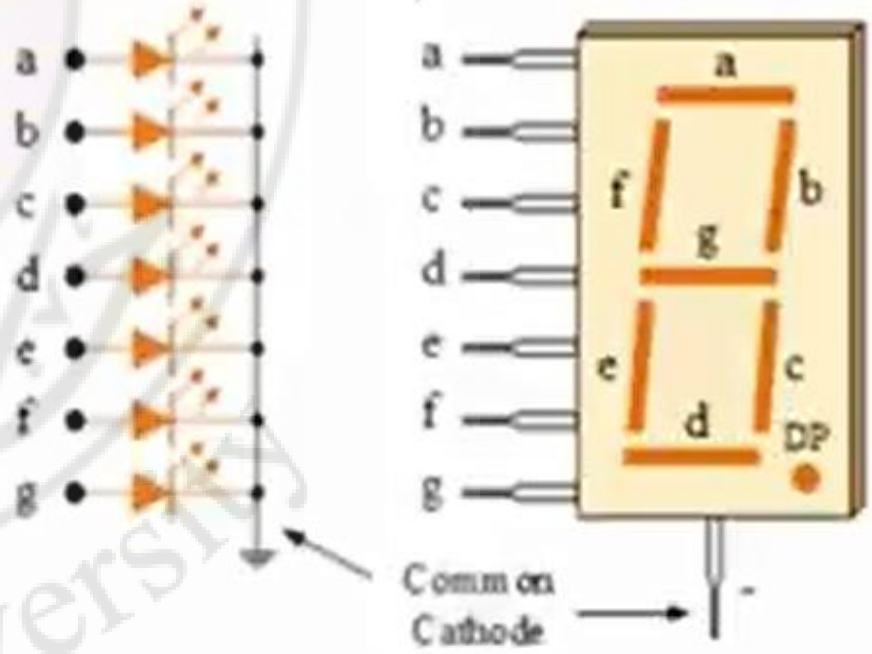
ملاحظة: بعض السيفن-سيجمنت المتوفرة في الأسواق **(خاصة صغيرة الحجم)** تحتوي على طرف إضافي وهو دايود ضوئي صغير موجود على الجانب الأيمن السفلي ويستخدم في بعرض الفاصلة العشرية (.) لكن برنامج بروتس لا يحتوي على هذا الدايود الضوئي لذا لن نستخدمه في المحاكاة).

# ربط 7-segment مع المتحكم المصغر

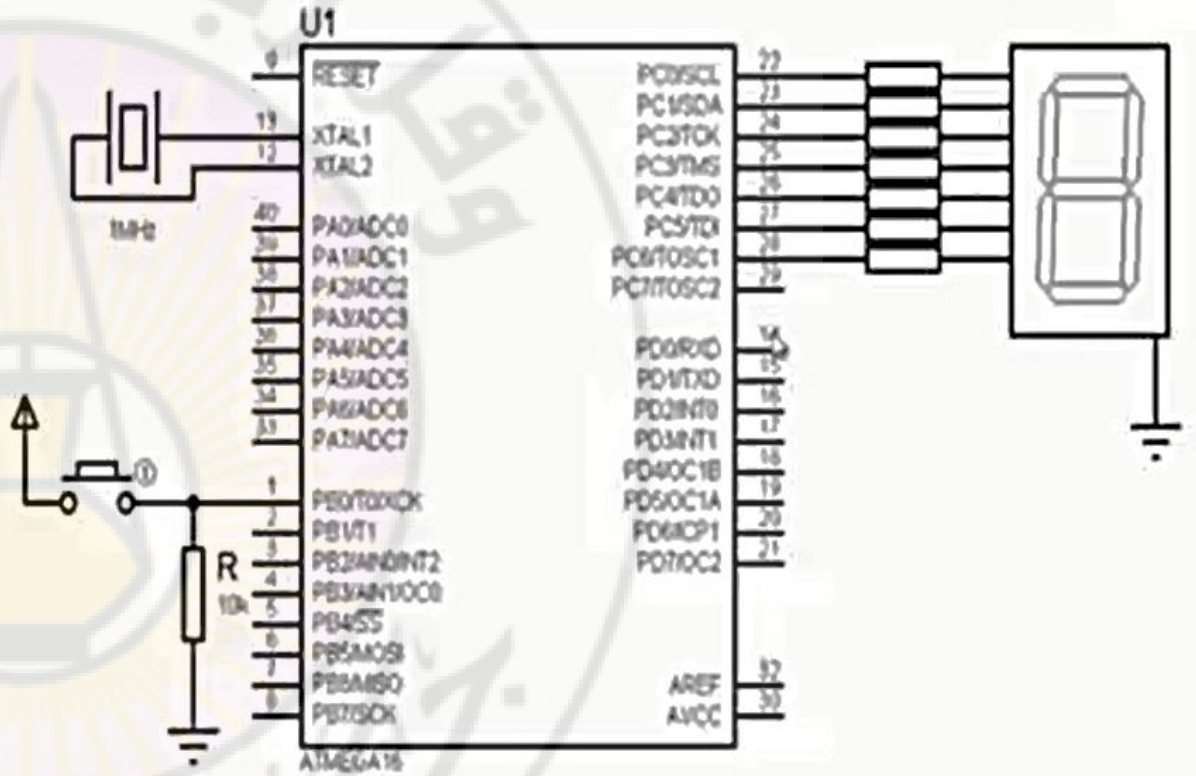
Common Anode 7-segment Display




Common Cathode 7-segment Displa

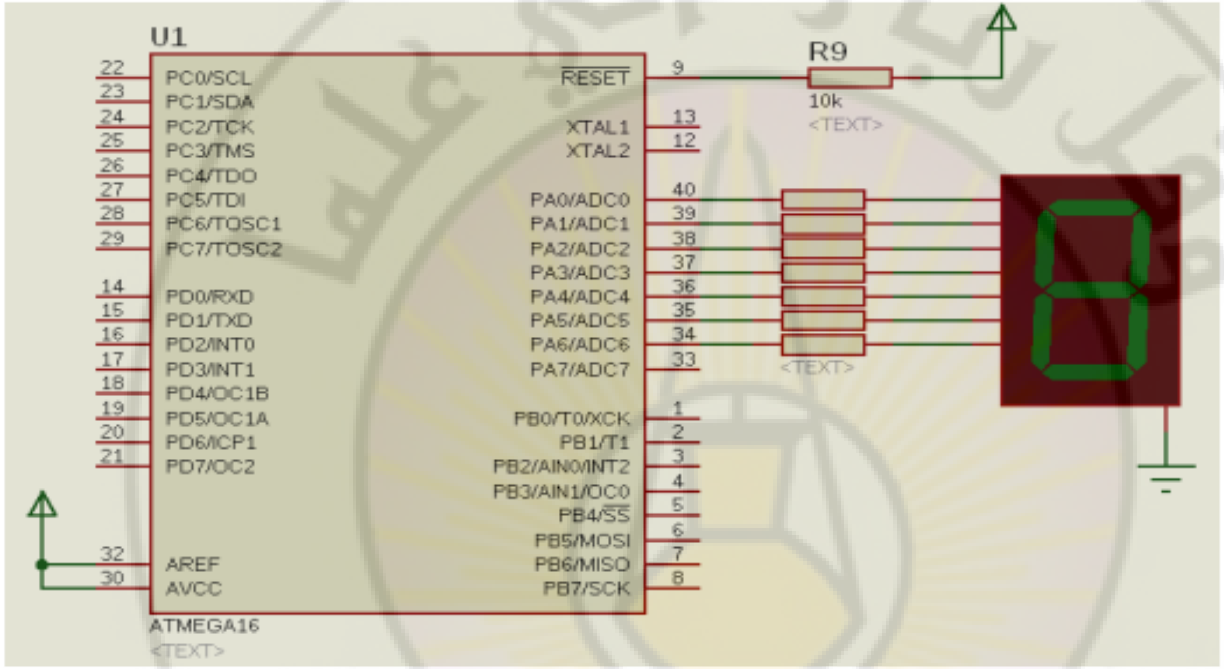


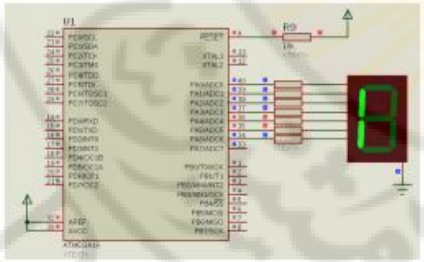
برمج الدارة بحيث كل كبسة على Button يظهر رقم 1 أو 2 بطريقة عشوائية .

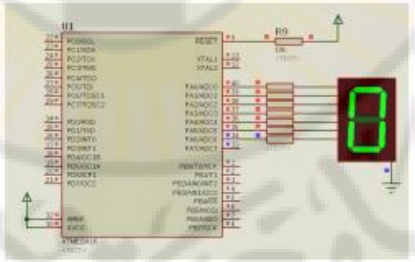


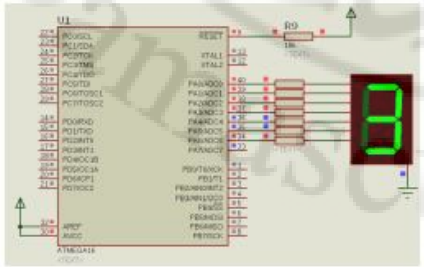

```
#include <mega16.h>
#include <delay.h>
void main(void)
{
DDRC=0B01111111;
while (1)
    {
  if (PINB.0==1)
    {
   PORTC=0B00000110;
     delay_ms(25);
    }
  if (PINB.0==1)
    {
   PORTC=0B01011011;
     delay_ms(25);
    }
   }
}
```

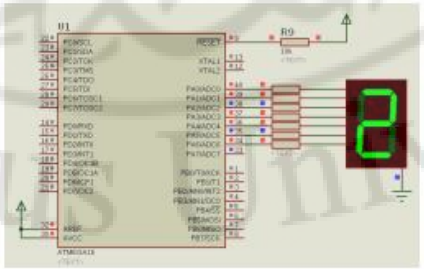
قم بتوصيل الدارة التالية وكتابة كود يقوم بعرض الأرقام من 1 – 9 بتأخير زمني 1 ثانية كما في صورة المحاكاة المرفقة



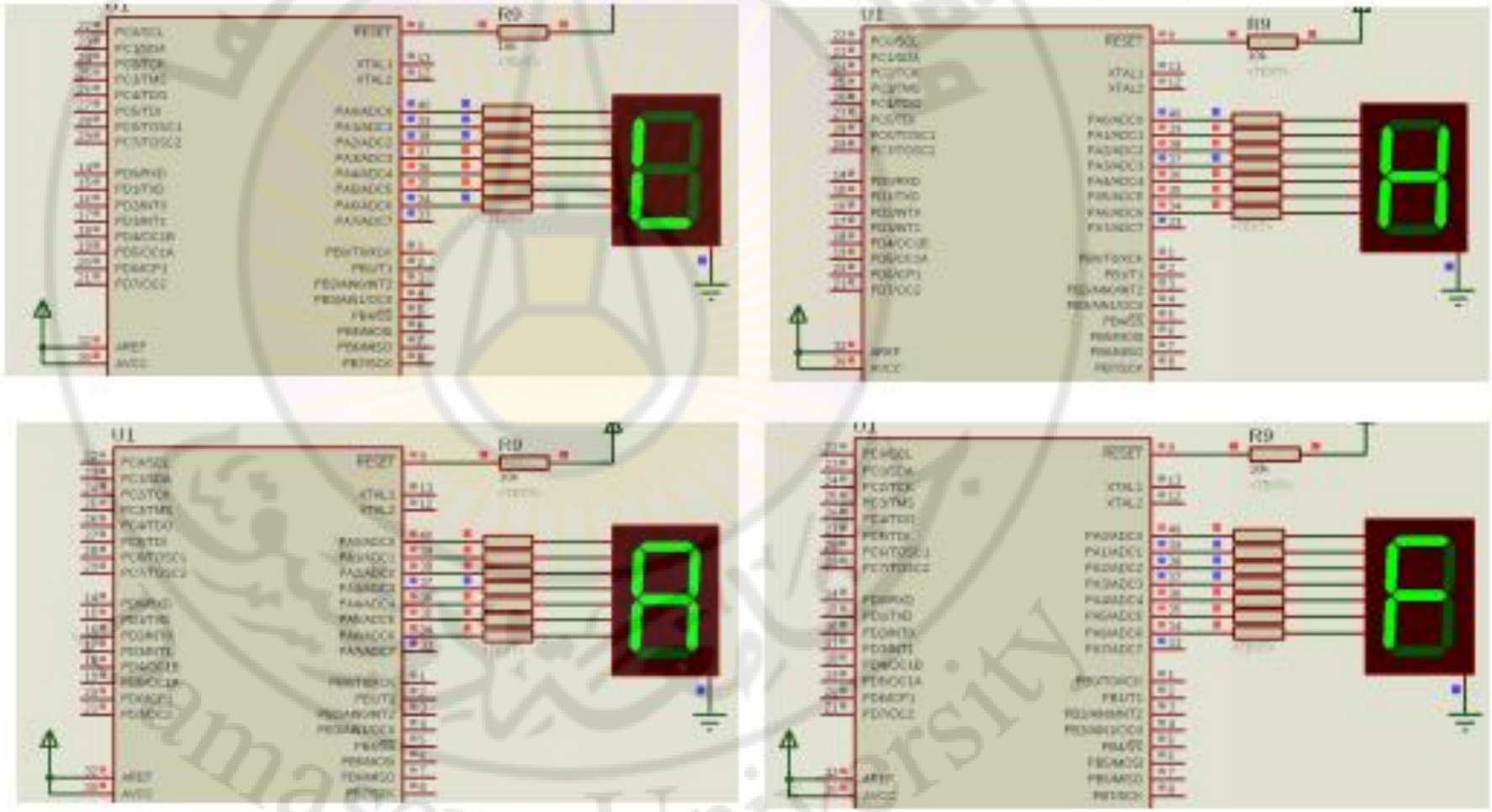









اكتب برنامج يقوم بعرض الأحرف الأبجدية الإنكليزية على شاشة 7-Segment كما في المحاكاة التالية :



# تداول البيانات داخل متحكمات AVR

# أنواع البيانات في الأنظمة المدمجة:

البيانات الرقمية : تعتبر اكثر الأنواع استخداما في عالم الأنظمة المدمجة فيه تعبر عن قيم المتغيرات مثل قراءة الحساسات او قيم المخارج مثل سرع المحركات

| نوع البيانات | استهلاك الذاكرة (عدد Bytes) | أقصى قيمة يمكن وضعها داخل المتغير |
|---|---|---|
| int | 2-bytes (16 bit) | -32,768 → +32,767 |
| unsigned int | 2-bytes (16 bit) | 65,535 |
| float | 4-bytes (32 bit) | 1.2E-38 to 3.4E+38 |
| double | 8-bytes (32 bit) | 2.3E-308 to 1.7E+308 |
| long | 4-bytes (32 bit) | -2,147,483,648 → +2,147,483,647 |
| unsigned long | 4-bytes (32 bit) | 4,294,967,295 |
| long long | 8-bytes (64 bit) | $-9.223372037\times10^{18}$ $+9.223372037\times10^{18}$ |
| unsigned long long | 8-bytes (64 bit) | $1.844674407\times10^{19}$ |

ملاحظة: نوعي البيانات float و double دائماً يكونا من نوع Signed ولا يمكن استخدام العلامة unsigned معهما

جدول البيانات التقليدية

البيانات المعيارية ANSI C - C99 وهي صورة محسنة لأنواع البيانات وتمكننا من اختيار قيمة المتغيرات بالدقة والطول المطلوب دون أي اهدار للذاكرة.

| Signed | أقصى قيمة | Unsigned | أقصى قيمة |
|---|---|---|---|
| int8_t | -128 → +127 | uint8_t | 0 → 255 |
| int16_t | -32,768 → +32,767 | uint16_t | 0 → 65,535 |
| int32_t | -2,147,483,648 → +2,147,483,647 | uint32_t | 0 → 4,294,967,295 |
| int64_t | $-9.223372037\times10^{18}$ $+9.223372037\times10^{18}$ | uint64_t | $1.844674407\times10^{19}$ |

جدول البيانات المعيارية

## أمثلة

**ما هو نوع المتغير المطلوب للقراءة أو الكتابة في أي بورت (مثل port A, B,C)؟**

- بما أن جميع البورتات يتم قراءتها من خلال المُسجِل PINx والذي يكون 8 بت وكذالك يتم كتابة في أي بورت باستخدام المُسجِل PORTx والذي يعتبر 8 بت أيضاً إذا أفضل نوع بيانات هو **uint8_t** حيث يمكنه تخزين الأرقام من 0b0000000 إلى 0b11111111

**ما هو المتغير المناسب لتخزين عداد رقمي من صفر إلى 10,000؟**

- بما أن الرقم لا يحتوي على أرقام سالبة وأقصى رقم مطلوب هو 10000 وهذا الرقم أكبر من 255 لذا لا يمكن استخدام متغير بمساحة 8 بت ويجب أن نستخدم متغير بمساحة 16 بت والذي يستطيع التعامل مع ارقام تصل إلى 65,535. إذا أفضل نوع بيانات هو **uint16_t**

**ما هو المتغير المناسب لتخزين درجة حرارة تتراوح بين سالب 50 إلى موجب 100؟**

- هناك إجابتين لهذا السؤال، الأولى: في حالة أننا نريد تخزين أرقام صحيحة فقط مثل 20 أو 50 أو أي رقم بدون كسر عشري فسنجد أن افضل نوع بيانات هو **int8_t** حيث يستطيع هذا النوع من البيانات أن يخزن أي قيمة تتراوح بين -128 إلى + 127 ويستهلك 1 بايت فقط من مساحة الذاكرة.
- الثانية: في حالة وجود كسر عشري مثل أن 25.1 أو 30.5 يجب أن نستخدم نوع البيانات **float** مع العلم أنه في المقابل سيتم استهلاك مساحة ذاكرة = 4 بايت (32 بت).

## مجموعة التعليمات

يمتلك AVR مجموعة تعليمات غنية مؤلفة من 131 تعليمة. تقسم هذه التعليمات إلى أربع مجموعات : التعليمات الحسابية ، تعليمات البتات ، تعليمات التفرع وتعليمات نقل البيانات.

## تعليمات التحميل ونقل البيانات

تقوم هذه التعليمات بنقل البيانات بين المسجلات أو بين مسجل والذاكرة وبالعكس أو بين مسجل ومسجل خاص وبالعكس أو تحميل قيمة فورية في مسجل ما.

**ملاحظات :**

1. التعليمة LDI ( وجميع التعليمات المنتهية بالحرف I ) تحمل قيمة فورية في أحد المسجلات من R16 حتى R31 ولا يمكن استخدام المسجلات الأخرى لتحميل القيم الفورية.
2. تختلف تعليمات النقل من وإلى الذاكرة بنمط العنونة ، أي بطريقة حساب عنوان الحجرة المقصودة.
3. التعليمات OUT و IN تتعامل مع فضاء عنونة الدخل والخرج وتستخدم مع المسجلات الخاصة SFR.
4. التعليمات LPM و SPM تتعاملان مع ذاكرة البرنامج ROM .

لنعود إلى المثال السابق ونقوم بحله بطريقة أخرى :

توصيل 8 دايودات ضوئية على المتحكم وكتابة برنامج يقوم بإخراج القيم من 0 إلى 255 على البورت PORT بتأخير زمني بين كل قيمة 100 ميللي ثانية :

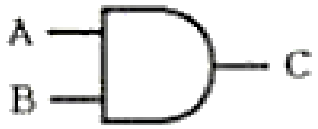
الكود البرمجي : لاحظ ان PORTC خرج

```
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) | (1<<DDC2) | (1<<DDC1) | (1<<DDC0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

for (counter=0;counter<=200;counter++)
    {
    PORTC=counter;
    delay_ms(100);
    }
}
```

# العمليات المنطقية

| | |
|---|---|
| OR | \| |
| AND | & |
| NOT | ~ |
| XOR | ^ |
| Shift Left | << |
| Shift Right | >> |

AND




| Inputs | | Output |
|---|---|---|
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

أمثلة على AND

0 & 0 = 0

0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

OR

| Inputs | | Output |
|---|---|---|
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

أمثلة على OR

0 | 0 = 0

0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

NOT

| Input | Output |
|---|---|
| A | C |
| 0 | 1 |
| 1 | 0 |

أمثلة على NOT

~ 0 = 1
~ 1 = 0

Exclusive-OR gate

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

أمثلة على XOR

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

# عمليات الإزاحة Shift Operation:

وهي عملية تحريك البتات إلى اليمين أو اليسار داخل متغير أو مسجل أو أي قيمة رقمية . تعد هذه العمليات من أهم الأوامر التي تستخدم في برمجة المعالجات والمتحكمات الصغرية

| 0b00000101 | الرقم الأصلي |
|---|---|
| 0b00001010 | إزاحة بمقدار 1 بت جهة اليسار << |
| 0b00010100 | إزاحة بمقدار 2 بت جهة اليسار << |
| 0b00101000 | إزاحة بمقدار 3 بت جهة اليسار << |

**مثال:** لنفرض أن لدينا 0b01100000 لنقم بتطبيق عمليات الإزاحة لجهة اليمين.

| 0b01100000 | الرقم الأصلي |
|---|---|
| 0b00110000 | << إزاحة بمقدار 1 بت جهة اليمين |
| 0b00011000 | << إزاحة بمقدار 1 بت جهة اليمين |
| 0b00001100 | << إزاحة بمقدار 1 بت جهة اليمين |

## التعامل مع بتات محددة داخل أي متغير :

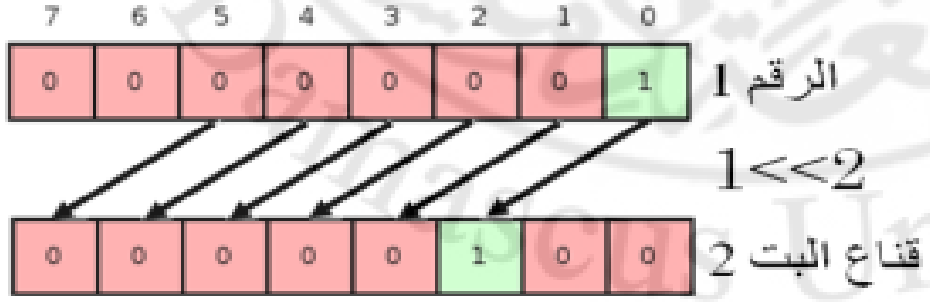
بفرض لدينا المتغير a=11110000 والمطلوب جعل البت رقم 1 يساوي القيمة 1 بدلا من 0 ، أي نريد جعل المتغير يصبح a=11110010 ، يمكن ذلك من خلال الأمر a=0B11110010 أو باستخدام القناع أو الإزاحة .

## القناع mask لأي بت :

القناع للبت رقم واحد هو (00000010) وهو رقم كله أصفار ماعدا البت رقم 1 يوضع بالقيمة (1) . بنفس الطريقة فإن قناع البت رقم 5 سيكون (00100000) أي أن كل البتات تساوي أصفار ماعدا البت رقم 5 .

وبهذا فإن قناع أي بت رقمه n سيكون رقم بتاته كلها أصفار ماعدا البت رقم n .

مثال الشكل التالي : لدينا الرقم الستعشري 0x01=00000001 بإجراء إزاحة ناحية اليسار بمقدار 2 بت على بتات الرقم 1 يصبح الناتج (00000100) . وهذه العملية يمكن التعبير عنها بلغة C كالتالي 1<<2 ، حيث ان الرمز << يعني إزاحة ناحية اليسار والرقم على يساره هو الرقم المراد إزاحته ، والرقم على يمين الرمز هو مقدار الإزاحة بالبتات، وبالتالي فإن 1<<2 يعني إزاحة بتات الرقم 1 ناحية اليسار بمقدار 2 بت . وعلى ذلك فإن قناع أي بت يمكن الحصول عليه بإزاحة الرقم 1 ناحية اليسار بمقدار n من البتات باستخدام التعبير (1<<n)



لو تم إجراء عملية OR المنطقية على محتويات الرقم a=11110000 مع قناع البت رقم 1 الذي نحصل عليه بإزاحة الرقم 1 بمقدار بت واحدة وهو 00000010 ، فإن الناتج سيكون 11110010.
وهو نفس الرقم a مع تغير البت رقم 1 فقط من 0 إلى 1 .

يمكن التعبير عن ذلك بالأمر التالي : ;(1<<1)=|a والذي يعني جعل الرقم a يساوي نفسه (a) بعد إجراء عملية OR على محتوياته مع ناتج إزاحة الرقم 1 بمقدار 1 بت .

التعبير (1<<2)=|a يكون كالتالي a تساوي a أور مع الرقم 1 المزاح بمقدار 2 بت .
وبالتالي فإن التعبير (1<<0)=|a سيكون ناتجه a=11110001 .
والتعبير (1<<3)=|a سيكون ناتجه a=11111000 ،
والتعبير (1<<5)=|a سيكون a=11110000 (أي لاتغيير) حيث أن 1 اور 1 هو 1 .
لجعل أي بت في أي رقم يساوي الواحد ، أي عمل setting لهذا البت فإننا نجري عملية OR على الرقم مع قناع mask هذه البت ..

إذا أردنا تصفير resetting أي بت من بتات أي رقم بدون تغيير بتاته الأخرى ، نقوم بإجراء عملية AND على محتويات الرقم مع معكوس قناع هذه البت .

مثلا : لدينا الرقم a=11110000 والمطلوب تصفير البت رقم 5 ، قناع البت رقم 5 هو 00100000 ومعكوس هذا القناع هو 11011111 وبإجراء عملية آند على هذا المعكوس مع الرقم a نحصل على الرقم a الجديد وهو a=11010000 ، بعد أن تم تصفير البت رقم 5 فيه . الأمر الذي سيقوم بهذه العملية هو

a&=~(1<<5) والذي ينطق a تساوي a بعد عمل آند لها مع معكوس قناع البت رقم 5.

**مثال برمجي :**

بالعودة إلى مثال تشغيل ثماني دايودات ضوئية باستخدام نظام العداد في حلقة for ، لاحظنا أثناء تنفيذ الكود على برنامج المحاكاة أن تشغيل وإطفاء الليدات لم يكن منتظما بالترتيب كما كان في الكود الطويل التقليدي الأول الذي قمنا فيه بتشغيل وإطفاء كل قطب PIN على سطر . ولحل هذه المشكلة سوف نعتمد نظام الإزاحة كما يلي :

قمنا بتعديل الكود البرمجي كما يلي :

```
// Port C initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) | (1<<DDC2) | (1<<DDC1) | (1<<DDC0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
while(1)
{
for (counter=0;counter<=255;counter++)
    {
    PORTC=(1<<counter);
    delay_ms(100);
    }
}
```

نتيجة تنفيذ الكود بدأت الديودات بالإضاءة واحدة تلو الأخرى بالترتيب وذلك باستخدام متغير counter لزيادة قيمة الإزاحة بصورة تلقائية وذلك ضمن حلقة while وباستخدام المحاذاة أصبحت الإضاءة للديودات بالترتيب واحد تلو الآخر

# تعديل الكود باستخدام حلقتي for لعكس اتجاه الإضاءة

```
50  // State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
51  PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
52
53  // Port D initialization
54  // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
55  DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
56  // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
57  PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
58  while(1)
59  {
60   for (counter=7;counter>=0;counter--)
61          {
62          PORTC=(1<<counter);
63          delay_ms(200); }
64          for (counter=0;counter<=7;counter++)
65              {
66              PORTC=(1<<counter);
67              delay_ms(200);
68              }
69          }
70      }
71
```

# عكس بت محددة Toggle Bit

تعد الصيغة البرمجة Toggle Bit من أفضل الصيغ المستخدمة في البرمجة وتعني أنه في حالة أن البت المطلوبة = 1 قم بعكسها إلى 0 وإذا كانت تساوي 0 قم بعكسها إلى 1. وتكون الصيغة القياسية كالتالي:

```
Resgister ^= (1 << BitName);
```

باستخدام هذه الصيغة يمكننا اختصار المثال blinking led بأمرين فقط (بدلاً من 4 أوامر).

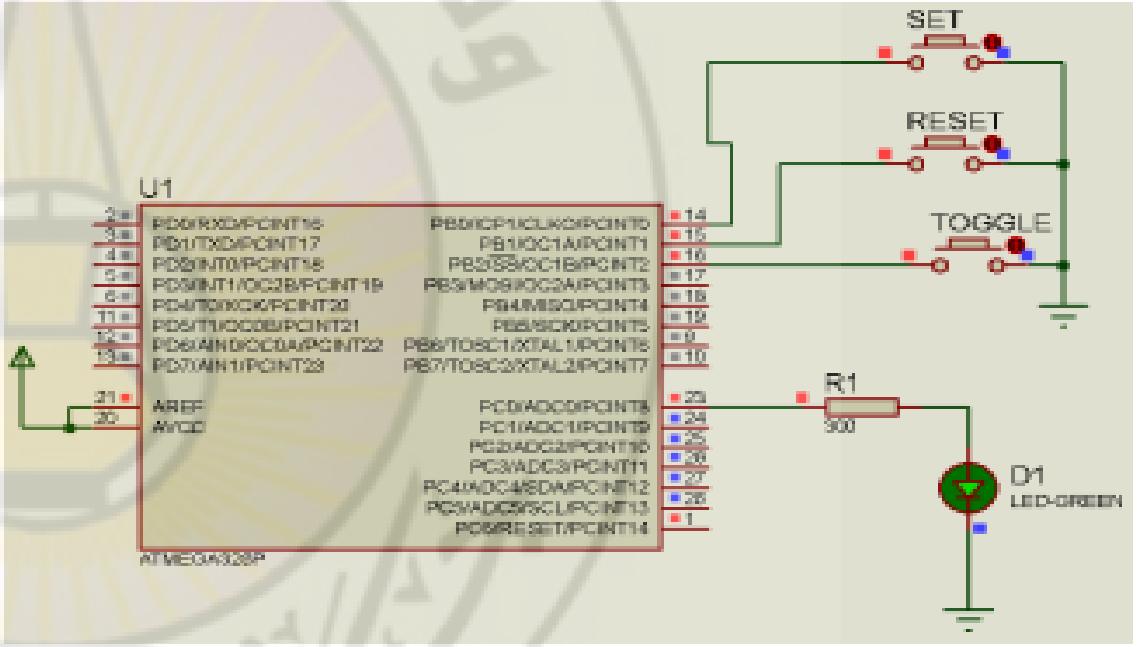
```
57
58    while (1)
59          {
60          // Place your code here
61          if (PINC.0==1){
62          PORTA^=(1<<PORTA.0);
63          delay_ms(100);
64          }
```

باستخدام 4 أوامر

```
57
58    while (1)
59          {
60          // Place your code here
61          if (PINC.0==1){
62          PORTA.0=1;
63          delay_ms(100);
64          PORTA.0=0;
65          delay_ms(100);
66          }
67
68          }
```

# التحكم فى دايود ضوئى من خلال ثلاث مفاتيح

سنحاول فى هذا المشروع التدريب على التعامل مع بت بعينها فى أحد المتغيرات من خلال الإزاحة. سنجعل البوابة C بوابة خرج مع وضع دايود ضوئى ومقاومة على المخرج الأول منها PC0 بحيث يضىء عند إخراج واحد على هذا الطرف. فى المقابل سنجعل البوابة B بوابة دخل مع وضع ثلاث مفاتيح على المداخل الثلاثة الأولى فيها، PB0 و PB1 و PB2. المطلوب عند الضغط على المفتاح الأول أن يضىء الدايود الضوئى، وإن كان مضيئا فى الأصل، يبقى كما هو، ولذلك سنسميه مفتاح الوضع SET. المفتاح الثانى عند ضغطه يطفىء الدايود الضوئى، وإن كان مطفأ فى الأصل يظل كما هو، ولذلك سنسمي مفتاح إعادة الوضع RESET. المفتاح الثالث عند الضغط عليه نريده أن يغير وضع الدايود الضوئى، فإن كان مطفأ يضيئه، وإن كان مضيئا يطفؤه، ولذلك سنسميه مفتاح تغيير الوضع TOGGLE. البرنامج سيكون كالتالى،



الكود البرمجي :

```
while (1)
    {
    // Place your code here
     if ((PINB&(1<<0))==0)
       {
       PORTC|=(1<<0);
       }
     if ((PINB&(1<<1))==0)
       {
        PORTC&=~(1<<0);
       }
     if ((PINB&(1<<2))==0)
       {
       PORTC^=(1<<0);
       }
    }
}
```

بعد تفعيل مقاومات الرفع الداخلية تم تنشيط حلقة لانهائية تحتوي ثلاث أوامر شرطية ، الأول(0==((PINB&(1<<0)) عملية آند على البوابة B مع الرقم 00000001 وهو (1<<0) والعملية تساوي الصفر عند ضغط المفتاح تكون البوابة صفر ويكون الخرج على PORTC هو واحد ;(1<<0)=|PORTC

الشرط الثاني (0==((PINB&(1<<1)) if يقوم بإجراء آند للبوابة B مع 00000010 وهو ناتج (1<<1) ليتاكد إذا كان المفتاح الثاني تم ضغطه أم لا بحيث أذا كان مضغوط يتم تصفير الطرف PC0 وبالتالي إطفاء الليد .

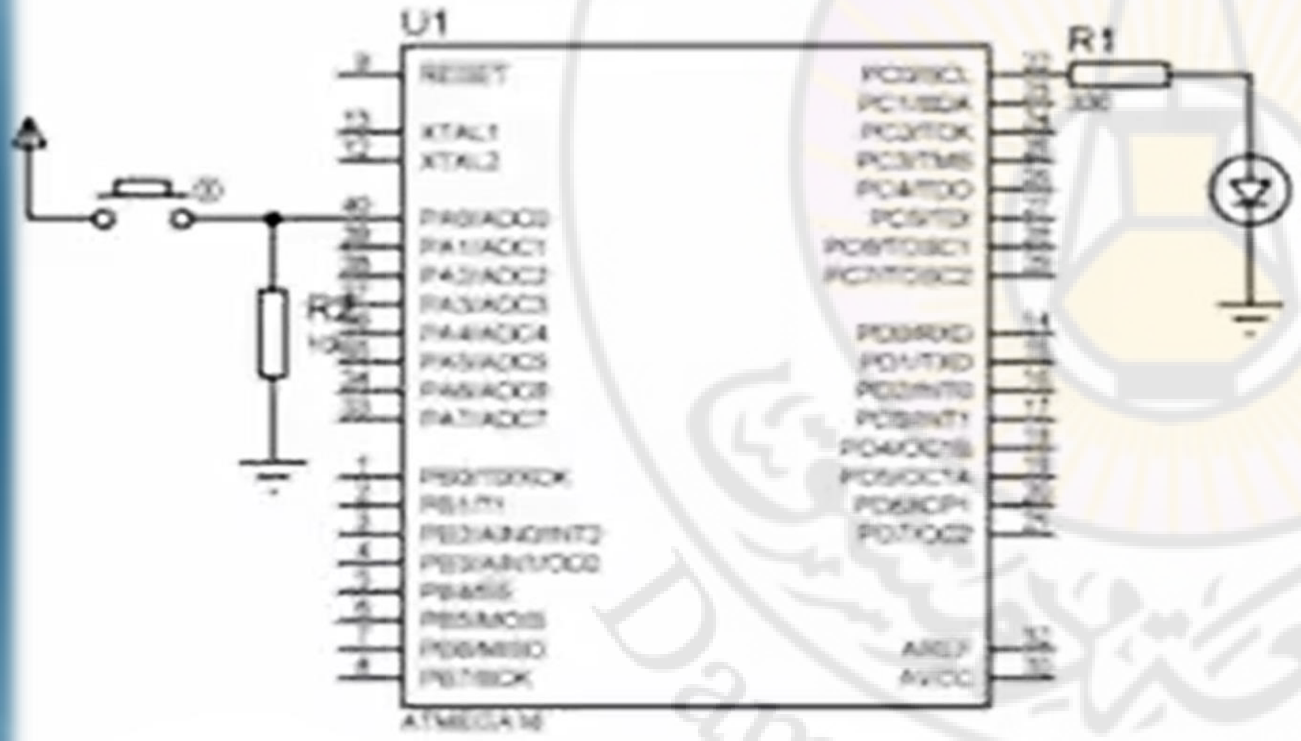
بنفس الطريقة الشرط الثالث

# قراءة الدخل الرقمي

# Inputs Reading

# تطبيق على استخدام PORT كدخل

اكتب برنامج يقوم بتشغل LED لمدة 1 s وإطفاء LED لمدة 1 s .... وهكذا يستمر بالعمل إذا تم ضغط button.




```
#include <mega16.h>
#include <delay.h>
void main(void)
{
DDRC.0=1;
Start:
if (PINA.0==1)
   {   PORTC.0= 1;
      delay_ms(1000);
       PORTC.0= 0;
      delay_ms(1000);}
goto Start;
```

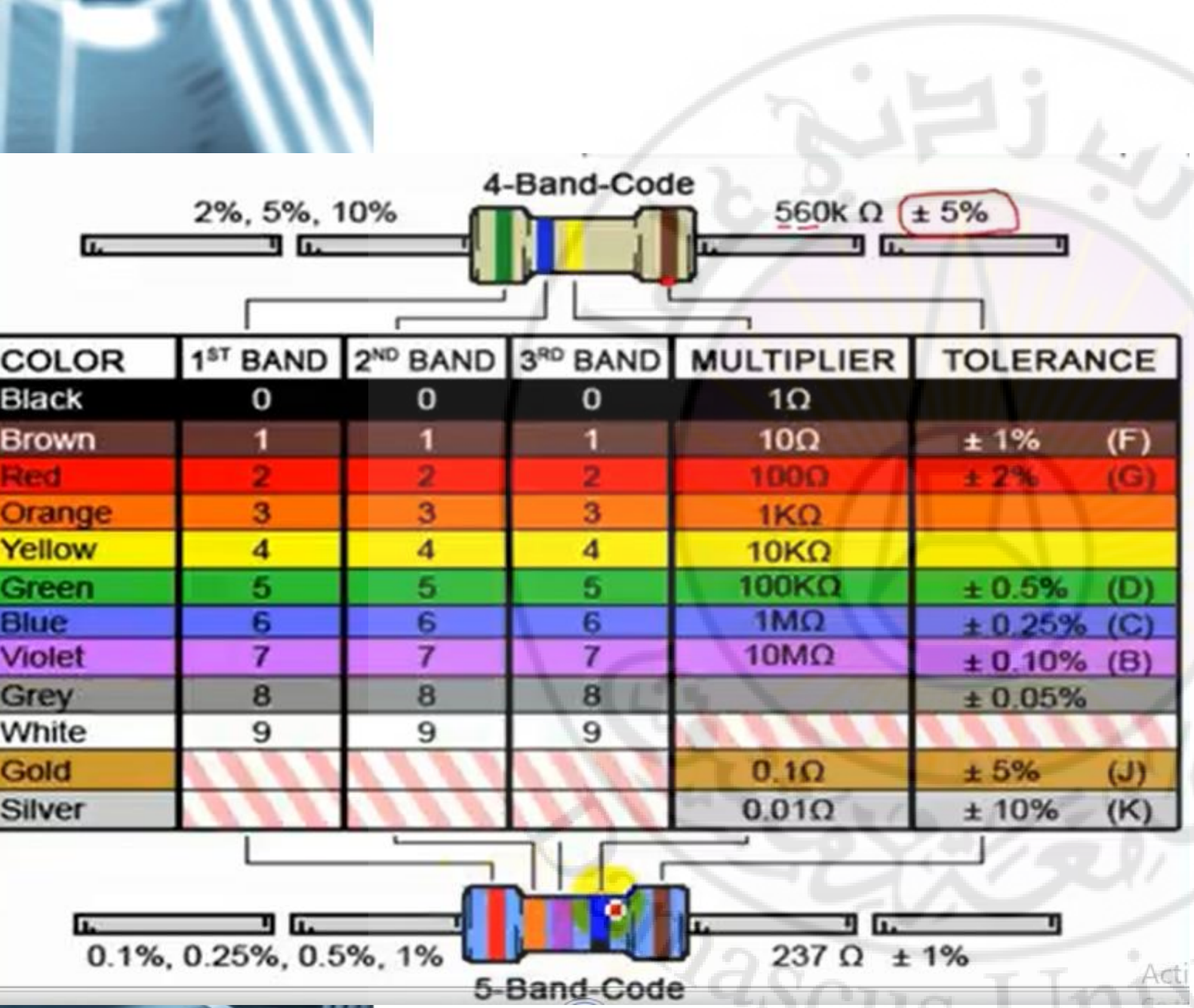
قراءة حالة PINA

قبل البدء سنتذكر أنواع وأهمية استخدام المقاومات في الدارات .

ان الاستعمالان الرئيسيان للمقاومة هما:

- تقسيم الجهد (مقسم للجهد)
- التحكم في شدة التيار لمراعاة شروط خروج ودخول التيار إلى المتحكم الصغري.

ويمكن قياسها إما عن طريق الآفو أو الألوان كما يلي :




| COLOR | 1ST BAND | 2ND BAND | 3RD BAND | MULTIPLIER | TOLERANCE |
|---|---|---|---|---|---|
| Black | 0 | 0 | 0 | 1Ω | |
| Brown | 1 | 1 | 1 | 10Ω | ± 1% (F) |
| Red | 2 | 2 | 2 | 100Ω | ± 2% (G) |
| Orange | 3 | 3 | 3 | 1KΩ | |
| Yellow | 4 | 4 | 4 | 10KΩ | |
| Green | 5 | 5 | 5 | 100KΩ | ± 0.5% (D) |
| Blue | 6 | 6 | 6 | 1MΩ | ± 0.25% (C) |
| Violet | 7 | 7 | 7 | 10MΩ | ± 0.10% (B) |
| Grey | 8 | 8 | 8 | | ± 0.05% |
| White | 9 | 9 | 9 | | |
| Gold | | | | 0.1Ω | ± 5% (J) |
| Silver | | | | 0.01Ω | ± 10% (K) |

# حساب المقاومة المستخدمة من قبل الأحمال:

الحِمل Load هو أي جهاز يتصل بخرج المُتحكِم الدقيق مثل الدايود الضوئي، في الأمثلة السابقة لاحظنا وجود مقاومة قبل كل دايود بقيمة (330 أوم)، لماذا استخدمنا هذه المقاومة وبتلك القيمة تحديداً؟

هناك سببان لهذا الأمر. الأول هو أن الدايود الضوئي (خاصة اللون الأحمر) يفضل أن يتم تشغيله بتيار كهربي ما بين 15 إلى 20 مللي أمبير عند تطبيق فرق جهد 5 فولت. وبالتعويض في قانون أوم الكهربي (فرق الجهد 5 فولت - التيار 15 مللي أمبير)

$$Voltage(V) = Current(I) \times Resistance(R)$$

**نجد أن المقاومة المناسبة هي 330 أوم .**

إذا لم تستخدم هذه المقاومة قد تحدث العديد من المشاكل. فمثلاً إذا تعرض الدايود الضوئي الأحمر لأكثر من 25 مللي أمبير لفترة طويلة نسبياً (ساعة أو أكثر) فأنه سيحترق وبالتأكيد أنت لا تريد دائرة إلكترونية تحتاج لتغيير دايود ضوئي كل ساعة !

ثانياً: قد يؤدي تشغيل الأحمال بدون مقاومة لفترات طويلة إلى تدمير أطراف (بورتات) المُتحكِم الدقيق. وذلك بسبب محدودية التيار الكهربي الذي يمكن سحبه من كل طرف، إذا نظرت إلى الصفحة الخاصة بالخصائص الكهربية للمتحكم Electrical Charchtaristics

## Electrical Characteristics

### Absolute Maximum Ratings*

| | |
|---|---|
| Operating Temperature | -55°C to +125°C |
| Storage Temperature | -65°C to +150°C |
| Voltage on any Pin except $\overline{RESET}$ with respect to Ground | -0.5V to $V_{CC}$+0.5V |
| Voltage on $\overline{RESET}$ with respect to Ground | -0.5V to +13.0V |
| Maximum Operating Voltage | 6.0V |
| DC Current per I/O Pin | 40.0 mA |
| DC Current $V_{CC}$ and GND Pins | 200.0 mA PDIP and 400.0 mA TQFP/MLF |

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**لاحظ السطر قبل الأخير حيث نجد DC current per I/O = 40 mA**

هذا يعني أن أقصى تيار يمكن سحبة من أي طرف من أطراف المُتحكِم = 40 مللي أمبير وإذا كان الحِمل يسحب أكثر من ذلك فهذا سيؤدي إلى تلف المُتحكِم الدقيق. عليك أن تتذكر هذا الرقم جيداً عند تصميم أي دائرة إلكترونية.

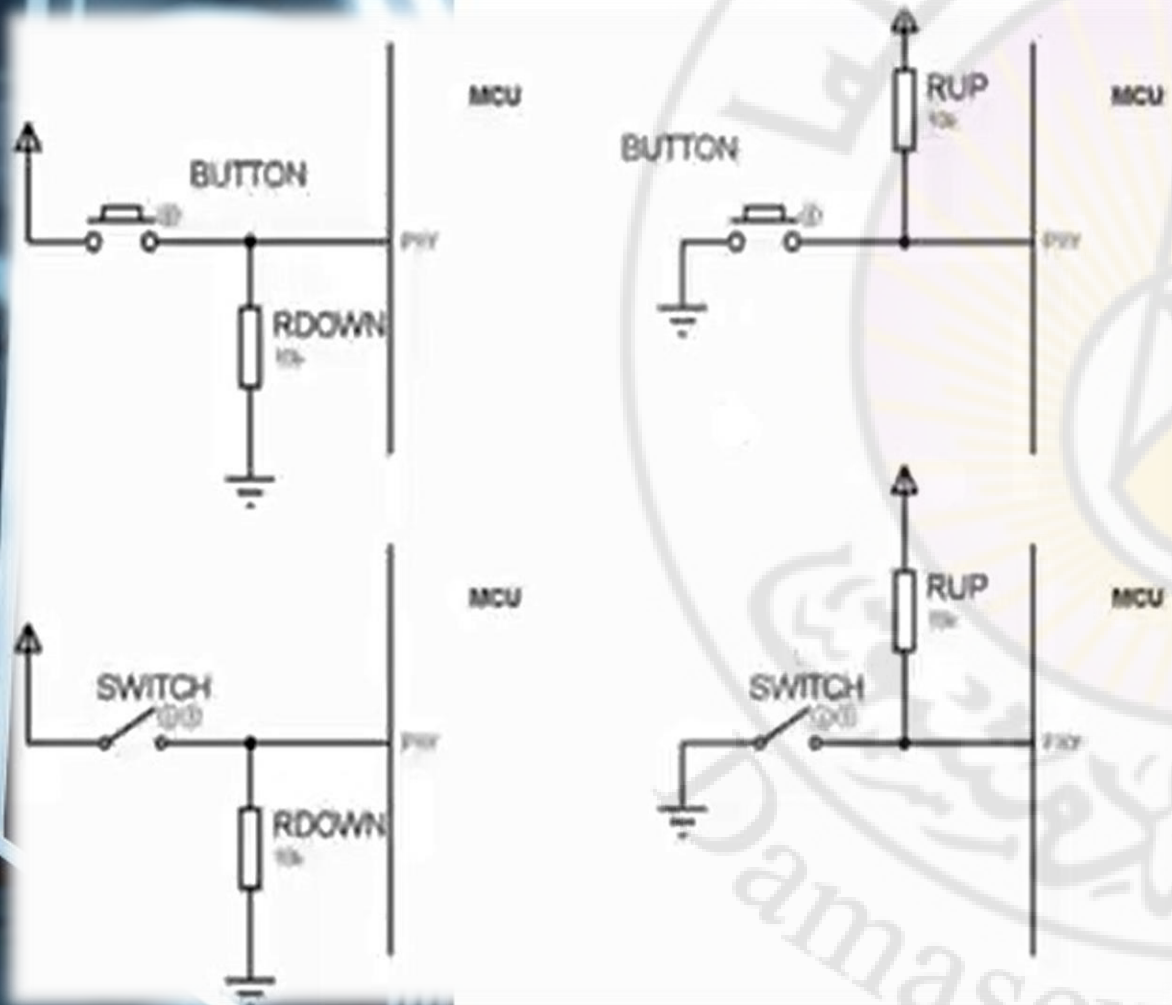
الأفضل أن تسحب الأحمال تيار = 85% من القيمة القصوى وذلك حتى تضمن أن يكون العمر الإفتراضي للمتحكم الدقيق أطول ما يكون. فمثلاً إذا طبقنا هذه القاعدة سنجد أن التيار الآمن هو 0.85 * 40 = 34 مللي أمبير.

**هذا يعني أن أقل مقاومة يمكن توصيلها على أي طرف من أطراف المُتحكِم (من قانون أوم) تساوي 147 أوم، مع ملاحظة أنه يمكن توصيل أي قيمة أعلى لأن المقاومات الأكبر ستمرر تيار كهربي أقل وهذا آمن تماماً.**

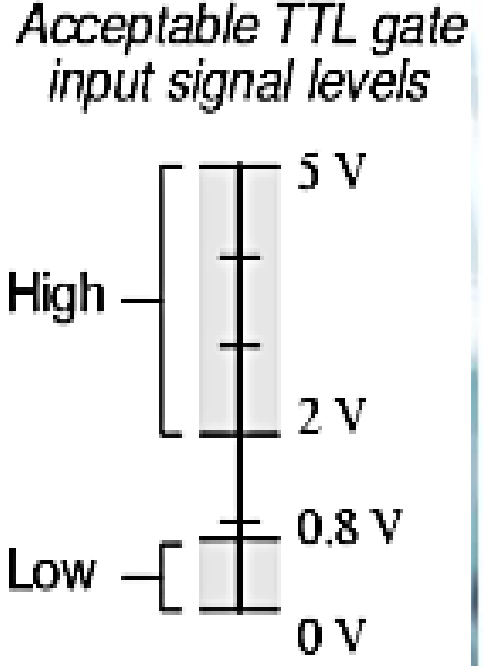
# ربط المفاتيح والكباسات اللحظية مع مداخل المتحكم AVR – مقاومة الرفع والخفض

هناك طريقتان :

1. طرف المفتاح موصول مع القطب الموجب (Vcc) والطرف الثاني موصول قطب المتحكم وموصول مع مقاومة ومع الأرضي . في حالة الزر غير مفعلOFF سيمر صفر منطقي عبر المقاومة ومنه للقطب ، وفي حالة التفعيل ON يمر واحد منطقي . هذه المقاومة تسمى مقاومة خفض .
2. الطريقة الثانية وهي الأكثر شيوعا ان يكون طرف الكباس موصول مع الأرضي Gnd ، والطرف الثاني موصول مع المقاومة ومع القطب ، فعندما يكون الزر غير مفعل يمر واحد منطقي وعند التفعيل يصبح واحد منطقي . وهذه المقاومة تسمى مقاومة رفع .

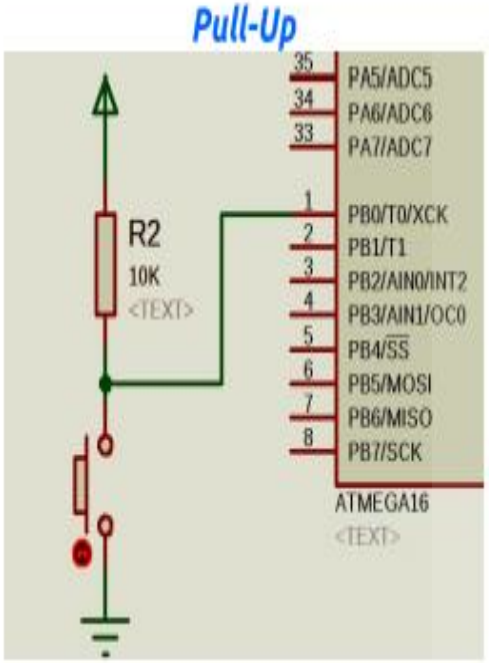


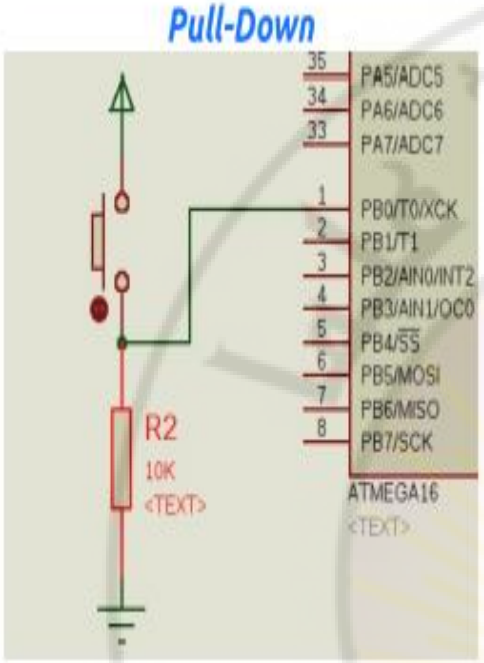
# استخدامات مقاومة الرفع ومقاومة الخفض Pull Up & Pull Down Resistor




هذه المقاومات تتراوح قيمتها بين 2.2 كيلو أوم إلى 10 كيلو أوم تستخدم بصورة أساسية داخل الأنظمة الرقمية Digital Inputs ، بداية من البوابات المنطقية البسيطة وانتهاء بالمتحكمات الدقيقة .
ومن أسباب استخدام هذا النوع من المقاومات مثل المقاومة 10 كيلو أوم مع مفتاح بدون الوصل إلى Vcc مباشرة هي مايلي:

الاستخدام الأول – إلغاء الدخل العائم : جميع المكونات الرقيمة التي تتعامل مع الصفر والواحد تعاني من مشكلة كبيرة وهي المنطقة العائمة Floating Area أو Noise Margin نطاق الشوشرة ، هذه المنطقة هي فارق الجهد بين الصفر المنطقي والواحد المنطقي ، وتستخدم مقاومات الرفع والخفض لحل هذه المشكلة .
والسبب في ذلك أن معظم المكونات الالكترونية تعتبر أي جهد دخل بين (0-0.8) فولت هو صفر (0) أو مايسمى Low . بينما أي جهد بين (2-5) فولت هو واحد (1) أو مايسمى HIGH .
لكن إذا كان هناك دخل بقيمة 0.9 فولت او 1.25 او 1.9 تحدث لدينا مشكلة المنطقة العائمة ، حيث لاتستطيع الالكترونيات الرقمية أن تتعرف على هذا الجهد وبالتالي لايمكنها أن تحدد بدقة ماإذا كانت القيمة (1) HIGH أو LOW(0) وتبدأ بالتصرف عشوائيا .
لذا عند ترك أي طرف دخل input فغنه يتعرض لشوشرة كهربية بفرق جهد صغير نسبيا مما يتسبب في دخول الجهاز بوضع floating area.
وجود مقاومة متصلة بطرف الدخل والطرف الأرضي يضمن تماما أن الطرف يساوي الصفر حتى مع وجود الشوشرة ، ولايتم تغيير هذا الجهد إلا عند إدخال جهد كبير نسبيا مثل 5 فولت ، وتكون قيمة المقاومة بين (2.2 – 10) كيلو أوم



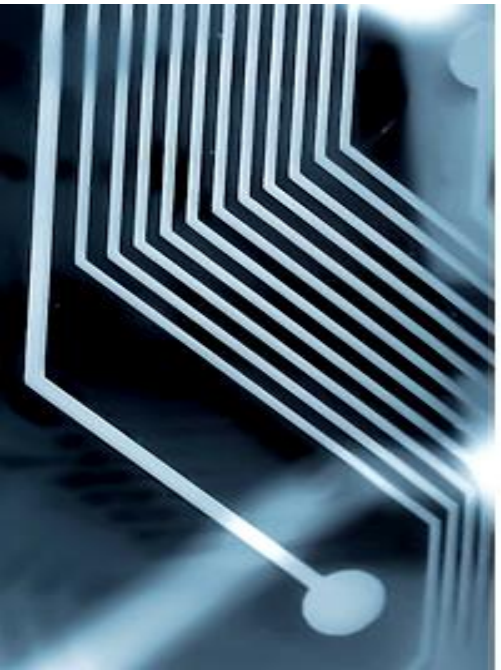



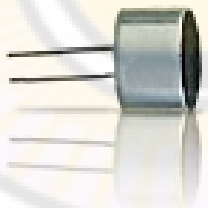

الاستخدام الثاني – عكس الجهد الداخل : في بعض الحالات يكون مطلوب عكس الجهد الداخل من المفتاح ، أي بأسلوب pull-Down مما يجعل المفتاح يدخل جهد 5 فولت عند الضغط على المفتاح ، و 0 فولت (أرضي) عند ترك المفتاح .

Press Button = HIGH  Releas Button = LOW

يمكن استخدام المقاومة الرافعة Pull-Up Resistor لعكس هذا الجهد وبذلك يصبح الجهد الأساسي للطرف هو HIGH وعند ضغط المفتاح يتحول هذا الجهد إلى صفر.

Press Button = LOW  Releas Button = HIGH





**الاستخدام الثالث (مُحول تيار إلى جهد):** بعض الحساسات الشهيرة مثل الميكروفون Microphone أو المقاومة الضوئية LDR أو الحرارية NTC تقوم بتحويل الحرارة أو الضوء إلى تغير في **التيار** الكهربي وليس تغير في **الجهد** مما يمثل مشكلة في فهم هذه الحساسات. حيث نجد أن جميع المُتحكِمات الدقيقة التي تحتوي على ADC يمكنها قراءة **تغير الجهد فقط** ولا تستطيع التعرف على التيار الكهربي المتغير لذا يتم استخدام مقاومة Pull-Down أو Pull-Up لتحويل التيار المتغير إلى جهد.

# خاصية Internal Pull-Up :

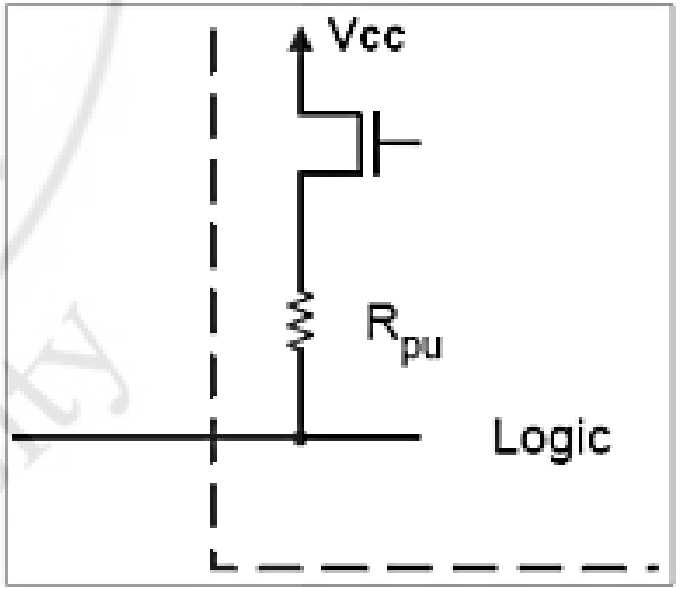
تمتلك متحكمات AVR خاصية أن أطراف البورتات (الأقطاب) تمتلك مقاومة رفع داخلية ، هذه المقاومة تمكننا من استخدام المفاتيح دون الحاجة إلى مقاومات إضافية ، الشكل التالي يوضح تركيب مقاومة الرفع الداخلية وهي عبارة عن :

ترانزستور+مقاومة$R_{Pu}$ ، حيث يتحكم الترانزستور في تفعيل المقاومة أو إلغائها.

بصورة افتراضية يتم إلغاء تفعيل هذه المقاومة ويتم تشغيلها عن طريق جعل الطرف المطلوب يعمل كدخل ، ومن ثم الكتابة في المسجل PORTx لتفعيل هذه المقاومة.

وبهذه الحالة فإن المسجل لن يعمل لكتابة قيم الخرج لأنه سوف يقوم بتفعيل الترانزستور الخاص بمقاومة الرفع فقط ، وأيضا بعد تفعيل المقاومة يصبح تأثير الضغط على المفتاح معكوس مما يعني أن

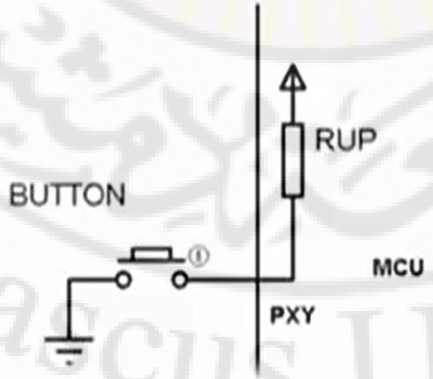
قراءة PINx ستصبح 0 في حالة الضغط على المفتاح (on) و 1 عند ترك المفتاح (off).




مقاومة رفع داخلية يمكن تفعيلها عندما يعمل القطب كدخل عن طريق المسجل PORTX وفق التالي:

DDRX BIT=0

PORTX BIT=1



مثال ١: برمج القطب PD0 في المنفذ PORTD
كمدخل وفعل مقاومته الرفع الداخلية RUP.

```
DDRD. 0=0;
PORTD. 0=1;
```

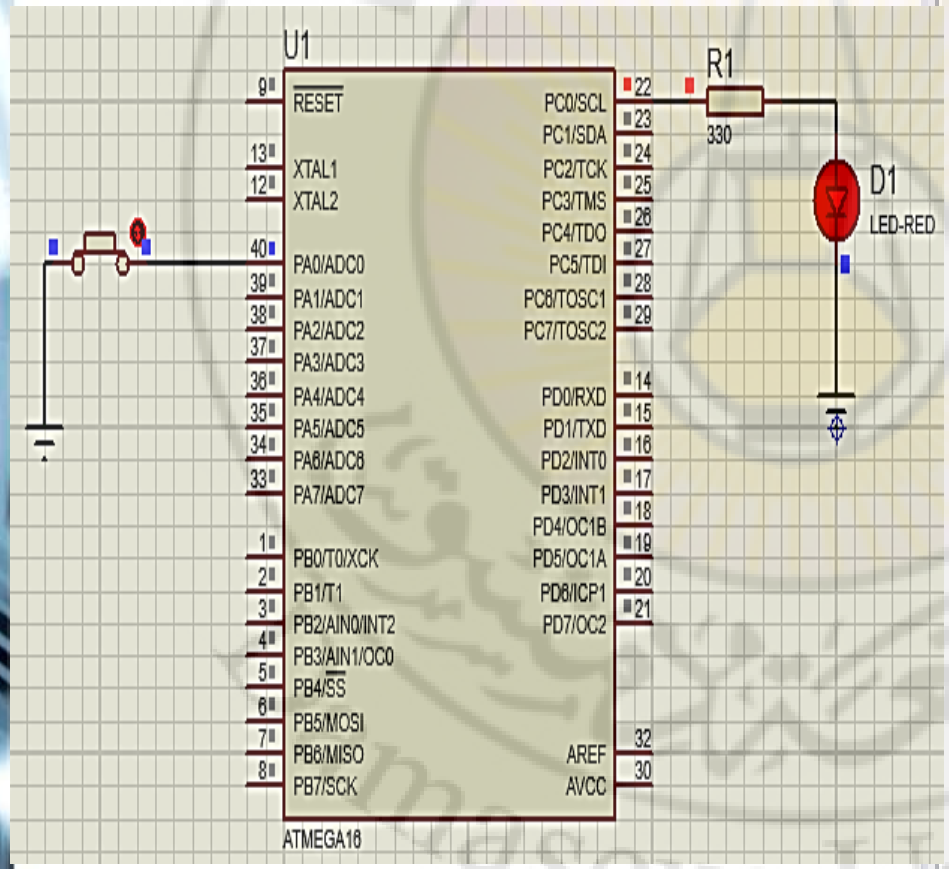
تم وصل مقاومة رفع داخلية على PIND0

مثال ٢: برمج االمنفذ PORTC كي تعمل أقطابه
كدخل وفعل مقاومة الرفع الداخلية RUP لكل قطب.

```
DDRC=0B00000000;
PORTC=0B11111111;
```

Damascus University

نعود للمثال السابق بتفعيل مقاومة رفع داخلية



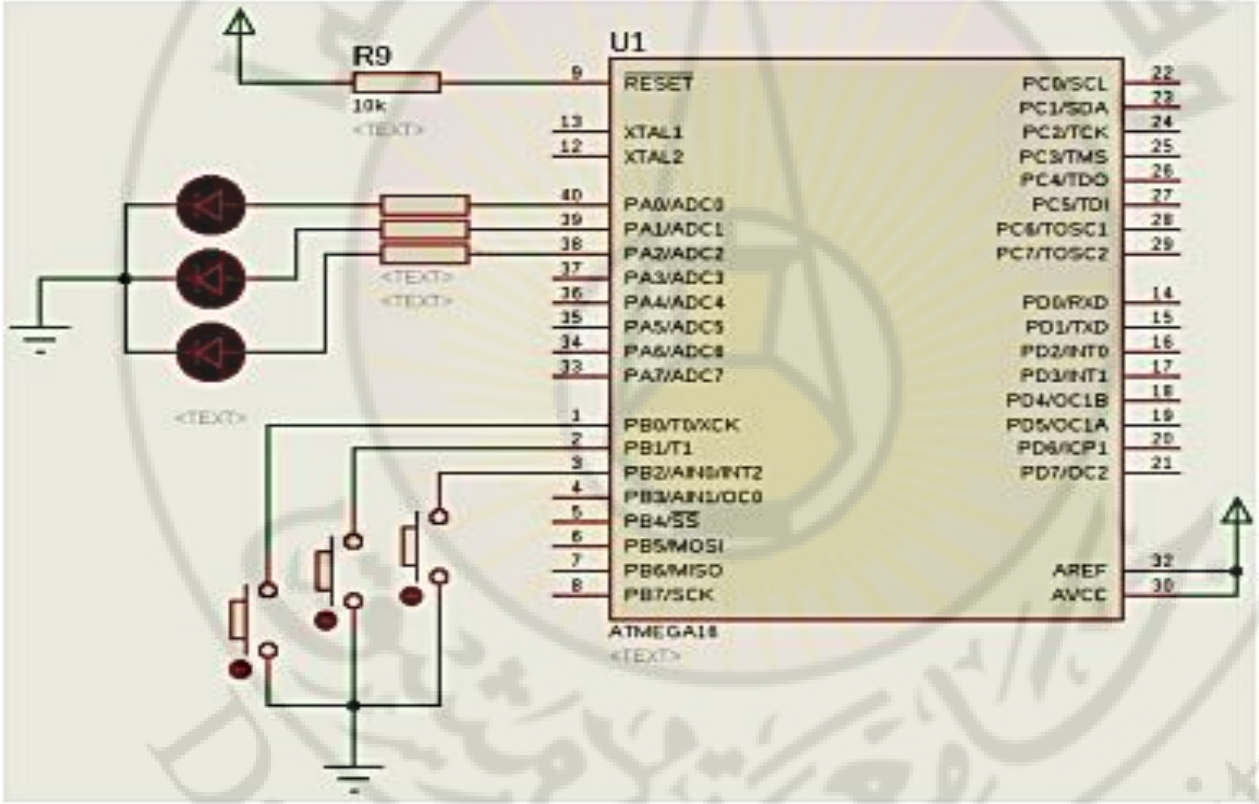

```
fx  main

1   #include <mega16.h>
2   #include <delay.h>
3   void main(void)
4   {
5   DDRC.0=1;
6   PORTA.0=1;
7   start:
8   if (PINA.0==0)
9   {
10  PORTC.0=1;
11  delay_ms(100);
12  PORTC.0=0;
13  delay_ms(100);}
14  goto start;
15  }
```

**PORTA افتراضيا هو دخل كتبنا عليه 1 لتفعيل مقاومة الرفع الداخلية .**
**وبذلك يصبح الشرط PINA.0==0**

# تشغيل 3 ديودات + 3 مفاتيح

في هذا المثال سنستخدم 3 مفاتيح مع تفعيل خاصية الـ Internal pull-up وبذلك سنوفر استخدام 3 مقاومات 10 كيلو. وسيتحكم كل مفتاح في تشغيل عدد من الدايودات بحيث:

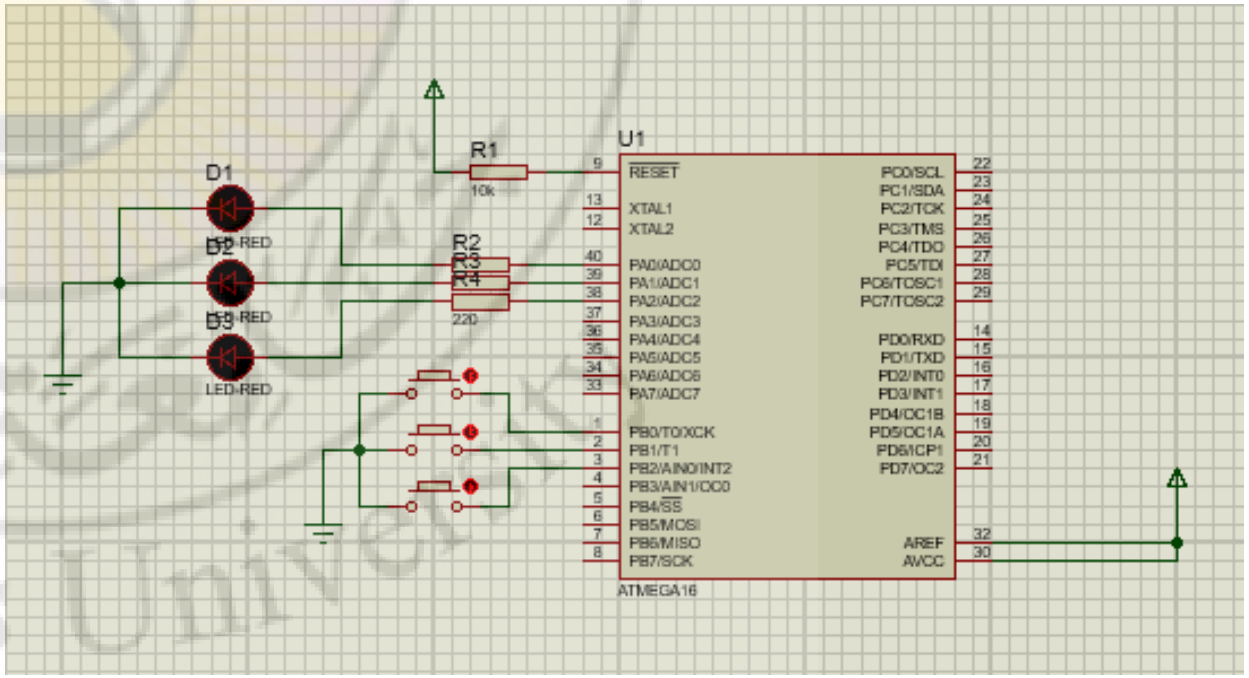
- إذا تم الضغط على المفتاح المتصل بالطرف PB0 يتم تشغيل دايود واحد فقط
- إذا تم الضغط على المفتاح المتصل بالطرف PB1 يتم تشغيل 2 دايود
- إذا تم الضغط على المفتاح المتصل بالطرف PB2 يتم تشغيل 3 دايود

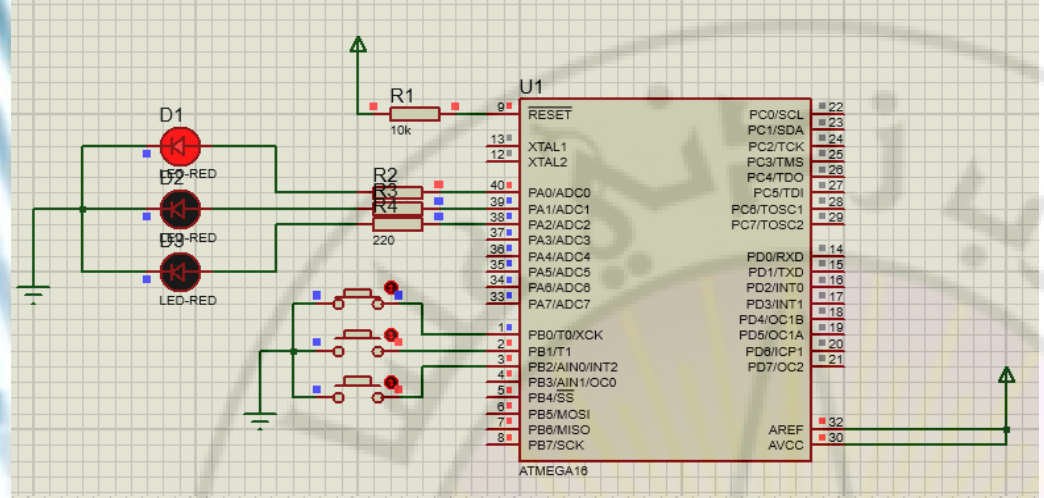


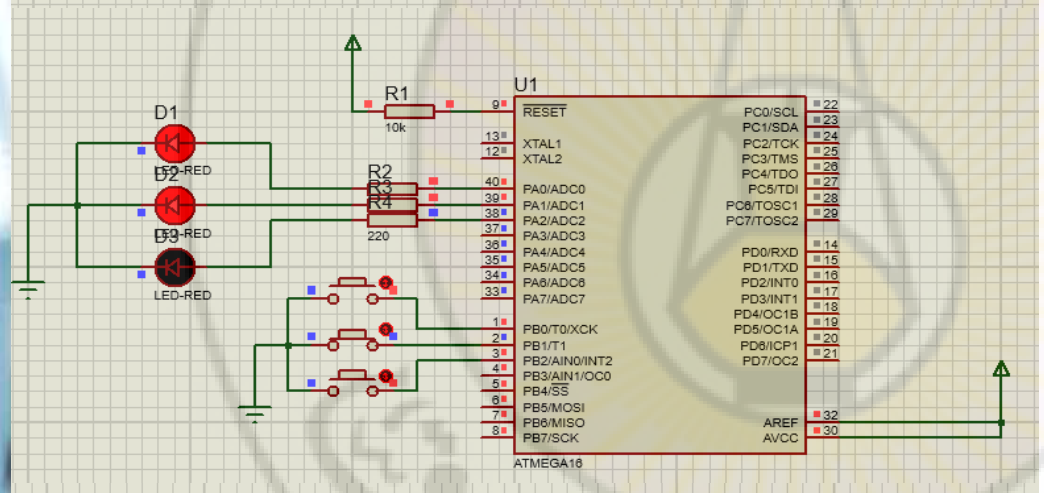

**تذكر أنه بعد تشغيل مقاومة الرفع تصبح قراءة المفتاح والمسجل PINx معكوسة** لذا سنجد القيمة الإفتراضية للمسجل PINx = 0b11111111 وعند الضغط على أي مفتاح ستتحول البت المقابلة له إلى صفر، فمثلاً الضغط على المفتاح المتصل ب PB0 سيجعل قيمة PINB تساوي 0b11111110

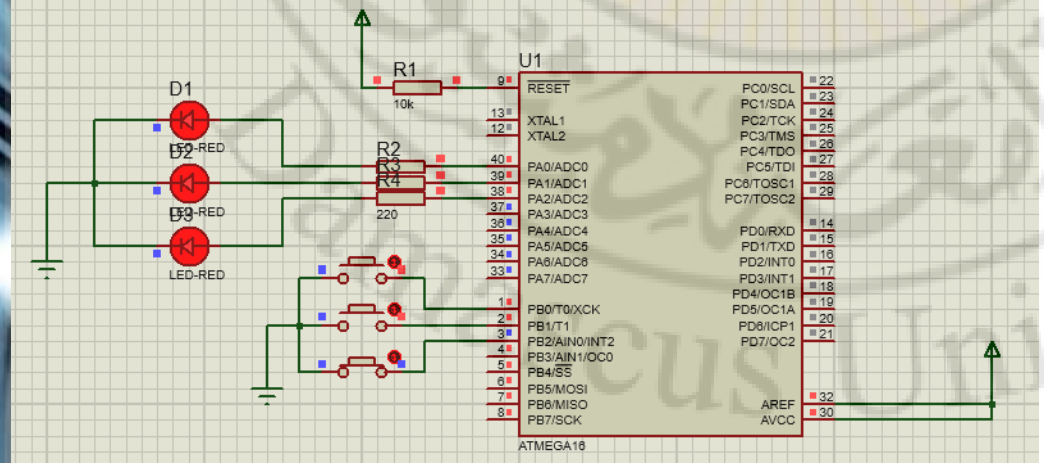
تم تشغيل أطراف PORTB لتعمل جميعها كدخل ، ثم قمنا بتفعيل جميع مقاومات الرفع الداخلية على PORTB

```
// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
PORTB=(1<<PORTB7) | (1<<PORTB6) | (1<<PORTB5) | (1<<PORTB4) | (1<<PORTB3) | (1<<PORTB2) | (1<<PORTB1) | (1<<PORTB0);
```






```
while (1)
{
if (PINB.0==0)
{ PORTA.0=1; }
else if (PINB.1==0)
{ PORTA.0=1;
PORTA.1=1;}
else  if (PINB.2==0)
{ PORTA.0=1;
PORTA.1=1;
PORTA.2=1;
else
{ PORTA.0=0;
PORTA.1=0;
PORTA.2=0; }
}
}
```

# الاهتزازات في المفاتيح الميكانيكية
# تأثير القفز
# Bouncing effect & De-bouncing

تأثير القفز أو الاهتزازات هي ظاهرة تحدث للمفاتيح Switches الميكانيكية خاصة مفاتيح الضغط Push Buttons حيث تتكون هذه المفاتيح عادة من شريحتين من الصفائح المعدنية والتي عند الضغط عليها يحدث بينها تلامس ، وبسبب طريقة صناعة هذه الصفائح فإنه عند الضغط عليها تقفز الألواح المعدنية أكثر من مرة (تهتز) هذا الأمر يجعل الصفائح تقوم بعشرات التلامسات في الثانية الواحدة قبل أن تستقر، أي ان التغيير لايتم من خلال نقلة واحدة وغنما بعد مجموعة اهتزازات غير مرغوبة .

تعتبر هذه الظاهرة سيئة بالنسبة لمصممي النظم المدمجة لأن أنظمة التحكم الالكترونية سريعة جدا (أسرع بكثير من حركة الانسان) فهي تلتقط كافة الإشارات من أجزاء الثواني وبهذا تنتج هذه المفاتيح إشارات غير مطلوبة تؤثر سلبيا على عمل النظام المدمج .

هناك عدة حلول لهذه الظاهرة :

## 1. الحل الأول - التأخير الزمني:

هذا يعتبر ابسط الطرق وتسمى software debouncing حيث تعتمد هذه الطريقة على التأكيد مرتين من الضغط على المفتاح مع وضع تأخير زمني بين كل مرة كما في الكود التالي لتشغيل دايود ضوئي عند الضغط على مفتاحين

الاهتزازات الميكانيكية

الناتجة عن تغير وضع أى مفتاح

الكود التقليدي لقراءة المفتاح بدون debouncing

```
if (PINB == 0b11111110) { PORTA = 0b00000001;}
```

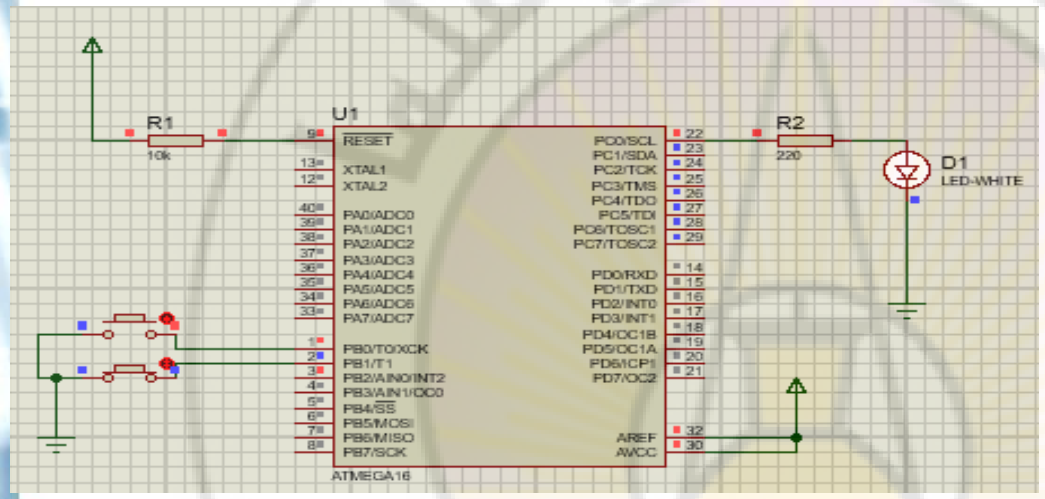
قراءة المفتاح باستخدام ال software debouncing (delay)

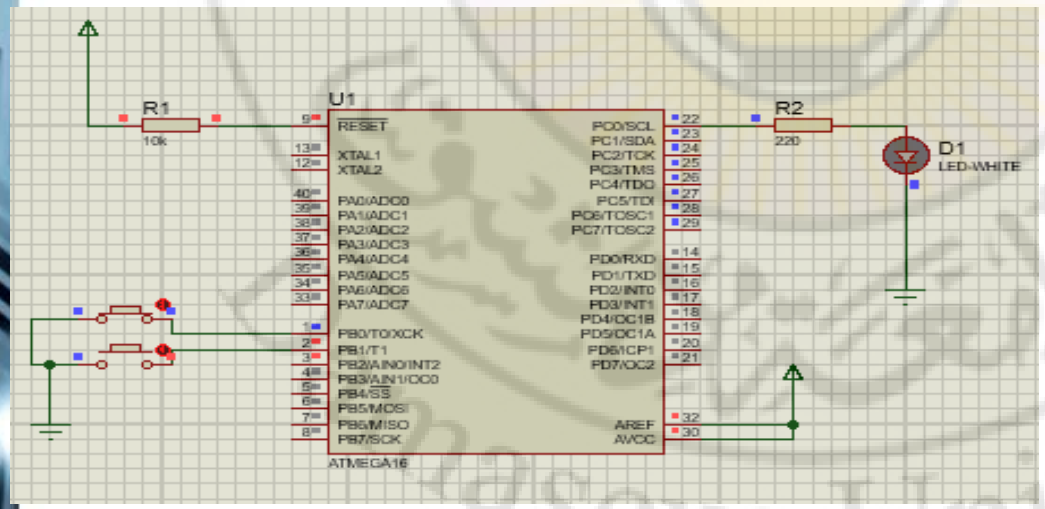
```
if (PINB == 0b11111110)
{
        delay_ms(10);                   //تأخير زمني
        if (PINB == 0b11111110)    // إعادة التأكد أن المفتاح مازال مضغوط
            { PORTA = 0b00000001;}
}
```

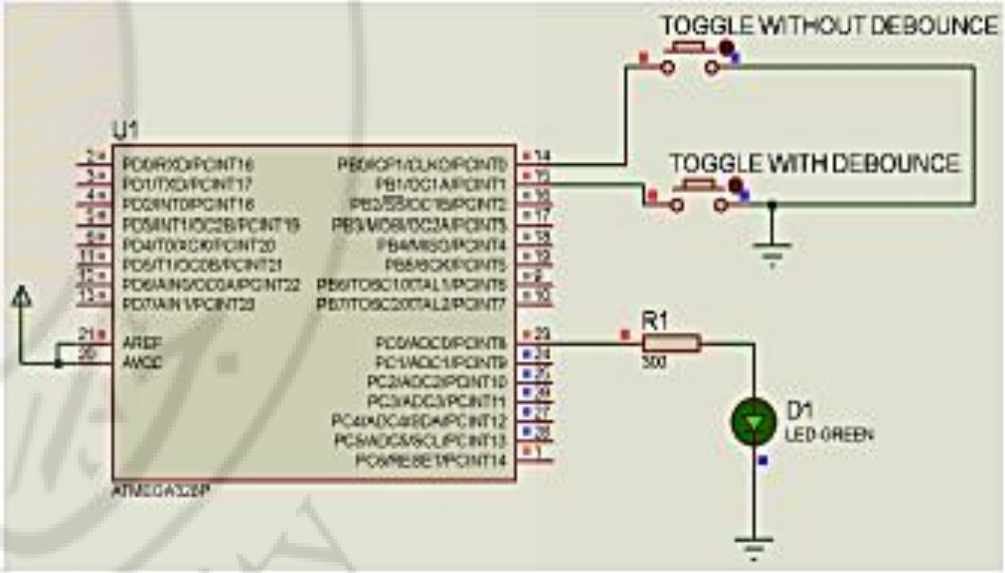
هذه الطريقة لها عيوب كثيرة جداً منها أنها لا تحل مشكلة الذبذبات الكهربية نفسها ولكنها تتأكد فقط من ضغط المفتاح لفترة كافية حتى تنتهي تلك الذبذبات

Damascus University

سنأخذ مثال دائرة الكترونية كما في الشكل : حيث ان المفتاح الأول لايهتم بالتخلص من الاهتزازات أما الثاني تم تعديل قراءته لنتخلص من الاهتزازات كما في الشرح التالي




تأثير التأخر الزمني على إضاءة الليد

الكود مع تأخير زمني وباستخدام الإزاحة:

```
while (1)
    {

     if ((PINB&(1<<0))==0)
       {
       PORTC^=(1<<0);
       }
     if ((PINB&(1<<PINB1))==0)
       {
       delay_ms(25);
       }
     if ((PINB&(1<<PINB1))==0)
       {
       PORTC^=(1<<0);
       }
    }
}
```

المفتاح الأول لايهتم بالتخلص من الاهتزازات أما المفتاح الثاني فقد تم تعديل طريقة قراءته بحيث تخلصنا من هذه الاهتزازات .

عند قراءة الأمر

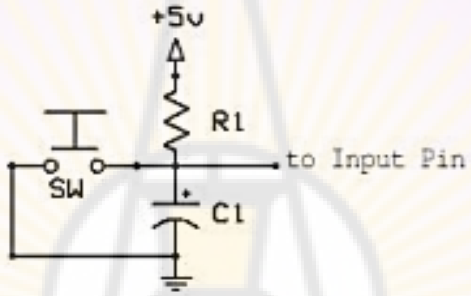
if ((PINB&(1<<PINB1))==0)

يتم بعدها تأخير زمني مقداره 25 ميلي ثانية ، ثم قراءة المفتاح مرة ثانية فإذا كان المفتاح لازال على وضعه ويساوي صفر ، فإن ذلك يعني ان مرحلة الاهتزاز قد مضت واستقر المفتاح على الصفر ، وفي هذه الحالة يتم عمل تغيير للطرف PC1 ، اما إذا كان يساوي 1 بعد مرور 25 ميلي ثانية يعني أن الواحد ناتج من الاهتزازات وفي هذه الحالة لايتم عمل أي إجراء .

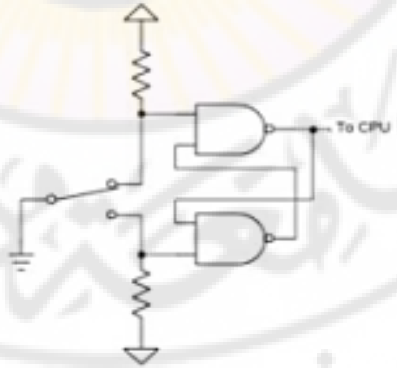
## 2. الحل الثاني- استخدام فلتر مرشح:

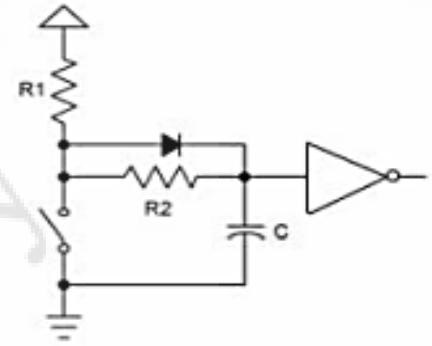
هذا الحل يعتمد على استخدام دائرة filter لترشيح الذبذبات الكهربية الناتجة عن الـ bouncing effect وتسمى Hardware Debouncing حيث تستخدم هذه الدائرة في امتصاص الذبذبات الكهربية وتمنع دخولها للمتحكم الدقيق.

أبسط دائرة مرشح هي توصيل مكثف سيراميكي على التوازي مع المفتاح Ceramic - capacitor بقيمة 0.1 ميكروفاراد (يكون مكتوب على المكثف الرقم 104). مع العلم أنه يمكن توصيل هذا المكثف سواء كنت تستخدم المفتاح مع Pull up أو Pull down resistor




هناك دوائر أخرى أكثر تعقيداً وقوة في الترشيح مثل استخدام مقاومة 10 كيلو + دايود + مكثف (بنفس القيمة). أو باستخدام دوائر رقمية مثل AND gates لعمل digital trigger.



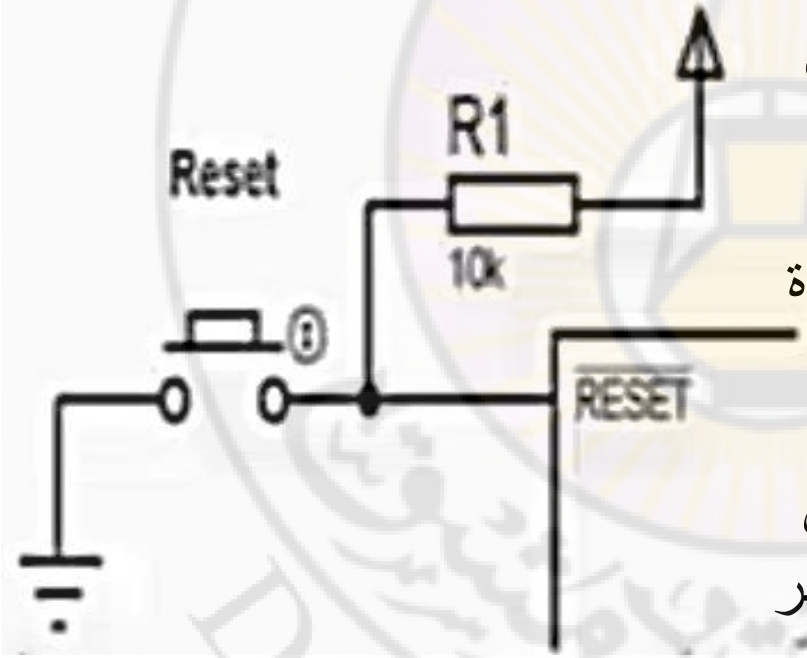




على أي حال، استخدام مكثف واحد فقط كافي لفلترة الذبذبات الكهربية الغير مرغوب فيها ولا داعي لبناء دوائر أكثر تعقيداً كما أنه أوفر من ناحية للتكلفة.

# قطب التصفير (إعادة التهيئة) RESET

يولد التصفير الخارجي عبر تطبيق نبضة كهربائية ذات منطق منخفض على القطب RESET تؤدي إلى :

- تصفير كل المسجلات الداخلية .
- تصفير عداد البرنامج ( \اي يبدا المعالج تنفيذ التعليمات من البداية ) .

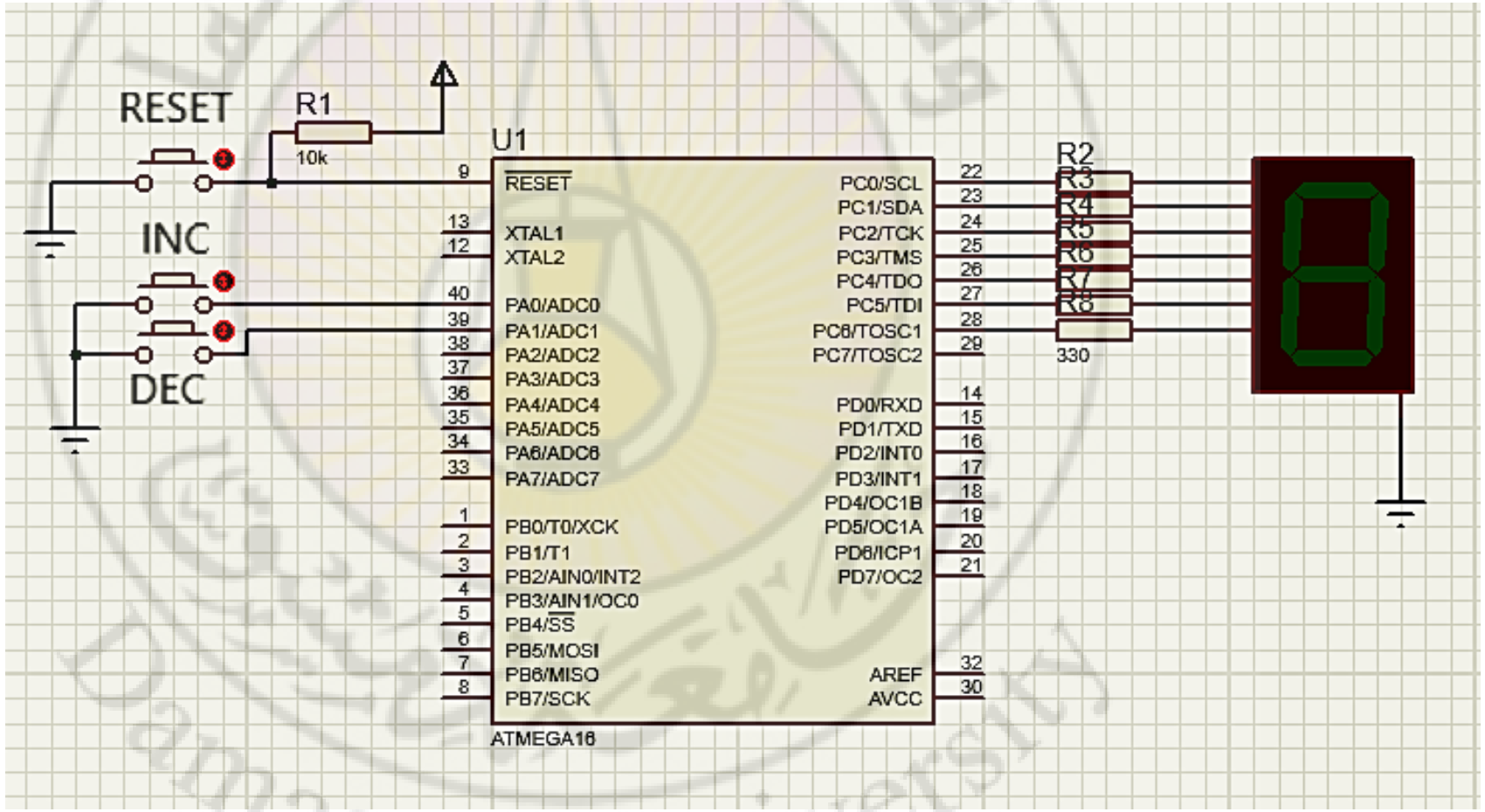



ملاحظة : إعادة التهيئة تكافئ فصل التغذية عن المتحكم وإعادة وصلها.

والشكل المجاور يوضح دارة التصفير وحسب الدارة يتم تفعيل القطب RESET عند الصفر منطقي أي عند الضغط يعطي صفر منطقي .

عادة مايتم وضع هذه الدارة بسبب معالجة الحالات غير المتوقعة التي يمكن ان يدخل بها الحاسب ويمكن الاستغناء عنها لكن لايمكن الاستغناء عن مقاومة الرفع

اكتب برنامج ليقوم بزيادة الرقم (عداد تزايد) على 7-seg بالضغط على INC
وبانقاص الرقم (عداد تناقص) بالضغط على DEC



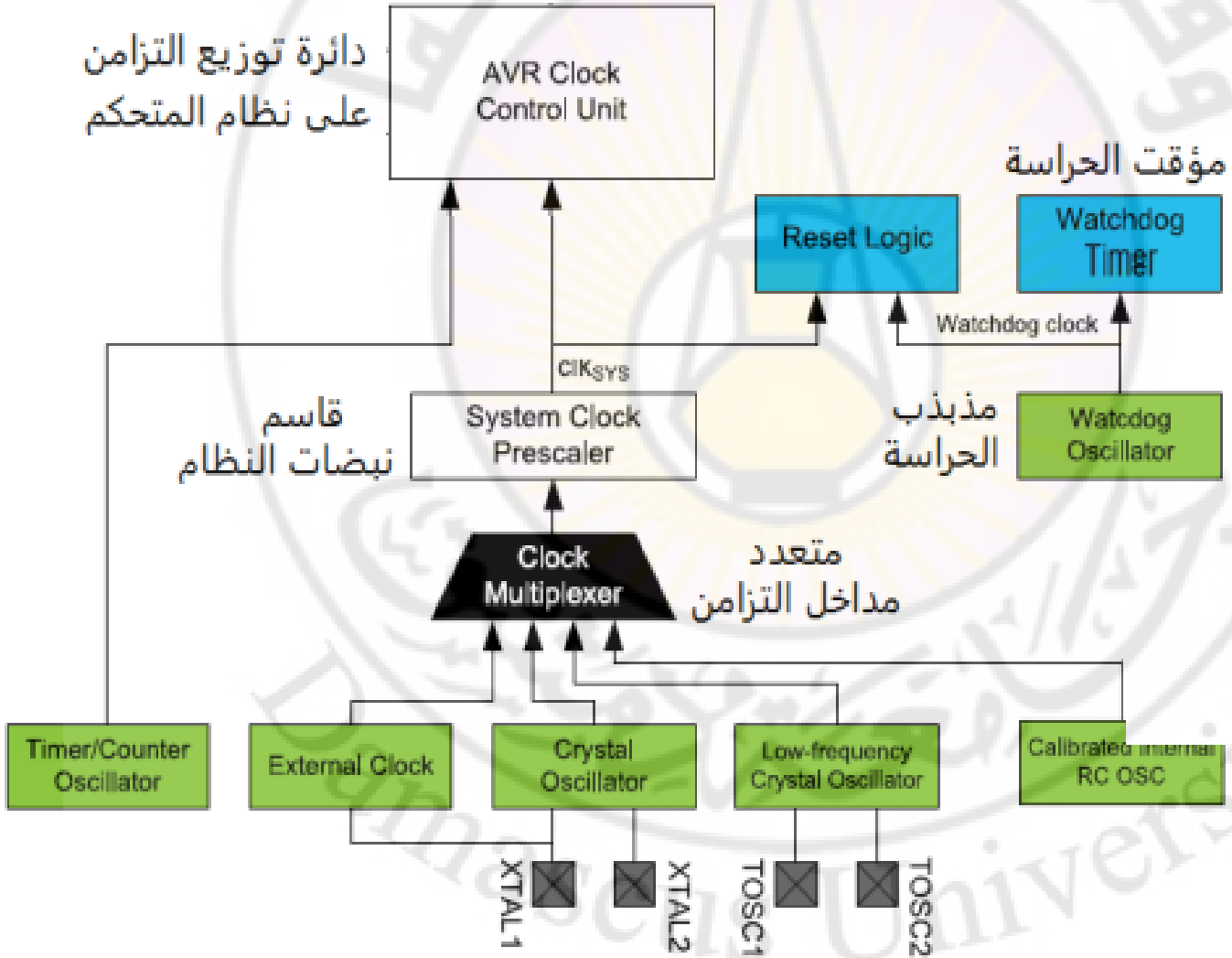
```
#include <mega16.h>
#include <delay.h>
char i=5;
char sevenseg_code[10]={0x3f,0x06,0x5B,0x4f,0x66,0x6d,0x7c,0x07,0x7f,0x6f};

void main(void)
{
DDRC=0xFF;
PORTA=0x03;
while (1)
    {
    if (PINA.0==0&&i<9)
   { i++;
    delay_ms(200);
   }
   if (PINA.1==0&&i>0)
      {i--;
          delay_ms(200);
      }
      PORTC=sevenseg_code[i];
    }
}
```

قمنا بتعريف متغيرين نوع char أحدهما مصفوفة للأعداد من 1 -9 وفق النظام الستعشري أي الهيكسا.

تم تعريف البورت C كمخرج ، البورت A افتراضيا هو دخل ولكن تم تفعيل مقاومة الرفع الداخلية لأول ثلاث بنات.
وبعد حلقتي الشرط وضعنا المصفوفة بدليل i الذي يعيد في كل عداد زيادة او نقصان الرقم المقابل في المصفوفة

# مصادر نبضات تزامن المتحكم المؤقتات

نعود لنكمل المؤقتات في المتحكمات بشكل عام وبالأخص متحكمات Atmega16 ، يبين الشكل المصادر المختلفة لنبضات التزامن في المتحكم AVR ، وهو يوضح بطريقة تفصيلية نظام نبضات الساعة في المتحكم .
كما ذكرنا هناك عدة مصادر : هزاز داخلي RC (1و2و4و8)MHZ ، وهزاز خارجي له نوعان بتردد عالي وتردد منخفض وبنفس طريقة التوصيل التردد المنخفض أقل من 1 ميغا هرتز والتردد العالي مابين (1-16) ميغا هرتز ، ومصدر آخر خارجي .




بالإضافة إلى Timer/Counter Oscillator أو مايسمى بكريستالة الساعة ، وهي ذات قيمة ثابتة بشكل دائم 32.768 KHZ .
يتم اختيار إحدى هذه المصادر عن طريق متعدد مداخل التزامن clock multiplexer الذي يتم التحكم به عن طريق الفيوز، وكما هو واضح من المخطط ان كل أجزاء المتحكم يتم تغذيتها من متعدد مداخل التزامن .
أما المزامن المراقب Watchdog فله نظام مختلف وهو مفصول عن نظام الساعة للمتحكم لأنه يراقب عمل المتحكم المصغر ، وسنتحدث عنه لاحقا

أما النبضات الخارجة من قاسم النبضات هي نبضات التزامن المستخدمة في كل نظام المتحكم مثل المحول التماثلي الرقمي ADC والمؤقتات ووحدة المعالجة المركزية.
وهناك مصدر نبضات خاص بالمؤقتات وهو يمثل النبضات الخارجية الداخلة لكل واحد من المؤقتات الثلاثة وهو (المؤقت 0 ، المؤقت 1 ، المؤقت 2 ) حيث يوجد طرف خاص بكل مؤقت تدخل منه النبضات المراد عدها عند استخدام المؤقت كعداد ، هذه النبضات لاتدخل على القاسم وانما تدخل مباشرة على وحدة توزيع نبضات التزامن.
اختيار مصدر نبضات التزامن التي سيعمل عندها المتحكم تتم من خلال بتات فيوزات تسمى بايتات الفيوزات ، كل بت من بتات هذه البايتات تقوم بوظيفة معينة عند تنشيطها أو برمجتها .

**أنماط عمل المؤقت**

1. **نمط العمل العادي**

في هذا النمط يعد العداد نبضات الدخل تصاعدياً حتى يصل إلى نهاية العد 0xFF حيث يعود إلى الصفر ويبدأ من جديد . عند الوصول إلى نهاية العد يقوم العداد بوضع 1 في العلم TOV0 ويمكن استخدام هذا العلم لتوليد مقاطعة. يستخدم هذا النمط لقياس فترة زمنية معينة أو عد أحداث خارجية، حيث يمكن قراءة المسجل TCNT0 لمعرفة العدد الحالي .

1. **نمط المقارنة والتصفير عند التطابق CTC**

في هذا النمط يقوم العداد بالمقارنة بين المسجل TCNT0 والمسجل OCR0 عند حدوث التطابق بين المسجلين يتم تصفير العداد والبدء من جديد ورفع العلم OCF0والذي يمكن استخدامه لتوليد مقاطعة. يستخدم هذا النمط لتوليد فواصل زمنية محددة حيث يستخدم المسجل OCR0 لتحديد الزمن المطلوب ويتميز هذا النط عن النمط العادي بسهولة استخدامه خاصة عند الحاجة لفواصل زمنية تكرارية.

1. **نمط PWM السريع**

يعد العداد بشكل تصاعدي من 0x00 حتى 0xFF . وعندما يصل إلى قيمة المسجل OCR0 يتم وضع 1 على قطب الخرج وعند الوصول إلى قمة العد يعود الخرج إلى القيمة 0 . أو بالعكس حسب إعدادات مسجل التحكم . وبالتالي يمكن تغيير عرض النبضة المولدة حسب قيمة OCR0 .

1. **نمط PWM مع تصحيح الطور**

يعد العداد بشكل تصاعدي من 0x00 حتى 0xFF ثم بشكل تنازلي حتى يعود إلى القيمة 0x00. يتم تغيير قيمة الخرج عند تطابق العداد مع OCR0 وذلك مرتان عند العد الصاعد وعند العد الهابط ويتم التغيير وفق مسجل التحكم. يتميز هذا النمط بأن الخرج لا يتغير طوره عند تغير قيمة OCR0وهذا أفضل عند التحكم بالمحركات.

# فيوزات المتحكم Atmega16

هذه الفيوزات هى فى الحقيقة عبارة مجموعة من بايتات ذاكرة القراءة فقط القابلة للقراءة والكتابة EEPROM، والتى من مميزاتها أنها لا تفقد محتوياتها بانقطاع القدرة عنها، وهذا بالطبع ضرورى لوضع هذه الفيوزات. هذه البايتات يكون لها عنوان خاص بكل منها مختلفا تماما عن عناوين الذاكرة EEPROM الأخرى الموجودة فى المتحكم والتى سبق

دائمة لا تتغير مع انقطاع الكهرباء عن المُتحكِم الدقيق، فمثلا مُتحكِمات الـ AVR تأتي بصورة افتراضية من المصنع تعمل بتردد 1 ميجاهرتز وقد نحتاج أن تغير هذه السرعة ليعمل المُتحكِم دائما بسرعة 16 ميجاهرتز، مثل هذه الإعدادات يجب أن تتم مرة واحدة ولا تتغير في المستقبل.. هنا يأتي دور الفيوزات Fuses.

الفيوزات هي وحدات ذاكرة (بتات Bits) يتم برمجتها بصورة مستديمة، هذا يعني أن محتواها لا يتغير بإعادة تشغيل المُتحكِم أو انقطاع الكهرباء أو حتى برمجة الذاكرة الرئيسية (Flash Memory)، هذه الفيوزات تحتفظ بمجموعة خاصة من الإعدادات المطلوبة وجودها بصورة مستديمة.

تنقسم الفيوزات في مُتحكِمات الـ AVR إلى 3 فيوزات رئيسية و هي:

Low Fuse Byte : يتكون من 8 بتات – كل 1 بت تتحكم في خاصية معينة في الـ AVR

HIGH Fuse Byte: أيضاً 8 بت - كل 1 بت تتحكم في مجموعة خصائص

Extended Fuse Byte: في معظم المُتحكِمات تكون أقل من 5 بتات

جميع البتات في الفيوزات الثلاثة السابقة لها وضعان من البرمجة

- **programmed = 0** الفيوز تم برمجته وتفعيل الخاصية التي يتحكم بها
- **unprogrammed = 1** الفيوز لم يُفعل وتم إلغاء الخاصية التي يتحكم بها

# فيوزات المتحكم Atmega16

جميع البتات في الفيوزات الثلاثة السابقة لها وضعان من البرمجة

- **programmed = 0** الفيوز تم برمجته وتفعيل الخاصية التي يتحكم بها
- **unprogrammed = 1** الفيوز لم يُفعل وتم إلغاء الخاصية التي يتحكم بها

يتم تفعيل أحد الأوضاع بكتابة 0 أو 1 في هذا الفيوز، مع ملاحظة أن صفر لا تعني unprogrammed بل العكس - حيث يمكن أن الفيوز يتم تفعيله programmed عندما تضع بداخله القيمة 0 ويتم إلغاؤه عندما تضع بداخله القيمة 1.

**تحذير: بعض الفيوزات إذا تم برمجتها بصورة خاطئة قد تتسبب في تعطيل المُتحكِم أو إلغاء إمكانية برمجته مرة أخرى إلا باستخدام مبرمجات خاصة High voltage programmers، لذا كن على حذر عند اختيار قيم الفيوزات وتأكد أكثر من مرة من جميع الإعدادات قبل برمجتها.**

يمكنك التعرف على تفاصيل الفيوزات وكيفة تفعيلها وذلك عبر فتح الـ Datasheet الخاصة بمتحكمات الـ AVR والبحث عن جداول Fuse High Byte , Fuse LOW Byte (موجودة في صفحة 260 للـ Datasheet في حالة المُتحكِم ATmega16).

## Fuse High Byte

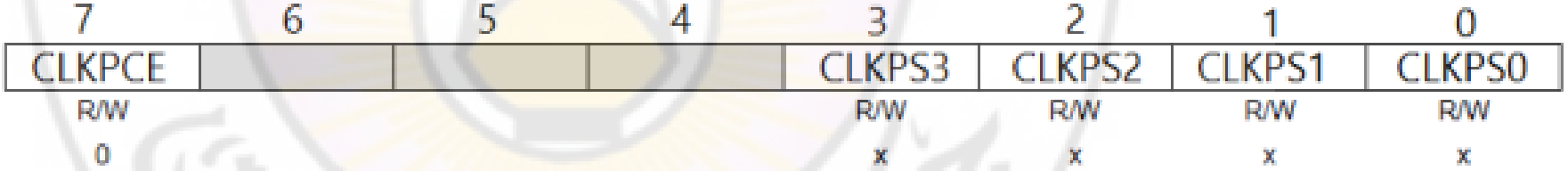
**Table 105.** Fuse High Byte

| Fuse High Byte | Bit No. | Description | Default Value |
|---|---|---|---|
| OCDEN[4] | 7 | Enable OCD | 1 (unprogrammed, OCD disabled) |
| JTAGEN[5] | 6 | Enable JTAG | 0 (programmed, JTAG enabled) |
| SPIEN[1] | 5 | Enable SPI Serial Program and Data Downloading | 0 (programmed, SPI prog. enabled) |
| CKOPT[2] | 4 | Oscillator options | 1 (unprogrammed) |
| EESAVE | 3 | EEPROM memory is preserved through the Chip Erase | 1 (unprogrammed, EEPROM not preserved) |
| BOOTSZ1 | 2 | Select Boot Size (see Table 100 for details) | 0 (programmed)[3] |
| BOOTSZ0 | 1 | Select Boot Size (see Table 100 for details) | 0 (programmed)[3] |
| BOOTRST | 0 | Select reset vector | 1 (unprogrammed) |

يوضح الجدول قيمة كل بت في جدول الفيوزات الـ High Byte مع القيمة الافتراضية لكل فيوز (هذه هي القيم التي تأتي مبرمجة مسبقاً من المصنع)، وكما نرى بعض الفيوزات تأتي

## مسجل معامل تقسيم نبضات التزامن :

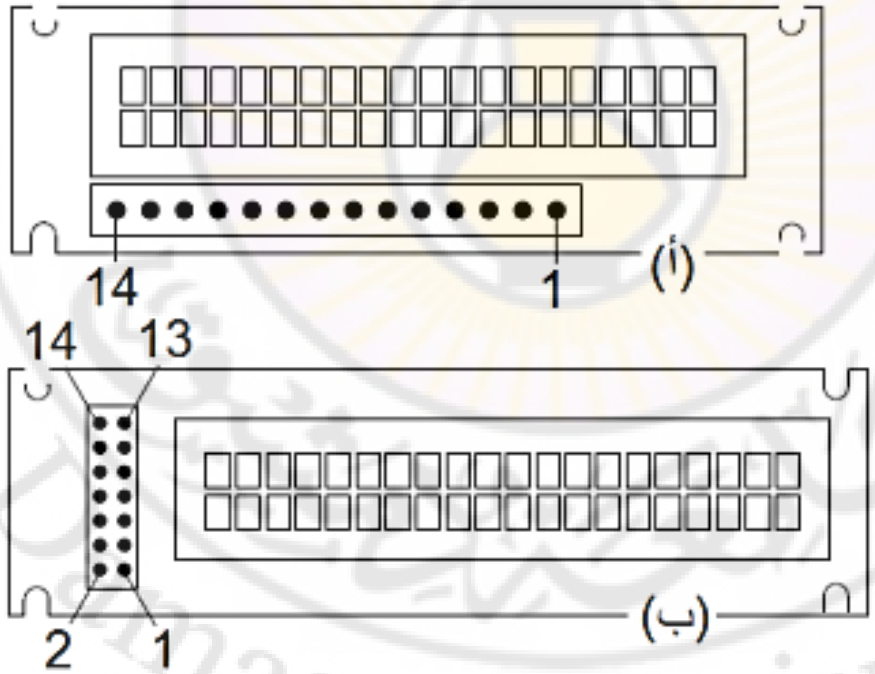
المسجل مكون من ثمان بتات مستخدم منها خمسة فقط ووظيفة كل بت مايلي :



# التعامل مع شاشات العرض البللورية LCD

شاشة الإظهار LCD: عبارة عن مصفوفة نقطية تستخدم لعرض المعلومات والنتائج حيث يمكن من خلالها عرض جميع رموز الآسكي .
وهي عبارة عن شاشة مؤلفة من سطر أو أكثر يحتوي كل سطر على عدد من الخانات ، والخانة عبارة عن مربع صغير يتم إظهار المحرف عليه ، أي أن كل خانة تستطيع إظهار محرف واحد فقط.

فى العادة تستخدم شاشات العرض البللورية Liquid Crystal Displays, LCDs فى الكثير من تطبيقات النظم الكامنة embedded systems التى تستخدم المتحكمات. من مميزات هذه الشاشات أنها صغيرة الحجم، ذات تكلفة معقولة، وسهولة تثبيتها فى أى مكان فى النظام الكامن أو المدمج. توجد هذه الشاشات فى نوعين، نوع خاص بعرض الحروف اللغوية، وتسمى الشاشات الأبجدية alphanumeric، وبالطبع المقصود هنا هو الحروف الإنجليزية، والنوع الثانى هو الشاشات الرسومية Graphical والتى يمكن بواسطتها عرض رسومات معينة يمكن من خلالها عرض الأحرف العربية.




بالنسبة لسوكيت التوصيل لهذه الشاشات فهى توجد فى نوعين وكلا النوعين يتكون من ١٤ نقطة توصيل قد تصل إلى ١٦ نقطة فى بعض أنواع الشاشات. النوع الأول يحتوى كل النقاط مرصوصة فى صف واحد، والنوع الثانى يحتوى هذه النقاط موضوعة فى صفين.

Damascus University

# وظائف الأطراف المختلفة لشاشات العرض البللورية

| رقم الطرف | رمزه | وظيفته |
|---|---|---|
| ١ | Vss | الأرضى |
| ٢ | Vcc | جهد مصدر القدرة، ٥ فولت |
| ٣ | Vee | جهد التحكم فى التباين contrast |
| ٤ | RS | إختيار مسجل البيانات أو مسجل الأوامر، Register Select<br>0= مسجل الأوامر، 1= مسجل البيانات |
| ٥ | R/W | القراءة أو الكتابة، 0= كتابة، 1= قراءة من الشاشة |
| ٦ | E | طرف تنشيط enable عالى الفعالية، أى بوضع بواحد ليتم التنشيط |
| ٧ | DB0 | الطرف 0 فى مسار البيانات |
| ٨ | DB1 | الطرف 1 فى مسار البيانات |
| ٩ | DB2 | الطرف 2 فى مسار البيانات |
| ١٠ | DB3 | الطرف 3 فى مسار البيانات |
| ١١ | DB4 | الطرف 4 فى مسار البيانات |
| ١٢ | DB5 | الطرف 5 فى مسار البيانات |
| ١٣ | DB6 | الطرف 6 فى مسار البيانات |
| ١٤ | DB7 | الطرف 7 فى مسار البيانات |
| ١٥ | A | الأنود أو طرف الموجب للتحكم فى شدة الإضاءة الخلفية (إختيارى) |
| ١٦ | K | الكاثود أو الطرف السالب للتحكم فى شدة إضاءة الخلفية (إختيارى) |

## أنماط عمل شاشة LCD

تعمل وفق نمطين

# أقطاب التحكم لشاشة LCD :

**1- القطب VSS :**

هو قطب التغذية لشاشة الـ LCD ،
وهو جهد الأرضي (0) منطقي ( GND ) .

**2 - القطب VDD :**

هو أيضاً قطب التغذية لشاشة الـ LCD ،
ولكن ذو القيمة ( +5V ) .

**3 - القطب VEE :**

ويدعى أيضاً بـ VO وهو قطب جهد التباين ،
ويقصد بالتباين هو حدة ظهور الرمز على
الشاشة وأقل تباين أن لا نرى شيئاً على الشاشة
يكون عند تطبيق ( +5V ) على هذا القطب ، وأعلى تباين للشاشة يكون عند تطبيق ( 0 V )
على هذا القطب .ويمكن التحكم بتباين الشاشة عن طريق وصل قطب التباين ( VO ) إلى
مقاومة متغيرة .

Activate
Go to Settin

**4 - القطب RS :**

وهو مسجل اختيار الدخل لشاشة الـ LCD وذلك في حال طبق عليه (0) منطقي عندها نريد إرسال كلمة تحكم ، بينما في حال طبق (1) منطقي فعندها نريد إرسال معطيات إلى الشاشة ، وفي الحالتين عن طريق أقطاب المعطيات .

**5 - القطب R/W :**

وهو للقراءة أو الكتابة إلى الشاشة ، نطبق (0) منطقي على هذا القطب عندما نريد كتابة ( إرسال ) المعلومات إلى شاشة الـ LCD ، ونطبق (1) منطقي عندما نريد قراءة ( استقبال ) المعلومات من شاشة الـ LCD ، وعادةً ما يوصل هذا القطب إلى الأرضي في حال لم يكن هناك داعٍ للقراءة من الشاشة .

**6 - القطب E :**

وهو قطب تمكين شاشة الـ LCD ، فكل معلومة يتم كتابتها أو قراءتها من شاشة الـ LCD يجب إرفاقها بنبضة تمكين على هذا القطب ، ونبضة التمكين هذه تحدث عند الجبهة الهابطة ، أي تتم هذه النبضة برفع القطب إلى الـ (1) منطقي وإنزاله إلى الـ (0) منطقي بعد تأخير مناسب .

Activate
Go to Settin

**7 - الأقطاب DB0 . . . DB7 :**

وهي أقطاب المعطيات ( DATA ) ،
حيث يتم كتابة المعطيات أو كلمات التحكم عبر هذه الخطوط إلى شاشة الـ LCD وكذلك قراءة المعطيات منها . . .

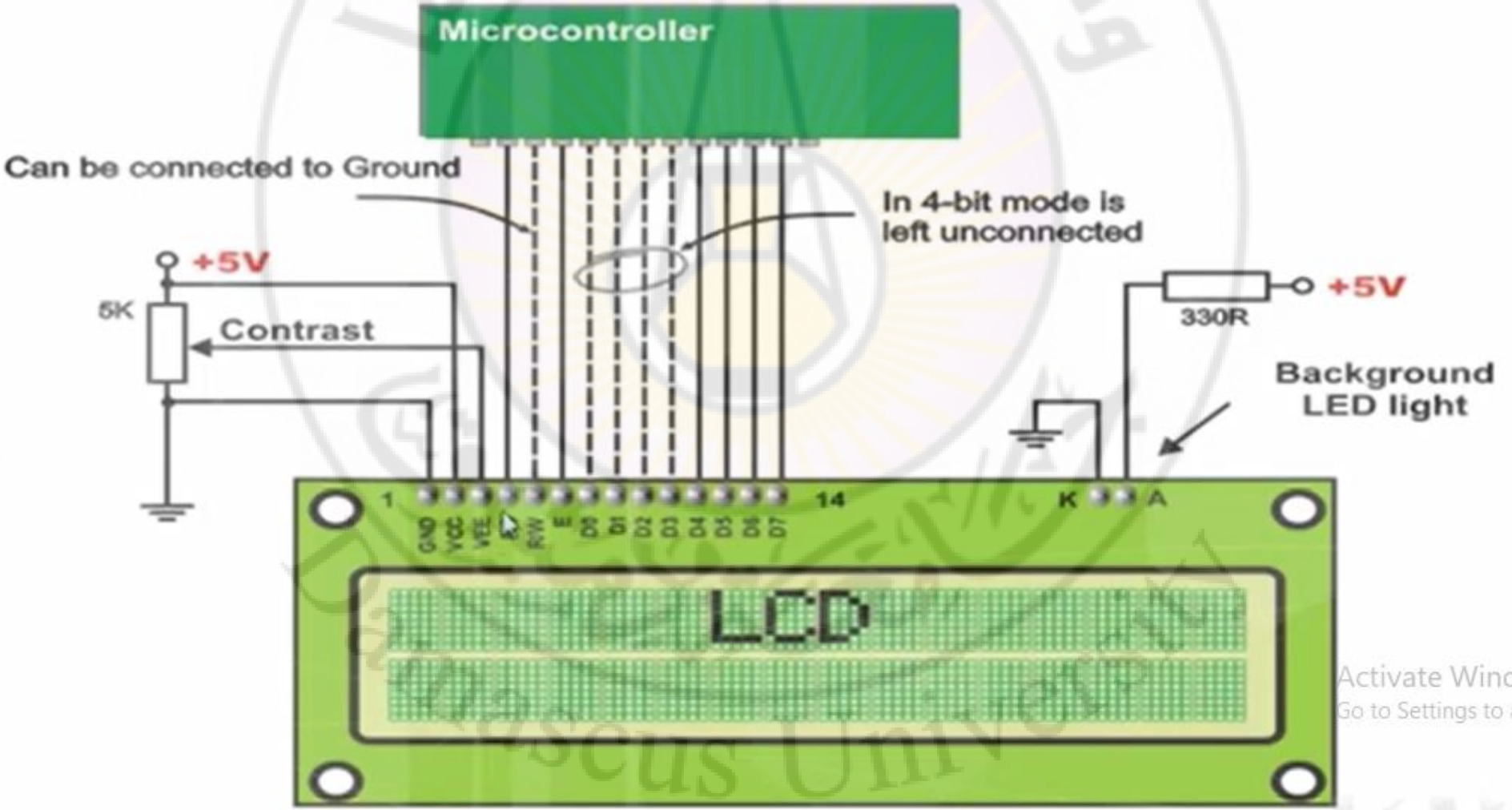
**8 - القطبين K و A :**

تزود بعض الشاشات بهذين القطبين ،
وهما على الترتيب قطبي المهبط و المصعد للّيد المسطّح المستطيل الموجود خلف شاشة الـ LCD ،
والذي يضاء عند تطبيق الـ (1) منطقي على المصعد ( A ) و الـ (0) منطقي على المهبط ( K ) .

وظيفة هذا الليد هو تأمين الإضاءة الكافية لشاشة الـ LCD ليتمكن المستخدم من رؤية العبارات المكتوبة عليها في ظلمة الليل . . .

Activate Windows
Go to Settings to activate Windows.

# طريقة التوصيل LCD مع Microcontroller :




Activate Windows
Go to Settings to activate Windows.

# تداول البيانات داخل متحكمات AVR

# أنواع البيانات في الأنظمة المدمجة:

البيانات الرقمية : تعتبر اكثر الأنواع استخداما في عالم الأنظمة المدمجة فيه تعبر عن قيم المتغيرات مثل قراءة الحساسات او قيم المخارج مثل سرع المحركات

| نوع البيانات | استهلاك الذاكرة (عدد Bytes) | أقصى قيمة يمكن وضعها داخل المتغير |
|---|---|---|
| int | 2-bytes (16 bit) | -32,768 → +32,767 |
| unsigned int | 2-bytes (16 bit) | 65,535 |
| float | 4-bytes (32 bit) | 1.2E-38 to 3.4E+38 |
| double | 8-bytes (32 bit) | 2.3E-308 to 1.7E+308 |
| long | 4-bytes (32 bit) | -2,147,483,648 → +2,147,483,647 |
| unsigned long | 4-bytes (32 bit) | 4,294,967,295 |
| long long | 8-bytes (64 bit) | $-9.223372037\times10^{18}$ $+9.223372037\times10^{18}$ |
| unsigned long long | 8-bytes (64 bit) | $1.844674407\times10^{19}$ |

ملاحظة: نوعي البيانات float و double دائماً يكونا من نوع Signed ولا يمكن استخدام العلامة unsigned معهما

جدول البيانات التقليدية

البيانات المعيارية ANSI C - C99 وهي صورة محسنة لأنواع البيانات وتمكننا من اختيار قيمة المتغيرات بالدقة والطول المطلوب دون أي اهدار للذاكرة.

| Signed | أقصى قيمة | Unsigned | أقصى قيمة |
|---|---|---|---|
| int8_t | -128 → +127 | uint8_t | 0 → 255 |
| int16_t | -32,768 → +32,767 | uint16_t | 0 → 65,535 |
| int32_t | -2,147,483,648 → +2,147,483,647 | uint32_t | 0 → 4,294,967,295 |
| int64_t | $-9.223372037\times10^{18}$ $+9.223372037\times10^{18}$ | uint64_t | $1.844674407\times10^{19}$ |

جدول البيانات المعيارية

## أمثلة

**ما هو نوع المتغير المطلوب للقراءة أو الكتابة في أي بورت (مثل port A, B,C)؟**

- بما أن جميع البورتات يتم قراءتها من خلال المُسجِل PINx والذي يكون 8 بت وكذالك يتم كتابة في أي بورت باستخدام المُسجِل PORTx والذي يعتبر 8 بت أيضاً إذا أفضل نوع بيانات هو **uint8_t** حيث يمكنه تخزين الأرقام من 0b0000000 إلى 0b11111111

**ما هو المتغير المناسب لتخزين عداد رقمي من صفر إلى 10,000؟**

- بما أن الرقم لا يحتوي على أرقام سالبة وأقصى رقم مطلوب هو 10000 وهذا الرقم أكبر من 255 لذا لا يمكن استخدام متغير بمساحة 8 بت ويجب أن نستخدم متغير بمساحة 16 بت والذي يستطيع التعامل مع ارقام تصل إلى 65,535. إذا أفضل نوع بيانات هو **uint16_t**

**ما هو المتغير المناسب لتخزين درجة حرارة تتراوح بين سالب 50 إلى موجب 100؟**

- هناك إجابتين لهذا السؤال، الأولى: في حالة أننا نريد تخزين أرقام صحيحة فقط مثل 20 أو 50 أو أي رقم بدون كسر عشري فسنجد أن افضل نوع بيانات هو **int8_t** حيث يستطيع هذا النوع من البيانات أن يخزن أي قيمة تتراوح بين -128 إلى + 127 ويستهلك 1 بايت فقط من مساحة الذاكرة.
- الثانية: في حالة وجود كسر عشري مثل أن 25.1 أو 30.5 يجب أن نستخدم نوع البيانات **float** مع العلم أنه في المقابل سيتم استهلاك مساحة ذاكرة = 4 بايت (32 بت).

## مجموعة التعليمات

يمتلك AVR مجموعة تعليمات غنية مؤلفة من 131 تعليمة. تقسم هذه التعليمات إلى أربع مجموعات : التعليمات الحسابية ، تعليمات البتات ، تعليمات التفرع وتعليمات نقل البيانات.

## تعليمات التحميل ونقل البيانات

تقوم هذه التعليمات بنقل البيانات بين المسجلات أو بين مسجل والذاكرة وبالعكس أو بين مسجل ومسجل خاص وبالعكس أو تحميل قيمة فورية في مسجل ما.

**ملاحظات :**

1. التعليمة LDI ( وجميع التعليمات المنتهية بالحرف I ) تحمل قيمة فورية في أحد المسجلات من R16 حتى R31 ولا يمكن استخدام المسجلات الأخرى لتحميل القيم الفورية.
2. تختلف تعليمات النقل من وإلى الذاكرة بنمط العنونة ، أي بطريقة حساب عنوان الحجرة المقصودة.
3. التعليمات OUT و IN تتعامل مع فضاء عنونة الدخل والخرج وتستخدم مع المسجلات الخاصة SFR.
4. التعليمات LPM و SPM تتعاملان مع ذاكرة البرنامج ROM .

لنعود إلى المثال السابق ونقوم بحله بطريقة أخرى :

توصيل 8 دايودات ضوئية على المتحكم وكتابة برنامج يقوم بإخراج القيم من 0 إلى 255 على البورت PORT بتأخير زمني بين كل قيمة 100 ميللي ثانية :

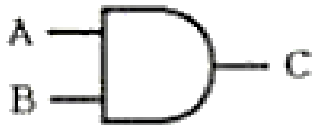
الكود البرمجي : لاحظ ان PORTC خرج

```
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTB=(0<<PORTB7) | (0<<PORTB6) | (0<<PORTB5) | (0<<PORTB4) | (0<<PORTB3) | (0<<PORTB2) | (0<<PORTB1) | (0<<PORTB0);

// Port C initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) | (1<<DDC2) | (1<<DDC1) | (1<<DDC0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);

for (counter=0;counter<=200;counter++)
    {
    PORTC=counter;
    delay_ms(100);
    }
}
```

# العمليات المنطقية

| | |
|---|---|
| OR | \| |
| AND | & |
| NOT | ~ |
| XOR | ^ |
| Shift Left | << |
| Shift Right | >> |

AND




| Inputs | | Output |
|---|---|---|
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

أمثلة على AND

0 & 0 = 0

0 & 1 = 0
1 & 0 = 0
1 & 1 = 1

OR

| Inputs | | Output |
|---|---|---|
| A | B | C |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

أمثلة على OR

0 | 0 = 0

0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

NOT

| Input | Output |
|---|---|
| A | C |
| 0 | 1 |
| 1 | 0 |

أمثلة على NOT

~ 0 = 1
~ 1 = 0

Exclusive-OR gate

| A | B | Output |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

أمثلة على XOR

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

# عمليات الإزاحة Shift Operation:

وهي عملية تحريك البتات إلى اليمين أو اليسار داخل متغير أو مسجل أو أي قيمة رقمية . تعد هذه العمليات من أهم الأوامر التي تستخدم في برمجة المعالجات والمتحكمات الصغرية

| 0b00000101 | الرقم الأصلي |
|---|---|
| 0b00001010 | إزاحة بمقدار 1 بت جهة اليسار << |
| 0b00010100 | إزاحة بمقدار 2 بت جهة اليسار << |
| 0b00101000 | إزاحة بمقدار 3 بت جهة اليسار << |

**مثال:** لنفرض أن لدينا 0b01100000 لنقم بتطبيق عمليات الإزاحة لجهة اليمين.

| 0b01100000 | الرقم الأصلي |
|---|---|
| 0b00110000 | << إزاحة بمقدار 1 بت جهة اليمين |
| 0b00011000 | << إزاحة بمقدار 1 بت جهة اليمين |
| 0b00001100 | << إزاحة بمقدار 1 بت جهة اليمين |

## التعامل مع بتات محددة داخل أي متغير :

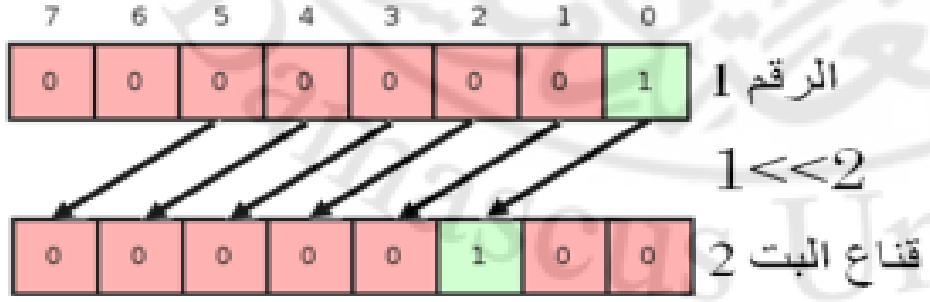
بفرض لدينا المتغير a=11110000 والمطلوب جعل البت رقم 1 يساوي القيمة 1 بدلا من 0 ، أي نريد جعل المتغير يصبح a=11110010 ، يمكن ذلك من خلال الأمر a=0B11110010 أو باستخدام القناع أو الإزاحة .

## القناع mask لأي بت :

القناع للبت رقم واحد هو (00000010) وهو رقم كله أصفار ماعدا البت رقم 1 يوضع بالقيمة (1) . بنفس الطريقة فإن قناع البت رقم 5 سيكون (00100000) أي أن كل البتات تساوي أصفار ماعدا البت رقم 5 .

وبهذا فإن قناع أي بت رقمه n سيكون رقم بتاته كلها أصفار ماعدا البت رقم n .

مثال الشكل التالي : لدينا الرقم الستعشري 0x01=00000001 بإجراء إزاحة ناحية اليسار بمقدار 2 بت على بتات الرقم 1 يصبح الناتج (00000100) . وهذه العملية يمكن التعبير عنها بلغة C كالتالي 1<<2 ، حيث ان الرمز << يعني إزاحة ناحية اليسار والرقم على يساره هو الرقم المراد إزاحته ، والرقم على يمين الرمز هو مقدار الإزاحة بالبتات، وبالتالي فإن 1<<2 يعني إزاحة بتات الرقم 1 ناحية اليسار بمقدار 2 بت . وعلى ذلك فإن قناع أي بت يمكن الحصول عليه بإزاحة الرقم 1 ناحية اليسار بمقدار n من البتات باستخدام التعبير (1<<n)



لو تم إجراء عملية OR المنطقية على محتويات الرقم a=11110000 مع قناع البت رقم 1 الذي نحصل عليه بإزاحة الرقم 1 بمقدار بت واحدة وهو 00000010 ، فإن الناتج سيكون 11110010.
وهو نفس الرقم a مع تغير البت رقم 1 فقط من 0 إلى 1 .

يمكن التعبير عن ذلك بالأمر التالي : ;(1<<1)=|a والذي يعني جعل الرقم a يساوي نفسه (a) بعد إجراء عملية OR على محتوياته مع ناتج إزاحة الرقم 1 بمقدار 1 بت .

التعبير (1<<2)=|a يكون كالتالي a تساوي a أور مع الرقم 1 المزاح بمقدار 2 بت .
وبالتالي فإن التعبير (1<<0)=|a سيكون ناتجه a=11110001 .
والتعبير (1<<3)=|a سيكون ناتجه a=11111000 ،
والتعبير (1<<5)=|a سيكون a=11110000 (أي لاتغيير) حيث أن 1 اور 1 هو 1 .
لجعل أي بت في أي رقم يساوي الواحد ، أي عمل setting لهذا البت فإننا نجري عملية OR على الرقم مع قناع mask هذه البت ..

إذا أردنا تصفير resetting أي بت من بتات أي رقم بدون تغيير بتاته الأخرى ، نقوم بإجراء عملية AND على محتويات الرقم مع معكوس قناع هذه البت .

مثلا : لدينا الرقم a=11110000 والمطلوب تصفير البت رقم 5 ، قناع البت رقم 5 هو 00100000 ومعكوس هذا القناع هو 11011111 وبإجراء عملية آند على هذا المعكوس مع الرقم a نحصل على الرقم a الجديد وهو a=11010000 ، بعد أن تم تصفير البت رقم 5 فيه . الأمر الذي سيقوم بهذه العملية هو

a&=~(1<<5) والذي ينطق a تساوي a بعد عمل آند لها مع معكوس قناع البت رقم 5.

**مثال برمجي :**

بالعودة إلى مثال تشغيل ثماني دايودات ضوئية باستخدام نظام العداد في حلقة for ، لاحظنا أثناء تنفيذ الكود على برنامج المحاكاة أن تشغيل وإطفاء الليدات لم يكن منتظما بالترتيب كما كان في الكود الطويل التقليدي الأول الذي قمنا فيه بتشغيل وإطفاء كل قطب PIN على سطر . ولحل هذه المشكلة سوف نعتمد نظام الإزاحة كما يلي :

قمنا بتعديل الكود البرمجي كما يلي :

```
// Port C initialization
// Function: Bit7=Out Bit6=Out Bit5=Out Bit4=Out Bit3=Out Bit2=Out Bit1=Out Bit0=Out
DDRC=(1<<DDC7) | (1<<DDC6) | (1<<DDC5) | (1<<DDC4) | (1<<DDC3) | (1<<DDC2) | (1<<DDC1) | (1<<DDC0);
// State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);

// Port D initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
// State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
while(1)
{
for (counter=0;counter<=255;counter++)
    {
    PORTC=(1<<counter);
    delay_ms(100);
    }
}
```

نتيجة تنفيذ الكود بدأت الديودات بالإضاءة واحدة تلو الأخرى بالترتيب وذلك باستخدام متغير counter لزيادة قيمة الإزاحة بصورة تلقائية وذلك ضمن حلقة while وباستخدام المحاذاة أصبحت الإضاءة للديودات بالترتيب واحد تلو الآخر

# تعديل الكود باستخدام حلقتي for لعكس اتجاه الإضاءة

```
50  // State: Bit7=0 Bit6=0 Bit5=0 Bit4=0 Bit3=0 Bit2=0 Bit1=0 Bit0=0
51  PORTC=(0<<PORTC7) | (0<<PORTC6) | (0<<PORTC5) | (0<<PORTC4) | (0<<PORTC3) | (0<<PORTC2) | (0<<PORTC1) | (0<<PORTC0);
52
53  // Port D initialization
54  // Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
55  DDRD=(0<<DDD7) | (0<<DDD6) | (0<<DDD5) | (0<<DDD4) | (0<<DDD3) | (0<<DDD2) | (0<<DDD1) | (0<<DDD0);
56  // State: Bit7=T Bit6=T Bit5=T Bit4=T Bit3=T Bit2=T Bit1=T Bit0=T
57  PORTD=(0<<PORTD7) | (0<<PORTD6) | (0<<PORTD5) | (0<<PORTD4) | (0<<PORTD3) | (0<<PORTD2) | (0<<PORTD1) | (0<<PORTD0);
58  while(1)
59  {
60   for (counter=7;counter>=0;counter--)
61          {
62          PORTC=(1<<counter);
63          delay_ms(200); }
64          for (counter=0;counter<=7;counter++)
65              {
66              PORTC=(1<<counter);
67              delay_ms(200);
68              }
69          }
70      }
71
```

# عكس بت محددة Toggle Bit

تعد الصيغة البرمجة Toggle Bit من أفضل الصيغ المستخدمة في البرمجة وتعني أنه في حالة أن البت المطلوبة = 1 قم بعكسها إلى 0 وإذا كانت تساوي 0 قم بعكسها إلى 1. وتكون الصيغة القياسية كالتالي:

```
Resgister ^= (1 << BitName);
```

باستخدام هذه الصيغة يمكننا اختصار المثال blinking led بأمرين فقط (بدلاً من 4 أوامر).

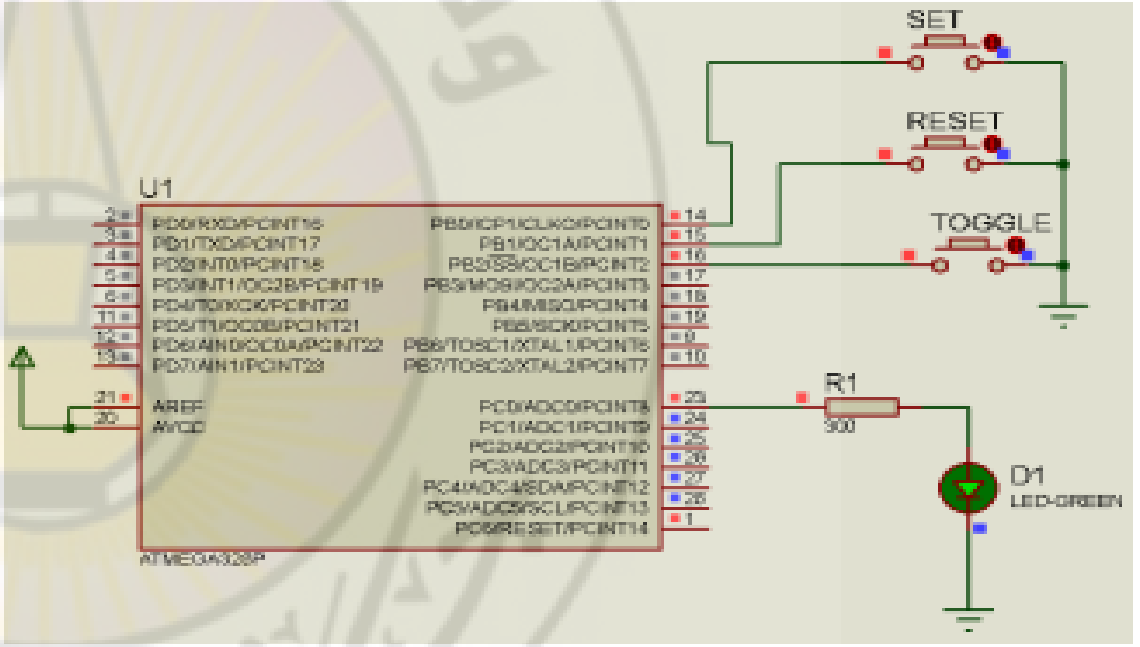
```
57
58    while (1)
59          {
60          // Place your code here
61          if (PINC.0==1){
62          PORTA^=(1<<PORTA.0);
63          delay_ms(100);
64          }
```

باستخدام 4 أوامر

```
57
58    while (1)
59          {
60          // Place your code here
61          if (PINC.0==1){
62          PORTA.0=1;
63          delay_ms(100);
64          PORTA.0=0;
65          delay_ms(100);
66          }
67
68          }
```

# التحكم فى دايود ضوئى من خلال ثلاث مفاتيح

سنحاول فى هذا المشروع التدريب على التعامل مع بت بعينها فى أحد المتغيرات من خلال الإزاحة. سنجعل البوابة C بوابة خرج مع وضع دايود ضوئى ومقاومة على المخرج الأول منها PC0 بحيث يضىء عند إخراج واحد على هذا الطرف. فى المقابل سنجعل البوابة B بوابة دخل مع وضع ثلاث مفاتيح على المداخل الثلاثة الأولى فيها، PB0 و PB1 و PB2. المطلوب عند الضغط على المفتاح الأول أن يضىء الدايود الضوئى، وإن كان مضيئا فى الأصل، يبقى كما هو، ولذلك سنسميه مفتاح الوضع SET. المفتاح الثانى عند ضغطه يطفىء الدايود الضوئى، وإن كان مطفأ فى الأصل يظل كما هو، ولذلك سنسمي مفتاح إعادة الوضع RESET. المفتاح الثالث عند الضغط عليه نريده أن يغير وضع الدايود الضوئى، فإن كان مطفأ يضيئه، وإن كان مضيئا يطفؤه، ولذلك سنسميه مفتاح تغيير الوضع TOGGLE. البرنامج سيكون كالتالى،



الكود البرمجي :

```
while (1)
    {
    // Place your code here
     if ((PINB&(1<<0))==0)
       {
       PORTC|=(1<<0);
       }
     if ((PINB&(1<<1))==0)
       {
        PORTC&=~(1<<0);
       }
     if ((PINB&(1<<2))==0)
       {
       PORTC^=(1<<0);
       }
    }
}
```

بعد تفعيل مقاومات الرفع الداخلية تم تنشيط حلقة لانهائية تحتوي ثلاث أوامر شرطية ، الأول(0==((PINB&(1<<0)) عملية آند على البوابة B مع الرقم 00000001 وهو (1<<0) والعملية تساوي الصفر عند ضغط المفتاح تكون البوابة صفر ويكون الخرج على PORTC هو واحد ;(1<<0)=|PORTC

الشرط الثاني (0==((PINB&(1<<1)) if يقوم بإجراء آند للبوابة B مع 00000010 وهو ناتج (1<<1) ليتاكد إذا كان المفتاح الثاني تم ضغطه أم لا بحيث أذا كان مضغوط يتم تصفير الطرف PC0 وبالتالي إطفاء الليد .

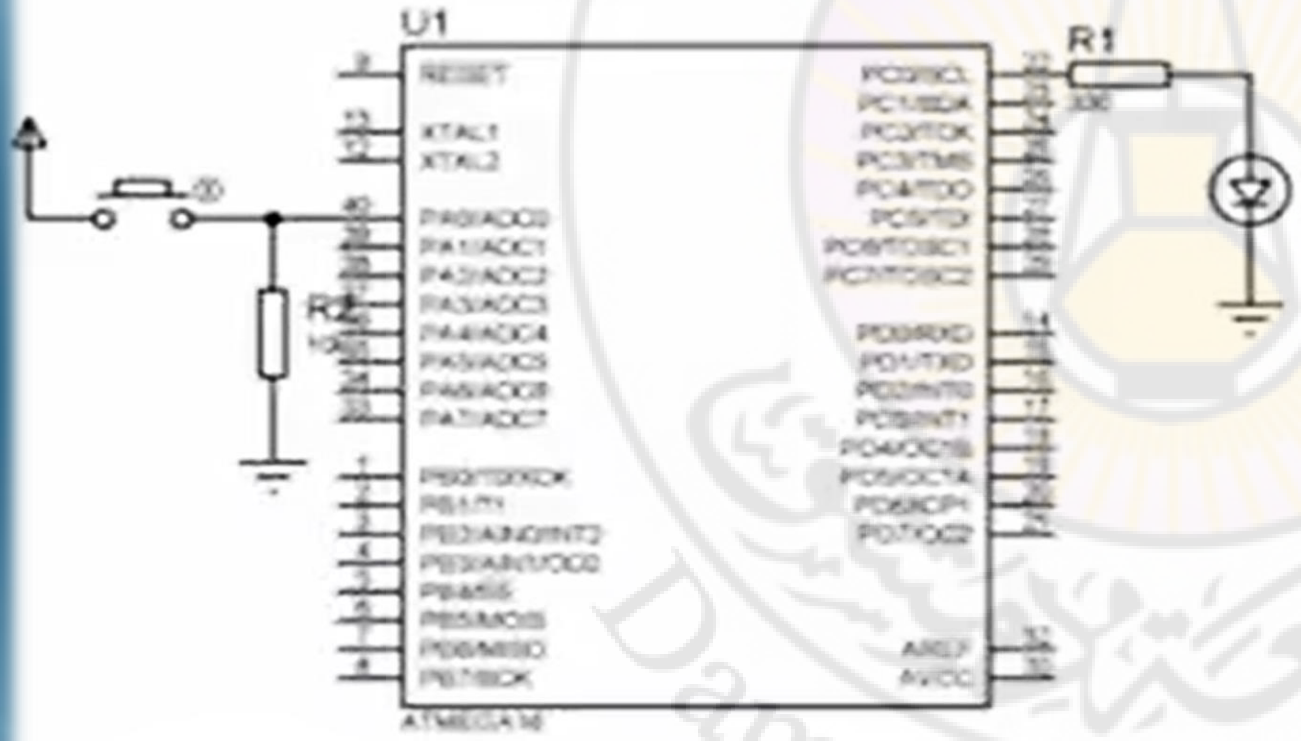
بنفس الطريقة الشرط الثالث

# قراءة الدخل الرقمي

# Inputs Reading

# تطبيق على استخدام PORT كدخل

اكتب برنامج يقوم بتشغل LED لمدة 1 s وإطفاء LED لمدة 1 s .... وهكذا يستمر بالعمل إذا تم ضغط button.




```
#include <mega16.h>
#include <delay.h>
void main(void)
{
DDRC.0=1;
Start:
if (PINA.0==1)
   {   PORTC.0= 1;
      delay_ms(1000);
       PORTC.0= 0;
      delay_ms(1000);}
goto Start;
```

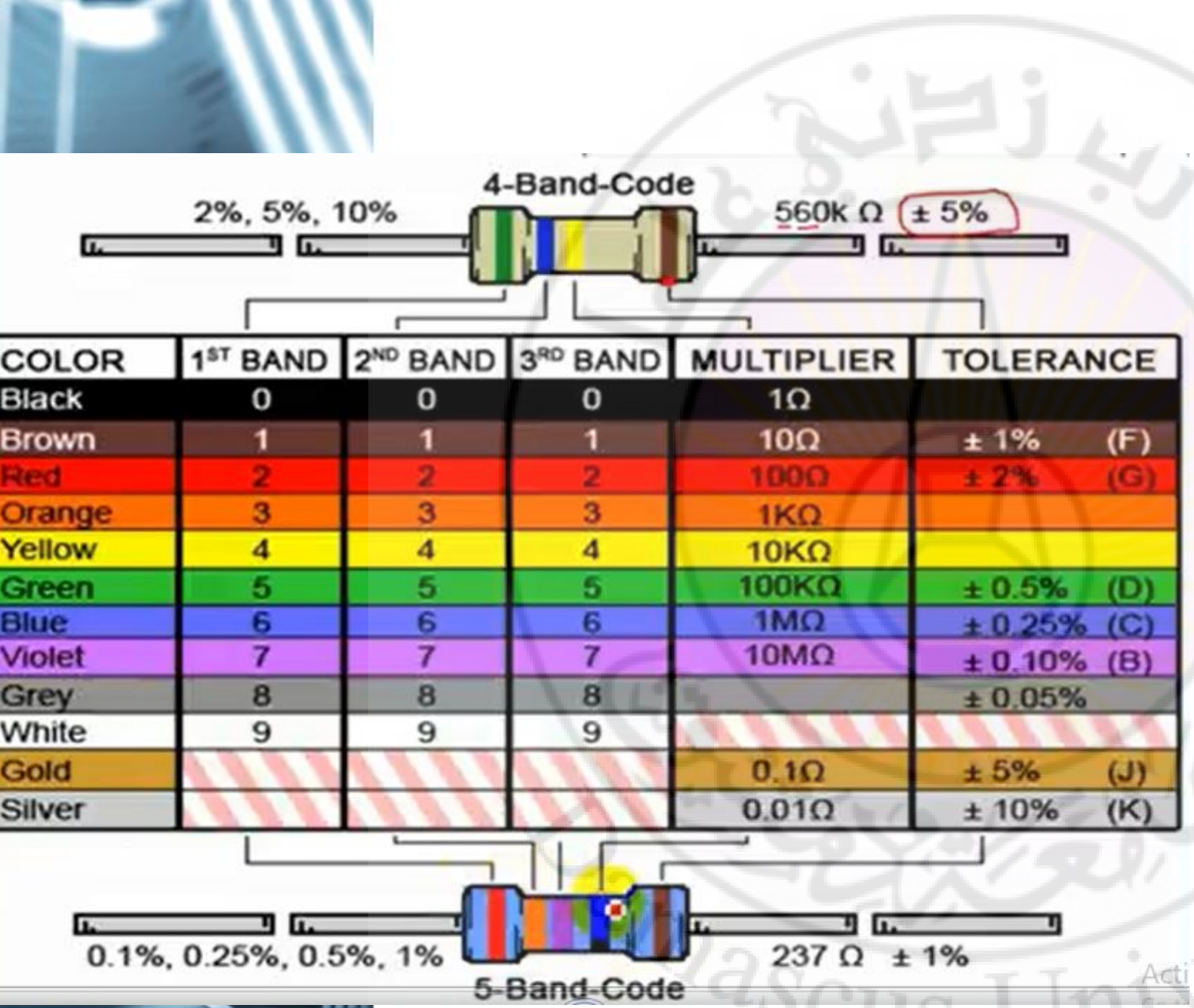
قراءة حالة PINA

قبل البدء سنتذكر أنواع وأهمية استخدام المقاومات في الدارات .

ان الاستعمالان الرئيسيان للمقاومة هما:

- تقسيم الجهد (مقسم للجهد)
- التحكم في شدة التيار لمراعاة شروط خروج ودخول التيار إلى المتحكم الصغري.

ويمكن قياسها إما عن طريق الآفو أو الألوان كما يلي :




| COLOR | 1ST BAND | 2ND BAND | 3RD BAND | MULTIPLIER | TOLERANCE |
|---|---|---|---|---|---|
| Black | 0 | 0 | 0 | 1Ω | |
| Brown | 1 | 1 | 1 | 10Ω | ± 1% (F) |
| Red | 2 | 2 | 2 | 100Ω | ± 2% (G) |
| Orange | 3 | 3 | 3 | 1KΩ | |
| Yellow | 4 | 4 | 4 | 10KΩ | |
| Green | 5 | 5 | 5 | 100KΩ | ± 0.5% (D) |
| Blue | 6 | 6 | 6 | 1MΩ | ± 0.25% (C) |
| Violet | 7 | 7 | 7 | 10MΩ | ± 0.10% (B) |
| Grey | 8 | 8 | 8 | | ± 0.05% |
| White | 9 | 9 | 9 | | |
| Gold | | | | 0.1Ω | ± 5% (J) |
| Silver | | | | 0.01Ω | ± 10% (K) |

# حساب المقاومة المستخدمة من قبل الأحمال:

الحِمل Load هو أي جهاز يتصل بخرج المُتحكِم الدقيق مثل الدايود الضوئي، في الأمثلة السابقة لاحظنا وجود مقاومة قبل كل دايود بقيمة (330 أوم)، لماذا استخدمنا هذه المقاومة وبتلك القيمة تحديداً؟

هناك سببان لهذا الأمر. الأول هو أن الدايود الضوئي (خاصة اللون الأحمر) يفضل أن يتم تشغيله بتيار كهربي ما بين 15 إلى 20 مللي أمبير عند تطبيق فرق جهد 5 فولت. وبالتعويض في قانون أوم الكهربي (فرق الجهد 5 فولت - التيار 15 مللي أمبير)

$$Voltage(V) = Current(I) \times Resistance(R)$$

**نجد أن المقاومة المناسبة هي 330 أوم .**

إذا لم تستخدم هذه المقاومة قد تحدث العديد من المشاكل. فمثلاً إذا تعرض الدايود الضوئي الأحمر لأكثر من 25 مللي أمبير لفترة طويلة نسبياً (ساعة أو أكثر) فأنه سيحترق وبالتأكيد أنت لا تريد دائرة إلكترونية تحتاج لتغيير دايود ضوئي كل ساعة !

ثانياً: قد يؤدي تشغيل الأحمال بدون مقاومة لفترات طويلة إلى تدمير أطراف (بورتات) المُتحكِم الدقيق. وذلك بسبب محدودية التيار الكهربي الذي يمكن سحبه من كل طرف، إذا نظرت إلى الصفحة الخاصة بالخصائص الكهربية للمتحكم Electrical Charchtaristics

## Electrical Characteristics

### Absolute Maximum Ratings*

| | |
|---|---|
| Operating Temperature | -55°C to +125°C |
| Storage Temperature | -65°C to +150°C |
| Voltage on any Pin except $\overline{RESET}$ with respect to Ground | -0.5V to $V_{CC}$+0.5V |
| Voltage on $\overline{RESET}$ with respect to Ground | -0.5V to +13.0V |
| Maximum Operating Voltage | 6.0V |
| DC Current per I/O Pin | 40.0 mA |
| DC Current $V_{CC}$ and GND Pins | 200.0 mA PDIP and 400.0 mA TQFP/MLF |

*NOTICE: Stresses beyond those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or other conditions beyond those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

**لاحظ السطر قبل الأخير حيث نجد DC current per I/O = 40 mA**

هذا يعني أن أقصى تيار يمكن سحبة من أي طرف من أطراف المُتحكِم = 40 مللي أمبير وإذا كان الحِمل يسحب أكثر من ذلك فهذا سيؤدي إلى تلف المُتحكِم الدقيق. عليك أن تتذكر هذا الرقم جيداً عند تصميم أي دائرة إلكترونية.

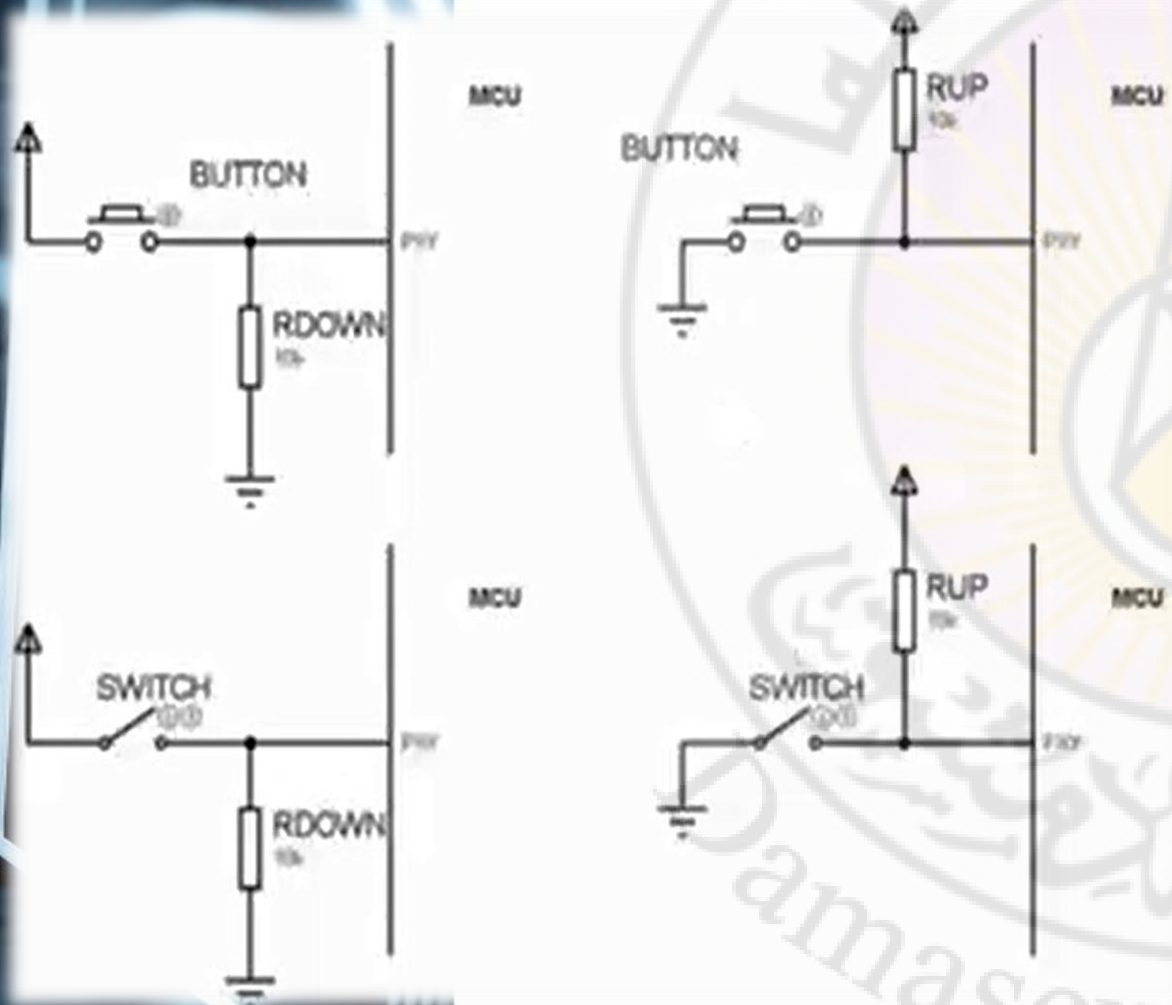
الأفضل أن تسحب الأحمال تيار = 85% من القيمة القصوى وذلك حتى تضمن أن يكون العمر الإفتراضي للمتحكم الدقيق أطول ما يكون. فمثلاً إذا طبقنا هذه القاعدة سنجد أن التيار الآمن هو 0.85 * 40 = 34 مللي أمبير.

**هذا يعني أن أقل مقاومة يمكن توصيلها على أي طرف من أطراف المُتحكِم (من قانون أوم) تساوي 147 أوم، مع ملاحظة أنه يمكن توصيل أي قيمة أعلى لأن المقاومات الأكبر ستمرر تيار كهربي أقل وهذا آمن تماماً.**

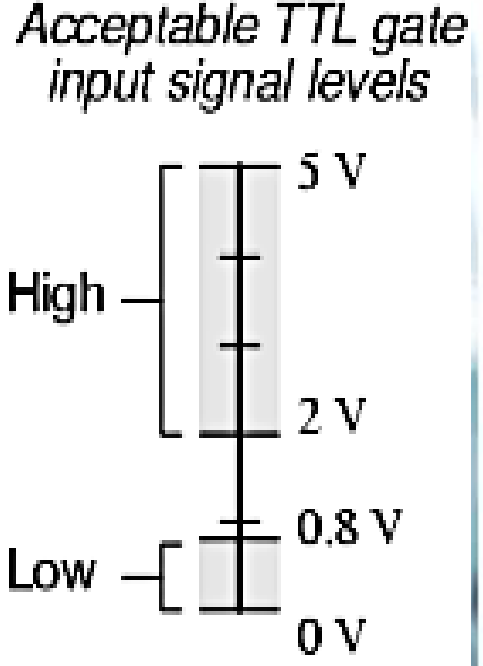
# ربط المفاتيح والكباسات اللحظية مع مداخل المتحكم AVR – مقاومة الرفع والخفض

هناك طريقتان :

1. طرف المفتاح موصول مع القطب الموجب (Vcc) والطرف الثاني موصول قطب المتحكم وموصول مع مقاومة ومع الأرضي . في حالة الزر غير مفعلOFF سيمر صفر منطقي عبر المقاومة ومنه للقطب ، وفي حالة التفعيل ON يمر واحد منطقي . هذه المقاومة تسمى مقاومة خفض .
2. الطريقة الثانية وهي الأكثر شيوعا ان يكون طرف الكباس موصول مع الأرضي Gnd ، والطرف الثاني موصول مع المقاومة ومع القطب ، فعندما يكون الزر غير مفعل يمر واحد منطقي وعند التفعيل يصبح واحد منطقي . وهذه المقاومة تسمى مقاومة رفع .

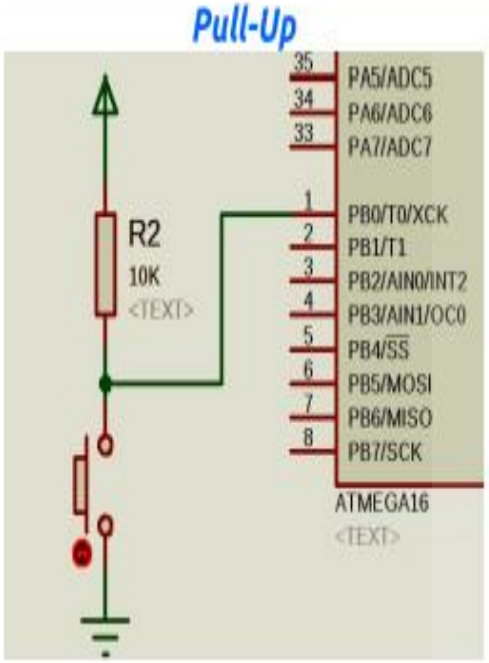


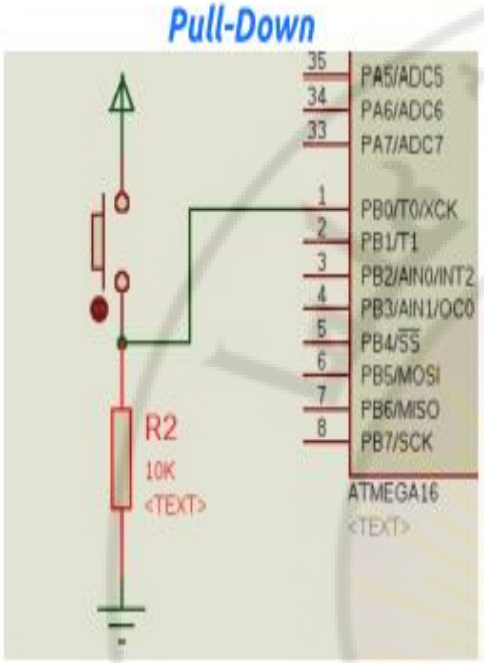
# استخدامات مقاومة الرفع ومقاومة الخفض Pull Up & Pull Down Resistor




هذه المقاومات تتراوح قيمتها بين 2.2 كيلو أوم إلى 10 كيلو أوم تستخدم بصورة أساسية داخل الأنظمة الرقمية Digital Inputs ، بداية من البوابات المنطقية البسيطة وانتهاء بالمتحكمات الدقيقة .
ومن أسباب استخدام هذا النوع من المقاومات مثل المقاومة 10 كيلو أوم مع مفتاح بدون الوصل إلى Vcc مباشرة هي مايلي:

الاستخدام الأول – إلغاء الدخل العائم : جميع المكونات الرقيمة التي تتعامل مع الصفر والواحد تعاني من مشكلة كبيرة وهي المنطقة العائمة Floating Area أو Noise Margin نطاق الشوشرة ، هذه المنطقة هي فارق الجهد بين الصفر المنطقي والواحد المنطقي ، وتستخدم مقاومات الرفع والخفض لحل هذه المشكلة .
والسبب في ذلك أن معظم المكونات الالكترونية تعتبر أي جهد دخل بين (0-0.8) فولت هو صفر (0) أو مايسمى Low . بينما أي جهد بين (2-5) فولت هو واحد (1) أو مايسمى HIGH .
لكن إذا كان هناك دخل بقيمة 0.9 فولت او 1.25 او 1.9 تحدث لدينا مشكلة المنطقة العائمة ، حيث لاتستطيع الالكترونيات الرقمية أن تتعرف على هذا الجهد وبالتالي لايمكنها أن تحدد بدقة ماإذا كانت القيمة (1) HIGH أو LOW(0) وتبدأ بالتصرف عشوائيا .
لذا عند ترك أي طرف دخل input فغنه يتعرض لشوشرة كهربية بفرق جهد صغير نسبيا مما يتسبب في دخول الجهاز بوضع floating area.
وجود مقاومة متصلة بطرف الدخل والطرف الأرضي يضمن تماما أن الطرف يساوي الصفر حتى مع وجود الشوشرة ، ولايتم تغيير هذا الجهد إلا عند إدخال جهد كبير نسبيا مثل 5 فولت ، وتكون قيمة المقاومة بين (2.2 – 10) كيلو أوم



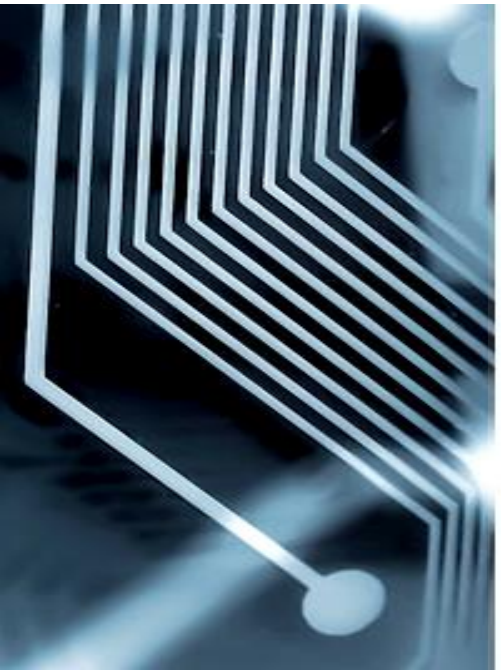



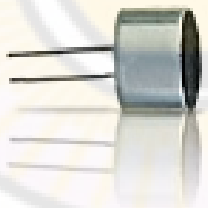

الاستخدام الثاني – عكس الجهد الداخل : في بعض الحالات يكون مطلوب عكس الجهد الداخل من المفتاح ، أي بأسلوب pull-Down مما يجعل المفتاح يدخل جهد 5 فولت عند الضغط على المفتاح ، و 0 فولت (أرضي) عند ترك المفتاح .

Press Button = HIGH  Releas Button = LOW

يمكن استخدام المقاومة الرافعة Pull-Up Resistor لعكس هذا الجهد وبذلك يصبح الجهد الأساسي للطرف هو HIGH وعند ضغط المفتاح يتحول هذا الجهد إلى صفر.

Press Button = LOW  Releas Button = HIGH





**الاستخدام الثالث (مُحول تيار إلى جهد):** بعض الحساسات الشهيرة مثل الميكروفون Microphone أو المقاومة الضوئية LDR أو الحرارية NTC تقوم بتحويل الحرارة أو الضوء إلى تغير في **التيار** الكهربي وليس تغير في **الجهد** مما يمثل مشكلة في فهم هذه الحساسات. حيث نجد أن جميع المُتحكِمات الدقيقة التي تحتوي على ADC يمكنها قراءة **تغير الجهد فقط** ولا تستطيع التعرف على التيار الكهربي المتغير لذا يتم استخدام مقاومة Pull-Down أو Pull-Up لتحويل التيار المتغير إلى جهد.

# خاصية Internal Pull-Up :

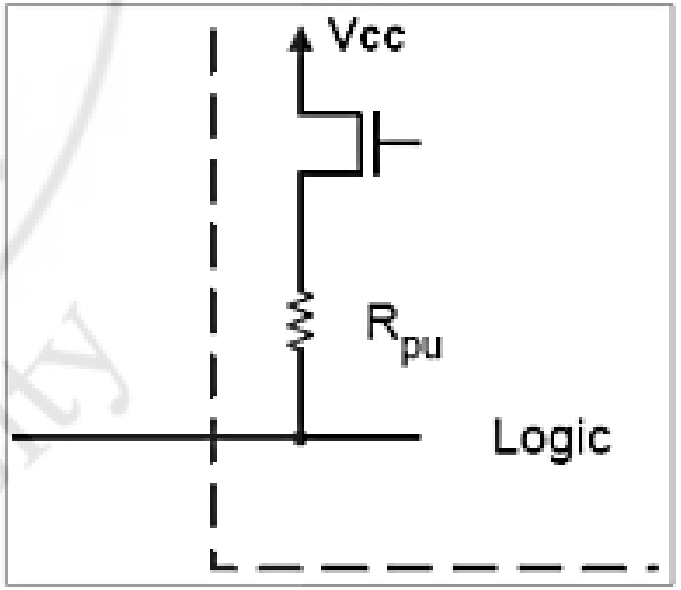
تمتلك متحكمات AVR خاصية أن أطراف البورتات (الأقطاب) تمتلك مقاومة رفع داخلية ، هذه المقاومة تمكننا من استخدام المفاتيح دون الحاجة إلى مقاومات إضافية ، الشكل التالي يوضح تركيب مقاومة الرفع الداخلية وهي عبارة عن :

ترانزستور+مقاومة$R_{Pu}$ ، حيث يتحكم الترانزستور في تفعيل المقاومة أو إلغائها.

بصورة افتراضية يتم إلغاء تفعيل هذه المقاومة ويتم تشغيلها عن طريق جعل الطرف المطلوب يعمل كدخل ، ومن ثم الكتابة في المسجل PORTx لتفعيل هذه المقاومة.

وبهذه الحالة فإن المسجل لن يعمل لكتابة قيم الخرج لأنه سوف يقوم بتفعيل الترانزستور الخاص بمقاومة الرفع فقط ، وأيضا بعد تفعيل المقاومة يصبح تأثير الضغط على المفتاح معكوس مما يعني أن

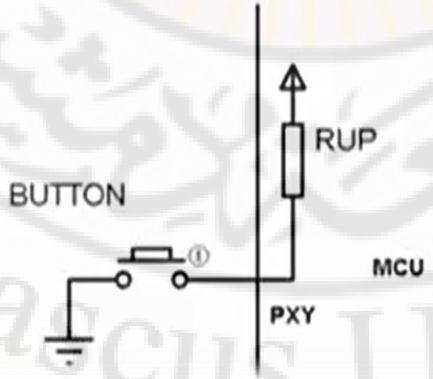
قراءة PINx ستصبح 0 في حالة الضغط على المفتاح (on) و 1 عند ترك المفتاح (off).




مقاومة رفع داخلية يمكن تفعيلها عندما يعمل القطب كدخل عن طريق المسجل PORTX وفق التالي:

DDRX BIT=0

PORTX BIT=1



مثال ١: برمج القطب PD0 في المنفذ PORTD
كمدخل وفعل مقاومته الرفع الداخلية RUP.

```
DDRD. 0=0;
PORTD. 0=1;
```

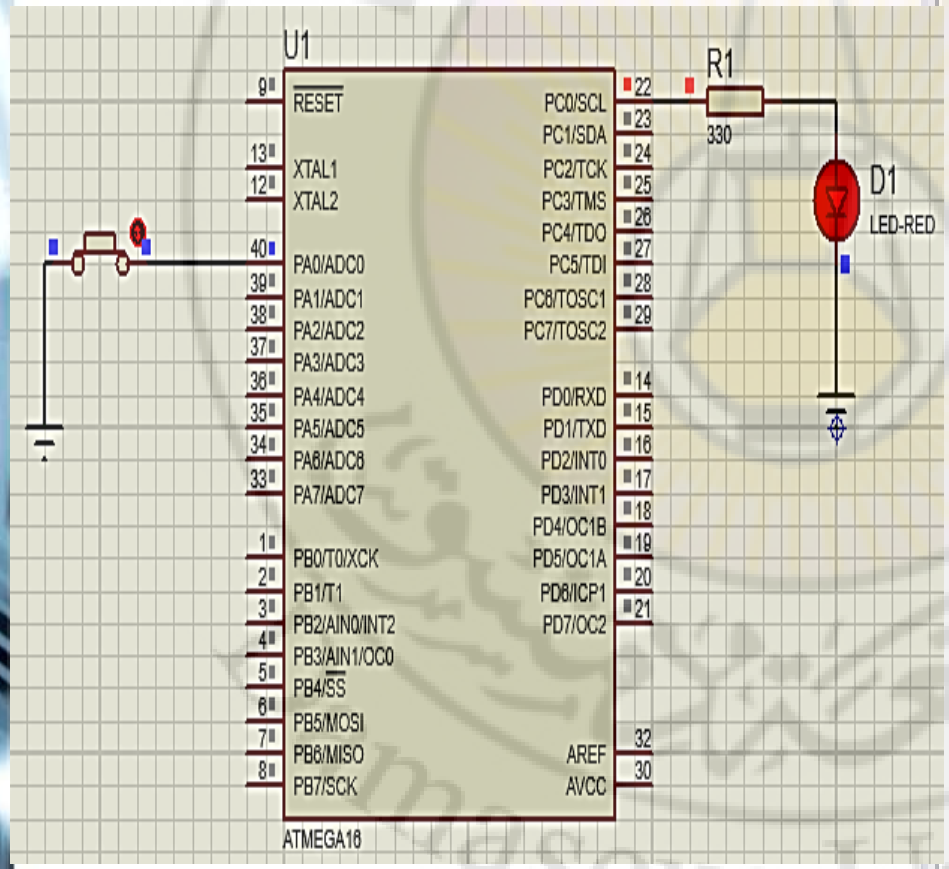
تم وصل مقاومة رفع داخلية على PIND0

مثال ٢: برمج االمنفذ PORTC كي تعمل أقطابه
كدخل وفعل مقاومة الرفع الداخلية RUP لكل قطب.

```
DDRC=0B00000000;
PORTC=0B11111111;
```

Damascus University

نعود للمثال السابق بتفعيل مقاومة رفع داخلية



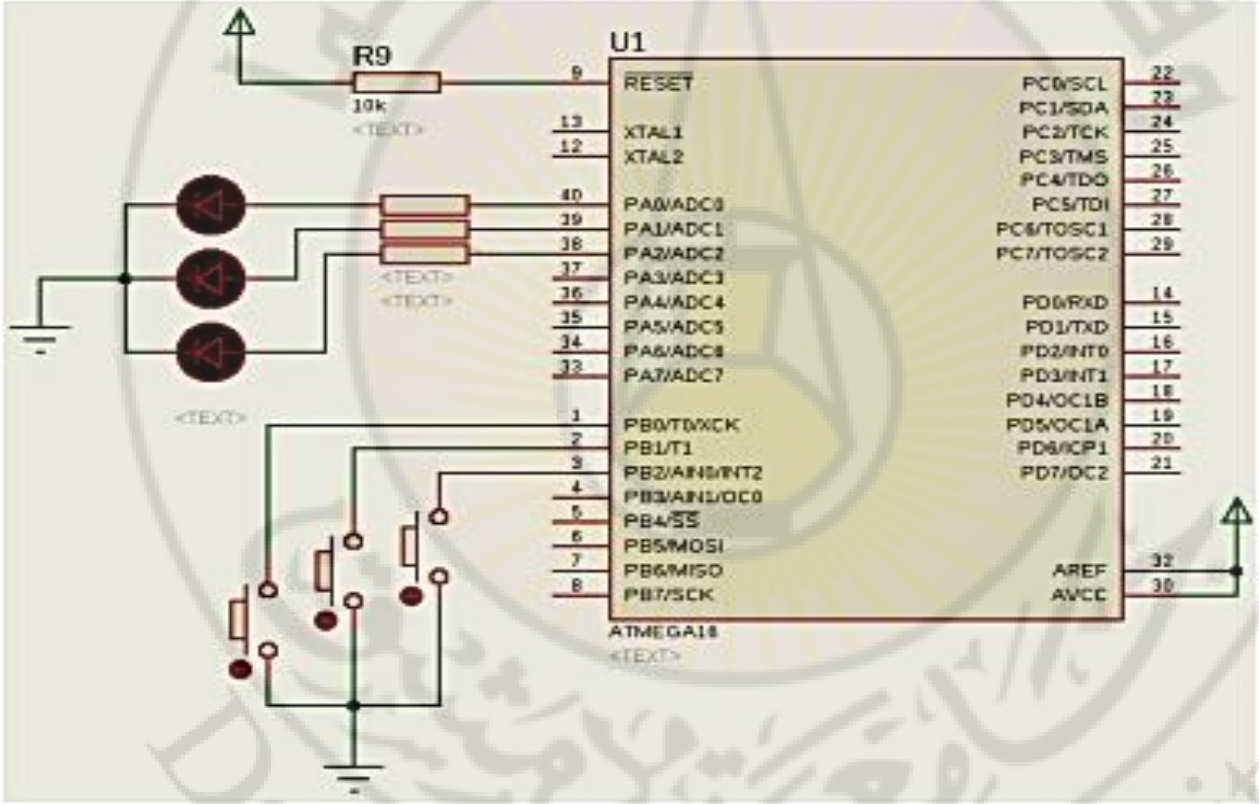

```
fx  main

1   #include <mega16.h>
2   #include <delay.h>
3   void main(void)
4   {
5   DDRC.0=1;
6   PORTA.0=1;
7   start:
8   if (PINA.0==0)
9   {
10  PORTC.0=1;
11  delay_ms(100);
12  PORTC.0=0;
13  delay_ms(100);}
14  goto start;
15  }
```

**PORTA افتراضيا هو دخل كتبنا عليه 1 لتفعيل مقاومة الرفع الداخلية .**
**وبذلك يصبح الشرط PINA.0==0**

# تشغيل 3 ديودات + 3 مفاتيح

في هذا المثال سنستخدم 3 مفاتيح مع تفعيل خاصية الـ Internal pull-up وبذلك سنوفر استخدام 3 مقاومات 10 كيلو. وسيتحكم كل مفتاح في تشغيل عدد من الدايودات بحيث:

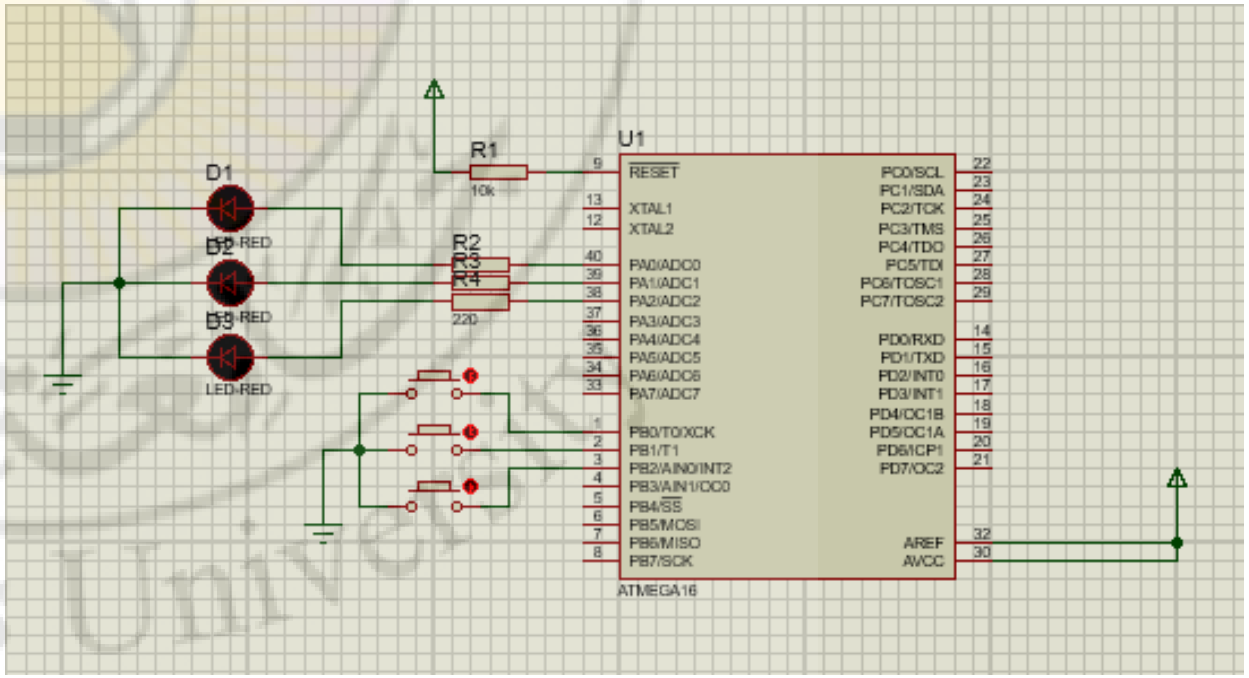
- إذا تم الضغط على المفتاح المتصل بالطرف PB0 يتم تشغيل دايود واحد فقط
- إذا تم الضغط على المفتاح المتصل بالطرف PB1 يتم تشغيل 2 دايود
- إذا تم الضغط على المفتاح المتصل بالطرف PB2 يتم تشغيل 3 دايود

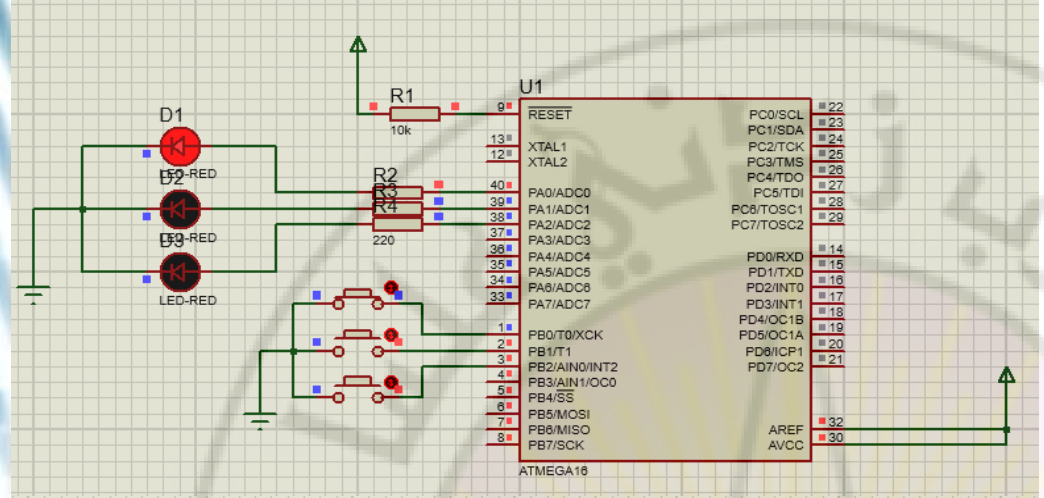


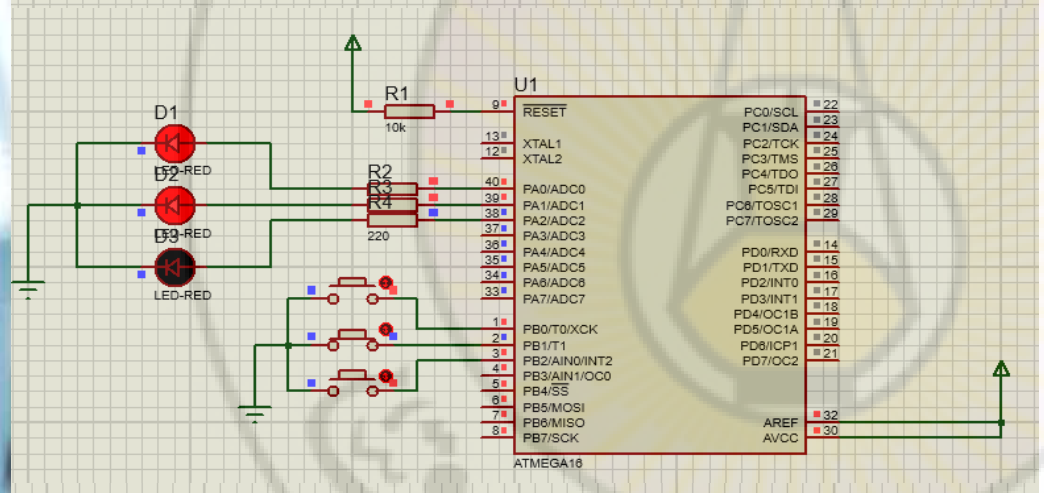

**تذكر أنه بعد تشغيل مقاومة الرفع تصبح قراءة المفتاح والمسجل PINx معكوسة** لذا سنجد القيمة الإفتراضية للمسجل PINx = 0b11111111 وعند الضغط على أي مفتاح ستتحول البت المقابلة له إلى صفر، فمثلاً الضغط على المفتاح المتصل ب PB0 سيجعل قيمة PINB تساوي 0b11111110

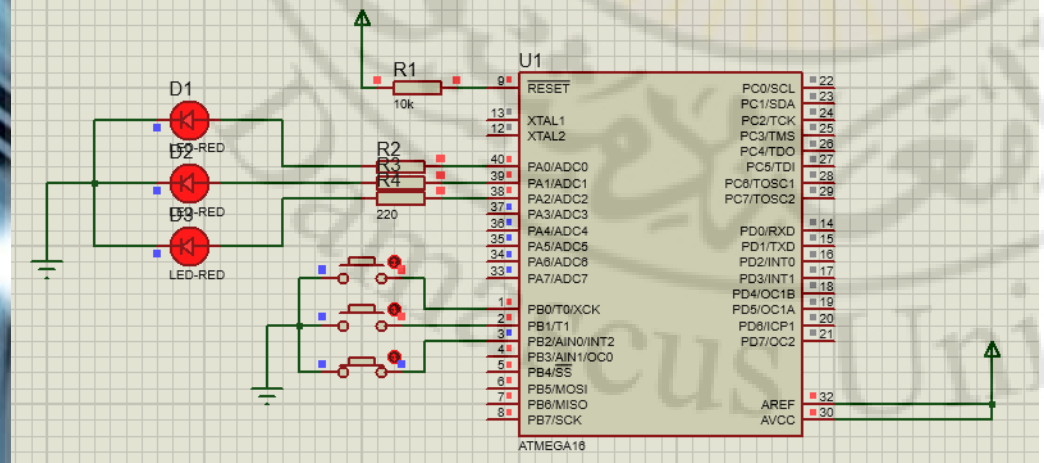
تم تشغيل أطراف PORTB لتعمل جميعها كدخل ، ثم قمنا بتفعيل جميع مقاومات الرفع الداخلية على PORTB

```
// Port B initialization
// Function: Bit7=In Bit6=In Bit5=In Bit4=In Bit3=In Bit2=In Bit1=In Bit0=In
DDRB=(0<<DDB7) | (0<<DDB6) | (0<<DDB5) | (0<<DDB4) | (0<<DDB3) | (0<<DDB2) | (0<<DDB1) | (0<<DDB0);
// State: Bit7=P Bit6=P Bit5=P Bit4=P Bit3=P Bit2=P Bit1=P Bit0=P
PORTB=(1<<PORTB7) | (1<<PORTB6) | (1<<PORTB5) | (1<<PORTB4) | (1<<PORTB3) | (1<<PORTB2) | (1<<PORTB1) | (1<<PORTB0);
```






```
while (1)
{
if (PINB.0==0)
{ PORTA.0=1; }
else if (PINB.1==0)
{ PORTA.0=1;
PORTA.1=1;}
else  if (PINB.2==0)
{ PORTA.0=1;
PORTA.1=1;
PORTA.2=1;
else
{ PORTA.0=0;
PORTA.1=0;
PORTA.2=0; }
}
}
```

# الاهتزازات في المفاتيح الميكانيكية
# تأثير القفز
# Bouncing effect & De-bouncing

تأثير القفز أو الاهتزازات هي ظاهرة تحدث للمفاتيح Switches الميكانيكية خاصة مفاتيح الضغط Push Buttons حيث تتكون هذه المفاتيح عادة من شريحتين من الصفائح المعدنية والتي عند الضغط عليها يحدث بينها تلامس ، وبسبب طريقة صناعة هذه الصفائح فإنه عند الضغط عليها تقفز الألواح المعدنية أكثر من مرة (تهتز) هذا الأمر يجعل الصفائح تقوم بعشرات التلامسات في الثانية الواحدة قبل أن تستقر، أي ان التغيير لايتم من خلال نقلة واحدة وغنما بعد مجموعة اهتزازات غير مرغوبة .

تعتبر هذه الظاهرة سيئة بالنسبة لمصممي النظم المدمجة لأن أنظمة التحكم الالكترونية سريعة جدا (أسرع بكثير من حركة الانسان) فهي تلتقط كافة الإشارات من أجزاء الثواني وبهذا تنتج هذه المفاتيح إشارات غير مطلوبة تؤثر سلبيا على عمل النظام المدمج .

هناك عدة حلول لهذه الظاهرة :

## 1. الحل الأول - التأخير الزمني:

هذا يعتبر ابسط الطرق وتسمى software debouncing حيث تعتمد هذه الطريقة على التأكيد مرتين من الضغط على المفتاح مع وضع تأخير زمني بين كل مرة كما في الكود التالي لتشغيل دايود ضوئي عند الضغط على مفتاحين

الاهتزازات الميكانيكية

الناتجة عن تغير وضع أى مفتاح

الكود التقليدي لقراءة المفتاح بدون debouncing

```
if (PINB == 0b11111110) { PORTA = 0b00000001;}
```

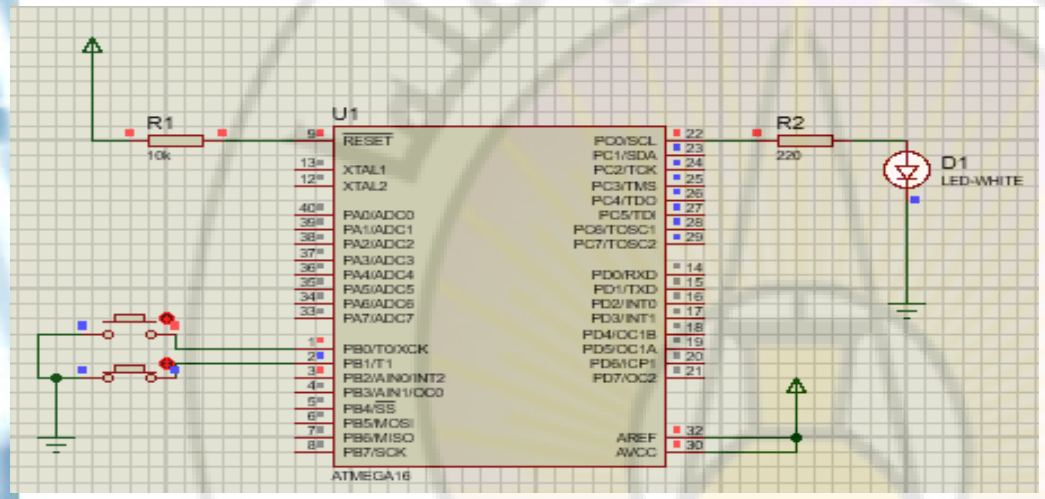
قراءة المفتاح باستخدام ال software debouncing (delay)

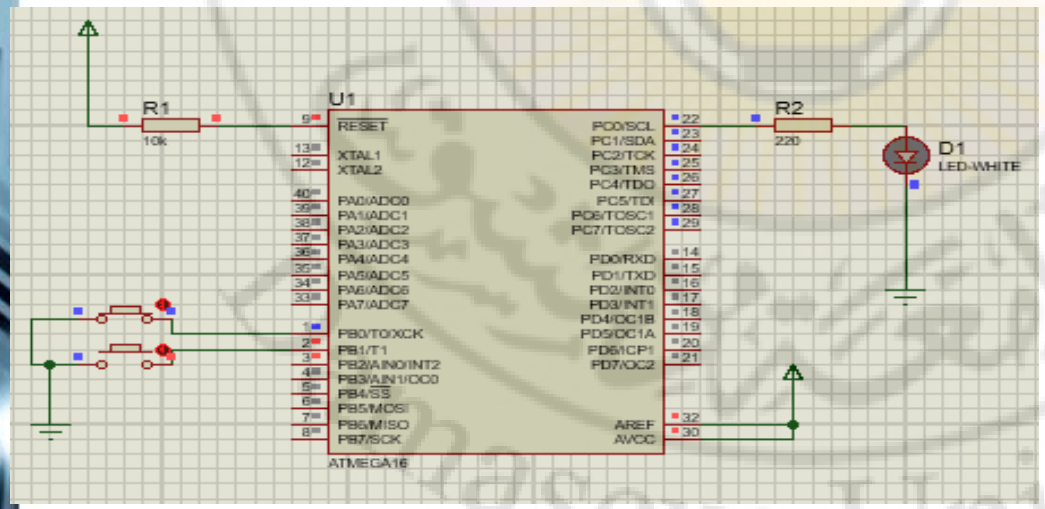
```
if (PINB == 0b11111110)
{
        delay_ms(10);                   //تأخير زمني
        if (PINB == 0b11111110)    // إعادة التأكد أن المفتاح مازال مضغوط
            { PORTA = 0b00000001;}
}
```

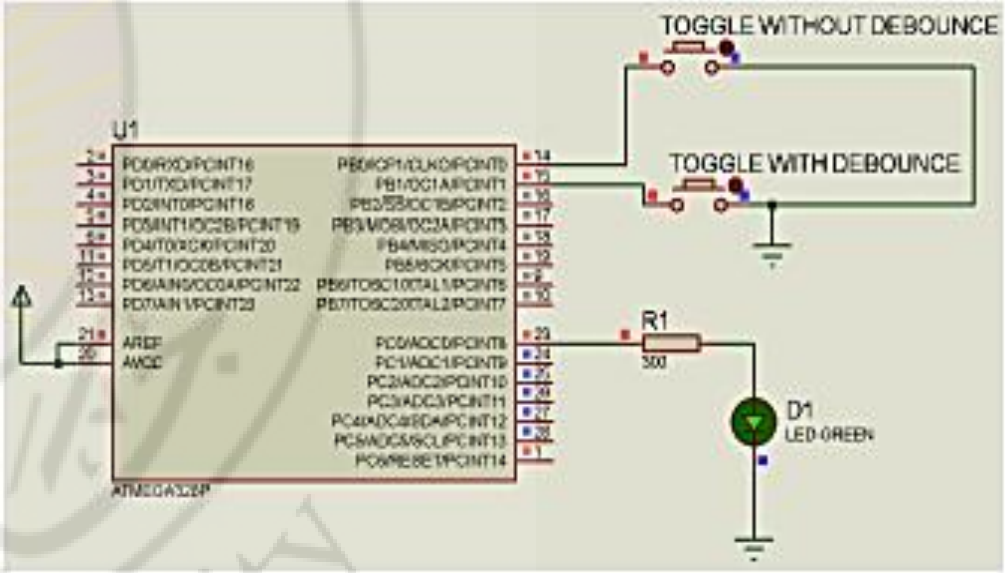
هذه الطريقة لها عيوب كثيرة جداً منها أنها لا تحل مشكلة الذبذبات الكهربية نفسها ولكنها تتأكد فقط من ضغط المفتاح لفترة كافية حتى تنتهي تلك الذبذبات

Damascus University

سنأخذ مثال دائرة الكترونية كما في الشكل : حيث ان المفتاح الأول لايهتم بالتخلص من الاهتزازات أما الثاني تم تعديل قراءته لنتخلص من الاهتزازات كما في الشرح التالي




تأثير التأخر الزمني على إضاءة الليد

الكود مع تأخير زمني وباستخدام الإزاحة:

```
while (1)
    {

     if ((PINB&(1<<0))==0)
       {
       PORTC^=(1<<0);
       }
     if ((PINB&(1<<PINB1))==0)
       {
       delay_ms(25);
       }
     if ((PINB&(1<<PINB1))==0)
       {
       PORTC^=(1<<0);
       }
    }
}
```

المفتاح الأول لايهتم بالتخلص من الاهتزازات أما المفتاح الثاني فقد تم تعديل طريقة قراءته بحيث تخلصنا من هذه الاهتزازات .

عند قراءة الأمر

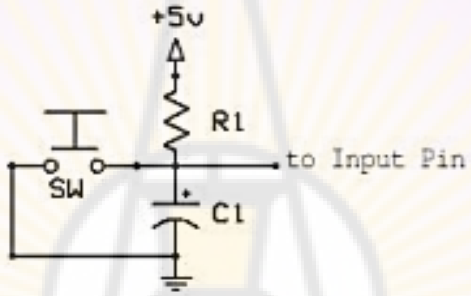
if ((PINB&(1<<PINB1))==0)

يتم بعدها تأخير زمني مقداره 25 ميلي ثانية ، ثم قراءة المفتاح مرة ثانية فإذا كان المفتاح لازال على وضعه ويساوي صفر ، فإن ذلك يعني ان مرحلة الاهتزاز قد مضت واستقر المفتاح على الصفر ، وفي هذه الحالة يتم عمل تغيير للطرف PC1 ، اما إذا كان يساوي 1 بعد مرور 25 ميلي ثانية يعني أن الواحد ناتج من الاهتزازات وفي هذه الحالة لايتم عمل أي إجراء .

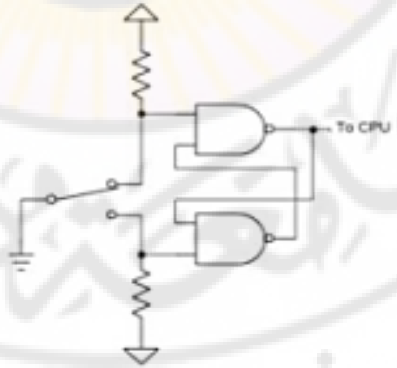
## 2. الحل الثاني- استخدام فلتر مرشح:

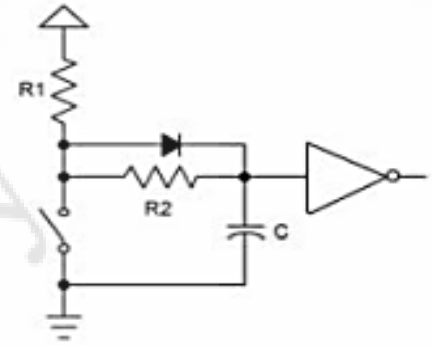
هذا الحل يعتمد على استخدام دائرة filter لترشيح الذبذبات الكهربية الناتجة عن الـ bouncing effect وتسمى Hardware Debouncing حيث تستخدم هذه الدائرة في امتصاص الذبذبات الكهربية وتمنع دخولها للمتحكم الدقيق.

أبسط دائرة مرشح هي توصيل مكثف سيراميكي على التوازي مع المفتاح Ceramic - capacitor بقيمة 0.1 ميكروفاراد (يكون مكتوب على المكثف الرقم 104). مع العلم أنه يمكن توصيل هذا المكثف سواء كنت تستخدم المفتاح مع Pull up أو Pull down resistor




هناك دوائر أخرى أكثر تعقيداً وقوة في الترشيح مثل استخدام مقاومة 10 كيلو + دايود + مكثف (بنفس القيمة). أو باستخدام دوائر رقمية مثل AND gates لعمل digital trigger.




على أي حال، استخدام مكثف واحد فقط كافي لفلترة الذبذبات الكهربية الغير مرغوب فيها ولا داعي لبناء دوائر أكثر تعقيداً كما أنه أوفر من ناحية للتكلفة.

# المقاطعات في المتحكمات

# Interrupts

**المقاطعة :** هي عبارة عن حادثة تسبب توقف المتحكم عن تنفيذ البرنامج الرئيس ليذهب لتنفيذ برنامج خدمة المقاطعة (برنامج فرعي) وبعد الانتهاء منه يعود لتنفيذ البرنامج الرئيس ابتداءاً من التعليمة التي توقف عندها عند حدوث المقاطعة .

أي أنها إشارة إلى المعالج او المتحكم آتية من مصدر عتادي Hardware او برامجي Software توضح وجود حدث يستوجب الانتباه والخدمة الفورية من قبل المعالج ، حيث يقوم المتحكم بتعليق نشاطه الحالي ويذهب لخدمة المقاطعة وبعد الانتهاء يعود لاستئناف نشاطه من نفس المكان الذي توقف فيه ،

الشكل التالي يبين رسم تخطيطي لمقاطعة المعالج او المتحكم.

## أنواع المقاطعات :

تقسم المقاطعات إلى أنواع عديدة وبطرق مختلفة ، منها مايكون مقاطعة عتادية والتي تحدث عادة من خلال أطراف المتحكم أو المعالج ، من امثلتها أن يتم مقاطعة المتحكم في حالة حدوث حريق مثلا أو ارتفاع درجة الحرارة حيث تتم هذه المقاطعة على أحد أطراف المتحكم وبمجرد قبول المقاطعة يذهب المتحكم إلى برنامج خدمة المقاطعة الذي يقوم بالعمل المناسب لحالة هذه المقاطعة مثل ضرب جرس انذار أو استدعاء مطافئ أو غير ذلك . من أمثلة هذه الأطراف الموجودة في جميع المعالجات والمتحكمات الطرف RESET والذي بتنشيطه يترك المتحكم أو المعالج ماينفذه ويذهب لتنفيذ برنامج معين خاص بإعداد وضع المعالج أو المتحكم.

أما المقاطعة البرامجية فهي تحدث لأسباب برامجية تحدث أثناء تنفيذ البرنامج مثل القسمة على الصفر مثلا ، التي قد تحدث عرضيا عندها يخرج المتحكم من البرنامج الذي ينفذه ويعطي رسالة خطأ تعرف المستخدم بهذا الخطأ.

هناك أيضا بعض الأوامر التي بتنفيذها تحدث مقاطعة للمتحكم ومن أمثلة ذلك الأمر reset والذي يمكن وضعه في البرامج بحيث إذا حدثت أخطاء معينة يقوم البرنامج بعمل إعادة وضع للمتحكم لتجنب أي مواقف خطيرة قد تحدث من الاستمرار في تنفيذ البرنامج .

## البعض يقسم المقاطعة حسب طبيعة الإشارة المستخدمة في المقاطعة :

مقاطعة المستوى level interrupt والتي فيها يتم وضع إشارة بمستوى معين (واحد أو صفر مثلا) وتظل الإشارة على هذا المستوى إلى أن تقبل المقاطعة، وإذا تغيرت الإشارة عن هذا المستوى قبل قبول المقاطعة فإن المقاطعة لن تقبل. في مقابل هذا النوع هناك مقاطعة الحافة edge interrupt (الحافة الصاعدة أو النازلة لإشارة المقاطعة). في هذا النوع يكفي أن تنتقل الإشارة من الصفر إلى الواحد (الحافة الصاعدة) مثلا لكي تقبل المقاطعة. عند حدوث هذه الحافة يتم تسجيل 1 في ماسك معين داخل المتحكم يقوم المتحكم بقراءة حالته قبل تنفيذ أي أمر، فإن وجد واحد في هذا الماسك يقوم فورا بقبول المقاطعة. لاحظ أن هذه المقاطعة لابد أن يقبلها المتحكم وليست هناك فرصة لعدم قبولها مثل مقاطعة المستوى السابقة. البعض يقسم المقاطعة إلى مقاطعة يمكن حجبها maskable interrupt وأخرى لا يمكن حجبها nonmaskable. في المقاطعة التي يمكن حجبها يكون قبول المقاطعة مرتبطا بحالة علم معين داخل المتحكم، فإذا كان هذا العلم نشطا يتم قبول المقاطعة، وإذا لم يكن هذا العلم نشطا فلن تقبل المقاطعة، وهذا العلم يمكن التحكم فيه عن طريق المستخدم. أما المقاطعة غير القابلة للحجب فهي مقاطعة ليست مرتبطة بعلم أو خلافة وعند حدوثها لابد من قبولها من قبل المتحكم، ولذلك يتم توصيل المقاطعة الأكثر خطورة على هذا النوع من المقاطعة حتى نضمن قبولها وتنفيذها.

وبشكل رئيسي تقسم المقاطعات إلى نوعين:

## ❑ مقاطعات خارجية :

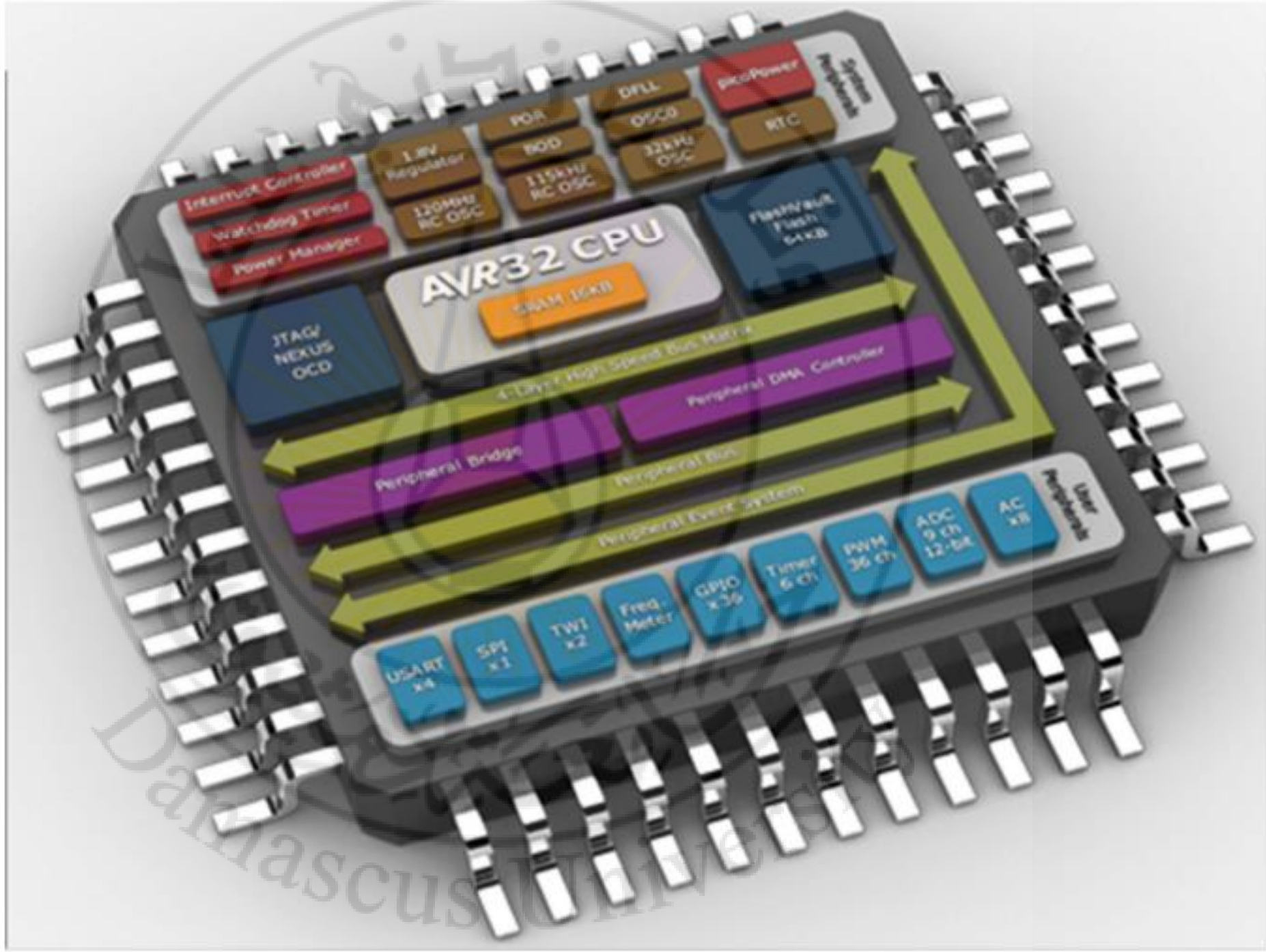
هي استجابة لحوادث خارجية (من خارج الشريحة) مثل :

- مقاطعات الطلب الخارجي تغير مستوى منطقي على القطب – تغير مستوى جهد معين (INT0~INT7) .
- مقاطعة التصفير reset.

## ❑ مقاطعات داخلية :

هي استجابة لحوادث داخلية (داخل الشريحة) مثل :

- مقاطعات المؤقتات / العدادات طفحان للمؤقت – طفحان للعداد
- مقاطعات المسك للمؤقتات/عدادات ICP.
- مقاطعة اكتمال التحويل للـ ADC.
- مقاطعة اكتمال الإرسال للنافذة التسلسلية SPI (STC)
- مقاطعة النافذة التسلسلية USART (RX,TX,UDR)
- مقاطعة المقارن التشابهي (ANALOG COMP).
- مقاطعة النافذة التسلسلية TWI.


Interrupt Controller
Watchdog Timer
Power Manager
1.8V Regulator
120MHz RC OSC
POR
BOD
115kHz RC OSC
DFLL
OSC0
32kHz OSC
picoPower
RTC
System Peripherals
AVR32 CPU
SRAM 16kB
Flash/NVRAM Flash 64kB
JTAG/ NEXUS OCD
4-Layer High Speed Bus Matrix
Peripheral DMA Controller
Peripheral Bridge
Peripheral Bus
Peripheral Event System
User Peripherals
USART x4
SPI x1
TWI x2
Freq. Meter
GPIO x36
Timer 6 ch
PWM 36 ch
ADC 9 ch 12-bit
AC x8

# المفاهيم الأساسية في المقاطعات

1. مسجلات التحكم والحالة للمقاطعة Control and Status :

عن طريق خانات هذه المسجلات يتم تحديد حدث المقاطعة وتفعيلها أو حجبها .

2. برنامج خدمة المقاطعة Interrupt routine :

هو برنامج فرعي يتم تنفيذه عند حدوث مقاطعة .

3. شعاع المقاطعة (عنوان المقاطعة) Interrupt Vector :

شعاع المقاطعة هو العنوان الموجود فيها بداية برنامج خدمة المقاطعة وترتيبها يحدد أفضلية المقاطعة.

4. علم المقاطعة Interrupt Flag :

هو عبارة عن خانة BIT في مسجل أعلام المقاطعة تصبح قيمته واحد عند حدوث المقاطعة (يرفع العلم) وبعد الانتهاء من برنامج خدمة المقاطعة تصبح قيمتها صفر .

## العمليات التى تجري عند حدوث مقاطعة :

وعند ورود إشارة المقاطعة تجري العمليات التالية :

1. ينتهي المتحكم من تنفيذ الأمر الحالي الذي يقوم بتنفيذه.
2. يتم رفع علم الخاص بالمقاطعة (تصبح قيمته واحد). ، لكي يقبل المتحكم عمل المقاطعة
3. يتم وضع عنوان بداية خدمة المقاطعة في مسجل عداد البرنامج (PC) ليقوم بتنفيذ برنامج المقاطعة . (يقفز المعالج إلى مكان محدد في متجه المقاطعة Interrupt vector ليعرف منه عنوان برنامج خدمة المقاطعة ISR).
4. يتم تنفيذ برنامج خدمة المقاطعة (Interrupt Routine).
5. عند الانتهاء من برنامج خدمة المقاطعة الذي يكون آخر أمر فيه هو RETI ، يتم تنزيل العلم الخاص بالمقاطعة (تصبح قيمته صفر).
6. يرجع المتحكم بمتابعة تنفيذ تعليمات البرنامج الرئيس ابتداءاً من التعليمة التي توقف عندهاعندما وردت إشترة المقاطعة .

كل هذه العمليات تتم عن طريق Hardware

Main Function

```
void main(void)
{
....
While(1)
{
Instruction 1
Instruction 2
Instruction 3
.
.
Instruction n
}
}
```

✓ **Interrupt event**

Interrupt Routine

```
interrupt [vector number] void lable (void)
{
...
...
...
}
```

Flag Interrupt=0

Damascus University

Activat
Go to Se

## أولويات المقاطعات :

ماذا سيحدث لو أنه حدثت مقاطعتان للمتحكم في نفس التوقيت ؟

- بشكل عام المقاطعة ذات الأولوية الأعلى هي التي يتم خدمتها أولا ، في المتحكمات AVR تم تحديد أولوية المقاطعات حسب ترتيب وضعها في متجه المقاطعة ، تأتي المقاطعة العتادية RESET في الأولوية .
- عند ورود طلب مقاطعة من النمط (int 0) إلى المتحكم أثناء تنفيذ برنامج مقاطعة من النمط (int 0) (من نفس النمط) فإن المتحكم سوف يهمل هذا الطلب ولن يستجيب لهذه المقاطعة (لأن خانة علم المقاطعة (INTF0) تحمل القيمة (1) منطقي ) ، والأمر نفسه بالنسبة للمقاطعة (int 1) .
- أما إذا ورد طلب مقاطعة من نمط مختلف أثناء تنفيذ برنامج مقاطعة ما فإن استجابة المتحكم لتلك المقاطعة مرتبطة بأولوية المقاطعة الأولى على الثانية ، وتتحدد أولويات المقاطعات المختلفة وفق القاعدة التالية :

  المقاطعة ذات العنوان الأصغر تمتلك الأولوية على المقاطعة ذات العنوان الأكبر فمثلا المقاطعة (RESRT) ذات العنوان ($000) لها أولوية على المقاطعة (Int0) ذات العنوان ($001) والتي لها الأولوية هي الأخرى على المقاطعة (Int 1) ذات العنوان (0X002 أو $002) وهكذا بالنسبة إلى بقية المقاطعات.

يبين الجدول التالي أشعة المقاطعة في ATMega16 .

وترتيب أولوية المقاطعات بين المقاطعات الداخلية والخارجية

**Table 18.** Reset and Interrupt Vectors

| Vector No. | Program Address(2) | Source | Interrupt Definition |
|---|---|---|---|
| 1 | $000(1) | RESET | External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset |
| 2 | $002 | INT0 | External Interrupt Request 0 |
| 3 | $004 | INT1 | External Interrupt Request 1 |
| 4 | $006 | TIMER2 COMP | Timer/Counter2 Compare Match |
| 5 | $008 | TIMER2 OVF | Timer/Counter2 Overflow |
| 6 | $00A | TIMER1 CAPT | Timer/Counter1 Capture Event |
| 7 | $00C | TIMER1 COMPA | Timer/Counter1 Compare Match A |
| 8 | $00E | TIMER1 COMPB | Timer/Counter1 Compare Match B |
| 9 | $010 | TIMER1 OVF | Timer/Counter1 Overflow |
| 10 | $012 | TIMER0 OVF | Timer/Counter0 Overflow |
| 11 | $014 | SPI, STC | Serial Transfer Complete |
| 12 | $016 | USART, RXC | USART, Rx Complete |
| 13 | $018 | USART, UDRE | USART Data Register Empty |
| 14 | $01A | USART, TXC | USART, Tx Complete |
| 15 | $01C | ADC | ADC Conversion Complete |
| 16 | $01E | EE_RDY | EEPROM Ready |
| 17 | $020 | ANA_COMP | Analog Comparator |
| 18 | $022 | TWI | Two-wire Serial Interface |
| 19 | $024 | INT2 | External Interrupt Request 2 |
| 20 | $026 | TIMER0 COMP | Timer/Counter0 Compare Match |
| 21 | $028 | SPM_RDY | Store Program Memory Ready |

أشعة المقاطعة في ATMega16 .

وترتيب أولوية المقاطعات بين المقاطعات الداخلية والخارجية

| رقم المتجه | العنوان فى ذاكرة البرمجة | مصدر المقاطعة | الوصف |
|---|---|---|---|
| 1 | 0x0000 | RESET | يمكن لهذه المقاطعة أن تحدث من الطرف reset، أو بعد تشغيل القدرة، أو بسبب دائرة مراقبة القدرة brown out، أو بسبب مؤقت كلب الحراسة |
| 2 | 0x0002 | INT0 | طلب مقاطعة من على الطرف INT0 للشريحة |
| 3 | 0x0004 | INT1 | طلب مقاطعة من على الطرف INT1 للشريحة |
| 4 | 0x0006 | PCINT0 | المقاطعة رقم 0 بسبب تغير فى جهد الطرف |
| 5 | 0x0008 | PCINT1 | المقاطعة رقم 1 بسبب تغير فى جهد الطرف |
| 6 | 0x000A | PCINT2 | المقاطعة رقم 2 بسبب تغير فى جهد الطرف |
| 7 | 0x000C | WDT | المقاطعة بسبب فيضان فى مؤقت كلب الحراسة |
| 8 | 0x000E | TIMER2_COMPA | تطابق مقارنة A فى المؤقت ٢ |
| 9 | 0x0010 | TIMER2_COMPB | تطابق مقارنة B فى المؤقت ٢ |
| 10 | 0x0012 | TIMER2_OVF | فيضان فى المؤقت ٢ |
| 11 | 0x0014 | TIMER1_CAPT | التقاط حدث للمؤقت١ |
| 12 | 0x0016 | TIMER1_COMPA | تطابق مقارنة A فى المؤقت ١ |
| 13 | 0x0018 | TIMER1_COMPB | تطابق مقارنة B فى المؤقت ١ |
| 14 | 0x001A | TIMER1_OVF | فيضان فى المؤقت ١ |
| 15 | 0x001C | TIMER0_COMPA | تطابق مقارنة A فى المؤقت 0 |
| 16 | 0x001E | TIMER0_COMPB | تطابق مقارنة B فى المؤقت 0 |
| 17 | 0x0020 | TIMER0_OVF | فيضان فى المؤقت 0 |
| 18 | 0x0022 | SPI STC | اكتمال الإرسال التتابعى بالنظام SPI |
| 19 | 0x0024 | USART_RX | اكتمال الاستقبال التتابعى Rx فى النظام USART |
| 20 | 0x0026 | USART_UDRE | مسجل البيانات فاضى فى النظام USART |
| 21 | 0x0028 | USART_TX | اكتمال الإرسال التتابعى Tx فى النظام USART |
| 22 | 0x002A | ADC | اكتمال التحويل فى المحول التماثلى الرقمى ADC |
| 23 | 0x002C | EE READY | الذاكرة EEPROM جاهزة |
| 24 | 0x002E | ANALOG COMP | المقارن التماثلى |
| 25 | 0x0030 | TWI | الواجهة مع نظام السلكين للتراسل I2C |
| 26 | 0x0032 | SPM READY | ذاكرة تخزين البرنامج جاهزة |

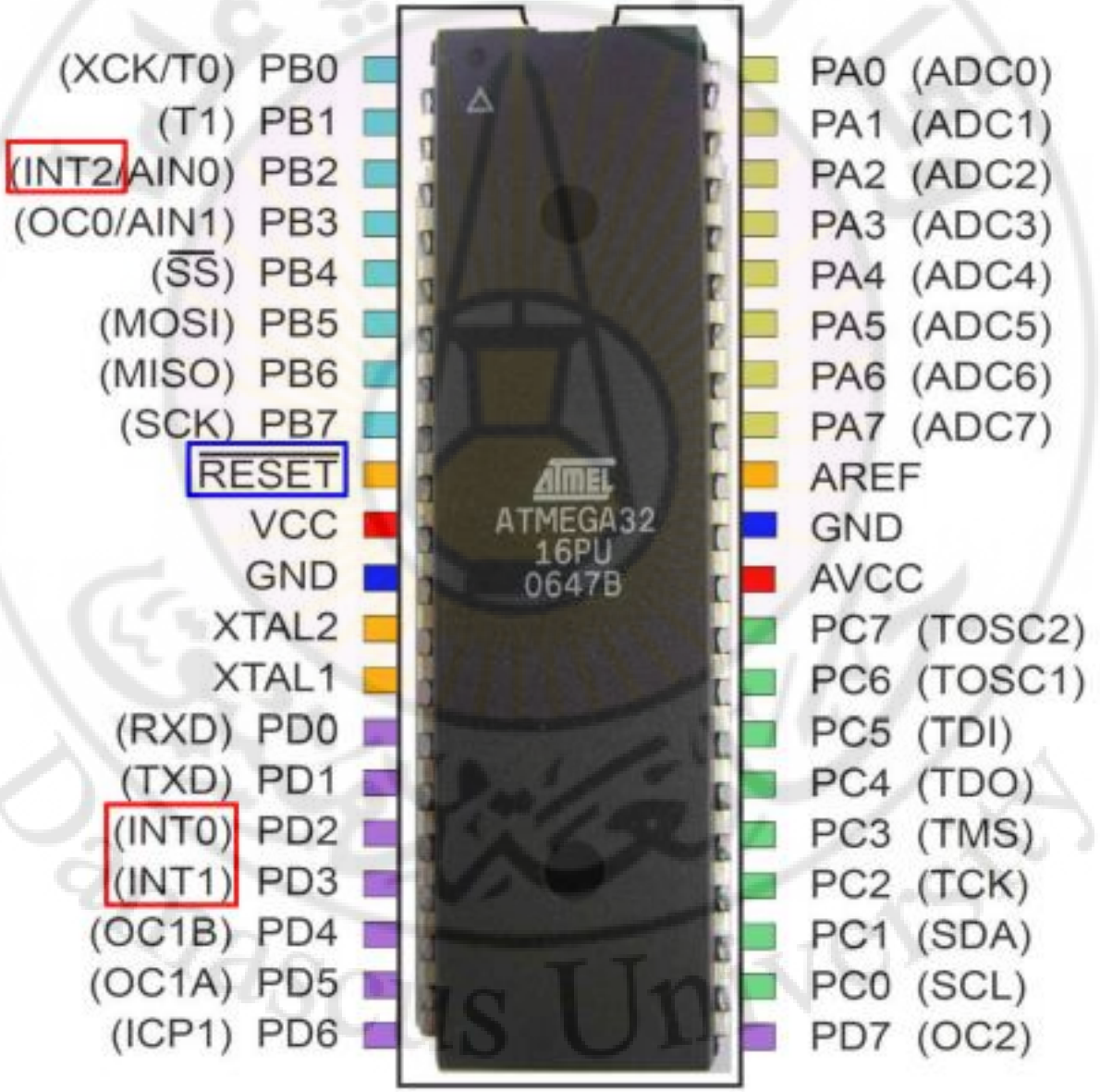
ملاحظة: الـ **RESET** في عالم المُتحكِمات الدقيقة يختلف قليلاً عن الحاسب الآلي. فمثلاً نجد في الحاسب الآلي (أو الهاتف الجوال) الزر RESET أو Restart والذي يعني **إيقاف تشغيل الطاقة** عن الحاسب ثم إعادة تشغيله. أما في المُتحكِمات الدقيقة الزر RESET هو مقاطعة خارجية تأمر المعالج **أن يترك ما يفعله الآن وينتقل إلى أول أمر في البرنامج المخزن بداخله** مع تصفير جميع المُسجِلات ووحدات الذاكرة (وهذا يحاكي إعادة تشغيل المُتحكِم الدقيق) ومع ذلك تظل الكهرباء **متصله بالمتحكم ولا يتم فصلها**. والسر وراء تصميم الـ RESET بهذه الطريقة هو أن إعادة فصل وتوصيل الكهرباء بالمتحكم قد يستغرق 60 مللي ثانية وهذا رقم كبير نسبياً في التطبيقات التي تحتاج استجابة سريعة بينما المُقاطعة يتم تشغيلها في أقل من 1 ميكروثانية (يعني أسرع بنحو 60,000 ضعف من فصل الكهرباء وإعادة توصيلها).

Damascus University

# المقاطعات الخارجية
# (External Interrupts)

## مقاطعات الطلب الخارجي على الأقطاب
## (INT0~INT2)

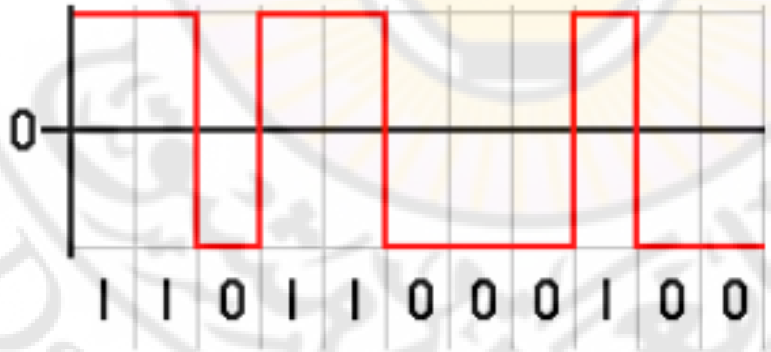
يقصد بالمقاطعات الخارجية تلك التي يمكن توليدها بإشارات من خارج المتحكم . يوجد في ATMega16 ثلاثة أقطاب مقاطعة يمكن من خلالها توليد مقاطعات وهي INT0 وINT1 و INT2.

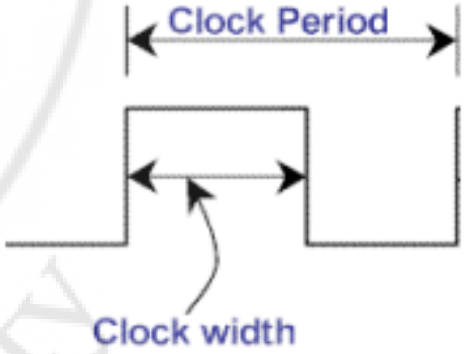


قبل شرح الأوامر الخاصة بضبط المقاطعات الخارجية ، سوف نتعرف على بعض الأشياء المتعلقة بالإشارات الكهربية الرقمية .

## الإشارات الرقمية:

تنقسم الإشارات الكهربية الرقمية إلى نوعين وهما logic level و falling or rising level النوع الأول معروف إما أن يكون HIGH أو LOW ويعبر عن القيم الرقمية (1 و0) وتكون كل إشارة لها زمن محدد يقاس على حسب ال clock المستخدمة لتشغيل المتحكم الدقيق.\فمثلا لو كان المتحكم يعمل بـ clock=1 MHZ (مليون هرتز) بأن زمن النبضة الواحد =1 ميكروثانية ويكون هذا الزمن هو نفس الزمن المطلوب لعمل إشارة كهربية بقيمة 1 أو 0 .
الصورة التالية توضح شكل إشارة رقمية مقسمة إلى واحدات وأصفار (حيث يمثل كل مربع رمادي اللون زمن إشارة واحدة ).

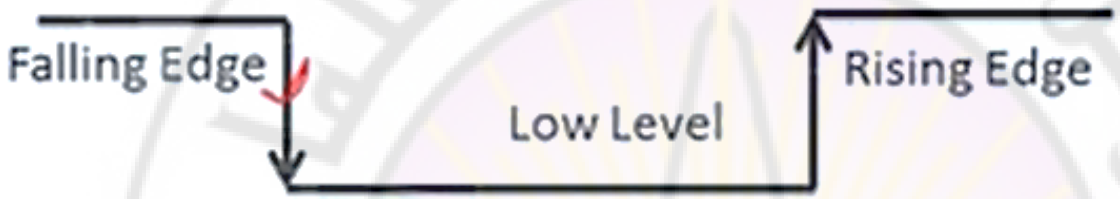


هو الاستجابة لحدث الجبهة الصاعدة أو الهابطة او المستوى صفر منطقي أو أي تغير على قطب المقاطعة الخارجية INTx

هناك ثلاثة أنواع (أقطاب):

- Int0
- Int1
- int2




**أنماط عمل المقاطعة :**

عمل المقاطعة يكون بشكل نبضات حيث يمكن توليد المقاطعات على الأقطاب -

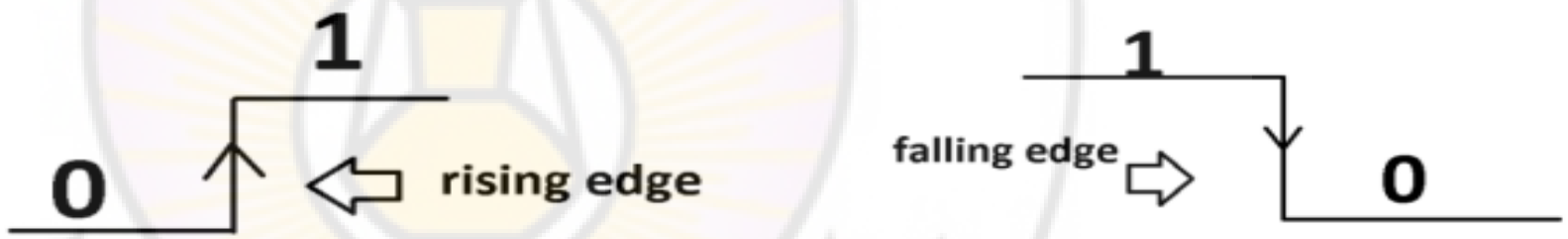
**يمكن تحديد نمط عمل المقاطعة الخارجية بحيث:**

1) تقدح عن الجبهة الصاعدة (Rising Edge).
2) تقدح عن الجبهة الهابطة (Falling Edge).
3) تقدح عن مستوى الجبهة (Low Level).
4) تقدح عن تغير المستوى (Level Change).

ملاحظة: المقاطعة int2 لها فقط حدثين جبهة صاعدة أو هابطة.

النوع الثاني من الإشارات الرقمية يسمى الحواف "Edges" والتي تنقسم إلى نوعين وهما الحافة الصاعدة Rising Edge والحافة الهابطة Falling Edge ، هذا النوع من الإشارات الكهربية يتميز بأنه فائق السرعة ولايلتزم بزمن محدد وغالبا مايحدث في زمن يقاس بالنانو ثانية (جزء من مليار من الثانية).

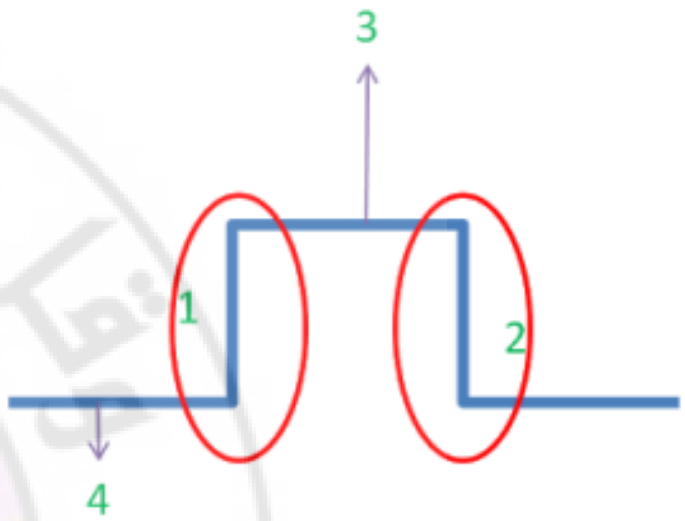
وتعتبر الحواف الصاعدة هي التي تحول الإشارة الكهربية من LOW level إلى HIGH level في زمن صغير جدا بينما الحواف الهابطة هي التي تحول الإشارة بالعكس من HIGH إلى LOW .




**الصور السابقة تثير تساؤل هام وهو: ألا يعني ذلك أن جميع الإشارات الرقمية تحتوي على falling و rising edges ؟**

الإجابة هي **نعم.** والحقيقة أن أي إشارة رقمية تحتوي على حواف عددها يساوي ضعف عدد الأصفار والوحايد والصورة التالية توضح الحافة الهابط والصاعدة مع إشارة LOW ثم HIGH

1:Rising
2:Falling
3:High
4:Low




الأطراف التقليدية التي تعمل كدخل للمتحكم الدقيق لا تستطيع أن تستشعر هذه الحواف سواء كانت الصاعدة أو الهابطة وذلك لأنها تتم في زمن صغير جداً. وهنا تظهر مشكلة خطيرة. حيث نجد أن بعض الحساسات يكون الخرج الكهربي الخاص بها سريع جداً لدرجة أن الإشارات الكهربية الناتجة منه تكون في زمن يقاس بالنانو ثانية (مثل الحواف الصاعدة والهابطة) وتمسى هذه الإشارات السريعة بالـ **Electric Impulse** أو **Electric Edges**.

لحل هذه المشكلة قام مصممو المُتحكِمات الدقيقة بصناعة دائرة إلكترونية خاصة تتصل بأطراف المقاطعات وتسمى بالـ **edge detector** (مكتشف الحواف). وتكون هذه الدوائر مسؤولة عن الإحساس بالإشارات الكهربية فائقة السرعة وإبلاغ المُتحكِم بأنه هناك مقاطعة مطلوبة فوراً.

تمتلك مُتحكِمات AVR هذه الدوائر الخاصة على جميع أطراف المقاطعات وبذلك يمكننا أن نجعل المُقاطعة تعمل على جميع المفاتيح أو الحساسات فائقة السرعة. حيث يمكن ضبط المُقاطعة أن تعمل إما بإشارة تقليدية 1 & 0 أو عن طريق إشارة سريعة **Edge**.

يتحكم في هذا الأمر المُسجِل MCUCR والذي يحتوي على مجموعة من البتات مسؤولة عن تحديد نوع إشارة المُقاطعة وتسمى **ISC**xx (يتم استبدال xx برقمي 0 و 1 كما سنرى في الشرح التالي). يمكنك الوصول لشرح هذا المُسجِل في الصفحة رقم 68 من دليل البيانات للمتحكم ATmega16.

**MCU Control Register – MCUCR**

The MCU Control Register contains control bits for interrupt sense control and general MCU functions.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

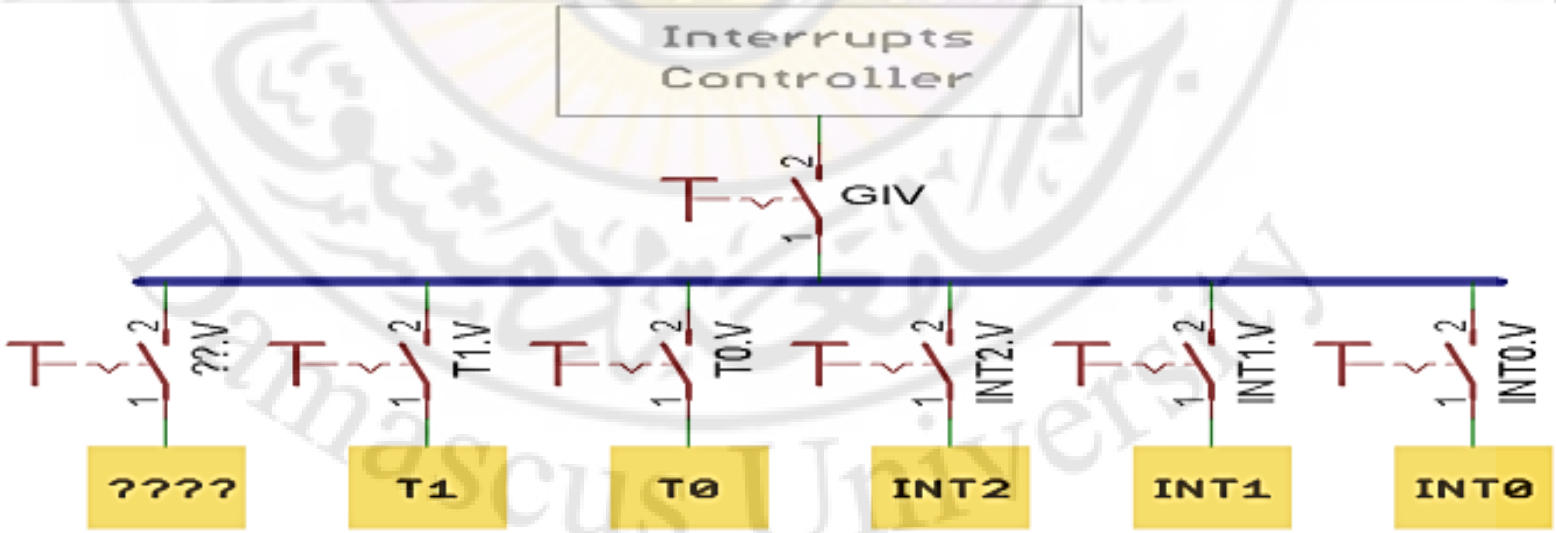
Damascus University

## مراحل تشغيل مقاطعات الطلب الخارجي (INTx ) :

1. تحديد نمط المقاطعة الخارجية (Rising – Falling – Low – Level)
2. تفعيل شعاع المقاطعة المطلوبة تشغيلها .
3. تفعيل شعاع المقاطعات العام .

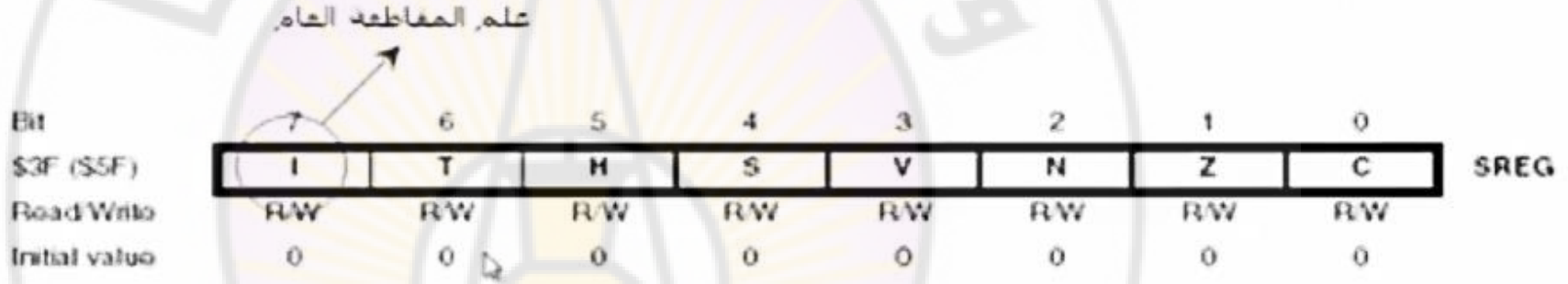
## تفعيل شعاع المقاطعات العام (Global Interrupt Vector):

يمكن ان يتم تشبيهه بالقاطع الرئيسي الذي يقوم بفصل الكهرباء كاملة عند فصله ولايمكن تشغيل أي قاطع فرعي إلا بعد ان يتم تشغيله ، أي من أجل تشغيل أي مقاطعة يجب تفعيل المقاطعة وتفعيل شعاع المقاطعات العام .



# علم المقاطعة العام I

هناك علم (Flag) في مسجل الحالة (SERG) يدعى بعلم المقاطعة العام (I) كما في الشكل وهو متوضع في الخانة الثامنة من مسجل الحالة :




فإذا وضعنا في هذه الخانة القيمة (1) منطقي فهذا يعني أننا فعلنا خدمة المقاطعات وبالتالي عند ورود أي مقاطعة للمتحكم سوف يستجيب لها بشكل طبيعي . وبالعكس إذا وضعنا فيه القيمة (0)م منطقي فهذا يعني أننا حجبنا المقاطعات بأكملها وبالتالي فإن أي مقاطعة ترد إلى المتحكم سوف يتم حجبها ولن يستجيب لها المتحكم أبدا إلى أن يتم تفعيل العلم (I) من جديد.

# مسجلات التحكم بالمقاطعات :

- يمكن التعامل مع المقاطعات عن طريق السجلات التالية:

*MCUCR : MCU control Register*
*MCUCSR : MCU control &status Register*
*GICR : General interrupt control Register*
*GIFR : General interrupt flag register*

يتم التعامل أولا عن طريق المسجلات الثلاثة الأولى : المسجلين الأول والثاني لتفعيل المقاطعات وأعلام المقاطعات وهي بطول 8 بت والثالث هو مسجل المقاطعة العام أما GIFR هو علم المقاطعة العام الذي هو عبارة عن بت يرتفع إلى واحد منطقي عند تفعيل المقاطعة ثم بعد الانتهاء من عمل المقاطعة يعود للصفر.
وسوف ندرسها لاحقا بالتفصيل مع أمثلة .....

## 1. مسجل التحكم المقاطعة العام General Interrupt Control:

هو مسجل التمكين ، وهو مسؤول عن تفعيل وحجب المقاطعات الخارجية ، فبالإضافة إلى علم المقاطعة العام (I) هناك خانة تمكين منفصلة لكل مقاطعة من المقاطعات الخارجية INTx ، فإذا وضعنا في هذه الخانة (1) واحد منطقي نكون قد فعلنا المقاطعة الخارجية المقابلة وإذا وضعنا فيها (0) صفر منطقي نكون قد حجبنا المقاطعة الخارجية المقابلة لهذه الخانة ، وخانتي تمكين المقاطعتين (INT0,INT1,INT2),

GICR – General Interrupt Control Register

| Bit Number | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| GICR | INT1 | INT0 | INT2 | — | — | — | IVSEL | IVCE |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# أما المسجل الثاني مسجل التحكم MCU (Control Register)

أهمها أول أربع بتات ، يتم تحديد الحالات أو الأخداث التي يستجيب لها قطب المقاطعة الخارجية ، عن طريق هذا المسجل يتم تحديد حدث المقاطعة (INT0, INT1 ) حيث عن طريق الخانتين (بت اول وثان) (ISC00,ISC01)

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

MCUCR

INT1

00 -> Low level
01 -> Logic Change
10 -> Falling Edge
11 -> Rising Edge

INT0

00 -> Low level
01 -> Logic Change
10 -> Falling Edge
11 -> Rising Edge

| ISC01 | ISC00 | Description |
|---|---|---|
| 0 | 0 | The low level of INT0 generates an interrupt request. |
| 0 | 1 | Any logical change on INT0 generates an interrupt request. |
| 1 | 0 | The falling edge of INT0 generates an interrupt request. |
| 1 | 1 | The rising edge of INT0 generates an interrupt request. |

يتم تحديد حدث المقاطعة INT0 وفق الجدول :

| ISC11 | ISC10 | Description |
|---|---|---|
| 0 | 0 | The low level of INT1 generates an interrupt request. |
| 0 | 1 | Any logical change on INT1 generates an interrupt request. |
| 1 | 0 | The falling edge of INT1 generates an interrupt request. |
| 1 | 1 | The rising edge of INT1 generates an interrupt request. |

وعن طريق الخانتين , ISC11

ISC10 يتم تحديد حدث المقاطعة

INT1 وفق الجدول التالي :

## المسجل الثالث: مسجل التحكم والحالة MCU (Control and Status Register)

أهمها البت السادس ، وعن طريق هذا المسجل يتم تحديد حدث المقاطعة لـ INT2 حيث عن طريق الخانة ISC2 وفق التالي:

ISC2=0 Falling edge

ISC2=1 Rising edge

**MCU Control and Status Register – MCUCSR**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | ISC2 | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

If ISC2 is written to zero, a falling edge on INT2 activates the interrupt.

If ISC2 is written to one, a rising edge on INT2 activates the interrupt

المسجل الرابع هو : **علم المقاطعة العام General Interrupt Flag :**

أهمها آخر ثلاث بتات ، الخامس هو علم المقاطعة الثانية – السادس علم المقاطعة الأولى – السابع علم المقاطعة الأولى

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INTF1 | INTF0 | INTF2 | – | – | – | – | – | GIFR |
| Read/Write | R/W | R/W | R/W | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

# برمجة المقاطعة INTx

أي برمجة الثلاث مسجلات الأول نحدد عن طريقه حدث المقاطعة ، والثاني لتفعيل أو حجب المقاطعة ، والثالث الذي يتم فيه برمجة المقاطعة

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | ISC2 | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

# برمجة المقاطعة INTx

## أمثلة لإعداد المسجلات :

مثال 1 : قم بإعداد المقاطعة INT0 لتعمل عند الجبهة الصاعدة :
مايهمنا هو من المسجل الأول البت صفر والبت واحد ويتم تحديد قيم الجبهة الصاعدة وفق الجدوال التي مرت سابقا ، أي يكون ISC00=1 و ISC01=1 وكما يجب تفعيل البت السادس في المسجل الثاني GICR أي وضع قيمته واحد لتفعيل هذه المقاطعة INT0=1، اما المسجل الثالث لانحتاج لبرمجته .

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | ISC2 | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

## أمثلة لإعداد المسجلات :

مثال 2 : قم بإعداد المقاطعة INT1 لتعمل عند الجبهة الهابطة :
البتات الثانية والثالثة في المسجل الأول (خانة 3و4) والبت السابع في المسجل الثاني ، ويتم اختيار قيم البتات من الجدول عندما تكون ISC01=0 و ISC11=1 والمسجل الثاني INT1=1 أي تفعيل المقاطعة عند البت السابعة.

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | INT2 | - | - | - | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | ISC2 | - | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

كل هذه العمليات تتم عن طريق Hardware

## أمثلة لإعداد المسجلات :

مثال 3 : قم بإعداد المقاطعة INT2 لتعمل عند الجبهة الصاعدة:
نقوم ببرمجة السمجلين الثاني والثالث ، المسجل الثاني GICR هو مسجل التفعيل نفعل البت الخامس INT2=1 ، وبما أننا نريد جبهة صاعدة فعند البت السادس في المسجل MCUCSR نفعل ISC2=1

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | SM2 | SE | SM1 | SM0 | ISC11 | ISC10 | ISC01 | ISC00 | MCUCR |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | INT1 | INT0 | INT2 | – | – | – | IVSEL | IVCE | GICR |
| Read/Write | R/W | R/W | R/W | R | R | R | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | JTD | ISC2 | – | JTRF | WDRF | BORF | EXTRF | PORF | MCUCSR |
| Read/Write | R/W | R/W | R | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | See Bit Description | | | | | |

# كيف تعمل المقاطعة الخارجية :

في جميع المعالجات والمتحكمات الدقيقة يتم تصميم الكود المسؤول عن معالجة المقاطعات بصورة مستقلة تماما عن البرنامج الرئيسي main program فنجد دائما أن برنامج المقاطعة ويسمى Interrupt Service Routing (يختصر بكلمة ISR) يكتب في جزء بعيد عن دالة ()main فمثلا يمكننا كتابة برنامج main ليقوم بعمل محدد إلى الأبد ، ثم يكتب برنامج الـ ISR ليقوم بوظيفة محددة وسريعة عند تشغيل حساس أو زر معين .
يمتلك المتحكم الدقيق من فئة Atmega 16/ATmega32 عدد أطراف 3 للمقاطعة الخارجية يمكن توصيلها بأي حساس أو مفتاح رقمي وهي :

**INT0** (pin 2 on port D)
**INT1** (pin 3 on port D)
**INT2** (pin 2 on port B)

# خطوات تفعيل المقاطعة الخارجية :

يتم تفعيل المقاطعة الخارجية بمجموعة من الإعدادات كالتالي :

1. ضبط الأطراف التي ستستخدم المقاطعة مثل INT0 أو INT1 لتعمل كدخل input .
2. ضبط نوع الإشارة الكهربية التي ستسبب المقاطعة على حسب نوع الحساس أو المفتاح الذي سيولد إشارة المقاطعة .
3. يتم تفعيل قبول استقبال المقاطعة على الطرف المطلوب مثل INT0 .
4. تفعيل قبول استقبال المقاطعة بشكل عام.
5. كتابة البرنامج الخاص بالمقاطعة ISR.

الكود التالي يمثل التركيب البسيط للـ ISR مع الـ main program

```
int main()
{
.....
}
ISR(interrupt_type)
{
.....
}
```

## خطوات استخدام وبرمجة المقاطعة:

1. إعداد حدث المقاطعة عن طريق مسجلات التحكم الخاصة بالمقاطعة.
2. تفعيل المقاطعة المطلوبة عن طريق المسجلات الخاصة بالمقاطعة .
3. تفعيل المقاطعات بشكل عام عن طريق تفعيل على المقاطعة العام I بإحدى الطريقتين:

SREG.7=1; // اسم المسجل ورقم الخانة

OR

#asm("sei") //باستخدام تعليمة الاسمبلي

## كتابة برنامج خدمة المقاطعة :

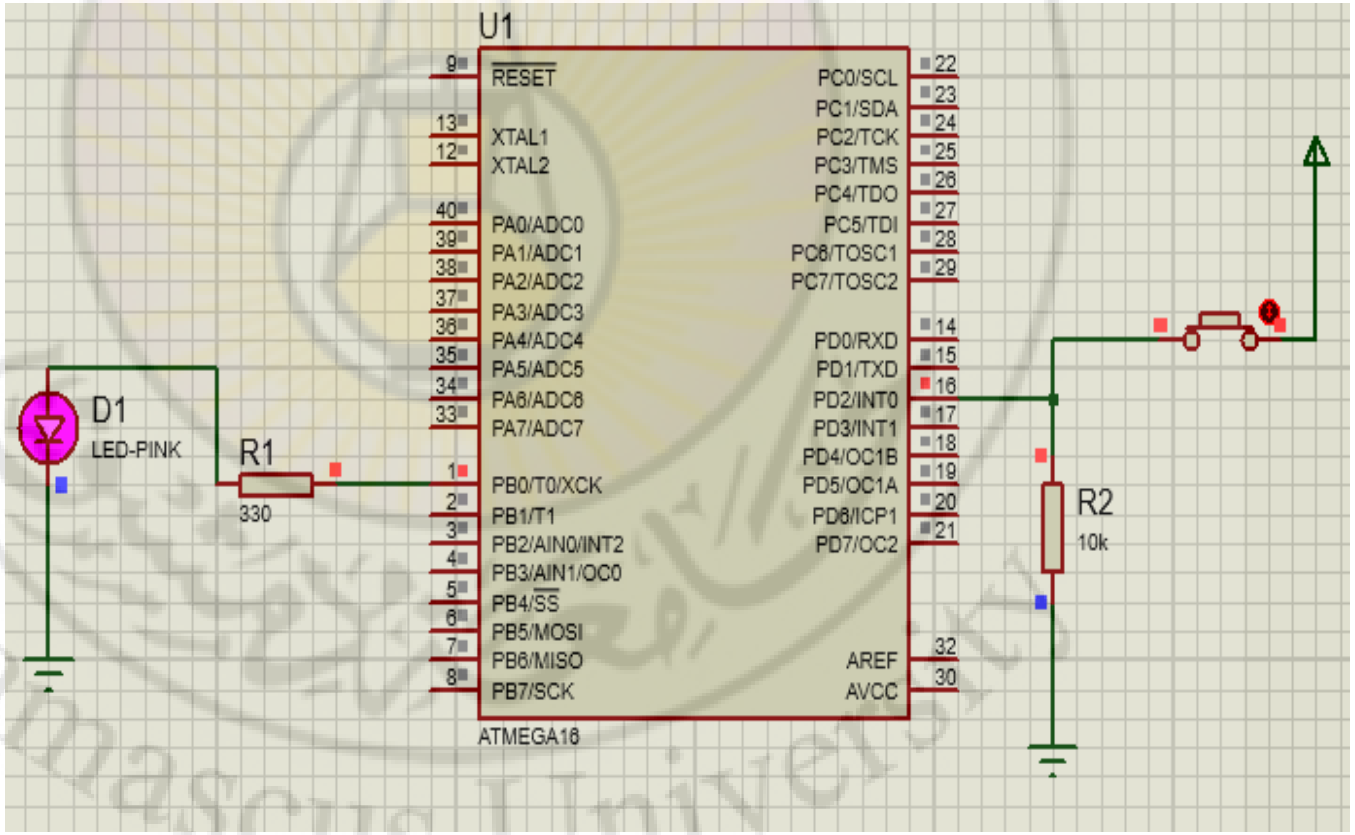
هو عبارة عن تابع فرعي يكتب قبل التابع الرئيسي.

```
Interrupt [vector number] void lable (void)      // اسم اختياري lable
{ ---                                            // -- رقم شعاع المقاطعة يؤخذ من vector number
--}                                                 جدول المقاطعة
```

# تمارين على المقاطعات:

تشغيل وإطفاء ليد باستخدام push button

عند الضغط على الزر يقوم بتشغيل الليد وعند الضغط عليه مرة ثانية يقوم بإطفاء الليد وهكذا



ممكن أن يكون إطفاء وتشغيل أي جهاز عن طريق كباس لحظي ، كما في التمرين تم استخدام متحكم Atmega16 والكباس موصول على زر المقاطعة الخارجية INT0 والذي هو الوظيفة الثانية لـ PD2 وموصول مع مقاومة خفض خارجية ، والليد أو أي جهاز موصول مع PB0 ، في لحظة ضغط الكباس سيعطي جبهة صاعدة أي من 0 إلى 1 منطقي ، وعندما نرفع الضغط يعطي جبهة هابطة أي ينتقل من 1 إلى 0 منطقي (لحظة الكبس تتشكل الجبهة الصاعدة وعند الترك تتشكل الهابطة)

```
#include <mega16.h>
interrupt[2]void external_int0(void)
{
PORTB.0=~ PORTB.0;
}
void main(void)
{
 DDRB.0=1;
 MCUCR=0B00000011; //RISING EDGE FOR INT0
 GICR=0B01000000; //ENABLE INT0
 #asm("sei")    //ENABLE GENERAL INT0
 while (1)
     {}
}
```

كتابة برنامج خدمة المقاطعة وهو عكس حالة البورت B

قطب خرج PB0

برمجة مسجل حدث المقاطعة وفق الجداول السابقة لتفعيل الجبهة الصاعدة

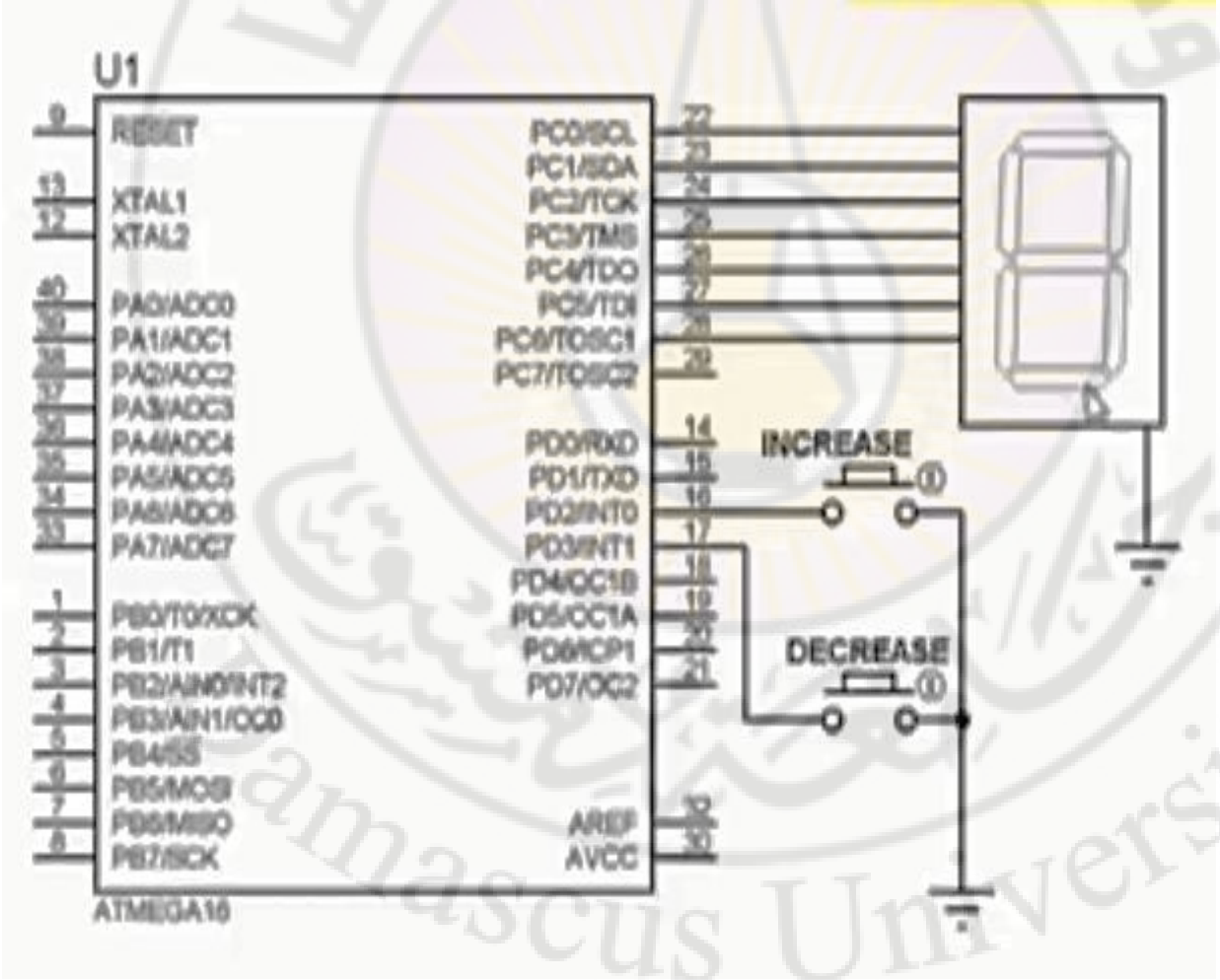
برمجة المسجل الثاني لتفعيل المقاطعة INT0

تعليمة الاسمبلي لتفعيل علم المقاطعة العامة

ثم كتابة حلقة لانهائية

أي يقوم المتحكم ببرمجة PB.0 كخرج ثم يقوم بتفعيل الجبهة الصاعدة ثم تفعيل المقاطعة ومن ثم تفعيل علم المقاطعة العام . ويدخل بالحلقة اللانهائية ولايتوقف حتى تأتيه مقاطعة خارجية والتي هي ضغط الزر

# تطبيق عداد الزيادة والنقصان من 0 إلى 9




برمجة المقاطعة السباعية عن طريق كباسين لحظيين الأول موصول مع المقاطعة INT0 يقوم بزيادة القيمة (العدد الذي يظهر على المقاطعة) عند الضغط عليه أي ورود جبهة هابطة على قطب المقاطعة وذلك بسبب تفعيل مقاومة الرفع الداخلية كما في الشكل . والثاني موصول مع INT1 يقوم بإنقاص القيمة عند الضغط عليه ، أي ورود جبهة هابطة لإنقاص العداد.

كرر نفس التمرين من أجل مقاطعة سباعية 4digit لزيادة العداد من 0 إلى 1000

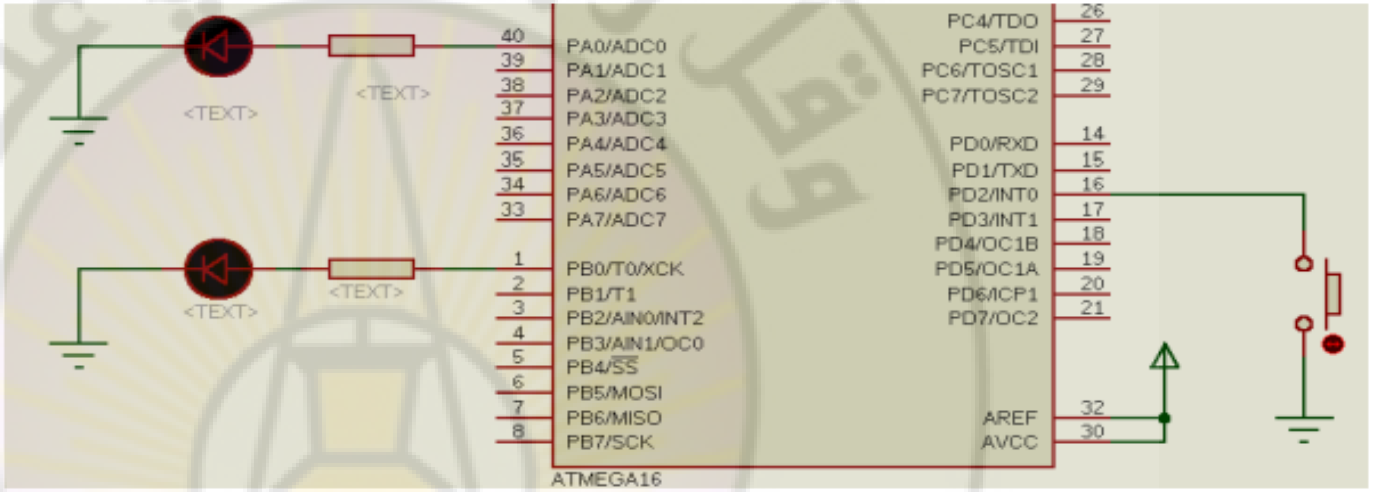
```
#include <mega16.h>
#include <delay.h>
char i=0;
interrupt [2] void ext_int0_isr(void)
{
 if (i<9)
 i++;
}
interrupt [3] void ext_int1_isr(void)
{
if (i>0)
i--;
}
void main(void)
{
char
sevenseg_code[10]={0x3f,0x06,0x5B,0x4f,0x66,0x6d,0x7c,0x07,0x7f,0x6f};
DDRC=0B11111111;
PORTD=0B00001111;

MCUCR=0B00001010;
GICR=0B11000000;
#asm("sei")
while (1)
    {
    PORTC=sevenseg_code[i];
    } }
```

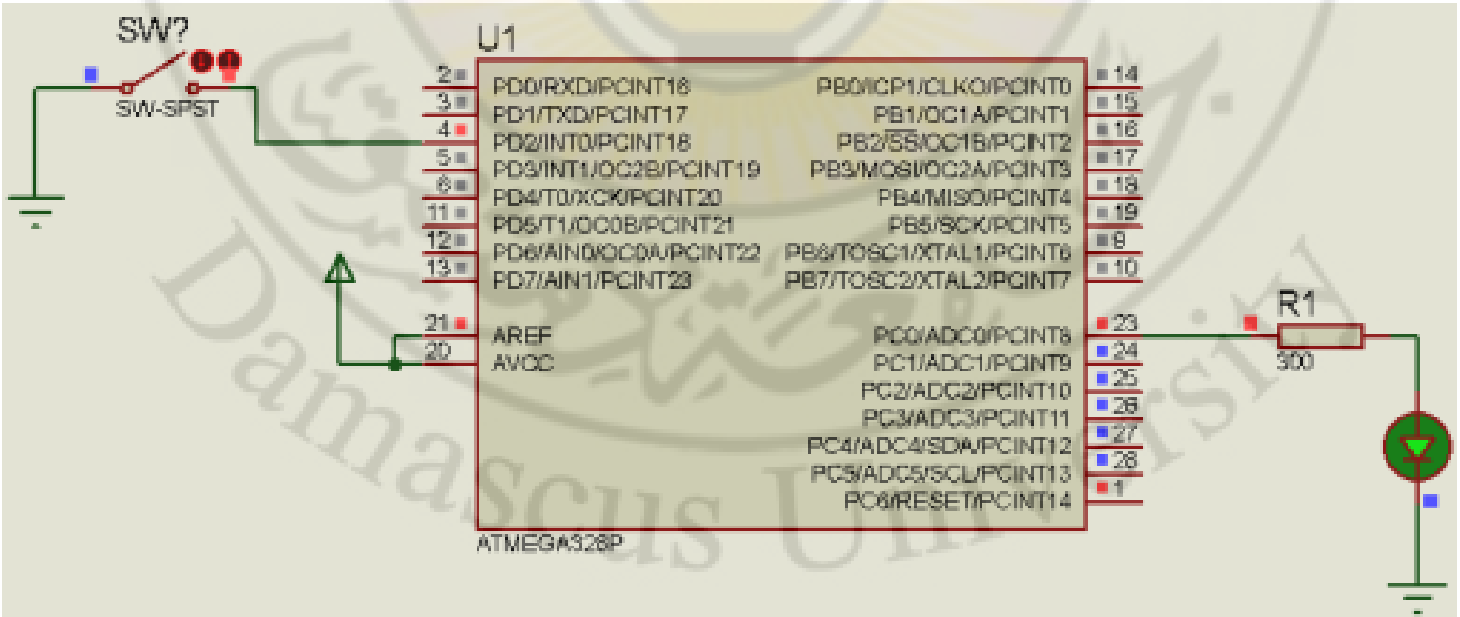
تم وضع متحول نوع char بقيمة 0
كتابة برنامج خدمة المقاطعة لـINT0 وفيه تعليمتين الشرط والزيادة حتى القيمة 9 كل ضغطة للكباس وذلك عند ورود جبهة هابطة على المقاطعة الأولى.

كتابة برنامج خدمة المقاطعة الثانية كما الأولى أيضا عند ورود جبهة هابطة على المقاطعة الثانية شعاع المقاطعة رقم 3

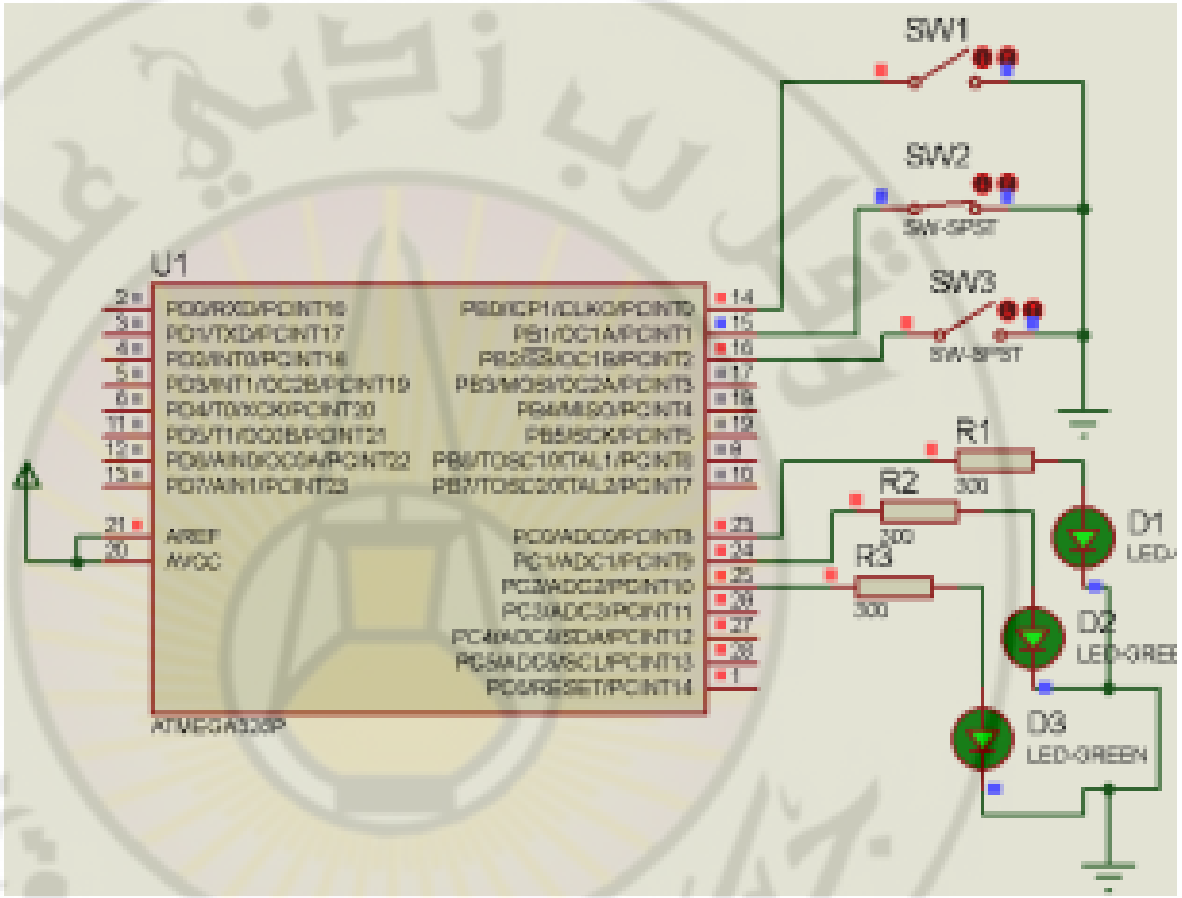
كتابة التابع الرئيسي :
تعريف مصفوفة كود ال7-seg.
تعريف الPORTC كخرج ، وتعريف مقاومة الرفع الداخلية للأقطاب INT0 وINT1 في الPORTD وذلك بإحضار القيم من الجداول.
تعريف المسجل GICR
تفعيل المقاطعتين INT0وINT1 (البت السادس والسابع)
كتابة تعليمة الاسمبلي لعلم المقاطعة العام
كتابة الحلقة اللانهائية while لإظهار قيمة ال-7 seg. التي تقوم بتنفيذ التعليمة بشكل دائم عند ورود المقاطعة

الدائرة التالية عبارة عن 2 دايود ضوئي + مفتاح الدايود المتصل بالطرف PA0 سنقوم بتشغيله بصورة طبيعية ليقوم بعمل Blink كل مئة مللي ثانية. أما الدايود المتصل بالطرف PC0 سيتم تشغيله أو إطفاؤه فقط عند حدوث مقاطعة على الطرف INT0.




تفعيل المقاطعة باستخدام switch




شكل ٦-١٤ المقاطعات الثلاث تذهب لنفس برنامج

خدمة المقاطعة

Damascus University