

بسم الله الرحمن الرحيم

# السلام عليكم ورحمة الله وبركاته



## دورة الأسمبلى

تأليف : MaX ( **MaX** )

منتديات : المدرسة العربية للبرمجة (وَقَلِّ رَبِّ زُنِّي عَلَّمًا)

الرابط الاصلى للدورة على المنتدى : <http://www.geek4arab.com/vb/forumdisplay.php?f=90>

جمع وترتيب وتنسيق : علي السيد محمد ( **الفدائي : Alfidai** )

اسألکم الدعاء لنا ولوالدینا والدعاء خاصة لصاحب الدورة .

**ملحوظة** : توجد دورة أخرى مكتملة لهذه الدورة في صفحة ( 40 ) من هذا الكتاب وهي أيضا ممتازة

كما توجد ملاحق للمسجلات والرايات وجداول الاسكي والتعليمات . وايضا دليل صغير يربطك بالهندسة العكسية

هذا النوع من ملف الـ PDF مصنوع بـ وورد 2010 لذا بالامكان عمل نسخ ولصق للنصوص على الوورد بكل سهولة ثم تحريرها ( طبعا انا جربت ذلك في وورد 2010 وادوبي 9 )

بسم الله الرحمن الرحيم

# السلام عليكم ورحمة الله وبركاته



دورة الأسمبلى - اليوم القبل الأول

## دورة الأسمبلى - اليوم القبل الأول

هذا الدرس هو أساس لغة الأسمبلى و تعتبر المواضيع التى ستناقش به هى أساس لغة الأسمبلى خاصا و أساس لغات الكمبيوتر عاماً، أعزرونى أنه جاء متأخرا والشكر الخاص للأستاذ / إبراهيم صالح الذى له الفضل فى تذكيرى بهذه المواضيع المهمة، شكرا لك يا أختي

سنبدأ بإذن الله و سنناقش المواضيع الآتية :-

- ما هو 0 و 1 (اللغة الثنائية) ؟
  - ما هى حقيقة وحدات التخزين داخل الكمبيوتر (المسجلات و الذاكرة) ؟
  - كيف يتم تناقل البيانات بين هذه الوحدات و بعضها البعض و بينها و بين المعالج ؟
  - مما يتركب المعالج ؟؟
  - البيانات و أنواعها ،
  - كيف تتم معالجة البرامج و تعليماتها؟؟
  - كيف يقوم المعالج بتنفيذ تعليمه ما ؟
- ملاحظه : لأهمية هذه الموضوعات يرجى الرجوع إلى الدروس السابقه و إلى المصادر المتاحة للأسمبلى و لبنية المعالجات و تركيبها

بسم الله،، Computer organization and structure

### ما هي حقيقة ال 0 و 1 ( اللغة الثانيه )؟؟

هذه القيم تعبر عن معنى كلمة يوجد و لا يوجد -صح أم خطأ - أسود أم أبيض - و تعبر عنها في علوم الإلكترونيات بوجود جهد كهربى على سلك ما أم لا يوجد ، بمعنى هناك سلك واحد نحاس مثلا و عليه فرق جهد بالنسبة للأرضى، فنقول هنا أن هذا السلك به فولت و يحمل القيمة 1 ، هذا السلك نسميه في الإلكترونيات (بت) ، 8 اسلاك بجانب بعضهم نطلق عليهم (بايت. )

أما في حالة عدم وجود به فولت فإنه يمثل القيمة 0بهذه الطريقة يمكن لمجموعه من الأسلاك متجاوره و منعزله عن بعضها البعض أن تمثل قيمه مثلا 10101 وهذا يعبر عن خمس أسلاك الأول به فولت و الثانى لا يوجد به و الثالث يوجد به و الرابع لا يوجد به و الخامس يوجد به فولت .

طيب فكر معى أختى ما هي احتمالات هذه الأسلاك أن تحمل قيم (بمعنى ما هو عدد الإحتمالات الغير متكرره التى يمك أن تمثلها هذه الاسلاك الخمسه ؟؟ )

سنجد أنها 32 إحتمال من 00000 إلى 11111 و هو ما يعبر عن  $2^5$

(أى إحتمال قيم السلك الواحد  $2^5$  عدد الأسلاك = إحتمال القيم على كل السلاك)

مثال بسيط ، بالنظام العشرى الطبيعى الذى نعمل به جميعا (الرقم الواحد إحتماله أن يكون 10 قيم من 0 إلى 9) فمثلا لو عندنا رقم مكون من 5 أماكن فما هي عدد الأرقام التى يمكن أن تمثل بهذه الأماكن الخمسه - على حسب القانون السابق نجد أنها  $(10^5 = 100000$  إحتمال) (من 00000 إلى 99999) ، فعلا القانون صحيح ....

من هذا المثال السابق يتبين لنا طريقة حساب عدد القيم التى يمكن أن يمثلها مجموعه من الأماكن (الأسلاك) فى أى نظام عددى (ونحن نتحدث هنا عن النظام الثانى)

تدريب : ما هو عدد القيم التى يمكن تمثيلها ب 8 أسلاك (بايت) و 16 سلك (2 بايت) و 32 سلك (4 بايت) الإجابة :

8بت  $2^8 = 256$  >>>إحتمال

16بت  $2^{16} = 65536$  >>>إحتمال

32بت  $2^{32} = 4294967296$  >>>إحتمال (وهو ما يمثل 4جيجا من وحدات الكمبيوتر)

باقى أن نذكر أن هذا النظام هو المتبع فى جميع الأجهزة الإلكترونية و الكمبيوترية و الأدوات الحاسبه التى يطلق عليها (أجهزه رقميه Digital Divices )

### -ما هي حقيقة وحدات التخزين داخل الكمبيوتر (المسجلات و الذاكره) ؟

وحدات التخزين فى الحقيقة تعتبر مجموعه من السلاك المتجاوره مع بعضهم و منعزلين عن بعضهم ، مع إمكانية أن كل قيمه على أى سلك يمكن تغييرها من 1 إلى 0أنظر أختى الكريم إلى هذا المثال :-

أنظر داخل جهازك ، ستجد أن الهارد ديسك Hard Disk موصل بكابيل بيانات Data Cable وهو مكون من 40 سلك Bin 40 في أغلب الأحيان ، هذه الأسلاك هي المسؤولة عن نقل البيانات التي تجدها أمامك على الشاشة الآن من الهارد ديسك إلى المعالج عبر مسارات اللوحة الأم .

لو تسنى لك النظر إلى اللوحة الأم Mother Board من الخلف ستجد عدة مسارات متوازية و بجانب بعضها تصل بين الذاكرة Ram وبين المعالج Processor تتفق جميع وحدات التخزين في قدرتها على تمثيل 0 أو 1 و هذا على حسب حجم الوحدة وتختلف وحدات التخزين في طبيعة هذه القدرة وناقش هذا سويا هنا،

المسجلات Registers :- تتكون من مجموعات منطقية بجانب بعضها يمكن للمجموعه الواحده تمثيل بت ( 0 أو 1 ) ( فولت أو لا فولت) وتطلق على المجموعه Flip Flop وهو مركب من وحدات منطقيه اصغر Logic Gates يستطيع الإحتفاظ بالقيم التي بداخله طوال تغذيته بالفولت ، ماذا تعني كلمة (طوال تغذيته بالفولت) ؟؟ تعنى أن أى مكون كهربى داخل أى جهاز يحتاج لمصدر أساسى للكهرباء وهذا ما يمكنه من التمييز بين 0 أو 1 ، بمعنى أن المسجل يكون موصل بهذا المصدر الأساسى VCC حتى لو ان القيمة التي تحملها أطرافه 000000 و هذاوحدات الذاكرة :- Ram بدون التوغل في تفاصيل - تتركب من مكونات شبيهه بالمسجلات أو بمكثفات و لها أنواع عده و لكنها هذه الأنواع التي تحتاج إلى مصدر كهربى أساسى للحفاظ على القيم داخلها كما هو الحال بالمسجلات .

الإقراص الصلبه :- Hard Disks يمكن تمثيل البيانات عليها على هيئة سطوح قابله للمغظه ، كل سطح مكون من مسارات ، كل مسار مكون من قطع ، كل قطع مكونة من وحدات صغيره فأصغر ، أصغر وحده هي النقطة و هذه النقطة يمكن تغيير خاصيتها المغناطيسيه من (توجد مغظه أم لا يوجد) و هذا ما يهنا أنه يمكن تمثيل الوحدات الثنائيه بهذه الطريقه ، و الجدير بالذكر هنا أن وحدات التخزين التي تعتمد على هذه الطريقه لا تحتاج إلى مصدر كهربى للحفاظ على البيانات التي على سطوحها طيب سؤال : لماذا إذن نوصلها بكابل طاقة Power Source عند تشغيلها ، الإجابة : لا ننسى أختى الأجزاء أنه يوجد موتور كهربى بالداخل و أجزاء ميكانيكيه أخرى تحتاج طبيعتها إلى المصدر الكهربى بالإضافة إلى الوحدات المنطقية والمسجلات التي تنقل البيانات من وإلى كابل البيانات . Data Cables

نعود للمسجلات ثانيا و هي أهم وحده نتعامل معها فى الأسمبلى ، تكون المسجلات موصله ببعضها و بعض و تصل مباشرة بالمسار الرئيسى للبيانات و المسار الرئيسى للتحكم . Data Bus and Control Bus تعتبر مسجلات المعالج AX , BX , ..... هي من أهم مكونات المعالج ، معظم المسجلات تحتوى - بالإضافة إلى وحدات تخزين البيانات داخلها - Flip Flops على بينات Bins للتحكم بالبيانات من و إليها فمثلا :-

-تحتوى على بن bin للقراءه  
-تحتوى على بن للكتابه  
-تحتوى على بن تصفير ( جعل القيمة التي يحملها صفر)  
-تحتوى على بن زياده ( زيادة القيمة الى بداخلها بمقدار 1 ) وهكذا ..... و تتيح بنات التحكم هذه للمعالج التحكم بالبيانات الخاصه بكل مسجل على حده ، و سنرى هذه الإمكانيه فى موضوع كيفية نقل البيانات داخل الكمبيوتر .

الجدير بالذكر هنا أيضا ، أن أى وحدة تخزين لها بنات تحكم مشابهه Control Bins بما فيهم الذاكرة والمسجلات و القرصات الصلبه و المرنه و .....

**-كيف يتم تناقل البيانات بين هذه الوحدات و بعضها البعض و بينها و بين المعالج ؟**  
**يوجد على اللوحة الأمثلث مسارات رئيسيه و لا يخلوا منها جهاز كمبيوتر منذ أخترع الكمبيوتر إلى الآن و هما :-**

### 1 - Data Bus

وهو المسار الذى يتم نقل البيانات عليه و يختلف عرض هذا المسار من معالج إلى الأخر (أو بمعنى أصح وأدق - من جيل إلى آخر) ، الجدير بالذكر هنا هو أن المسار يتكون من عدة أسلاك بجانب بعضها البعض و هنا تستخدم لنقل البيانات من و إلى مكونات الكمبيوتر (المعالج - الذاكرة - أجهزة الإخراج و الإدخال ) ، يكون كل من هذه المكونات موصل مباشرة بهذا المسار، عرض هذا المسار دائما يتساوى مع عرض مسجلات المعالج و عرض الذاكرة (بمعنى أن الأجهزة القديمه 16 بت ، كان عرض كل مسجل 16 بت و عرض الذاكرة 16 بت و عرض مسار البيانات Data Bus 16 بت أيضا ، و بنفس الطريقه مع 32 بت و 64 بت الذى ظهر قريبا)

### 2 - Address Bus

هذا المسار مسؤول على حمل قيم العناوين و هو يمثل (الورقه التى تكون بيدك عندما تذهب لتبحث عن بيت الرجل الذى يريدك والدك و كتب لك العنوان بهذه الورقه )

يتصل بهذا المسار المعالج (الذى يكتب العناوين) و وحدات التخزين القابله للعنونه (الذاكرة و المسجلات و أجهزة الإدخال و الإخراج )

الجدير بالذكر هو أن عرض هذا المسار يحدد كمية الذاكرة التى يمكن ان يدعمها النظام ، فمثلا فى المعالجات 8086 كان عرض هذا المسار 20بت ، إذن عدد الاحتمالات التى يمكن أن تمثل على هذا العدد و هو يعبر عن الأماكن التى يمكن عنونتها باستخدامه  $2^{20} = 1$  ميغا ذاكره تدريب : ما هو أقصى حجم ذاكره يمكن أن يدعمه جهازك الحالى إذا علمت أن عرض هذا المسار 32 بت؟؟؟

### 3 - Control Bus

وهو المسار التى ترسل عليه إشارات التحكم التى تكلمنا عليها فى الجزء الثانى من هذا الدرس ،،، و يكون هذا المسار متصل به كل مكونات الكمبيوتر و ذلك ليتمكن التحكم بالبيانات من جانب المعالج ،،، إذن كيف يمكن نقل البيانات ،،

لوافتراضنا ان المعالج ينفذ التعليمه : `MOV AX,[1000]`والتي معناها "إنقل محتويات المكان بالذاكرة المعنون ب 1000 داخل مقطع البيانات الحالى إلى المسجل AX يقوم المعالج بحساب العنوان الحقيقى من العنوان 1000 فى التعليمه و عنوان المقطع (راجع جزء العنونه فى دروس الأسمبلى) و يقوم بوضع الناتج فى مسجل العناوين Address Register وهو المسجل المسؤول عن العناوين ، ثم يرسل المعالج إشارة قراءة من الذاكرة، ستقوم دوائر البحث فى الذاكرة بالبحث عن هذا العنوان المسجل فى مسجل الذاكرة AR ومن ثم تحميل ما يحتويه هذا المكان على مسار البيانات Data Bus ، فى نفس الوقت يكون المعالج قد أرسل رسالة كتابه فى المسجل AX ومن ثم تنتقل البيانات التى على مسار الداتا (والتي خرجت من الذاكرة لتوها) إلى المسجل و بهذا تكون التعليمه تم تنفيذها وبهذه الطريقه يتم نقل البيانات جميعها عبر مكونات الكمبيوتر ، ونلخص هذا فى الخطوات الآتية -

- 1- تحديد عنوان مكان البيانات المستخدمه فى التعليمه الحاليه .
  - 2 - تحديد المكان الذى سيتم إرسال البيانات له و من ثم فتح بن الكتابه به .
  - 3 - تكون البيانات متاحه على مسار البيانات والتي ستنقل إلى المكون المفتوح لديه بن الكتابه حاليا .
- يقوم المعالج بالتحكم فى مسار هذه الإشارات و التحكم بها و موازنة كل خطوه من الخطوات السابقه

😊 Synchronization لينتج لك ما تعمل عليه الآن

## -مما يتركب المعالج؟؟

يتركب المعالج من الأتى :-

- المسجلات Registers
- وحدة الحساب و المنطق Arithmetic and Logic Unit (ALU)
- المسارات المختلفه بداخله و التى تحدثنا عنها سابقا Internal Bus
- وحدة التحكم Control Unit
- باقى المكونات الأخرى ....

## البيانات و أنواعها ،،،

-كما لابدأن تعلم أختى/أختى الكريم/الكريمه أن كل ما هو مخزن على وحدات التخزين المختلفه فى الحاسب تكون مخزنه على هيئة 0 و 1 و تكون فيما بينها مجموعه من (البيانات والتعليمات)

-فمثلا ، ملفات ال txtتحتوى على بيانات ، ملفات ال doc تحتوى على بيانات ، ملفات exeتحتوى على تعليمات و بيانات ، ملفات لل dllتحتوى على تعليمات فقط و هكذا

-تتواجد التعليمات عادة فى الملفات التنفيذيه ومكتبات التشغيل (exe , dll , ocx , com , bin)

-يمكن التفريق بين البيانات و التعليمات باستخدام الحقائق الأتية :-

- الملفات التشغيليه تحتوى على أكواد تعليمات معلومه لدى المعالج و نظام المعالجه ككل .
- الملفات التشغيليه لها إمتدادات معروفه . (ocx , dll , exe , ....)
- الملفات التنفيذيه لها تركيب معين ، كل البرامج العامله أمامك الآن لها نفس التركيب ،ملحوظه :لزيادة المعرفه حول تركيب الملفات التنفيذيه من الأنواع (exe , com) ، يرجى قراءة دروس Xacker فى موضوع صناعة الفيروسات.
- فيما عدا ذلك ، يعتبره نظام التشغيل بيانات .مثال - برنامج بسيط يقوم بطباعة رسالة hello على الشاشة ومن ثم ينتظر لأن يضغظ المستخدم على حرف من لوحة المفاتيح "Press Any Key To Exit" لينتهى البرنامج ،

### ----- التحليل -----

- هذا البرنامج مكون من التعليمات الآتية :

- تحضير لبدأ البرنامج و ذلك بتحميل مسجل المقطع بعنوان مقطع الكود للبرنامج .
- تعليمات الطباعه على الشاشة .
- تعليمات إنتظار حرف من وحدة الإدخال (لوحة المفاتيح . (Keyboard

- ويحتوى على البيانات الآتية :

- + النص . "Hello"
- + النص "Press Any Key To Exit"

### ----- نهاية التحليل -----

- البيانات ممكن أن تكون صور ، ألوان ، أصوات ، نصوص ، نصوص مشكله ، بيانات مبهمه .....  
- يمكن التفريق بين أنواع البيانات المختلفه (بالنسبه لنظام التشغيل:-) (

- إمتداد الملف نفسه الذى يحتوى على البيانات .
- تركيب الملف نفسه الذى يحتوى على البيانات .
- أن تكون البيانات التى بداخل الملف لهاصيغ مفهومه للبرنامج المشغل لها و إلا سيقوم بإضهار رساله خطأ

للمزيد عن تركيب الملفات ، يرجى زيارة المواقع الآتية :-

<http://www.onicos.com/staff/iz/formats>  
<http://whatis.techtarget.com/fileFormatA>  
<http://myfileformats.com/>

- لا يمكن للبيانات أن يكون لها أهميه إلا بوجود التعليمات (البرامج المشغله) .
- لا يمكن للبرامج أن تكون لها فأنده إلا بتعاملها و معالجتها للبيانات المختلفه .
- أى برنامج يتكون من جزء بيانات **Data Segment** وجزء تعليمات **Code Segment** .

## CODE

Each program must consist from some code Instructions and  
some data blocks

## كيف تتم معالجة البرامج و تعليماتها؟؟

-نظام التشغيل هو المسؤول عن تحضير الملفات التنفيذية و إختبار صحة تركيبها و من ثم تكون البرامج جاهزه للتحميل على الذاكره .

-لا يتعامل المعالج مع الأقراص الصلبه مباشراً ، فالمعالج لايعرف إلا الذاكره و المسارات و المسجلات و فقط ،

-نظام التشغيل المسؤول عن تحميل البرامج فى الذاكره و إصدار الأمر للمعالج بأن يبدأ بتنفيذ تعليمات البرنامج .

-علمتم أختوى من القسم السابق (البيانات و أنواعها) فإن كل برنامج يتكون من تعليمات و بيانات ، وكل نوع من الملفات له الصيغه المعروفه من جانب نظام التشغيل، حيث يقوم نظام التشغيل بتحميل هذه التعليمات و البيانات فى الذاكره و من ثم يسلم المعالج زمام الأمور لتنفيذ هذه التعليمات .

-يحتوى المعالج على مسجل مقطع، يكتب به عنوان مقطع الكود الحالى بالذاكره و الذى قام نظام التشغيل بتحميلها فيه .

-يحتوى المعالج على مسجل مقطع يحتوى بداخله على عنوان مقطع البيانات الحالى فى الذاكره و الذى قام نظام التشغيل بتحميله به .

-يحتوى المعالج على مسجل يوضع به عنوان التعليمه بالذاكره التى عليها الدور فى التنفيذ .

## خطوات تنفيذ أى برنامج :-

1 - يتأكد نظام التشغيل من أن تركيبه الملف التنفيذى سليمه و معروفه (ملفات ال exe مثلاً تركيبها غير معروف لدى نظام التشغيل لينوكس لذا لا يستطيع التعامل معها ولا تشغيلها إلا بإضافة مفسرات لها )

2 -يقوم نظام التشغيل بقراءة نوع الملف من ناحيت إحتياجاته للذاكره (توجد عدة أنواع تختلف فى عدد مقاطع الدااتا و الكود الذى سيحتاج إليها البرنامج ويكون المبرمج مسؤول أو لغة البرمجه العاليه المستوى مسؤوله عن كتابة هذه الأنواع فى أول البرنامج) - إنتظر دروس [الأسمبلى](#) القادمه ، سوف اشرح هذا الجزء برمجيا بالتفصيل

3 -يقوم نظام التشغيل بالبحث لديه فى الذاكره على المقاطع الفارغه ، فإن لم يجد مساحه كافيه لتحميل البرنامج و بياناته ، فإنه يصدر رساله خطأ .

4 -يقوم نظام التشغيل بتحميل تعليمات البرنامج و بياناته (لو وجدوا فى ملف واحد -بمعنى أنه يمكن لبرنامج أن يضع بياناته الأساسيه بجانبه فى ملف منفصل و من ثم يقوم البرنامج نفسه بطلب تحميلها بعد ذلك. )

5 -تحميل كل من مسجلات المقاطع و مسجل التعليمات Instruction Register IR بالقيم المطلوبه ، والجدير بالذكر هنا أن مسجل التعليمات يتم تصفيره فى حالة بدأ تشغيل برنامج جديد وذلك لأن أو تعليمه فى البرنامج يجب أن توضع فى المكان صفر من مقطع الكود بالذاكره .



6- يبدا المعالج بأخذ أول تعليمة والمقابلة للعنوان صفر داخل مقطع الكود و يقوم بتنفيذها ومن ثم يزيد قيمة مسجل التعليمات بواحد (IR = IR + 1) وذلك للتعليمة الآتية وهكذا حتى يصل إلى تعليمة التي تسلم نظام التشغيل التحكم ثانياً .

7- الجدير بالذكر أن نظم التشغيل الحديثه لا تترك للمعالج (البرنامج) السيطرة الكامله على الجهاز و مكوناته و لكن تختبر حالة البرنامج كل وقت معين و ذلك للتأكد من أنه يعمل بحاله جيده و لا يسبب أخطاء أو تلف للعتاد (تتذكر أنه فى الدوس لو قمت بكتابة برنامج صغير يدخل فى دوره لا نهائيه Infinite Loop و قمت بتشغيله على الدوس ، لن تستطيع أن توقف البرنامج إلا بعمل إعادة تحميل للجهاز كله . ( Restart

8- معظم نظم التشغيل الحديثه تقوم بتغيير تعليمات البرنامج و ذلك لجماية مكونات الجهاز الحقيقيه من الوصول المباشر لها .

### كيف يقوم المعالج بتنفيذ تعليمة ما ؟

-كل تعليمة لها تركيب ثنائى مختلف عن الباقي ، ومن ثم يستطيع المعالج التمييز بين كل تعليمة و أخرى .

-يدخل التركيب الثنائى على المعالج لتفسيرها ومن ثم تنفيذها مباشراً ،

-مثال :- دعنا نتخيل أن القيمة (01110) هي المقابلة للتعليمة ADD AX,BX ،  
-إذن عند وصول القيمة (01110) إلى المعالج للتنفيذ ، فإن مكونات المعالج المنطقية تقوم بإدخال محتويات المسجل BX و المسجل AX على وحدة الحساب والمنطق

و من ثم تشغيل Activate عملية الجمع داخل وحدة الحساب و المنطق ALU

-تكون وحدة الحساب و المنطق موصله بمسار البيانات ، فيتم كتابة الناتج على المسار ،

-ومن ثم فتح Bin الكتابة على المسجل AX و بهذا يكون تم إنهاء تنفيذ هذه التعليمة .

-و بنفس الطريقة يتم تنفيذ كل التعليمات و لكن النظرها لسابقه بسيطه جدا عن الواقع

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



### دورة الأسمبلي - اليوم الأول

ماهي الأسمبلي :-

في قديم الزمان أيام بدايات الكمبيوتر كانت برمجة الكمبيوتر تتم بواسطة لغة الآله (Machine Language اختصاراً ML) لغة الآله هي اللغة التي تفهمها الآله مباشرة دون الحاجة الى تفسيري وهي تخزن بصورة ثنائية [ تركيبة من الأصفار والواحد ] في الذاكرة على شكل تعليمات ووسائط تأخذ كل واحد منها عادة مقدار 8بت (=1 بايت ) وكان هذا النوع من البرمجة صعب جداً عندها طور المبرمجون أول لغة برمجة وهذه اللغة فكرتها بسيطة جداً حيث أنه بدل أن تكتب رموز الآله يتم كتابة كلمات مختصرة تدل على نوع العمليه مثال ( MOV,ADD,CMP) ثم برنامج بسيط يتم تحويل هذه الشفرة الى لغة الآله باستخدام تخطيط واحد-الى-واحد أي أن كل سطر أو عبارة في الأسمبلي تحول الى تعليمة واحدة مقابله في لغة الآله (مثال بدل كتابة 01100000000101 يتم كتابة ( mov al,5) يعرف البرنامج الذي يقوم بعملية التحويل بالأسمبلي Assembler، علماً بأن هناك عدة أنواع من الأسمبلي كل نوع يختص بتقنية معينة وبعائلة معينة من المعالجات ونحن هنا بصدد تعلم البرمجة بالأسمبلي للمعالجات المبنية على تقنية IBM-PC والمنتجة من شركة إنتل وهي العائلة 80×86 ويرمز لها اختصاراً X86 وهي تضم :

( 80286 / 80186 / 8088 / 8086 ) لمعالجات ال 16 بت و ( 80386 / 80486 /

80586=بنتيوم1 / 80686=بنتيوم2 =80786=بنتيوم3=80886 / بنتيوم4 ) لمعالجات ال 32 بت وسوف أتطرق في دروس متقدمة الى المعالج أنتيوم 64 بت الميني بتقنية جديدة كلياً لمن يرغب بمعرفة مسبقة لهذا المعالج الجديد كذلك سوف أتطرق بأذن الله الى **الكروس أسمبلي** وهي مجموعة برامج خاصة مصممه للتحويل من لغة أسمبلي لعائلة معالجات معينة الى عائله أخرى .تعريف لغة الأسمبلي الأسمبلي هي لغة برمجة تتكون من سلسلة من التعليمات المتتابعة كل تعليمة فيها تحول الى تعليمة مقابلة بلغة الآله .تعريف الأسمبلي :-

الأسمبلي هو برنامج يقوم بتحويل التعليمات المكتوبة بلأسمبلي الى لغة الآله .لماذا أريد استخدام الأسمبلي :-

بتعلمك لغة الأسمبلي فأنت تكشف النقاب عن الأسرار المخفيه وراء الكمبيوتر وتصبح قادراً على الفهم تماماً كيف يعمل المعالج وكيف يعمل البرنامج وبذلك تزيد خبرتك كمبرمج وبالطبع فإن الأسمبلي أقوى من اللغات العالية المستوى في التعامل مع العتاد وتعطيك مرونة عالية وقدرة وصول الى أشياء لم تكن تستطيع الوصول اليها من قبل ، كذلك هناك نوعيات من البرامج لايمكن البرمجتها بالأسمبلي مثل الدرايفات(سواقات) الأجهزة ، كذلك فإن الأسمبلي يعطيك برامج سريعة جداً ، وبالطبع فإن بناء برنامج متطور بالأسمبلي أشبه بحفر حفرة بواسطة الملعقة فالبرغم أنك تحفر الا أنك أنتاجيتك قليلة ولكن من المحبذ جداً برمجة بعض الدوال و الأجزاء من البرامج بالأسمبلي وبقية البرنامج بواسطة لغة عالية المستوى مثل السي. ++

بسم الله الرحمن الرحيم  
**السلام عليكم ورحمة الله وبركاته**



دورة الأسمبلى - اليوم الثاني

كيان الحاسوب الصلب :-

يتألف الحاسوب بشكل أساسي من اللوحة الأم Mother Board والمعالج Microprocessor وذاكرة القراءة فقط ROM=Read-Only Memory وذاكرة الوصل العشوائي=الرام RAM=Random Access Memory ووحدة التغذية Power Supply والمنافذ التوسعية Expansion Slots مثل فتحات توصيل الكروت ( كروت الشاشة والصوت و ما إلى ذلك . )

**المعالج :-**

يمثل المعالج عقل الحاسوب وهي الوحدة المسؤولة عن القيام بإدارة الحاسوب والقيام بالعمليات الرياضية والمنطقية ونحن هنا كما أوضحت ندرس معالجات أنتل من العائلة X86 لأنها العائلة الأشهر والأكثر استخداماً بين الناس .

**وحدة التنفيذ ووحدة ملاءمة الممر :- Execution Unit And Bus Interface Unit**

يتألف المعالج من وحدتين هما وحدة التنفيذ Execution Unit وأختصاراً EU ومهمتها تنفيذ التعليمات ، ووحدة ملاءمة الممر Bus Interface Unit وأختصاراً BIU ومهمتها نقل البيانات والمعطيات الى وحدة التنفيذ . تحتوي وحدة التنفيذ على وحدة الحساب والمنطق Arithmetic And Logic Unit وأختصاراً ALU ووحدة التحكم Control Unit وأختصاراً CU ومجموعة من المسجلات.

تتألف وحدة ملاءمة الممر من وحدة التحكم بالممر Bus Control Unit ومسجلات المقاطع Segment Registers ورتل=كيبو التعليمات ( Instruction Queue الرتل أو الكيو هو نوع من إدارة الذاكرة تكون فيه المعلومة الداخلة أو لاخارجة أولاً ) . FIFO=First In First Out .

وتقوم وحدة ملاءمة الممر بعمليات التحكم بالممر ونقل المعطيات بين كل من وحدة التنفيذ والذاكرة وأجهزة الإدخال والأخراج الخارجية، كما تقوم مسجلات المقاطع بعملية التحكم في عنوانة الذاكرة .

تضع وحدة ملاءمة الممر تضع التعليمات في رتلها المخصص لها في وحدة التنفيذ بعد أن تقوم بجلبها من الذاكرة . يخصص رتل التعليمات لوضع التعليمات فيه بعد جلبها من الذاكرة بواسطة وحدة ملاءمة الممر ، ولذلك يوجد دائماً رتل من التعليمات جاهزه لتنفيذها من قبل وحدة التنفيذ . تعمل وحدة التنفيذ ووحدة ملاءمة الممر على

التوازي (في نفس الوقت) ، بينما تحتفظ وحدة ملاءمة الممر بخطوة نحو الأمام، فعندما تقوم وحدة التنفيذ بتنفيذ تعليمة ما ، تعمل وحدة ملاءمة الممر أما على جلب تعليمة من الذاكرة ووضعها في رتل التعليمات لكي تنتظر دورها في التنفيذ ، أو على جلب معطيات من الذاكرة أو أحد أجهزة الإدخال أو الأخراج . وخلافاً للطريقة التسلسلية في المعالجة فإن هذه العملية تحقق حدوث عمليتي الجلب **fetching** والتنفيذ **execution** في وقت واحد الأمر الذي يزيد بدورة من سرعة المعالج .

### ذاكرة القراءة فقط :- ROM = Read-Only Memory

وهي عبارة عن شريحة دائرة متكاملة IC تحوي على ذاكرة فيها بيانات غير قابلة لإعادة الكتابة عليها (أفترضياً - شرائح ال ROM الحديثة يمكن إعادة الكتابة عليها بطرق مختلفة ) ، تحتوي هذه الذاكرة على برنامج ال BIOS=Basic Input Output System أو نظام الإدخال والأخراج الأساسي، ولا يمكن للمعالج القراءة من هذه الذاكرة مباشرة ولكن أول شئ يفعل المعالج عند تشغيله في عملية الاستنهاض هي تحميل البيانات الموجودة في الروم ونقلها في الرام أو بالأحرى الى القسم الأخير من الرام ذا العنوان الأكبر . تتجلى فائدة ال BIOS في القيام بعملية الفحص الذاتي عند الاستنهاض POST=Power On Self Test بالإضافة الى تحميل برنامج محمل نظام التشغيل بالإضافة الى توفير دوال ومقاطعات قياسية في ذاكرة الرام تستطيع أن تستخدمها البرامج للرسم على الشاشة مثلاً أو التعامل مع لوحة المفاتيح أو القراءة والكتابة من والى القرص الصلب .

### ذاكرة الوصول العشوائي :- RAM = Random Access Memory

هذه الذاكرة مهمة جداً حيث أن أي برنامج لا يمكن أن يعمل الا اذا حمل الى هذه الذاكرة كذلك فهي تستخدم لحفظ المتغيرات وحفظ برامج النظام الأساسية ومنها جداول المقاطعات والمقاطعات أنفسها والرويتينات الفرعية ..... الخ ولا يتم استخدام القرص الصلب لحفظ مثل هذه المعلومات لأن وقت الوصول فيه أبطأ بكثير من ذاكرة الرام (ولو أن القرص الصلب يستخدم في توفير ذاكرة افتراضية عن طريق القيام بعمليات مبادلة للصفحات مع ذاكرة الرام ) ، وبالطبع فإن هذه الذاكرة يمكن الكتابة اليها أو القراءة منها عن طريق عنوانها . فباستخدام العنوان يمكننا أن نصل الى مكان محدد في الذاكرة لنعمل عليه كل عمليات التحرير المطلوبة .

المقدمه السابقه سنتخذ أنها كلام نظري وغير مفيد و لكن لتعلم أحي أنه هو أساس لغة الأسمبلي بل أساس الحاسب ككل ولو أنك أردت أن تفهم حقيقة التعامل داخل الكمبيوتر وحلقة الوصل بين البرامج والأنظمة التي تعمل عليها من جهه و العتاد من جهه أخرى فعليك الإنتباه له و التأكد التام من أنك إستوعبته جيداً وفهمت كل حرف به ،

### تمثيل الأعداد والحروف

قد تتسائل ما علاقة تمثيل البيانات والعد الثنائي بالأسمبلي ؟ حسناً كما وضحت من قبل فإن الأسمبلي هي لغة قريبة جداً من لغة الآله وهي لغة منخفضة المستوى تتعامل مع العتاد والمعالج بصورة مباشرة ولكي نحقق فهماً أوسع لهذه اللغة يجب أن نفهم بعض الأشياء المهمة جداً في بنية المعالج .

العد الثنائي :-

يتم تمثيل الشفرات والبيانات في ذاكرة الكمبيوتر كتوالييف من الشحنات الكهربائية تأخذ قيمتين الأولى وهي وجود الشحنة ويرمز لها ب ON أو صحيح TRUE أو '1' والأخرى وهي غياب الشحنة ويرمز لها ب OFF أو خطأ

FALSE أو '0' ، ووجود الشحنة يكون عادة بين 4.5 الى 5.5 فولت ( المعالجات الحديثه بين 2.5 الى 3.5 فولت ) وغياب الشحنة يكون بين +0.5 فولت و -0.5 فولت .

وحدات الذاكره الأساسيه في الذاكره والوحدات التي سنتعامل معها كثيراً هي :-

1بت = بت و هو إما يساوى 0 أو 1 (وجود شحنه أو عدم وجود شحنه ) ( عدد 2 احتمال Bit )  
8بت = 1 بايت (عدد 256 احتمال أى ما يعادل 2 أس 8 Byte )  
16بت = 2بايت(عدد 65536 احتمال أى ما يعادل 2 أس 16 Word )  
32بت = 4 بايت(عدد 4294967296 احتمال أى ما يعادل 2 أس 32 = 4 جيجا احتمال )  
DWord=DoubleWord  
64بت = 8 بايت ( عدد 2 أس 64 احتمال = 16 جيجاجيجا احتمال )

وحدات أكبر و تختص بقياس البيانات فى مختلف أجزاء الكمبيوتر :-

1024بايت =KB =الكيلوبايت  
1024كيلوبايت =MB =الميغابايت  
1024ميغابايت =GB =الجيجابايت  
1024جيجابايت =TB =التيرابايت  
1024تيرابايت =PB =البيتابايت  
1024بيتابايت =EB =الأكسابايت  
1024أكسابايت =ZB =الزيتابايت  
1024الزيتابايت =YB =اليوتابايت

الأسكي كود :-ASCII

يتم في الحاسوب وبقية توحيد استخدام الرموز استخدام شفرة الأسكي كود (حالياً يعمل على تبني شفرة ال unicode وهي تسمح بتعدد اللغات في مستند واحد حيث يتم تمثيل كل حرف باستخدام كلمة واحده=2بايت) كلمة ASCII هي اختصار ل :

American National Standard Code For Information Interchange

ويتم استخدام هذا الكود الموحد لتسهيل تناقل البيانات ويمثل كل رمز فيه بعدد ثنائي بطول 1بايت=8بت=256أحتمال .

مما يعنى أن أى حرف نكتبه أو نراه على الشاشة يكون له مقابل رقمى يسمى الأسكى كود الخاص به و سنستخدم هذا الموضوع كثيراً كثيراً جداً ، لذلك يجب عليك أخذ نظره على جدول الأسكى التالى :-

الجدول الكامل للأسكى كود من الرقم 0 إلى 127

جدول الأسكى الموسع من رقم 128 إلى 255

( انظر الملحقات فى نهاية الكتاب توضح جداول الاسكى )

## طريقة كتابة الأرقام في الأسيمبلر :-

لكتابة عدد ثنائي يوضع في آخر الرمز ( b ) لدلالة على أنه باينري مثال Binary=11010010B أما العدد العشري فلا يحتاج الى إضافة وأما العدد لأساس 8 فيكتب مع المرمز (Q) في نهايته Octal=1276Q أو الرمز (O) في نهايته Octal=1276O أما العدد السداسي عشر فيكتب بوضع H في نهايته Hexadecimal=0AB9CDH مع مراعاة وضع 0 إذا كان العدد يبدأ بحرف كما المثال .

يجب أن تعرف الفرق بين تخزين الرقم كرقم أو تخزينه كنص فتخزين الرقم 201 مثلاً كرقم سسيأخذ بايت واحد وهو جاهز للقيام بعمليات رياضية ومنطقيه عليه أما تخزينه كنص فسيأخذ ثلاثة بايت في البايث الأول سيخزن الرقم الخاص بالأسكي كود للرمز '2' وكما قلت يخزن كرقم يدل على الرمز أما البايث الثاني فسيخزن رقم الأسكي كود للرمز '0' أما البايث الثالث فيأخذ القيمة الخاصة بالرمز '1' في الأسكي كود أي أن الرقم خزن بطريقة "102" وليس 102 وهذه الطريقة ليست جاهز للجمع أو الطرح ولكنها ممتاز للطباعة على الشاشة ويمكن تحويل النص الى رقم والعكس .

## الأعداد ذات الأشاره :-

يتم تخزين الأعداد ذي الأشاره كالتالي :-

العدد موجب إذا كانت البت الأخيره صفر وقيمة الرقم هي باقي البتات أي لو أخذنا رقماً من بايت واحد فإن البت رقم 7 (الثامن - الترقيم يبدأ من الصفر ) يجب أن تكون صفراً ليكون العدد موجب أما البتات من 0 الى 6 ( السبعة الأولى ) فتشكل قيمة الرقم أما إذا كان العدد سالب فإن البت الأخيرة تساوي واحد أما قيمة الرقم فتساوي سالب المكمل الثنائي للعدد أي لو أخذنا رقم مخزن في واحد بايت مثال = 11110110 بما أن البت السابعه = 1 فإن الرقم سالب / نأخذ الآن المكمل الثنائي للعدد وهو 00001001 / القيمة تساوي 00001010-أي سالب عشرة .

## ملاحظات مهمه :-

- 1 البايث في نظام التمثيل العددي بدون إشاره ( على إعتبار أن العدد موجب ) نطاقه من 0 إلى 255 .
- 2 البايث في نظام التمثيل العددي بالإشاره ( البت الأخير للإشاره ) نطاقه من 128- إلى +127 .
- 3 عند تعريف متغير من النوع Integer في السى أو البيسيك فإنه يحجز للرقم 2 بايت .

إلى هنا ينتهى الدرس الثانى ،

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



### دورة الأسمبلى - اليوم الثالث

كما ذكر من قبل أن لغة الأسمبلى تتعامل مع الهارد وير ، كيف هذا ؟ وما معنى هذا ؟  
سأسرد كيفية التعامل العامه مع الهاردوير عاماً :-

البيانات المخزنه أو التى يتعامل معها أى جزء من أجزاء الهارد وير تنقسم إلى قسمين :-  
1- البيانات: هي أرقام ، حروف ، ... كل هذا مخزن بالأسكى كود.

2- أوامر : وهى أيضاً أصفار ووحايد مخزنه بطريقة ما يتم عمل تنفيذ لها على البيانات السابقه . Decoding

مثلا عند تشغيل برنامج ما يطبع كلمة "Hello" على شاشة الكمبيوتر عند الضغط على حرف 'P' ، فإننا هنا نتعامل مع :-

- **1 البروسسور** : هو وهو المعالج الذى يقوم بتنفيذ كل تعليمه و التوصيل ما بين كل جوانب التعليمه الواحده .  
- **2 الرامات** : وتتعامل معها هنا على اساس عنوان كلمة "Hello" ، فمثلا نقول أن هذه الكلمه مخزنه في العنوان H 1000 وهذا يعنى عند عد 1000 أماكن بالنظام السداسى عشر (أى 4096 بالنظام العشرى العادى ) ، فإن أحرف الكلمه ستجدها مخزنه من عند هذا المكان ، ويتم أخذ حرف حرف إلى أن يصل إلى حرف معين و هو الذى ينهى عملية الطباعة و هو '\$' فعندما يجدها الحرف يقوم بإنهاء عملية الطباعة و كما نرى أن هذه طريقة التعامل مع النصوص عند كتابتها أو طباعتها على الشاشة عاماً.

- **3 الشاشة** : يكون هناك مخزن على كارت الشاشة أو الذاكره الخاصه بالشاشه على الرامات ، يتم وضع حرف حرف بها و يتم إعطاء أمر الطباعة وهو يكون كتابة سلسله من الأصفار و الواحيد الخاصه و التى تعنى بالنسبه لنظام التعامل مع الشاشه بأن إطبوع هذا الحرف ، طيب أين نطبع هذا الحرف ، يوجد مخزن Buffer للإحداثى السينى و آخر للإحداثى الصادى ، فعند طباعة حرف على الشاشه فيتم زيادة الإحداثى السينى بواحد (على اساس أن عرض كل حرف 1 وحده) و عند الوصول إلى آخر السطر (تخطى عدد معين من الأحرف وهو طول الشاشه) فإن النظام نفسه يتم زيادة الإحداثى الصادى بواحد و تفسير الإحداثى السينى للبدأ من أول السطر وهكذا .

- **4 الكيبورد** : يجب وجود نظام ايضا يسمع إلى الحروف أو الأزرار التى يتم ضغطها على الكيبورد (فى الحقيقه : عند الضغط على أى زر على الكيبورد فإن الأسكى المقابل له يسرى عبر الأسلاك إلى وحة التعامل مع الكيبورد على اللوحه الأم و من ثم يخزن على ال Buffers التى توجد هناك . ( بعد سماع كل حرف من هذه المخازن Buffers فإننا نختبر الأسكى لهذا الحرف فلو أنه يقابل قيمة 80 ( المقابل لحرف 'P' فى جدول الأسكى كود ) فنقوم بتشغيل دالة الطباعة من المكان 1000 فى الذاكره ، هذا مثال بسيط لما يتم فعليا داخل الحاسب عند تشغيل هذا البرنامج البسيط ، تخيلوا ما يحدث عند تشغيل البرامج المعقده الكبيره ، فعلا كان يجب تطوير اللغات فوق بعضها ، كل تعليمه فى لغه عاليه المستوى تقابلها مجموعه تعليمات فى اللغات الأقل مستوى و هكذا ، فقط تتميز لغة الأسمبلى بأن كل تعليمه تقابل تعليمه فى لغة الآله بالظبط .

توجد وحدات تخزين داخل البروسسور يتم التعامل معها مباشرة في كل التعاملات داخل الحاسب وهو تماثل ال Buffers في أي جزء آخر من الهاردوير وتسمى مسجلات Registers ويقوم تركيبها من لبوابات المنطقيه عالية السرعة High Speed Logic Gates، وهي أهم أجزاء البروسسور التي يتم التعامل معها في نظام السوفتوير كله (التعامل مع هذا المسجلات مباشرة من خلال لغة الآله ومن ثم لغة الأسمبلى )

نقول أن البروسسور يدعم 32 بت ... هذا يعني أن طول المسجلات الرئيسييه = 32 بت .  
مثلا البروسسور ال 8088 كان معالج 8 بت ... وهذا يعني أن طول المسجلات فيه = 8 بت .

**الجزء التالي يتحدث على المسجلات بالتفصيل فإنتبه إليه :-**

### **المسجلات : REGISTERS**

الكمبيوتر يحتاج في تعاملته الى ذاكرة سريعة جداً ومتصلة بالمعالج مباشرة حتى يمكن له أن يخزن فيها المعلومات المطلوبة لعملية حسابية معينة أو عداد لحلقة معينة ، هذه الذاكرة تعرف بالمسجلات REGISTERS وهي ذاكرة سريعة جداً تفيد المعالج في إجراء العمليات بسرعه وكفاءة أكبر هناك خمسة أنواع أو تصنيفات للمسجلات ( تختلف هذه التصنيفات قليلاً من مرجع لآخر ) وهي مسجلات الأغراض العامة -General Purpose Registers ( تعرف في بعض المراجع بمسجلات المعطيات ) Data Registers وهناك مسجلات الأقسام Segment Registers والمسجلات الدليلية Index Registers ومسجلات التأشير Pointer Registers بالإضافة الى مسجلات الحالة والتحكم Status and Control Registers.

**المخطط 4 1 : رسم تخطيطي يوضح المسجلات في معالجات ال16 بت**

**المخطط 4 2 : رسم تخطيطي يوضح المسجلات بعد توسعتها في معالجات ال32بت**

**( ستجد ملحقاً في نهاية الكتاب باسم Registers يوضح جدول المسجلات )**

### **[1] مسجلات الأغراض العامة :- General-Purpose Registers**

وهي عبارة عن كل من المسجلات AX و BX و CX و DX، طول كل منها 16بت أي كلمة أي 2بايت البايت اليساري فيهما يعرف بالعلوي (High) أما البايت الأيمن فيهما فيعرف بالمنخفض (Low) فمثلاً المسجل AX يتألف من مسجلين العلوي وهو AH والمنخفض وهو AL، أن تعديل المسجل الجزئي سوف يؤثر في المسجل الأم لأنه جزء منها وأيضاً التعديل في المسجل الأم سوف يؤثر في المسجل الجزء ، تم توسيع المسجلات في معالجات ال32 بت مع بقاء المسجلات نفسها ولكن كل منها أصبح جزء من مسجل موسع بطول 32بت وهي EAX,EBX,ECX,EDX، أي أن المسجل EAX هو بطول 32بت وكجزء منه هناك المسجل AX بطول 16بت والذي يتألف هو الآخر من مسجلين هما AL و AH بطول 8بت لكل منهما.

**المسجل ( AX مسجل المرمك :- Accumulator Register )**

هذا المسجل كان من أهم المسجلات في معالجات ال 8بت القديمة جداً حيث كانت كل العمليات الرياضية والمنطقيه تجري من خلاله ولذلك كان يسمى بمسجل المرمك لتراكم النواتج فيه لكن معالجات ال16 بت وسعت المرونة وجعلت كل مسجلات الأغراض العامة تستطيع أن تجري من خلالها العمليات الرياضيه والمنطقيه الا أن المسجل AX مازال المفضل لأجرائها حيث أن استخدام المسجل AX أو أحد أجزاءه يؤدي مع بعض التعليمات الى توليد



شفره أقل اختصاراً (الفرق بايت واحد فقط لكل تعليمية) ، يمكن استخدام المسجل AX كمسجلين هما AL و AH حيث تعرف ال 8بتات الأولى التي في اليسار بالمسجل AL وال 8بتات الأخيره التي في اليمين بالمسجل AH ، أما بالنسبة ل EAX وهو المسجل الموسع ل AX فهو بطول 32بتويعتبر المسجل AX كجزء منه.

**(المسجل BX) مسجل القاعدة:- (Base Register)**

هو المسجل الوحيد من بين مسجلات الأغراض العامة الذي يمكن استخدامه كدليل (INDEX) ، يمكن استخدام هذا المسجل للعمليات الرياضية والمنطقية وكما المسجلات الأخرى ينقسم هذا المسجل الى قسمين بطول 8 بت هما BL و BH وهو ضمن مسجل أوسع هو EBX بطول 32بت.

**(المسجل CX) مسجل العداد:- (Counter Register)**

يستخدم عادة كعداد ويستخدم هذا المسجل بشكل خاص مع تعليمة التكرار LOOP حيث يعمل كعداد لها وبالطبع يمكن استخدامه في العمليات الرياضية والمنطقية ، وكما المسجلات الأخرى ينقسم هذا المسجل الى قسمين بطول 8 بت هما CL و CH وهو ضمن مسجل أوسع هو ECX بطول 32بت.

**(المسجل DX) مسجل المعطيات:- (Data Register)**

يفضل استخدام هذا المسجل لتخزين المعطيات في عمليات الدخل والخرج والمقاطع وبالطبع فإنه يمكن استخدامه كباقي المسجلات في العمليات الرياضية والمنطقية وكما المسجلات الأخرى ينقسم هذا المسجل الى قسمين بطول 8 بت هما DL و DH وهو ضمن مسجل أوسع هو EDX بطول 32بت.

## [2] مسجلات الأقسام :- Segment Registers

كانت العنوان الحقيقية في معالجات ال 16بت تتم باستخدام خطوط عرض 20بت وهي تكفي لعنونة 1ميغابايت من الرام فقط ولصعوبة التأشير للرام باستخدام مسجلات من 16بت نشأت فكرة الأقسام والعنونة المنطقية وقد قسمت الرام لمقاطع كل منها بطول 64كيلوبايت (الحد الأقصى الذي يمكن عونته ب 16بت) وهذه الأقسام لا يبدأ كل واحد فيها بعد الآخر وإنما هي متداخلة حيث يبدأ كل 16بت قسم جديد وللتأشير على موقع ما يلزمنا عنوان المقطع والذي بطول 16بت بالإضافة الى قيمة الأزرحة من بداية هذا المقطع وهي بطول 16بت أيضاً لذلك لج أمصموا المعالج على وضع مسجلات خاصة بالأقسام الشائعة في البرنامج وهي قسم الشفرة Code Segment وقسم البيانات Data segment وقسم المكدهه Stack Segment وقسم المقطع الإضافي Extra Segment وبالرغم أنه في معالجات ال 32بت يمكن العنونة باستخدام 32بت ذاكره حقيقية أي ميساوي 4جيجابايت من الرام الا أن طريقة الأقسام مازالت موجودة حتى يتم خزن عناوين كثيرة باستخدام 2بايت بدل 4بايت داخل المقطع الواحد مع وجود عنوان مقطع واحد فقط مخزن في المسجل المناسب . (ملاحظة : تم في معالجات ال 32بت إضافة مسجلين أقسام جديدين بطول 16 بت إضافة الى مسجلات الأقسام السابقة والمسجلين هما FS و GS هذان القسمان يمكن استخدامهما كما المسجل ES كمسجل قسم بيانات إضافي )

**قسم الشفرة:- CS - Code Segment**

يحمل هذا المسجل عنوان بداية القسم الخاص بالشفرة في البرنامج .

**قسم البيانات :- DS - Data Segment**

يحمل هذا المسجل عنوان بداية قسم البيانات في البرنامج .

### قسم المكذسة :- **Stack Segment -CS**

يحمل هذا المسجل عنوان بداية قسم المكذسة في البرنامج .

### قسم الأضافي :- **Extra Segment -ES**

يحمل هذا المسجل عنوان بداية قسم أضافي يمكن أن يستعمل هذا القسم الأضافي كقسم بيانات آخر.

### [3] مسجلات التأشير :- **Pointer Registers**

تحتوي مسجلات التأشير وهي بطول 16بت على عنوان من 16 بت وهي تستخدم بشكل خاص مع العمليات الخاصة بالمكذسة وعادة تشكل العناوين التي بها الأزاحة بالنسبة لمسجل قسم المكذسة SS ومسجلات التأشير هي مسجلان مسجل مؤشر القاعدة Base Pointer BP ومسجل مؤشر المكذسة Stack Pointer SP .

### مسجل مؤشر القاعدة :- **Base Pointer -BP**

يعمل هذا المسجل على تسهيل الوصول الى الوسيطات(البارمترات) والتي تحتوي على عناوين ومعطيات والتي دفعت PUSH بشكل مؤقت الى المكذسة عند استدعاء روتينات فرعية من البرامج مع وسيطات ممرة ، وسع هذا المسجل في معالجات ال32بت وأصبح جزء من مسجل أوسع بطول 32بت هو EBP.

### مسجل مؤشر المكذسة :- **Stack Pointer -SP**

يحتوي المسجل SP على كلمة الذاكرة الحالية التي ستعالج في المكذس ، وسع هذا المسجل في معالجات ال32بت ليصبح جزء من مسجل أوسع بطول 32بت هو ESP . هذا المسجل يعدل ألياً بواسطة المعالج مع عملية دفع PUSH أو سحب POP في المكذس ليشير دوماً الى قمة المكذس .

### [4] المسجلات الدليلية :- **Index Registers**

هي مسجلات بطول 16بت تستخدم في عنوانة بيتات مقطع البيانات وكذلك في عمليات التأشير الى السلاسل النصية Strings ، وهناك مسجلات دليليان هما SI و DI وعادة ما يستخدمان معاً دائماً بغيره تنفيذ عملية ما .

### المسجل الدليلي المصدري :- **Source Index -SI**

يستخدم هذا المسجل في التأشير على النص المصدر وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في عنوانة بيتات مقطع البيانات ، وسع هذا المسجل في معالجات ال32بت ليصبح جزء من مسجل أوسع بطول 32بت هو ESI .

### المسجل الدليلي الهدفي :- **Destination Index -DI**

يستخدم هذا المسجل في التأشير على النص الهدف وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في عنوانة بيتات مقطع البيانات ، وسع هذا المسجل في معالجات ال32بت ليصبح جزء من مسجل أوسع بطول 32بت هو EDI .

## [5] مسجلات الحالة والتحكم :- Status and Control Registers

تتكون هذه المسجلات من مسجلين كل بطول 16 بت هما مسجل الأعلام **Flags Register** ومسجل مؤشر التعليمية **IP - Instruction Pointer**.

### مسجل مؤشر التعليمية :- **IP - Instruction Pointer**

يحتوي المسجل **IP** على أزرحة التعليمية التالية التي ستنفذ ، أي أن المسجل عبارة عن مؤشر إلى التعليمية التالية الموجودة في مقطع الشفرة **CS-Code Segment** المنفذ حالياً ، وسعه ذا المسجل في معالجات ال 32 بت ليصبح جزء من مسجل أوسع بطول 32 بت هو **EIP**. وماتعليمات القفز والتكرار التي تعديل للمسجل **IP**.

### مسجل الأعلام :- **Flags Register**

وهو مسجل بطول 16 بت يحتوي على أعلام طول كل منها 1 بت فقط وتستخدم لتعكس حالة آخر عملية أو للتحكم بعمليات معينه وكل علم له موقع معين في ال 16 بت علماً بأن 9 فقط منها محجوزة والباقي غير معرفة وليس عليك حفظ مواقع هذه الأعلام لأنك سستعامل معها بالرمز الذي يرمز لها وكل علم له رمز مكون من حرف واحد فقط يدل عليه ويتبعه أحياناً الحرف **F** اختصار **FLAG** للتوضيح أنه علم ، كل علم إما يكون واحد أي **SET** أو صفر أي **CLEAR**.

### مخطط 4 3 : هذا المخطط يوضح مسجل الأعلام الذي طوله ال 16 بت

( ستجد ملحقاً في نهاية الكتاب باسم **Registers** يوضح جدول المسجلات والرايات معا )

وهناك نوعين من الأعلام هي أعلام الحالة **Status Flags** وأعلام التحكم **Control Flags**.

### #أعلام التحكم :- **Control Flags**

وهي أعلام مخصصة لضبط قيمتها من قبل المبرمج أو العتاد ويتم ضبطها عند القيام بالمقاطعات أو استدعاء البرامج الفرعية أو بعض الأوامر بهدف التحكم بشئ ما وهذه الأعلام هي :

#### 1. علم الاتجاه :- (DF=direction flag)

يؤثر في التعليمات التي تقوم بنقل البيانات مثل **MOVS,CMPS,SCAS** عندما يكون العلم **UP 1** يأخذ الانتقال اتجاه الطبيعي أما عندما يكون **DOWN 0** يأخذ انتقال البيانات اتجاه معاكساً ( قيمة العلم **DF** عند بداية البرنامج ). (لضبط العلم بواحد نستخدم التعليمية **std** ولضبطه بصفر **cld**).

#### 2. علم المقاطعه :- (if=intreupt flag)

يحدد هذا العلم إذا ما كان بمقدور النظام إجراء مقاطعات أو لا ، ويضبط هذا العلم بواسطة أجهزة الهاردوير وكذلك وقت النظام ، تستطيع أنت ضبطه أو تصفيره إذا كنت تريد حدوث مقاطعات خارجية أولاً ، إذا كانت قيمة العلم =1

فهذا يعني مفعل **enable** ويمكن إجراء المقاطعات أما إذا كان صفر فإنه غير-مفعل **disabled** ولا يمكن إجراء المقاطعات ( قيمة العلم **IF** عند بداية البرنامج .(1= لضبط العلم بواحد نستخدم **sti** لضبطه بصفر . **CLI**

### 3. علم المصيدة - : (tf=trap flag)

يتيح هذا العلم وضع المعالج في نمط الخطوة الواحدة في الوقت الواحد (**single step mode**) مما يسمح لبرنامج فحص الأخطاء كالديبجر بتتبع البرنامج ، إذا كانت قيمة العلم واحد **ON**=فإن النمط يعمل أما إذا كان صفر **off**=فإن النمط لا يعمل ( قيمة العلم **TF** عند بداية البرنامج =0 . )

### #أعلام الحالة :- Status Flags

هذه الأعلام تضبط آلياً بعد كل عملية رياضية أو منطقية وهي تعكس هذه العملية ، ويمكن بعد العملية التحقق من قيم هذه الأعلام لتنفيذ عمليات مثل الشروط والحلقات وهذه الأعلام هي :

### 1. علم الحمل - : (cf=carry flag)

يضبط لهذا العلم =1 إذا كان نتيجة آخر عملية كبيرة جداً على الهدف أو المقصد (في الأعداد التي بدون إشارة فقط ) ، مثال هذا البرنامج :

```
mov ah,200
add ah,100
```

بما أن المسجل **AH** هو 8بت فإن أقصى قيمة يتحملها هي 256 وبما أن القيمة في المسجل هي 200 ثم أضفنا لها 100 فإن الجواب أكبر من الهدف (**ah**) هنا لذلك العلم **CF** سوف يضبط =1 بعد عملية الجمع لضبط لعلم برمجياً **stc** ولتصغيره **clc**

### 2. علم الفيض - : (OF=overflow flag)

هو نفس علم الحمل لكن مع العمليات ذي الإشارة أي أنه يضبط إذا كان ناتج آخر عملية أكبر أو أصغر من حدود الهدف ، مثال :

```
mov ah,-100
add ah,-50
```

بما أن أصغر قيمة يتحملها المسجل **AH** هي -128 لكن ناتج العملية -150 فإن علم الفيض يضبط =1

### 3. علم الإشارة - : (sf=sign flag)

يضبط هذا العلم إذا كان ناتج آخر عملية رياضية أو منطقية سالب ويصفر إذا موجب (في الواقع أن العلم نسخة من البت الأخيرة للجواب (بت الإشارة) - كما ذكرنا سابقاً فإن العدد سالب إذا البت الأخيرة 1 وموجب إذا صفر . )

### 4. علم الصفر - : (zf=zero flag)

يضبط هذا العلم =1 إذا كانت نتيجة آخر عملية رياضية أو منطقية تساوي صفر .

## 5. علم الحمل المساعد- : (af=auxiliary carry flag)

يضبط العلم=1 إذا تسببت آخر عملية رياضية أو منطقية حمل من البت الثالثة الى البت الرابعة أو استلاف من البت الرابعه الى البت الثالثه . هذا العلم لا توجد له فائدة واضحة وهو قليل الاستخدام برمجياً .

## 6. علم الأزدواجية - التحقق - : (pf=parity flag)

ببساطة يضبط =1 هذا العلم اذا كان عدد الواحد في ناتج آخر عملية رياضية أو منطقية زوجياً ويصفر اذا كان فردياً ، مثال لو كان جواب آخر عملية = 00100010 فإن العلم سوف يضبط=1 لأن عدد البتات التي تحتوي وحيد =2 وهو عدد زوجي أما اذا كان الجواب مثلاً = 11100000 فإن العلم يصفر لأن عدد البتات التي تحوي وحيد=3 وهو عدد فردي . وكما علم الحمل المساعد AF فإن استعماله قليل برمجياً ويستخدم عادة من قبل نظام التشغيل لإدارة الذاكرة وكذلك برامج الاتصال لتحقيق من سلامة البيانات المرسله .

لاحظ أن كل من الأعلام [علم المصيدة (tf=trap flag)/ علم الفيض (OF=overflow flag)/ علم الإشارة (sf=sign flag)/ علم الصفر (zf=zero flag)/ علم الحمل المساعد (af=auxiliary carry flag)/ علم الأزدواجية - التحقق (pf=parity flag)] لا يوجد لهم تعليمات مباشرة لضبطهم أو تصفيرهم وتحتاج أن تستخدم طريق فيها أنحناء بسيطة لتعديل قيم هذه الأعلام سوف يتم شرحها في دروس قادمة علماً بأن البرامج العادية لن تحتاج لتعديل قيم هذه الأعلام وكل ما ستحتاجه هو قراءة القيم التي بها .

إلى هنا ينتهي الدرس **الثالث** ، ونبدأ من الدرس الرابع إن شاء الله تعليمات لغة **الأسمبلي** (البدايه الفعلية) ، نشكركم لحسن إستماعكم معنا ، رجاء الإهتمام التام بهذه الدروس الثلاثة السابقه لأهميتها الشديده.

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



### دورة الأسمبلى - اليوم الرابع

أصبحنا فى اليوم الرابع و لم ندخل بعد فى اللغة نفسها ، يبدو أن الموضوع ليس كما تصور .....،،، اليوم سنفتح باب اللغة إن شاء الله و ندخلها ولكن بخطوات قليلة جدا، هيا جهزوا أنفسكم و الباب يحتاج مفتاح !!.

سنتناول الأتى فى هذا الدرس :-

- 1- نظم العنونه Addressing Modes
- 2- تعليمات الأسمبلى Asseply Instructions

### - 1 نظم العنونه :- Addressing Modes

قلنا أننا نتعامل مباشرة مع الذاكره و أماكنها المختلفه فيجب أن نتعرف على ما هى نظم عنوانه الذاكره و التى تمكن لنا وصولا مضمونا إلى القيم الأخيره (المستهدفه) Operands (معلومات عامه :-

1- الذاكره تكون مقسمه تخيليا (عن طريق نظام التشغيل و البروسور) إلى أجزاء تسمى مقاطع (Segments) هذه المقاطع تعمل كاقسام بيانات وهى أجزاء محددة الطول ب 64 كيلو بايت ، كل مقطعه رقمه ( عنوانه ) و يمكن الوصول إلى مقطع محدد عن طريق مسجلات المقاطع التى تم ذكرها فى الدرس السابق ، عند أى إستخدام للذاكره ، يجب تحديد المقطع أولا عن طريق Data Segment أو أى مسجل مقاطع آخر ومن ثم تحديد أى مكان داخل هذا المقطع عن طريق مسجل يحمل العنوان المراد وهو بالطبع من 0 إلى 65535 ومن هنا يتم تحديد مكان البيانات بالظبط ،ممكن أن تتخيلها بطريقه أخر : تخيل أن مدينه معينه وأريد الوصول إلى شخص ما فيها أو منزل بها ، المدينه مقسمه إلى شوارع ، كل شارع برقم معين بالترتيب و داخل كل شارع توجد منازل بأرقام مرتبه أيضا من 1 إلى 100 منزل ، كنت أريد الوصول إلى المنزل 45 فى الشارع ال 20 . عندها سأحاول البحث عن الشارع أولا ومن ثم البحث داخله عن المنزل ، نفس طريقه العمل بالظبط يتم التعامل بها فى إستخدام الذاكره و تخزين و إخراج البيانات منها .

2- عند تشغيل أى برنامج ، فإن نظام التشغيل يقوم بتحميل البرنامج فى الذاكره .. أين و كم مقطع سيترك للبرنامج؟؟ فى الحقيقه ، كل برنامج يتم تشغيله على الأقل يأخذ 2 مقطع ، المقطع للشفره Code Segment يتم وضع عنوان هذا المقطع داخل CS مادام البرنامج يتم العمل داخله ، والمقطع الآخر يوضع به البيانات لذلك سمي ب Data Segment ويوضع عنوان هذا المقطع فى DS ، كل برنامج يكون مكتوب به عدد المقاطع التى يحتاجها أولا ، هذا لخدمة البرامج الكبيره و التى تكون حجمها أكبر من وضعها فى مقطع واحد ، يتم فعليا تحديد حجم البرنامج فى أوله عن طريق تحديد نوعه ما بين صغير ، كبير ، متوسط ، كبير جدا .

## أنواع العنونه :-

### 1 عنونه فعليه (حقيقه-: Physical Address )

وهي العنونه الحقيقه التي يتم التعامل بها بين داخل الذاكره ، بمعنى لو أن الذاكره 128 ميجا ، فإنها ستكون مرتبه ترتيبيا تصاعديا من المكان صفر إلى المكان 128 ميجا ، نظام الهارد وير الفعلي على شريحة الذاكره يعمل بهذا النوع ( طبيعياً . )  
مثال بالنظام السادس عشر ، على أساس الذاكره مثلا 1 ميجا ، فإنها بها مليون مكان يتم تحديدهم Decoding ب عشرين مكان ثنائي . AF67B

### 2 عنونه تخيليه :- Logical Address

وهو النوع الذي تكلمت عنه بأعلى ، الذاكره مقسمه إلى أقسام Segments و كل قسم به محتوياته من المكان Offset صفر إلى المكان 64 كيلو ، وهذا النظام يتم التعامل به من جانب البروسسور مع نظام التشغيل و البرامج الأخرى و من ثم يكون المعالج مسؤولاً على تحويل هذا العنوان إلى العنوان الفعلي عن طريق عمليه حسابيه بسيطه وهي عن طريق  
[أضرب عنوان المقطع في 10 و من ثم أضيف عليه ال Offset العنوان الفعلي ]  
مثال [A100 : 9C00] :المكان الأول هو عنوان المقطع و الثاني هو ال Offset داخل هذا المقطع، يتم التعامل بهذا النظام فعليا في الأسمبلى و بهذه الطريقه بالظبط .

أنظمة عنونه و سائط التخزين المختلفه والوصول إلى القيم المستهدفه ( Operand مسجلات و ذاكره :- ) ...

### ( :- Impiled Mode - 1 الصراحه الترجمه العربيه مخرفه معايا شويا ، نأسف لهذا العطل ) ..

وهو لا يتم ذكر به العنوان و تكون التعليمه مفهومه للبروسسور بدون أى برامياتارات إضافيه ، مثال HLT : هذه التعليمه سنتعرض لها فى جزء التعليمات .

### 2 - Immediate Mode :-

و يتم ذكر العنوان صراحاً به ، مثال ADD AX,1000B : وهذه التعليمه ذكر بها القيمه صراحاً التي سيتم التعامل بها .

### 3 - Register mode :-

فى هذا النظام تكون القيمه المراد التعامل معها فى مسجل Register ما ، لذلك يتم ذكر اسم المسجل بجانب التعليمه .  
مثال MOV AX,BX : وهنا سيتم التعامل مع المسجلان المذكوران .

### 4 - Direct memory mode :-

وهنا يتم ذكر العنوان فى الذاكره الذى يحتوى القيمه المستهدفه .  
مثال ADD AX,[1000:200A] : حيث أننا نكتب العنوان الذى يوجد به القيمه المستهدفه داخل التعليمه كما هو موضح .

## 5 - Indirect memory mode :-

وهنا يكون عنوان القيمة في الذاكرة موجود بداخل مسجل و فيها يأخذ المعالج القيمة التي فالمسجل و تكون هي عنوان الذاكرة الذي يحتوى على القيمة .  
مثال . `add BX,[SI]` :

ملاحظه : المسجلات التي من الممكن إستخدامها في هذا النظام 4 وهما `BX , SI , BP , DI`

### 1تعليقات الأسمبلي :- Assemly Instructions

تتكون التعليمة الواحدة في الأسمبلي من تمثيل بسيط بالأحرف الأنجليزية يقابله بالأرقام تعليمة لغة آله ، تتكون كل تعليمة من مائلي : أولاً جزء الأمر وهو أمر يدل على نوع العملية المطلوبة مثل ( `ADD` للجمع) ، الجزء الثاني هو الوسائط علماً بأن بعض التعليمات لا يأخذ وسائط والجزء الآخر بسيطة واحدة فقط والبعض الآخر أكثر من ذلك ، تحدد هذه الوسائط الشئ الذي سيعمل عليه الأمر ، فالأمر `ADD` لوحده عقيم لا يدل على شئ لكن الأمر `ADD AX,5` يدل على جمع الرقم 5 مع القيمة الموجودة في المسجل `AX` ويوضح المثال التالي بعض الأوامر

CODE

`clc` ; فقط أمر بدون وسائط

`dec ax` ; وسيطة واحدة فقط

`mov cx,dx` ; وسيطتين

لاحظ أن أي نص في شفرة الأسمبلي يأتي بعد الفاصلة المنقوطة هو مجرد تعليق الوسائط ممكن تكون عدة أنواع :-

1. بيانات فورية (مباشرة) (أي ثابتة) مثال : `'a' 30 / 10`

2. مسجل مثال `AX / EAX / BL` :

3. موقع ذاكره (يتم تحديده عن طريق العنوان) مثال : `[100]200 / [bx] / ]`

4. متغير (وهو نفس السابق لكن بدل أن تحفظ أو تحسب العنوان يدوياً يقوم الأسمبلر بأستبدال المتغير برقم يدل على عنوانه ) مثال `count / VAR1 / INTVAL / STR1` :

### مدخل الى الديوغر :- Debugge

ها قد وصلنا الى واحد من أقوى البرامج المبيته في النظام فبواسطة الديوغ تستطيع عمل أشياء عجيبة وغريبة ، حسناً شغل الدوس وعند محث الأوامر أطلع `debug` ثم أنتر وستظهر لك علامه '-' دليل على أستعداد الديوغر على أستقبال أوامرك .

الآن دعنا نكتب هذا البرنامج الصغير

CODE

`mov ax,2` ; نقل العدد 2 ك معلومة مباشرة الى المسجل أي-أكس

`mov bx,3` ; نقل العدد 3 ك معلومة مباشرة الى المسجل بي-أكس

`add ax,bx` ; جمع أي-أكس مع بي-أكس مع وضع الجواب في أي-أكس / أي-أكس=أي-أكس + بي-أكس



## كيف تقوم بأدخال هذا الكود :-

1. عند المحث '-' أدخل a100 أي أننا سنبدأ نكتب الكود من العنوان 100 ثم أضغط أنتر بالطبع.
2. الآن أدخل كل تعليميه ثم أضغط أنتر ومع نهاية التعليمية الأخيرة أضغط أنتر مرتين .

الآن قم بأدخال الرمز R ثم أنتر لترى حالة المسجلات لاحظ أن المسجل AX يساوي صفروستري أيضاً ظهور التعليمية MOV ax,0002 وهي التعليمية التي عليها الدور في التنفيذ وليس المعلومة المنفذه ، الآن قم بطباعة الرمز T ثم أنتر لتنفيذ التعليمية التي عليها الدور هنا هي MOV AX,0002 ستري الآن أن المسجل AX أصبح يساوي 2 وهذا ماتوقعه بالضبط وستري أيضاً التعليمية التي عليها دور التنفيذ وهي MOV BX,0003 أدخل الرمز T ثم أنتر لتنفيذها لترى أن المسجل BX أصبح يساوي 3 وستري أيضاً التعليمية التي عليها الدور في التنفيذ وهي ADD AX,BX قم بأدخال الرمز T لتنفيذها ولاحظ كيف أن المسجل AX أصبحت قيمته مجموع العددين 2+3 وهو خمسة بينما بقي المسجل BX يساوي 3 .

الآن بعدما عرفت كيف تكتب كود بسيط أخرج من الديبغر بالضبط على Q ثم أدخل مرة أخرى بكتابة الأمر Debug حتى تصفر المسجلات مرة أخرى أدخل التعليمية A100 ثم جرب تكتب كود من عندك ومع كل نهاية تعليميه أضغط أنتر وفي نهاية التعليمية الأخيرة أضغط أنتر مرتين (ملاحظة لترى شفرتك بلغة الآله والأسملي أدخل الرمز U ثم أنتر مباشرة بعد إدخال الكود وقبل إدخال الرمز R ) أضغط R ثم أنتر لترى المسجلات قبل تنفيذ أي عملية ولترى التعليمية التي عليها الدور في التنفيذ أضغط T ثم أنتر لتنفيذ التعليمية وترى النتائج والتعليميه التي بعدها وهكذا ولا تنسى إذا أردت أن تدخل كود جديد الخروج والعودة مرة أخرى الى الديبغر لتصفر المسجلات والذاكره

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



دورة الأسمبلى - اليوم الخامس

سنبدأ بتعليمات وأوامر الأسمبلى،،،

### التعليمات الحسابية Arithmetic Instructions

- 1 الجمع :-

التعليمه Add :

وتأخذ في الحالة العادية 2 وسيط ، المصدر و الهدف وهى تكافىء  $==$  المصدر = المصدر + الهدف  
**ADD Source , Destination  $==$  Source = Source + Destination**

أمثله :

CODE

```
; A2H + 20H = C2  
MOV AX , A2; 1st Number  
ADD AX , 20; 2nd Num
```

نأخذ في إعتبارنا أن العدد كون بالنظام السادس عشر إفتراضيا حالما يتم تغييره إلى عشرى أوثنائى بمعنى فى المثال السابق أول خطوه نقوم بتحميل A2 بالنظام السادس عشر إلى المسجل AX وفى الخطوه الثانيه قمنا بجمع على ما بداخل المسجل الرقم السادس عشر A2

### مثال آخر :

CODE

[/u]

```
; A111 + 2302 + 1203 + 4099 = 116AF
; OF COURSE AX IS 16 BIT RWG. SO IT WILL CONTAINS 16AF
AFTER THE OPERATION
MOV AX , A111
ADD AX , 2302
ADD AX , 1203
ADD AX , 4099
; AX = 16AF AND CARRY FLAG IS BEEN SET
```

[U]

مثال آخر (باستخدام الذاكرة : (نضيف محتويات الذاكرة بالعنوان >>>> [A200] + [3409]

CODE

```
MOV AX , [A200]
ADD AX , [3409]
```

ملاحظه: يمكن القيام بكل هذه الأمثلة على برنامج ال Debug والذي تم شرحه في الدرس السابق ،،،

- 2 الطرح :-

-----  
التعليمة SUB :

بالظبط نفس تعليمة الجمع مثال:- نطرح محتويات المسجل CX من محتويات المسجل SI

CODE

```
SUB SI , CX
```

مثال آخر: نطرح محتويات المكان في الذاكرة المعنون بالعنوان الموجود في المسجل BX من FFFF

CODE

```
MOV AX , FFFF
SUB AX , [BX]
```

### - 3 الضرب :-

#### -----التعليمه MUL :-----

يتم ضرب أي رقمين في لغة الأسميلي في ثلاث أنظم إلى الآن :- الأول : ال-8 بت (البايت) :- (يتم وضع إحدى قيم الضرب في AL و الآخر في أي ريجستر آخر له نفس الحجم أو مكان في الذاكرة بجانب كود الضرب الذي هو في حالتنا هنا [MUL] للأرقام الغير محددة الإشارة . و [IMUL] للأرقام المحددة الإشارة . وتوضع النتيجة في ال AX .

مثال : للحصول على حاصل ضرب 120 \* 30 نقوم بالآتي :-

#### CODE

```
-----  
MOV CL,30D  
MOV AL,120D  
MUL CL  
MOV [1000H] , AX ; لتخزين الناتج في الذاكرة المعنونه ب1000 بالنظام السادس عشر  
-----
```

الثاني : ال-16 بت (الكلمه) :- (يتم وضع إحدى القيم في AX و الآخر في أي ريجستر آخر له نفس الحجم أو مكان في الذاكرة بجانب كود الضرب و توضع النتيجة في DX:AX . وهذا معناه أن الجزء ال Low من ناتج القسمة سيخزن في ال AX و الجزء ال High سيوضع في ال DX .

مثال : للحصول على حاصل ضرب 2365 \* 20000 سنقوم بالآتي :-

#### CODE

```
-----  
MOV CX,2365D  
MOV AX,20000D MUL CX  
MOV [1000H],AX ; تخزين الجزء الأول في المكان في الذاكرة المعنون ب 1000  
MOV [1001H],DX ; تخزين الجزء الثاني في المكان في الذاكرة الذي يلي المكان السابق  
-----
```

الثالث : ال-32 بت :- يتم وضع إحدى القيم في EAX و الآخر في أي ريجستر آخر له نفس الحجم أو مكان في الذاكرة بجانب كود الضرب و توضع النتيجة (64 بت) في EDX:EAX . وهذا معناه أن الجزء ال Low من ناتج القسمة سيخزن في ال EAX و الجزء ال High سيوضع في ال EDX .

بالطبع نحن نتعامل كبدايه على الأنظمه إلى ال 16 بت ،،،

#### - 4 القسمه :-

#### التعليمه DIV :

أيضاً تتم في نفس النظم السابقه والخاصه بالضرب :-

**الأول:** ال8بت (بايت) :- (يتم وضع المقسوم في AX و المقسوم عليه في أى ريجستر حجمه بايت مثل BL,CL,DL,... أو مكان فى الذاكره لها حجم بايت مثل BYTE PTR [1234H] , BYTE PTR [BX] و يوضع ناتج القسمه فى AL و باقى القسمه فى AH باستخدام [DIV] .  
**لاحظ :-** أنه فى حالة قسمة الأرقام المحددة بالإشارة باستخدام [IDIV] تكون إشارة ناتج القسمه هى الإشارة العاديه فى هذه الحالات . وتكون إشارة خارج القسمه Remender دائماً موجب و صحيحه Integer.

**لاحظ:-** كل القيم هنا -8بت إذا , لابد من تحويل القيمه ال -8بت للمقسوم إلى 16-بت ليمن نقلها إلى AX ويتم ذلك فى حالة القيم غير محددة بالإشاره بمسح ال AH ليكون كله أصفار و تكون قيمة المقسوم 16-بت . أما فى حالة القيم محددة بالإشاره يتم ذلك عن طريق كود [CBW] المسنول عن تحويل الباييت (8-بت) إلى كلمه (16-بت (لل AX فقط.

**مثال :-** للحصول على ناتج قسمة (+16) على (-5) يمكننا عمل الأتى :-

CODE

```
-----  
;-----  
MOV AL,16D  
CBW  
MOV BL,5D  
NEG BL  
IDIV BL  
MOV BYTE PTR[1000H] , AL  
MOV BYTE PTR[2000H] , AH  
;-----
```

وهنا تم نقل المقسوم إلى AL وتم عمل مد له عن طريق [CBW] ثم تم نقل قيمة المقسوم عليه إلى BL بعد وضع الإشارة السالبه عن طريق [NEG]. إذا القيم جاهزه لعملية القسمه ويتم حفظ الناتج فى الذاكره بعدها.

باقى النظم كما سبق من الممكن أن تقوم باستنتاجها .

**ملاحظه أخيره:** استخدمنا التعليمه MOV وهى مسنوله -كما توقعتم- على تحميل المسجلات أو أماكن الذاكره بقيم معينه أو محتويات مسجل أو أى قيم أخرى ,,

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



### دورة الأسمبلى - اليوم السادس

السلام عليكم اليوم السادس فى الأسمبلى، سنكمل بإذن الله ما بدأناه فى اليوم السابق من سرد تعليمات الأسمبلى، ولقد إستعرضنا فى اليوم السابق (الخامس) التعليمات الحسابيه مثل ADD, SUB, MUL, DIV

اليوم ، نبدأ بتعليمات عامه منها ما يستخدم بكثره ولقد حاولت قدر الإمكان أن أبتعد عن التعليمات التى لا تستخدم إطلاقا إلى فى حالات نادره يمكن أن نتحدث عنها آن حدوثها ، المهم ... خذ نفس عميق ... ركز .... و Let's Go

### MOV

-----التعليمة تستخدم فى المساواة بدلا من "=" أو لتحميل مكان ما بالذاكره ( سواء مسجلات أو عناوين ذاكره رام ) بقيم أو بمحتويات ذاكره أخرى و من هنا نستنتج أن هذه التعليمة تحتاج إلى 2مدخل **Argument** المصدر و الجهه . **Source and destination** ويذكر أن هذه التعليمة أشهر تعليمات **الأسمبلى** و أكثرها إستخداما ، أمثله :-

### CODE

```
MOV AX,2345H ; AX = 2345H
MOV AL,34 ;AL = 34 Decimal
MOV [2312] , BX ;Memory location 2312 IN current data
segment = BX Contents
MOV [SI] , 30 ;Memory location in SI Contents = 30 Decimal
.
.
.
```

### لاحظ :-

-لا يمكن نقل محتويات ذاكره رام إلى محتويات ذاكره رام بنفس التعليمة ، أو أى تعليمة أخرى .... وذلك لأن البروسسور لا يستطيع القيام بالتعامل مع الذاكره أكثر من مره واحده لكل تعليمة أسمبلى (المقابل له بلغة الآله )  
>>>

CODE

```
MOV [1000],[2000] ;Error
```

-لا يمكن النقل بين مكانين مختلفي الحجم ، بمعنى أنه لا يمكننا أن ننقل ما بداخل مسجل بعرض 16 بت إلى مسجل آخر بعرض 32 بت أو العكس .

CODE

```
MOV AX,AL ;Error  
MOV AL,AX ;Error
```

-عند النقل من مكان في الذاكرة إلى مسجل فإن البيانات التي تنتقل حقيقياً هي بيانات إبتداءً من هذا العنوان و بعرض المسجل ، وخذ في إعتبارك ان الذاكرة مقسمة **BYTES** بمعنى أنك أردت تحميل **AX** وهو بعرض 32 بت بمحتويات الذاكرة عند 1000 فإنه يبدأ بتحميل 4 بايت من عند هذا العنوان .

CODE

```
MOV CX,[0110] ;CX = 4 BYTES Contents starts from address  
0110 at current data segment
```

**NOP**

-----  
هذه التعليمه تعبر عن إضاعة الوقت و هي إختصار ل **No Operation** و هي تضيع وقت تعليمه قياسيـه بحيث أنك إذا أردت عمل إنتظار **Delay** فإن هذه التعليمه قياسيـه و تستخدم و طبعا تستخدم في دوره بحيث أنك مثلا لو اردت إنتظار ثانيه فإنك تكررها **100000** مره على الأقل ( لم احسبها و هتتوقف على سرعة المعالج لديك . )

أمثله :-

CODE

```
NOP ;Instruction tells the processor to do nothing this
instruction cycle time
;and used to waste a small time according to cpu clocking
speed
```

**HLT**

-----  
وتستخدم في إنهاء عمل البروسور ولست اقصد الجهاز نفسه ، يعنى احتمال يهنج و احتمال لا يفعل أى شىء ، يعنى هى إستخدامها كان زمان فى الإصدارات القديمه من اللغه ، أما فى الوقت الحالى ، فإنه توجد تعليمات أخرى تخبر المعالج بإيقاف البرنامج أو إنهاءه وما شابه ...

أمثله:-

CODE

```
HLT ;No command
```

**INC**

-----  
تستخدم فى عمل زياده للمعطى بمقدار 1، وهى تقابل عملية جمع واحد + القيمه الى بداخل المعطى و من ثم تخزين القيمه الجديده .

أمثله :-

CODE

```
INC AX ;AX = AX + 1 === ADD AX,1
INC [BX] ;[BX] = [BX] + 1 === ADD [BX] , 1
INC CX ;CX = CX + 1
```

**DEC**

-----  
وهى عكس سابقتها و تستخدم في إنقاص المعطى Operand بواحد . و هى تقابل التعليمه SUB ,1.....و بالطبع فإن المكان الخالى يتم وضع ال Operand

أمثله :-

CODE

```
DEC [SI] ;[SI] = [SI] -1
DEC AX ;AX = AX - 1 === SUB AX,1
DEX CX
```



- عند استخدام هذه التعليمة مع مسجل أو مكان بالذاكرة يحتوى على 0 فإنها تدخل فى النطاق السالب بعده ويتم تغيير علم الإشارة **SIGN Flag** فى مسجل الأعلام.

### NEG

-----وهى تستخدم فى تغيير إشارة المعطى **Operand**، وهى إختصار للكلمة الإنجليزية **Negiate** بمعنى جعل القيمة سالبة وطبعاً تعمل مع جميع القيم السالبة لتجعلها موجبة و الموجبة التى تحولها إلى سالبة  
أمثله :-

CODE

```
MOV AX,100 ;AX = 100
NEG AX ;AX = FF00 = -100
NEG AX ;AX = 100 Again
```

سنتقل الآن إلى تعليمات **الأسمبلى المنطقية Logic Instructions**

### AND

-----وهى تقوم بعملية "و" المنطقية **AND Gate** بين **Operrand 2** ومن ثم تخزن القيمة فى الأول كالعاده: أمثله :- ( 0 and 1=0 , 0 and 0=0 , 1 and 0=0 , 1 and 1=1 ) وجود صفر النتيجة صفر  
CODE

```
MOV AX,10 ;AX = 10
MOV BX,5 ;BX = 5
AND AX,BX ;AX = 0 Because that 10 (1010) and 5(0101) = 0000
:-)
```

تستخدم هذه التعليمة فى عملية تسمى **Masking**،

### OR

-----وهذه التعليمة كما أستنتجت فهى تعبر عن العملية المنطقية "أو" **OR Gate** "و لها شروط أمثله :- ( 0 or 1=1 , 0 or 0=0 , 1 or 0=1 , 1 or 1=1 ) وجود واحد النتيجة واحد

CODE

```
OR AX,CX
OR [1020],CL
OR AX,0000
```

## XOR

----- هذه التعليمه تستخدم فى العمليه المنطقيه XOR، كما سنرى فى الأمثله :-  
( 1 xor 1=0 , 1 xor 0=1 , 0 xor 0=0 , 0 xor 1=1 ) التشابه النتيجة صفر

### CODE

```
MOV AX,1000 ;AX = 1000
XOR AX , 2000 ;AX = 1080
XOR AX,2000 ;AX = 1000
```

لاحظ أن هذه التعليمه تستخدم فى التشفير الأحادى **Single Encyption** لأنه كما لاحظنا فى المثال السابق ، فإننا عملنا تشفير للقيمه بداخل المسجل **AX** فإستخدمنا المفتاح (القيمه الأخرى التى تستخدم فى فك التشفير أو إرجاع القيمه الأصلية ثانيا) و هى **2000** ومن ثم عند القيام بنفس العمليه مع نفس القيمه فإنه يتم إرجاع القيمه الأصلية

"لقد صممت برنامج بسيك بالأسمبلى يقوم بتشفير نصوص مدخله من قبل المستخدم و من ثم يعيد فكها ثانيا باستخدام مفتاح ثابت بداخل البرنامج و لرؤية البرنامج و السورس كود الخاص به ، يمكنك الإطلاع على الدرس التاسع فى موقعى <http://ezzuz.tk> !!!!!!!"

-لا حظ أيضاً أن التعليمات المنطقيه **Logic Instructions** تستخدم فى عمليات البت **Bit Operation** وكماعلم أن البت هى اصغر وحدة تخزين بالحاسب ، ومن ثم تستطيع تسخير هذه التعليمات لمساعدتك فى تكوين و التعامل مع المتغيرات المنطقيه **Boolean Variables** والتى تحتل قيمتين فقط أما صح أو خطأ **True** ،،،or False

بسم الله الرحمن الرحيم

## السلام عليكم ورحمة الله وبركاته



### دورة الأسمبلى - اليوم السابع

مازلنا بتعليمات الأسمبلى ، هذا آخر درس بإذن الله فى تعليمات الأسمبلى الأساسيه قبل أن نبدأ بكتابة برامج بالأسمبلى من اليوم القادم ،،، و من ثم نكمل باقى تعليمات التحكم و الدورات و باقى التعليمات المهمه ...

أتذكر أن آخر تعليمات تناولناها سويا الدرس السابق هى التعليمات المنطقيه ، و نسيت ذكر واحده فقط ، سأذكرها هنا أولا :

**NOT** هذه التعليمه مسؤوله عن عكس القيمه التى بداخل المكان الذى يذكر بعدها ، بمعنى ...مهممممم ، أنظر المثال الأتى :

مثال :

-----

لو أنه يوجد بالمسجل **AL** قيمه ثنائيه على هذا الشكل **01110010** ونريد عكس هذه القيمه لتكون هكذا **10001101** ، نستخدم هذه التعليمه .

CODE

```
MOV AL,01110010B
NOT AL
;AL = 10001101B NOW
```

-----

**نأتى لتعليمات الإزاحة و الدوران -Shift And Rotate Instructions** هذه التعليمات تتعامل مع التشكيل الثنائى المنطقى كما تتعامل التعليمات المنطقية السابقة مع القيم المختلفه **Logic Bit Instructions** -تستخدم هذه التعليمات بكثرة فى دوال التحكم فى وحدات الإدخال و الإخراج **Control I/O Divices** -تستخدم مع جميع أنواع المسجلات و الذاكرة

**أولا :تعليمات الإزاحة -Shift Instructions:** معنى الإزاحة هى إزاحة القيم الثنائيه المتتاليه إلى اليمين أو اليسار (<<< || >>>)

-تستخدم فى العمليات الحسابيه مثلا لضرب و القسمة (إزاحة لليساى بت واحده تعنى ضرب فى 2 و إزاحة لليمين بت واحد تعنى قسمة على 2)

-هناك أربع تعليمات إزاحة عباره عن مجموعتين (إزاحة منطقيه **Logical Shift**) و إزاحة حسابيه (**Arithmetic Shift**)

-جميع هذه التعليمات تتم من خلال علم الحمل **Carry Flag** وكما ترى بالشكل الأتى (انظر ملحق التعليمات نهاية الكتاب) ، كيفية عمل هذه التعليمات و الفرق بينهما .

-تلاحظ أن الإزاحة المنطقيه إلى اليسار **SHL** تتم الإزاحة و يتم وضع من اليمين صفر ، وفى الإزاحة المنطقيه إلى اليمين **SHR** يتم وضع صفر من اليسار .

-تلاحظ فى تعليمات الإزاحة الحسابيه إلى اليسار **SAL** مثل الإزاحة المنطقيه إلى اليسار ، أما الإزاحة الحسابيه إلى اليمين فيتم وضع نفس القيمة فى البت الأخير كما هى و هذا بسبب عدم تأثر إشارة القيمة الموجوده و لذلك سميت إزاحة حسابيه .

-يمكن إستعمال الإزاحة المنطقيه مع القيم التى بدون إشارة . **Unsigned Numbers**

-يمكن إستعمال الإزاحة الحسابيه مع القيم التى بإشاره . **Signed Number**

-لا يمكن إستعمال الإزاحة مع مسجلات المقاطع مطلقا وفيما عدا ذلك يمكن إستخدامها.

أمثله لتعليمات الإزاحة المختلفه :-

## CODE

```
SHR AH,3 ;AX is logically shifted Right 3 places
SHL CX,12 ;CX is logically shifted right 12 places
SHR [1000] , 3 ;The memory location adressed by 1000 in the
data segment is right shifted by 3 places
MOV CL,10
SAR DX,CL ;DX is arithmetically shefted to right by number
in CL
;Note that CL Here called the sheft count register and can
be used as above
SAL EDX,1 ;EDX is arithmeticaly shifted by 1 place
```

مثال لإستخدام عملية الإزاحة إلى اليسار فى ضرب القيمة الموجوده داخل المسجل AX فى القيمة 1010 وهى مساويه ل10 فى النظام العشرى .

## CODE

```
SHL AX,1
;AX = AX * 2B
MOV BX,AX
;BX = AX * 2
SHL AX,2
AX = AX * 8
ADD AX , BX
;AX =AX * 10
;AX = AX * 1010B
```



;



**ثانيا :تعليمات الدوران -Rotate Instructions:** تقوم هذه التعليمات بعمل إزاحه من طرف المسجل أو المكان بالذاكره إلى الطرف الأخر.

-أربع تعليمات مقسمين إلى مجموعتين ( دوران خلال علم الحمل Rotate أو دوران خارج علم الحمل Roate through carry )  
-وكما ترى بالشكل الأتى ، كيفية عمل هذه التعليمات و الفرق بينهما .

-يتضح من الشكل المبين سابقا أن تعليمات الدوران من خلال علمالحمل RCL , RCR ,تدور البتات من طرف إلى علم الحمل إلى الطرف الأخر .  
-أماالتعليمات الأخرى ROR , RRL تدور البتات من طرف إلى الأخر ومن ثم وضع البت المنقوله إلى علم الحمل كما هو مبين بالشكل. (انظر ملحق التعليمات نهاية الكتاب)  
-أمثله :-

## CODE

```
ROL SI,14 ;SI rotated left by 14 places
RCL BL,6 ;BL rotates left through carry by 6 places
ROL ECX,18 ;ECX rotates left by 18 places
RCR AH,CL ;AH rotates through carry by the number of places
in CL
ROR WORD PTR[BP] , 2 ;The word contents of the stack
segment memory location addressed by BP rotate right by 2
places
```

---

**تعليمات البحث عن البت 1 - Bit Scan Instructions:** وظيفة هذه التعليمات هي البحث عن بت واحد داخل مسجل ما أو مكان بالذاكرة .

- هذه التعليمات لها صيغتين وهما 2 (Bit Scan Forward) & BSR (Bit Scan Reverse) BSF  
- BSF - وتقوم بالبحث من اليسار إلى اليمين (من البت 0 إلى البت 15 في AX)  
- BSR - وتقوم بالبحث من اليمين إلى اليسار (من البت 15 إلى البت 0 في AX)  
- تأخذ التعليمه مدخلان (Source & Destination) ،

- عندما يجد المعالج 1 فإنه يقوم بتصحيح القيمة التي بعلم الصفر  
- flag is set . ثم يوضع رقم هذه البت في المدخل الأول . Destination مثال :-

## CODE

```
MOV AL,00101000
BSF BL,AL
;Zero Flag is set
;BL contains 3 , the 1-bit position in Al from left
BSR CL,AL
;Zero Flag is set
;CL Contains 4 , the 1-bit position in AL from right
```

بعد الوصول الى هنا اظن انك شبه اكلت اساسيات لغة الاسبمبلي

لكن لازال هناك اشياء اخرى لم يذكرها المؤلف في الدورة مثل

المصفوفات والتعامل مع السلاسل النصية وكتابة برنامج كامل

وغيرها هذه ستجدها بالتفصيل في كتاب اسمه :

( مرجع في البرمجة بلغة الاسبمبلي 16 ) للدكتور حميد المسمري .

متوفر في النت على الروابط الاتية .

<http://www.kutub.info/library/book/7528>

<http://www.boosla.com/showArticle.php?Sec=Programm&id=166>

<http://www.kutub.info/library/book/7471>

<http://forum.kku.edu.sa/showthread.php?t=13839&page=2>

ننتقل الي الدورة الثانية وهي من منتديات ستارتايمز هذه

الدورة فيها كتابة البرامج بالاسبمبلي وأشياء أخرى .

# دورة كاملة للأسمبلي

تأليف : [stoune](#) (  )

منتديات : ستارتايمز و كوووووورة

الرابط الاصيلي للدورة على المنتدى : <http://www.startimes.com/f.aspx?t=18803509>

جمع وترتيب وتنسيق : علي السيد محمد ( [الفدائي : Alfidai](#) )

اسألکم الدعاء لنا ولوالدينا والدعاء خاصة لصاحب الدورة .

**ملحوظة :** هذه الدورة قام ايضا بتجميعها الاخ [mourou riahi](#) في كتاب الكتروني اسمه (الاسمبلي من الالف الى الياء ) بصيغة EXE وهو موجود في موقع البوصة التقنية قسم لغة برمجة الاسمبلي لمن اراد الاستزادة :

<http://www.boosla.com/articlesList.php?Sec=Programm&menu=Assembly>





دورة الأسمبلي الدرس الرابع ◀▶ الجزئ الثاني

الدرس 7

دورة الأسمبلي الدرس الرابع ◀▶ الجزئ الثالث

الدرس 8

دورة الأسمبلي الدرس الرابع ◀▶ الجزئ الأخير

الدرس 9

دورة الأسمبلي الدرس الخامس ◀▶ أول برنامج لك بالأسمبلي

الدرس 10

دورة الأسمبلي الدرس السادس ◀▶ أسمبلي 16 بت أم 32 بت ؟ لماذا ؟

الدرس 11

دورة الأسمبلي الدرس السابع ◀▶ المقاطعات + اول برامج على 16 بت ▶

الدرس 12

دورة الأسمبلي الدرس الثامن ◀▶ المقاطعات + التعامل مع الملفات ▶

الدرس 13

دورة الأسمبلي الدرس التاسع ▶ التعامل مع الـ API بهم ايضا اللغات الاخرى

والفهرس تحت التحديث المستمر

--- نقاش حول دروس الأسمبلي ---

والسلام عليكم

لمادا ساتعلم هذه اللغة الصعبة نوعا ما Asseply ؟

أليس هناك لغات من الجيل الثالث توفر الأدوات الخاصة التي يمكن من خلالها ان نستغني عن Asseply ؟

وان تعلمتها ... فما هي الإنجازات التي يمكن ان احققها من خلال هذه اللغة ؟

لمادا يهمل المبرمجون لغة الأسمبلي ؟

كل هذه الاسئلة ساجيب عنها في هذا المقال , لهذا ارجو قرائته كاملا

-----\*

-----\*

لكي تستطيع أن تفهم من أمامك لابد أن تتعلم لغته , و لغة الحاسوب ليست السي أو الجافا....

و لكن لغة الحاسوب **صفر** و **واحد**

لو تعلمت الأسمبلي سوف ترى أشياء لم تراها عينك من قبل قط و سوف تتعلم أشياء لن تتعلمها قط  
من غير تعلم الأسمبلي و من قبلها **الدوائر**

**الرقمية و المنطقية**

صحيح لا يلزمك أن تكون على دراية بها و لكن تعلم **الدوائر المنطقية** سوف يفتح لك آفاق معرفة و  
نقاط قوة تميزك عن أي شخص آخر يعرف

لغة الاسمبلي وحدها دون أن يكون على دراية بعلم الدوائر المنطقية و الرقمية

-----\*

ساوضح لك اولا ماهي لغة الاسبلي:

في القديم كانت لغة الالة وحدها وكان المبرمجون علماء ليس كالان ، ياتي شخص يتعلم لغة برمجة في شهر او اسبوع ويسمي نفسه مبرمج

كان المبرمجون يجدون صعوبة ان وجد خطأ في البرنامج فيصعب عليه تحديد موقع الخطأ ، تخيل معي برنامج مكون من الف سطر كلها بالصفر

والواحد 010101010101 وهكذا ، قبل ان تجد الخطأ سيذهبو بك الى مستشفى الامراض العقليةمن هنا جائت الاسبلي

بكل بساطة لغة الاسبلي هي لغة مطورة للغة الالة ومن طورها ليس كبار المبرمجين بل هم اكثر من ذلك بكثير لا اعرف كيف يفكر هاؤلاء

الناس فانا الان ادرس لغة الالة وكتابة البرامج بلغة الالة مباشرة لكن الامر معقد اكثر مما تضنون

في الاسبلي كل امر يكافئ امر بلغة الالة وكل امر يكتب في سطر

لغة assembly يتم ترجمتها لحظيا الى لغة الالة يعني ينفذ البرنامج المكتوب في الاسبلي سطرًا سطرًا بالتوالي

وليس مثل اللغات العالية تترجم كاملة بالاول ومن تم يتم تنفيذها.

\*\*\*\*\*

تعلم الاسبلي يعني:

\_\_\*\*\*\*\*\_\_

فهم أعمق بالنظام الذي تكتب برنامجا له

\_\_\*\*\*\*\*\_\_

انت مبرمج سي او باسكال او ... صنعت برنامجا حجمه 800ميجا يعني برنامج ضخم ، تم قمت ببيعه الى المستعملين

وبعد شهر من إصدار البرنامج اكتشفت فيه خلل لا بد من إصلاحه ، وإلا سيفقد المشتري الثقة

قد تكون المشكلة مشكلة منطقية في الخوارزميات او قد تكون مشكلة برمجية كانك اعطيت احد المتغيرات قيمة خاطئة

الحل هو اولا إيجاد المشكلة في البرنامج، تانيا إيجاد حل لها، ثالثا ارسال الحل للمستخدم..

الشيء البديهي الذي ستقوم به هو تتبع الكود سطرا سطرا حتى تعثر على الخطأ

نفترض انك وجدت الخطأ واصلحته كيف سترسله للمستخدم ؟ هل ترسل له 100ميجا مرة اخرى

يعني قرص جديد للبرنامج

الحل بالنسبة لمن يتقن الاسمبلي بسيط جدا وقد لا يستغرق اكثر من بضع دقائق

افتح البرنامج في `debugger` ضع بعض نقاط التوقف, شغل البرنامج وتابع ما يحدث في `debugger`, عندما تعثر على الخطأ غير بحيث

يعمل البرنامج بالطريقة المطلوبة, ضع التغييرات في ملف `patch` حجمه لا يتجاوز 1ميجا على الاكثر ارسله إلى المستخدم

ما هو الفرق ؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟؟

بدون ان اتكلم الفرق شاسع وواضح ومثل هذه العمليات ساشرحها ان كان هناك من يريد التعلم فعلا

\*\*\*\*\*

نفس الموقف السابق لكن المستخدم طلب منك اضافة زر جديد مثلا يقوم بعمل ما

الله يكون في عون مبرمجي سي بلس بلس وامتالها حيث ستضطر الى فتح الكود سورس وكتابة كود جديد تم عمل ترجمة للكود من جديد

تم ارسال 100ميجا للمستخدم

بالنسبة للاسمبلي كل ما عليك هو استخدام `debugger` تم اضافة الشفرة وبعد ذلك تقوم بعمل `patch` للتغييرات التي قمت بها

وترسل الباتش للمستخدم..

في مثل هذه الحالات يكون حجم الباتش صغير جدا ولا يتجاوز عدة مئات من الكيلوبايت

قد ياتي احد محترفي ++c ويقول لي انه يوجد `debugger` في ++c واقول له انا هذا `debugger` ضعيف جدا مقارنة مع `debugger` حقيقي

مثل الوحش IDA PRO او OllyDBG

debugger +سي بلس بلس يتطلب فهم لغة الاسبلي , اما بالنسبة لمبرمجي ال VB فلا اعرف إذا كان هناك debugger اصلا

او حتى إذا كانوا قد سمعو بهذا المصطلح من قبل... مع الاعتذار الشديد لهم

و للأسف الشديد سمعة لغة الاسبلي اصبحت سيئة لان الكثيرين يعتقدون انها لغة لإختراق البرامج والبعض الآخر يرى انها لغة صعبة ولا تستحق

هذا الجهد والعناء..

ولكن من خلال تجاربي ارى انها اسهل من السي والسي بلس وحتى ال VB لانك في لغة الاسبلي تعرف ما يحدث جيدا بعكس باقي اللغات

وإذا كنت لا تعرف ما يحدث خلف الكواليس فمن الصعب عليك حل المشكلة دون اللجوء للمساعدة من قبل الخبراء.

\_\_\_\_\_

لا يوجد هاكر بالعالم لا يعرف الاسبلي

(انا لا اتكلم عن الاطفال اتكلم عن الهاكر وليس من يستعمل برنامج وسرفر مشفر و وتعلم sql injection..... لو يسمي نفسه هاكر)

لا يوجد خبير الحماية لا يعرف الاسبلي

لو اعطيناك برنامج ما كيف ما كان نوعه مثلا photoshop وقلنا لك اضف لنا خاصية جديدة في قائمة File وليس لديك الكود سورس

كيف تعمل هذا هل بواسطة سي او جافا او غيرها طبعا لا وهذا يتم عن طريق الاسبلي ببرامج التتبع حيث يمكن العبث بالبرنامج وضافة اكواد

اسبلي له.

مبرمج الاسبلي لا يعرف ما معنى كيف ظهر هذا الخطأ ؟ ، كيف عمل البرنامج هذا ؟ ، لماذا ظهرت هذه الرسالة ؟

فبكل بساطة يمكنه معرفة السبب

مبرمج الاسمبلي مستحيل تكون برامجها بها ثغرات ، عكس ذلك نجد اكثر البرامج بها ثغرات من نوع **Buffer OverFlow**

وهذه الثغرة قاتلة حيث تمكن المخترق بالكتابة على عنوان العودة وتنفيذ **shellcode**

وهو ليس خطأ في البرمجة بل هو **غياب** المبرمج وعدم معرفته بآلية عمل برنامج

ولا تعتقد انها غير موجودة بالبرامج الكبيرة ، هي منتشرة بكثرة وقد وجدت في الوندوز **xp service pack 2**

ومن نوع **Remot** هل تعرف ما معنى **Remot**؟

يعني استغلال عن بعد فبمجرد ان يحصل الهاكر على **P** الجهاز يتم تنفيذ **shellcode** عن بعد واختراق الجهاز

وقد اصلحت مايكروسوفت الثغرة في تحديث للويندوز لهذا عليكم دائما تحديث الوندوز

وملايين البرامج مصابة بهذه الثغرة منها **remot** ومنها **local**

المهم ليس موضوعنا شرح الثغرة

سأشرح هذه الثغرة في المواضيع القادمة على **C++** ليفهم المبرمجون اين يظهر غائبهم

\_\_\*\*\*\*\*\_\_

لغة الاسمبلي هي اساس الهندسة العكسية

والهندسة العكسية لها مجالات كثيرة منها مجال الـ **Cracking** يعني كسر التطبيقات والبرامج وصنع الكراكات للبرامج

وفك شفرات القنوات وكسر الكروت التلفزيونية...

\_\_\*\*\*\*\*\_\_

لغة الاسمبلي هي اللغة المعتمدة التي يبرمج بها **المتحكمات و المعالجات الدقيقة** التي تستخدم في الانظمة المتضمنة

و طبعا من خلالها تستطيع الوصول لمخابىء المعالج بكل سهولة و كتابة برنامج اصغر في الحجم و هذا شيء مهم جدا

واعرف مبرمجا محترفا في الاسمبلي يقوم الان بعمل برنامج اختراق بالاسمبلي

وسيكون حجم السرفر صغير صغير جدا ولا يصدق 1 ko نعم 1 ko

ويعمل الكثير من الامور وهو اول سرفر بالعالم بهذا الحجم

وهذا يعني ان برامج الاسبلي صغيرة الحجم

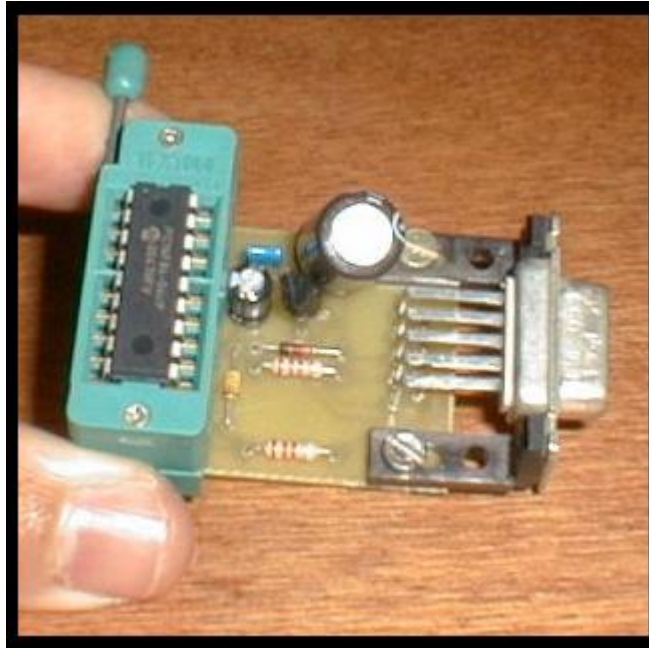
عندما نكتب في اللغات العالية "hello world"

فان ذلك يحتاج الى 8000بت

بينما في الاسبلي يحتاج ذلك الى 600بت فقط

\_\_\*\*\*\*\*\_\_

في مستقبلات القنوات الفضائية DEMO يتم فك تشفير القنوات المشفرة باستخدام Module صغير مبني على PIC 16F84



ياي لغة تتم برمجته في نظرك ؟ ولماذا !

الاجابة هي احد فوائد لغة Assembly

\_\_\*\*\*\*\*\_\_



في برمجة الالعب قد تجد حساب الجذر التربيعي مهم جدا لكي تأتي بالمسافة بين نقطتين و هذه عملية مرهقة للمعالج ، لذا نكتبها بالاسمبلي للتسريع و ايضا

هناك تعليمات مخصصة بالاسمبلي لتسريع العمليات علي الاعداد العشرية و هناك معالجات لها تعليمات مخصصة لها لن تصل اليها الا عن طريق لغة الاسمبلي

\_\_\*\*\*\*\*\_\_

استخدام الفيديو و الرسم علي شاشات التايفزيون او الكومبيوتر او اجهزة الالعب مثل الاتاري و الكونصول مثل البلاي ستيشن

فانت تحتاج تحكم دقيق جدا في الزمن يصل الي الميكروثانية و للوصول لهذه الدقة تحتاج الي حساب عدد الدورات التي تحتاجها للتعليمة الواحدة للوصول لزمن

دقيق مثل هذا و اللغات العالية المستوى مثل السي لن تعطي لك هذه الخاصية و لن تعرف زمن التعليمة الواحدة و كم دورة تاخذها لتنفيذه .

وامور عديدة غير محدودة....

\_\_\*\*\*\*\*\_\_

دراسة الفايروسات والملفات المحقونة بكود خبيث وتتبعها يكون بالاسمبلي مهما كانت خطورتها

ستفهم طريقة عملها تم برمجة اداة للقضاء عليها

\_\_\*\*\*\*\*\_\_

بصفة عامة الاسمبلي هي لغة برمجة العتاد في أي زمان و مكان و مهما تم تطوير أي لغة برمجة أخرى فلن تكون مثل الاسمبلي أبدا

ولا يوجد شيء لا يمكن عمله بالاسمبلي

ولن تستطيع لغة تصدير كميات كبيرة لمستشفى الامراض العقلية مثل الاسمبلي

اخيرا اود ان اشير الي اني لم اكتب هذا الموضوع لاتهجم على مبرمجي سي بلس بس او VB لكن

قلت الامر الواقع فقط

|

والسلام عليكم

## السلام عليكم

بإسم الله نفتح معكم دورة الأسمبلي من الصفر

( الدورة تمت الموافقة عنها )

هذه اللغة تختلف عن اللغات الأخرى من ناحية منهجية الشرح نأخذ متالا عن دورة للباسكال بمادا ستبدأ

طبعا أول شيء هو شرح انواع المتغيرات في هذه اللغة تم هيكلية البرنامج تم كتابة اول برنامج يطبع كلمة **Hello** على الشاشة

في الأسمبلي الأمر يختلف حيث انه لا يمكنك البرمجة بهذه اللغة دون ان تكون لديك خبرة في التعامل مع مكونات الحاسوب

لهذا سنبدأ من نقطة الصفر

(((((1 درس))))))

أول الاساسيات التي يجب ان تكون لك دراية بها هو أنظمة العد

النظام الثنائي Binary

النظام السداسي العشري Hexadecimal

النظام الثماني Octal

النظام العشري Decimal

طبعا هذه هي انواع انظمة العد الاساسية

سنشرحها واحدا واحدا في هذا الدرس

ملاحظة؛ [ انظمة العد ليست محصورة بين اربعة فهناك الثلاثي و التساعي و الأحدى عشري ... الخ



## النظام الثنائي Binary

يستعمل هذا النظام من قبل الحاسوب والدارات الكهربائية بشكل مباشر لفهم التعليمات البرمجية و يأخذ

قيمتان 0 و 1

مثال عن عدد ثنائي

10101B

وإذا حولناه للنظام العشري سنجد انه يمثل العدد 21

حرف B يشير الى انه بالبينائي



## النظام السداسي العشري Hexadecimal

يستعمل لعنونة أماكن الذاكرة العشوائية RAM حيث يأخذ كل قسم من الذاكرة رقم سداسي عشري

وهو افضل نظام في تمثيل الاعداد في عالم البرمجيات ولغة الاسميلي الارقام تبدأ فيه من الصفر وتنتهي عند

الحرف F

9 8 7 6 5 4 3 2 1 0

A B C D E F

ونكتب الاعداد في هذا النظام بهذا الشكل

مثلا العدد A12

يكتب هكذا

12AH

## النظام العشري Decimal

وهو النظام المتبادل بين الجميع وهو المستخدم في الحياة العملية ودرسناه في السلك الاول ابتدائي

9 8 7 6 5 4 3 2 1 0

في لغة الاسبلي ان ردنا كتابة عدد عشري وليكن العدد 105 مثلا نكتبه على هذا الشكل

105D

## النظام الثماني Octal

هذا النظام لن نستعمله كثيرا لكن لا بأس في ذكره

يبدأ من 0 وينتهي عند العدد 7

7 6 5 4 3 2 1 0

وهذا مثلا عن عدد ثماني

105O

الى هنا يطرح السؤال نفسه

هل يمكن التحويل بين الانظمة ؟ ( تحويل عدد من نظام الى نظام اخر )

وهذا ما سنراه في الدرس القادم بادن الله

والسلام عليكم

# لسلام عليكم

تكمّل معكم دورة الأسملي

في الدرس السابق تعرفنا على انواع انظمة العد

|

\\* / دورة في لغة الأسملي (لغة التجميع) من نقطة الصفر \\* / الدرس الأول

اليوم معنا درس مهم

|

\\* /-----\\* /

==> طريقة العد في الانظمة <==

==> تعريفات <==

\\* /-----\\* /

|

بإسم الله نبدأ

|

////////////////////

|

|

طريقة العد في الانظمة

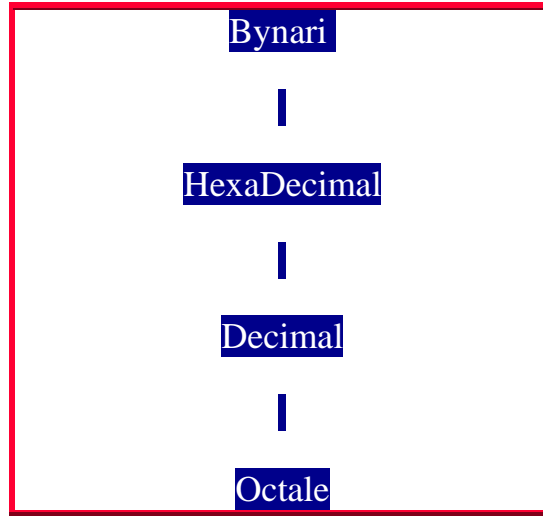
|

\\*/-----\\*/

|

كما درسنا في الدرس الاول انظمة العد هي اربعة اساسية

|



|

\\*/-----\\*/

النظام التنائي Bynari

|

لكي ابسط الامور لاحظ معي طريقة العد في النظام العشري

نبدأ هكذا : 0 1 2 3 4 5 6 7 8 9 تم 10

|

كيف اتت 10 ؟

|

نعلم ان النظام العشري مكون من عشرة رموز وهي 9876543210

|

بعد الوصول الى اخر رمز الذي هو 9 تنتهي الارقام

هل من دكي وجد الاجابة؟

الاجابة هي اننا نبدأ من جديد فتقلب 9 صفرا ونضيف 1 الى الخانة التالية

ومن هذا نستنتج ان 9 تليها 10

متلا عندما نصل الى 799

خانت الوحدات وصلت الى اخر رقم الذي هو 9 ادن ستصبح 10

ادن نضع مكانه 0 ونضيف 1 لخانة التالية

الخانة التالية بها 9 اذا اضفنا لها 1 ستصبح 10 ادن نضع 0 ونضيف 1 للخانة التالية فيصبح 8

==> 800 <==

اوكي الى هنا كل شيء واضح

الان لاحظ معي كيف يتم العد بالنظام الثنائي Binary الذي يتكون من 0 و 1 فقط ( اساسه 2 )

10 <-- 1 <-- 0

|

لاحظ : 0 تم 1 تم 10

|

ماذا وقع هنا ؟

|

كنا نعد بشكل طبيعي 0 يليه 1 تم يليه لا شيء

|

<== في النظام الثنائي 1 هو اخر رقم

0 -> 1 -> ?

|

كما ذكرنا سابقا سنقوم هنا بوضع 0 مكان علامة الاستفهام و 1 لخانة التي تليه فتصبح 10

|

تم 11 تم 100 كيف ؟

|

10 يليها 11 تم 100

|

لاحظ 11 خانة الوحدات وصلت الى اخر عدد ادن نستبدلها بصفر ونضيف واحد للخانة التي تليها

|

نلاحظ ان هذه الخانة ايضا وصلت الى اخر عدد ادن عندما نضيف 1 -> 1 ستصبح 10

|

يعني اننا نبقى 0 وننقل 1 للخانة التي تليها



لاحظ معي الجدول الاتي :

Decimal	Bynary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001

\\*/-----\\*/

النظام السداسي العشري HexaDecimal

يتكون هذا النظام من 16 رمز ( اساسه 16 )

==> FEDCDBA9876543210

لاحظ معي كيف يتم العد في هذا النظام

(ساكتب عموديا ، افقا يقع خلط في الترتيب )

0

1

2

3

4

5

6

7

8

9

A

B

C

D

E

F

10

كنا نعد بشكل طبيعي تم ووصلنا لآخر عدد هو **F**



التفسير هو اننا لما وصلنا لآخر عدد قمنا باستبداله بـ **صفر** وازافة **1** للخانة التالية

متال

405

406

407

408

409

A40

B40

C40

D40

E40

F40

410

411

412

\\*/-----\\*/

النظام الثماني Octale

يتكون هذا النظام من ثمانية رموز 7 6 5 4 3 2 1 0

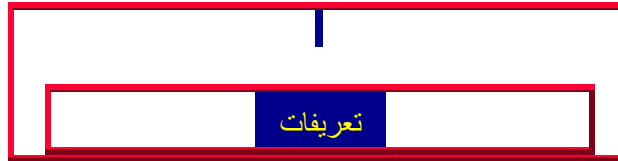
طريقة العد كالآتي

32 31 30 27 26 25 24 23 22 21 20 17 16 15 14 13 12 11 10 7 6 5 4 3 2 1 0 <==

33

\\*/-----\\*/

|



|

\\*/-----\\*/

bit & byte & two bytes & word & two words & double words & nibble

-----\\*/

تعريف البت

البت هو اصغر وحدة حاملة او ناقلة للمعلومات و يقوم الحاسوب بتخزين ومعالجة البيانات على

شكل bits

-----

كل رقم في النظام الثنائي يسمى bit

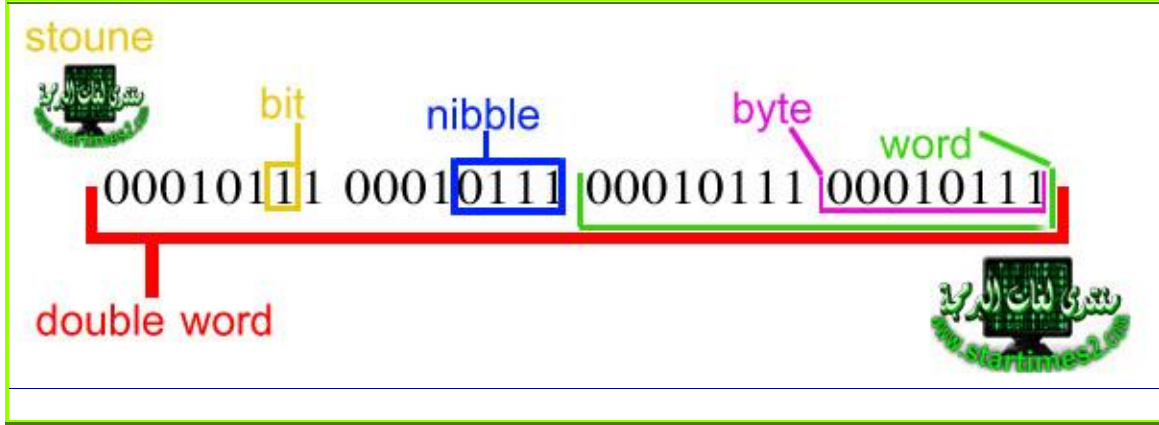
4bits --> NIBBLE

8bits-->BYTE

two bytes-->WORD

two words --> DOUBLE WORDS

لاحظ معي الشكل الاتي



اول بت ( في اليمين ) يسمى LSB

اخر بت ( في اليسار ) يسمى MSB

(لن اتطرق لدرس التحويل بين الانظمة سياخد مني وقتا طويلا ، هذا الدرس لوحده استغرق مني 3 ساعات )

|

السلام عليكم

دائما في إطار دورة الاسبلي نفتح الدرس التالت من الدورة

|

-----/\*

ها نحن قد وصلنا الى زمام الامور واليوم معنا اهم درس في مسيرة تعلم الاسبلي

حيث يبدأ الغموض والتكثر الاسئلة

بعد هذا الدرس سندخل في لغة الاسبلي

-----/\*

جهاز الحاسوب

-----/\*

الدرس مقسم الى جزئين ، الجزء الاول هو الذي ساشرحه الان والجزئ الثاني في موضوع منفرد  
غدا بادن الله



/\*\-----/\\*

### متطلبات الدرس

اهم واهم شيء ورقة وقلم لتسجيل بعض النقاط

القراءة مرارة عديدة بتمعن

قهوة كالعادة

/\*\-----/\\*

**ملاحظة:** هذا الدرس هو اساس الاسمبلي وكل شيء مبني عليه

كل من قال ان الاسمبلي لغة معقدة كن على يقين انه وصل الى هذا الجزئ وفقد الامل نظرا لانه لم  
يفهم شيئا

وانا متفق معه بان هذه الفقرة معقدة وغير مفهومة لكن كل شيء سيتضح مع تقدمنا في الدروس  
والامثلة القادمة ستصبح هذه الفقرة مهضومة الفهم

ركز جيدا وان شاء الله تفهم ولو القليل

اما الاسمبلي كلغة فهي سهلة جدا لان تعليماتها بسيطة وقليلة ومفهومة

/\*\-----/\\*



فهرس هذا الجزئ

تعريف بسيط للكمبيوتر

الذاكرة

وحدة المعالجة CPU

نضرة عن المعالجات

المسجلات

الرايات

/\*\-----\\*/

تعريف بسيط للكمبيوتر

جهازك الكمبيوتر عبارة عن آلة كهربائية تأخذ البيانات والتعليمات "inputs"

تم تقوم بمعالجتها ، ثم تخرج لنا المخرجات "outputs"

/\*\-----\\*/

الذاكرة

تستخدم الذاكرة لتخزين البيانات بها وتقسم الذاكرة الى عدة اقسام واهمها ذاكرة القراءة فقط ROM

بحيث يمكن القراءة منها ولا يمكن الكتابة عليها وعند انقطاع التيار الكهربائي تبقى البيانات مخزنة بها

**ملاحظة:** يمكن للكرار الوصول هذه الذاكرة وقراءة محتواتها بل وكسر حمايتها والتعديل بها

وهناك ذاكرة الوصول العشوائي RAM وهي ذاكرة للتخزين المؤقت وعند انقطاع التيار الكهربائي عنها تمحى البيانات الموجود فيها

تتكون الذاكرة من خلايا , لكل خلية عنوان Address وكل خلية تتكون من 8 بايت بتات اي بطول بايت

ويمكن تخزين رمزا واحدا فقط في كل خلية  
والعناوين في الذاكرة تبدأ من 0 الى FFFF

وهي بالنظام السادس عشر



/\*-----/\*

وحدة المعالجة CPU

هذه الوحدة هي عقل الحاسوب، ففيها يتم تنفيذ الأوامر أو التعليمات الصادرة من البرنامج

وتقوم بتنفيذ العمليات الحسابية والمنطقية وتقوم بتنظيم تزامن العمليات الأخرى في الحاسوب

وتقاس قدرة الحاسب بقياس قدرتها

وهي ما يعرف اليوم بالمعالج **MircoProcesseur**

ويتكون CPU من:

وحدات الحساب والمنطق

المسجلات Registers

وحدة التحكم Control Unit

الممر الداخلي Internal Bus



/\*-----\*/

## نضرة عن المعالجات

في القديم صممت شركة Intel معالج 4004 يحتوي على ناقل بيانات باتساع 4bits

كان قادرا على عنونة ذاكرة بحجم 640 بايت فقط

تم جاء المعالج 8086 بناقل بيانات bus باتساع 16 bits وناقل عناوين address bus باتساع 20 بت

وكان قادرا على عنونة ذاكرة بحجم 1 ميجابايت

بعد ذلك ظهر الـ Pentium بناقل بيانات باتساع 32 بت وايضا هناك معالجات حديثة بناقل بيانات باتساع 64 بت

للمزيد حول المعالجات من هنا

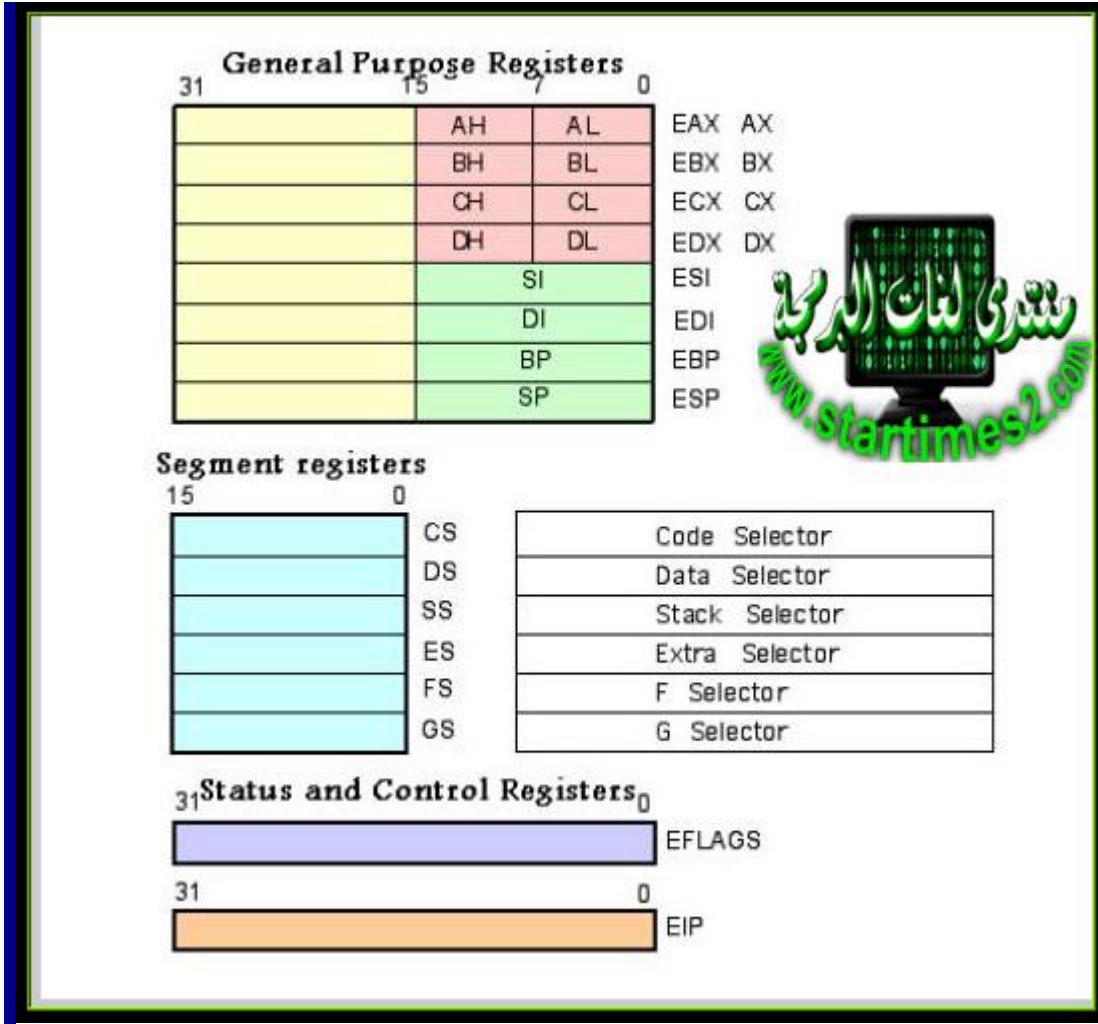
[http://fr.wikipedia.org/wiki/Liste\\_de\\_microprocesseurs](http://fr.wikipedia.org/wiki/Liste_de_microprocesseurs)

/\*-----\*/

## المسجلات

هذه الصورة توضح المسجلات وسيلي شرحها واحد تلو الآخر





### مسجلات معطيات

**EAX** و يستخدم كمراكم للمعاملات و لتخزين البيانات

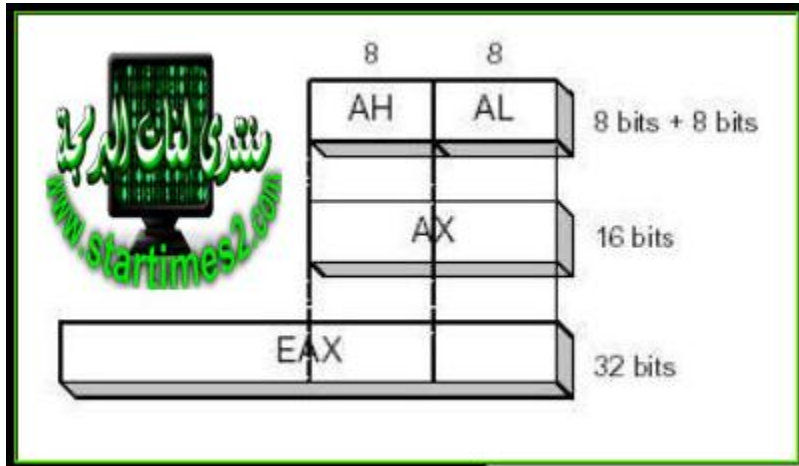
**EBX** يستخدم في فهرسة الذاكرة ( يشير الى عنوان القاعدة في الذاكرة )

**ECX** يستخدم كعداد للتحكم بعدد مرات تكرار الازاحة

**EDX** يستخدم في نقل البيانات وفي اعازات الادخال والايخراج ويستخدم كذلك في العمليات الحسابية



وهذه المسجلات الاربعة لها التركيب الاتي:



عند التعامل مع هذه المسجلات يمكن الوصول الى ال 32بت عن طريق المسجل EAX او الى او 16بت عن طريق AX

او حتى الى اول وتاني 8بت عن طريق AL و AH على الترتيب

وما ينطبق على EAX على ينطبق على EBX,ECX,EDX

/\*-----\*/

### مسجلات المقاطع

مسجل مقطع الشيفرة : CS يحتوي على عنوان اول حجرة في مقطع شيفرة البرنامج في الذاكرة، أي أنه يشير إلى بداية مقطع الشيفرة

مسجل مقطع المعطيات : DS يحتوي على عنوان اول حجرة في مقطع المعطيات في الذاكرة، أي أنه يشير إلى بداية مقطع المعطيات

مسجل مقطع المكس : SS يحتوي على عنوان اول حجرة في مقطع المكس في الذاكرة، أي أنه يشير إلى بداية مقطع المكس

مسجل مقطع المعطيات الإضافي : ES يحتوي على عنوان اول حجرة في مقطع المعطيات الإضافي في الذاكرة، أي أنه يشير إلى بداية مقطع

### المعطيات الإضافي

المقطع هو مكان في الذاكرة ، ويمكن للبرنامج ان يحتوي على اربعة مقاطع او يمكن اقل حسب

الحاجة وهي كالآتي:

مقطع Stack Segment

مقطع البيانات Data Segment يحتوي على المتغيرات والتوابت

مقطع البيانات الإضافي Extra Segment يستخدم في تخزين المعلومات الإضافية في حالة إذا لم يمكن هناك مكاناً في مقطع البيانات

ويمكن أيضاً استخدامه حتى ولو كانت هناك أماكن فارغة في مقطع البيانات

مقطع الأوامر Code Segment يحتوي على أوامر البرنامج

( كل هذا سنراه في الدروس القادمة الخاصة بالبرمجة بالاسمبلي )

/\*-----\*/

مسجلات التأشير والفهرسة

وهي عبارة عن أربعة مسجلات مساعدة تساعد في إيجاد العنوان الفيزيائي بالتعاون مع مسجلات المقاطع

مسجل دليل المصدر : SI يخزن فيه عنوان يدل على الإزاحة ضمن مقطع المعطيات DS و بمعنى آخر يستعمل في إمساك العناوين الفعالة من

أجل التعليمات التي تتناول المعطيات المخزنة في مقطع المعطيات في الذاكرة .

مسجل دليل الهدف : DI يخزن فيه عنوان يدل على الإزاحة ضمن مقطع المعطيات الإضافي ES، و بمعنى آخر يستعمل مسجل دليل الهدف DI

من أجل استنتاج العنوان الفيزيائي الذي يحدد حجرة متحول الهدف

مسجل مؤشر المكس : SP يسمح مؤشر المكس بوصول سهل للحجرات في مقطع المكس (ساشر حها) الموجود في الذاكرة حيث أن القيمة ف

في SP تمثل العنوان الفعال لحجرة المكس التالية التي يمكن الوصول إليها نسبة إلى العنوان الحالي الموجود في مسجل مقطع المكس SS و

يحتفظ SP دوماً بقيمة تدل على قمة المكس ، هذا و إن قيمة هذا المسجل تتعدل تلقائياً عند وضع أو سحب معلومة بالمكس .

مسجل مؤشر القاعدة : BP يحوي قيمة تدل على الإزاحة بالنسبة لمقطع المكس SS و هو يستخدم لقراءة المعطيات ضمن مقطع المكس بدون

إزالتها من المكس

//\*-----//

ان مسجلات المقاطع تعمل جنباً الى جنب مع مسجلات البيانات للوصول الى اي عنوان في الذاكرة

نأخذ مثالا : نريد الوصول الى عنوان الذاكرة الفيزيائي 12345h

يجب ان نضع DS = 1230h و SI = 0045h

كيف يسقوم بالوصول الى العنوان الفيزيائي 12345h؟

انه يقوم بعملية حسابية تتمثل في ضرب مسجل المقاطع في 10h وجمعه مع مسجل البيانات

هكذا:

$$1230 * 10 = 12300$$

$$12300 + 0045 = 12345$$

////////////////////

ان كل عنوان مكون بمسجلين يسمى effective address

ان BX,SI,DI تعمل مع DS اما BP,SP اما فتعمل مع SS

المسجلات العامة الاخرى لا يمكنها ان تكون effective address

//\*-----//

مؤشر التعليمات أو الأوامر EIP

EIP دائما يعمل مع CS وهو دائما يشير الى التعليمه يجري تنفيذها حاليا

//\*-----//

## الرايات

ان الرايات يتم تغييرها تلقائيا من قبل CPU بعد تنفيذ عمليات رياضية ومنطقية , انها تسمح بمعرفة نتيجة العملية وتحديد الشروط لنقل التحكم الى

اجزاء اخرى من البرنامج

تنقسم الرايات لثلاثة اقسام :

**رايات غير مستعملة** : اي انها لا تفيد في الحكم على اخر عملية وهي موجودة فقط في حالة تطوير المعالج ربما يحتاجون رايات اضافية ويمكن

استعمال هذه الرايات

**رايات الوضع** : وهي الرايات التي تتاثر وتتغير حسب وضع العمليات التي تقوم بها وحدة الحساب والمنطق في المعالج

**رايات السيطرة** : وهي رايات المبرمج يتحكم بها فمثلا ادى وضع بها القيمة 1 تبقى هذه القيمة حتى يغيرها المبرمج في البرنامج عن طريق اوامر

خاصة بها

مسجل الرايات في المعالج يحتوي على 7 رايات غير مستعملة و 6 رايات وضع و 3 رايات سيطرة واهمها :

الرمز	اسم الراية	النوع	ملاحظات
CF	Carry Flags	وضع	بأخذ 1 اذا اعطت البت الاخير عن اليسار 1 الى الخارج او اخذت 1 من الخارج والا يأخذ صفر
PF	Parity Flags	وضع	اذا كان عدد 1 في اول 8 بتات من النتيجة زوجي تأخذ هذه الراية 1 والا تأخذ صفر
AF	Auxiliary Flags	وضع	اذا اعطت البت الرابعة البت الخامسة 1 او اخذت البت الرابعة من البت الخامسة 1 تأخذ هذه الراية 1 والا صفر
ZF	Zero Flags	وضع	اذا كانت النتيجة صفر تأخذ هذه الراية 1 والا تأخذ 0
CF	Sing Flags	وضع	اذا كانت النتيجة سالبة تأخذ هذه الراية 1 والا صفر
TF	Trap Flags	سيطرة	اذا وضع المبرمج بها قيمة 1 يتم تنفيذ امر واحد فقط من الاوامر وبعدها تأخذ صفر
IF	Interrupt Flags	سيطرة	نضع 1 للسماح لعمليات interrupt في مدخل interrupt للمعالج سيطرة
DF	Direction	سيطرة	لتحديد اتجاه الحركة في حالة قراءة النصوص

	Flag		
OF	Overflow Flags	وضع	تأخذ 1 اذا كانت النتيجة اكبر من المكان المخصص لحفظها والا تأخذ صفر

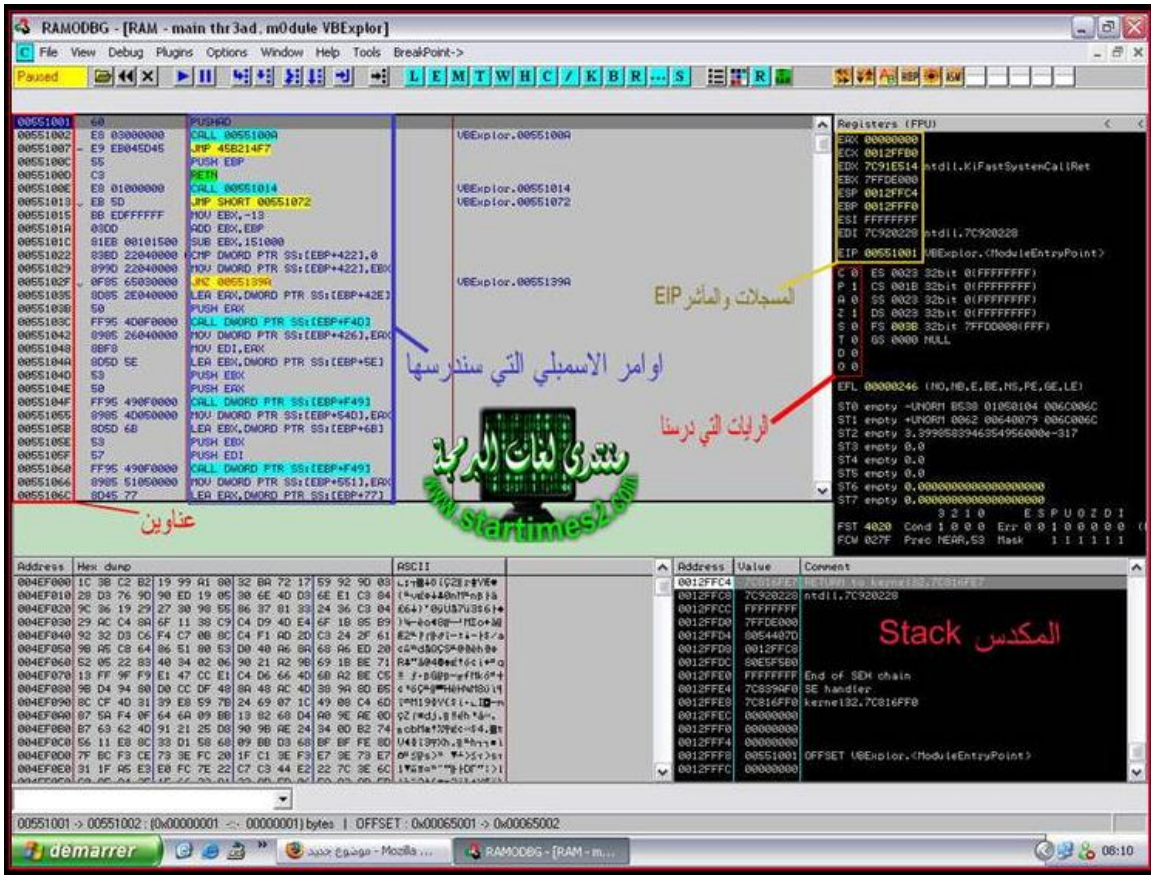
-----\*\*

نظرة قريبة من برنامج OillyDBG على كل ما سبق وذكرته

قمتم بفتح احد البرامج ببرنامج OillyDBG وهذه صورة مع الشرح

(شاهد الصورة هنا لتظهر لك بشكل واضح)

<http://img205.imageshack.us/img205/3535/reda.jpg>



انتهى هذا الجزئ

يبدا ان درس سهل ويفتح الشهية

في الجزئ الثاني سنرى

شرح المكس Stack

العنونة

شرح تفصيلي لل Effective Adresses

ارجو استوعاب هذا الدرس ولو بنسبة 60%

لاني ساتطرق في الموضوع الاتي الى امر معقد وهو العنونة و قضية العنوان الفيزيائي والمنطقي  
على 16 بت

رغم انها لم تعد الان مع معالجات 32 بت لكن مهم فهمها

تم الدرس الذي يليه ندخل في لغة الاسمبلي بادن الله

اي سؤال حول الدرس ؟





/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*/\*\*

دائماً في نفس الاطار لدي سؤال اخر

هل تعرف البرنامج الذي يفتح الملفات التنفيذية؟

حسناً نأخذ مثالا عن ملفين الأول بامتداد mp3 والثاني exe

او كي الان قم بتغيير امتداد الملفين mp3 اجعله exe و exe اجعله mp3

ماذا تلاحظ؟ تم عطب الملفين ولا يشتغلان لكن لماذا؟

كما هو معلوم فان ملفات mp3 يقوم بفتحها برنامج قارئ ملفات mp3 مثل Windows Media Player

ويتعرف على الملف من خلال امتداده

ولكن من الذي يفتح برامج exe؟

إنه Loader الويندوز كيف؟

عندما تعمل دابل كليك على الملف يقوم loader بتحميل الملف من القرص الصلب الى الذاكرة ثم يملأ المعلومات الخاصة

به في قائمة ال process ثم يقوم المعالج بحجز مسجلات لهذا الملف eax, ebx... ثم مؤشر على على التعليمية التي

سينفذ ... EIP وأول تعليمية تنفذ من قسم الكود هي نقطة البداية , وكل برنامج له نقطة بداية خاصة به...

هل عرفت الان لماذا ندرس الذاكرة وكيف تعمل وماذا يوجد بها والمعالج والمسجلات وكيف يعمل وكيف ينسق مع الذاكرة...

لهذا تجد مبرجي الاسمبلي يمتلكون قدرة هائلة في المجال لانهم يستطيعون معرفة سير اي تطبيق او برنامج

ومعرفة جميع دواله وكيف عمل هذا ولماذا ظهرت هذه الرسالة ....

سنرى كل هذا في الدروس القادمة بادن الله.

كانت هذه لمحة بسيطة فقط لأقرب للمتبعين ما الذي ندرسه ولمادا

\*/---/\*---\*/

بإسم الله نبدأ درسنا لهذا اليوم

سنشرح اليوم:

\*/-----\*/

<== بنية المعالج 8086 ==>

<== بنية الذاكرة ==>

Stack <== المكس ==>

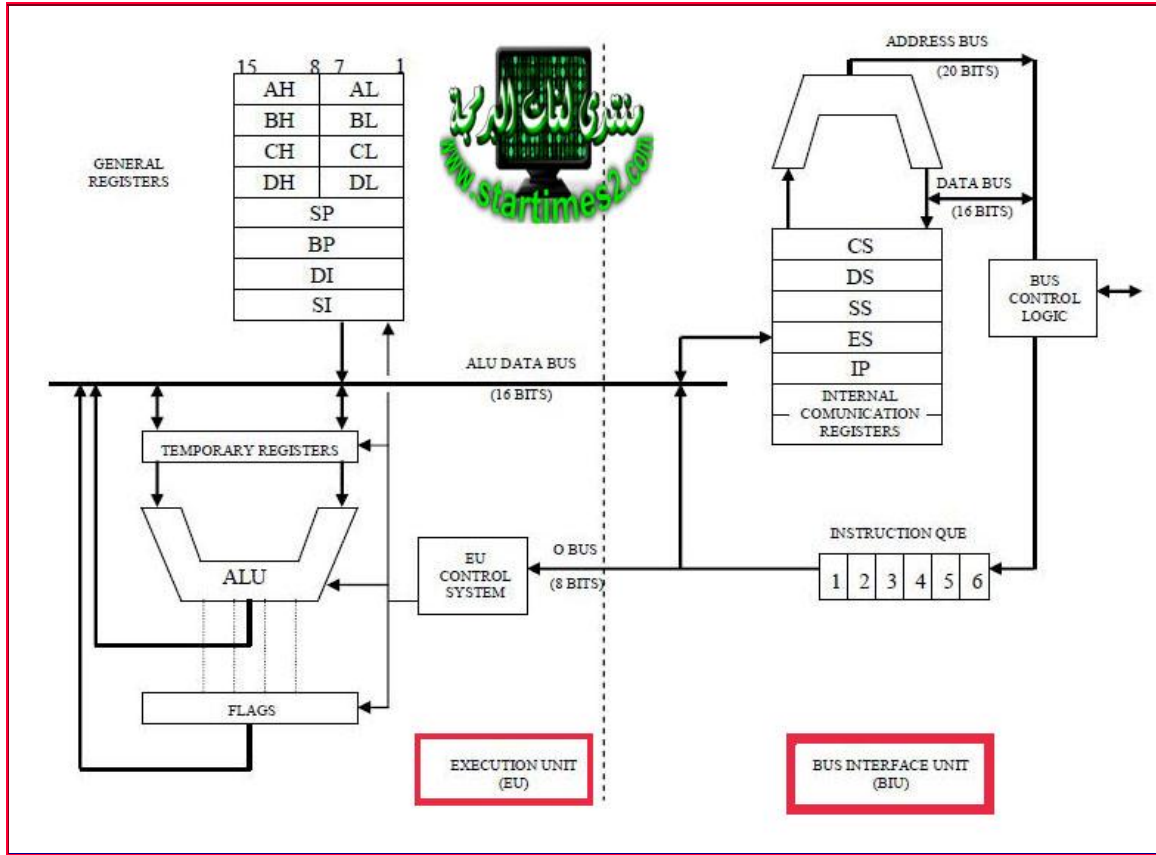
==> Memory Segmentation <==

\*/-----\*/

بنية المعالج 8086

لتشاهد الصورة الآتية:





الآن نشرح الصورة:

كما هو ملاحظ في الصورة هناك وحدتان وهما:

**Bus interface Unit** وحدة ملائمة الممرات , ونرمز لها بـ **BIU**

**Execution Unit** وحدة التنفيذ , ونرمز لها بـ **EU**

<b>BIU</b>	مسؤولة عن معظم الأعمال مثل : إحضار التعليمية، قراءة و كتابة المتحولات في الذاكرة، إدخال وإخراج المعطيات من و إلى الأجهزة المحيطة
<b>EU</b>	مسؤولة عن تنفيذ التعليمات. و كلا الوجدتين تعملان بشكل متواز لتخفيض الزمن المطلوب لإحضار عدة تعليمات و تنفيذها.

في الحاسوب هناك ثلاث ممرات وهي :

==> 1 ممر المعطيات : DATA BUS يصل بين المعالج و الذاكرة وظيفته نقل المعطيات من و إلى الذاكرة

==> 2 ممر العناوين : ADDRESS BUS يصل بين المعالج و الذاكرة أيضا و وظيفته نقل العناوين من المعالج إلى الذاكرة

==> 3 ممر التحكم : CONTROL BUS لتنسيق عمل الممرين السابقين

\*\*\*\*\*

الان نوضح عمل الوحدتين بشكل دقيق :

### وحدة ملائمة الممرات Bus Interface Unit

و تستخدم لملائمة المعالج مع العالم الخارجي. و تتألف من:

جامع العناوين، مسجلات المقاطع، وحدة التحكم بالمحرف وصف التعليمات .

تقوم وحدة BIU بالتحكم بممر المعطيات و ممر العناوين و ممر التحكم

تحضر BIU التعليمات من الذاكرة بايت بايت و تضعها فيما يسمى بصف التعليمات الذي يتسع لست بايتات

كحد أعظمي و من الطبيعي أن التعليمات التي تدخل صف التعليمات أولا يتم تنفيذها أولا للمحافظة على ترتيب التعليمات و

يدعى هذا المبدأ بالداخل أولا خارج أولا First In Last Out و نرسم لهذا المبدأ ب FILO

إن إحضار شيفرة التعليمات التالية يتم عندما تكون وحدة التنفيذ EU مشغولة بتنفيذ التعليمات الحالية ( هذه إحدى محسنات

المعالج 8086 عن أسلافه حيث كانت ال CPU في المعالجات السابقة للمعالج 8086 تتوقف عن العمل خلال فترة

تنفيذ التعليمات الحالية )

عندما تفك وحدة التنفيذ EU شيفرة تعليمات ما من صف التعليمات و تكون هذه التعليمات تعليمات تؤدي إلى تغيير تسلسل

تعليمات البرنامج) قفز إلى برنامج فرعي مثلا (عندها يتم تصفير صف التعليمات و إعادة ملئه من جديد بتعليمات البرنامج



## تلك الحجرة.

يوضع داخل كل حجرة رقم ست عشري يتراوح بين 0 و FF يدعى هذا الرقم بمحتوى تلك الحجرة

لاحظ الصورة من الدرس السابق:

FFFFFH
-----
-----
00004H
00003H
00002H
00001H
00000H

حسنا نكمل

يوجد بين المعالج و الذاكرة ممران هما ممر المعطيات بعرض 16 بت و ممر العناوين بعرض 20 بت

فمثلا عندما يحتاج المعالج إلى القيمة المخزنة في الحجرة ذات الرقم ( 100 عنوانها 100 ) فإن الرقم 100 يمثل

بشكل ثنائي و يوضع على ممر العناوين و يرسل إلى الذاكرة، و حالما تستلم الذاكرة هذا العنوان فإن محتوى الحجرة

100 يرسل إلى المعالج عن طريق ممر المعطيات

ان كون ممر العناوين ذو عرض 20بت ( 20 خط نقل ) هذا يعني أنه يستطيع نقل رقم ثنائي ذو 20 خانة أي أن

أكبر قيمة يمكن وضعها على ممر العناوين هي:

$$1\text{MB} \text{ (تعني اس } ^{\wedge}\text{) تساوي تقريبا } 1048576 = 2^{20}$$





وهذا ما يسمى مبدأ **Last In First Out** آخر ما يدخل اول ما يخرج ( زربان )

تحتاج البرامج التنفيذية الى منطقة في الذاكرة تسمى **المكدس** ، تفيد هذه المنطقة في توفير حيز من الذاكرة لتخزين العناوين

والمعطيات بشكل عام ، علما ان طول عنصر المعطيات الاساسي في المكدس هل كلمة واحدة **DWORD** في بيئة **32 بت**

وينبغي على المبرمج تعريف المكدس في برامج **exe** في حينه الا انه لا حاجة لذلك في برامج **com**

حيث يتولى نظام التشغيل تعريف مكدسة اليا . يقوم **DOS** بتهيئ مسجل المقطع **SS** بعنوان بداية المقطع المكدس

كما يهئ المسجل **ESP** بحجم المكدس ليشير الى نهايته او او قيمته الحالية .تتعامل مجموعة من التعليمات مع المكدس

بشكل مباشر كالتعليمات **PUSH / POP / CALL**

فالتعليمة **PUSH** تقوم بنقص محتويات المسجل **ESP** بمقدار كلمتين **DWORD** عند تنفيذها وتخزن قيمة ما في

تلك الكلمة المحجوزة التي اصبح **ESP** يشير اليها .

اما تعليمة **POP** فهي على العكس حيث تنقل الكلمة التي يشير اليها **ESP** الى موقع ما

( مسجل او مكان في الذاكرة ) تم تسحبها من قمة المكدس وتزيد قيمة المسجل **ESP** بمقدار كلمتين

(سندرس تعليمات الاسملي في الدرس القادم بشكل واضح )

°+-----+°

دينا بعض القيم مخزنة في بعض المسجلات

**AX = 01234H**

**BX = 04BA7H**

CX = 0FF17H

DX = 034E0H

لنرى ماذا يجري عندما نطبق عليها اوامر المكس

سنقوم بتنفيذ التعليمات الآتية:

PUSH AX

PUSH BX

PUSH CX

PUSH DX

ننظر الى مثال عملي:

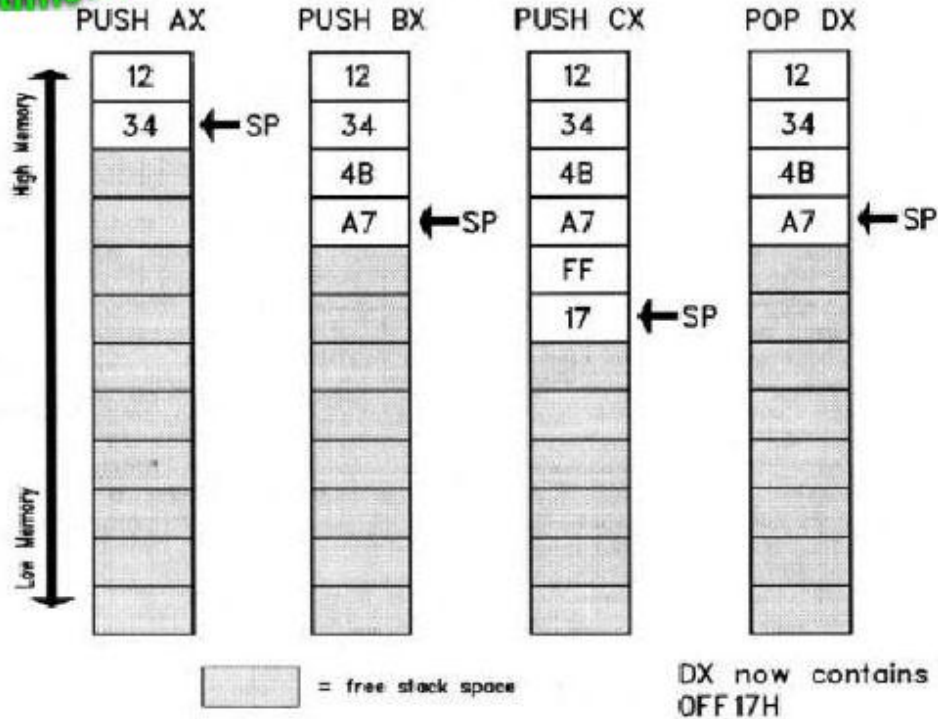




Given these  
Initial register  
values:

AX = 01234H  
BX = 04BA7H  
CX = 0FF17H  
DX = 034E0H

After...



فلاحظ انه اذا حفظنا قيم المسجلات AX, BX, DX, CX على الترتيب باستعمال التعليمات:

PUSH AX

PUSH BX

PUSH DX

PUSH CX

فعلينا استرجاعها هكذا

POP CX

POP DX

POP BX

POP AX

وهو هو مبدأ LIFO

(last in first out) اي ان اخر قيمة تنقل الى المكس هي اول قيمة تخرج منه

لاحظ ان القيم التي تم استرجاعها من المكس تبقى موجودة فيه لكن لا يآشر عليها المسجل SP

مما ينبغي ذكره هنا هو وجود حالتين يجب الا يمر بهما المكس

|

الاولى هي محاولة سحبنا قيمة من المكس وهو في حالة Underflow اي فارغ يحتوي على القيمة  
FFFFH

الثانية هي محاول دفع قيمة الى المكس وهو في حالة overflow يعني انه ممتلأ

ولربما سمعتم بتغرات فيض المكس BUFFER OVERFLOW وما اكثرها وتم اختراق الاف  
السرفرات بواسطتها

وهو اشبه بخطأ برمجي ويقع فيه كبار المبرمجين وسأشرح هذه التغرة بالتفصيل في دروس قادمة

لكي يتوخى المبرمجون الحذر

\*/---\*/\*\*\*/---\*/

|

## Memory Segmentation

(الشرح على بيئة 16 بت وصالح لبيئة 32 بت)

فقرتتا كلها عن مسجلات المقاطع Segment Registers

ادا رجعت للدرس السابق بالطبط فقررة مسجلات المقاطع ستلاحظ انها كلها تحتوي على  
عناوين لأجزاء معينة في الذاكرة

ركز معي تحتوي على "عناوين"

معالج 8086 هو معالج 16بت ، بمعنى أن حجم المسجلات فيه هي 16بت (2بايت)

ذكرنا أن المعالج 8086 يعنون 1 ميغا بايت من الذاكرة والسبب أن العنوان يتكون من 20 بت ( عدد خانات العنوان 20 خانة بت )

وبالنظام السادس عشر تكون (5 Hex Digits)

أي أن الذاكرة ستكون بالشكل التالي - بالنظام الثنائي :-

العنوانين من صفر الى آخر عنوان :

```
0000 0000 0000 0000 0000
0001 0000 0000 0000 0000
0010 0000 0000 0000 0000
0011 0000 0000 0000 0000
....بقية العنوانين هنا.....
....بقية العنوانين هنا.....
1111 1111 1111 1111 1111
```

والى آخر عنوان وهو بالنظام العشري .1048575

طبعا التعامل بالنظام السادس عشر أسهل كثيرا ، نشاهد العنوانين مرة أخرى ولكن بالنظام السادس عشر :



حسنا ، الى هنا الأمر جميل .. لدينا عنوان بمقدار 20 خانة ، ومسجل بمقدار 16 خانة ، ومسجلات **Segment** تخزن بها

العنوانين و مهلا مهلا مهلا !! هنا تكمن المشكلة ، كيف يمكن أن نحفظ بعنوان بطول 20 بت في مسجل بطول 16 بت ، لا يمكن

بالطبع ، اذا ما العمل؟؟

هنا قامت الشركات بدلا من توسيع المسجلات في تلك الفترة بعمل طريقة بسيطة تكمننا بالوصول الى أي عنوان في الذاكرة عن

طريق استخدام مسجلين مع بعض وجعل الذاكرة في صورة مقاطع **Segment** ، وهنا تبدأ قصتنا

الفكرة تكون بتقسيم الذاكرة الى أقسام تسمى **Segment** كل قسم منها حجمه 64 KB ، ومن داخل هذا القسم أو المقطع المعين لكي

تصل اليه تستخدم عنوان آخر يسمى **offset**

إذا لكي تصل لأي موقع في الذاكرة عليك أن تستخدم عنوان المقطع أولا للوصول للمقطع المحدد ثم تستخدم **offset**

للوصول الى المنطقة المعينة داخل المقطع..

هذا العنوان **Segment : offset** يسمى العنوان منطقي **Logical Address** ، ومن خلاله نستطيع الحصول على العنوان الحقيقي

في الذاكرة .. **Physical Address** بالتأكيد في حال كانت لديك فكرة عن هذا المعالج أن التحويل من ال **Logical Address** الى

العنوان الفيزيائي يتطلب الضرب ب16) اضافه 0 في البدايه (ثم نقوم بجمعه مع ال **Offset** وبالتالي نحصل على العنوان الفيزيائي

إذا للوصول الى أي عنوان في الذاكرة سنستخدم ( **Segment : Offset** عنوان : عنوان ) العنوان الأول **Segment** يكون موجود

في أحد مسجلات المقاطع **Segment Registers** والعنوان الثاني يكون في أحد المسجلات مثل **IP** أو **SP**

التحويل من العنوان المنطقي للفيزيائي كما ذكرنا أعلاه يتم من خلال اضافة صفر لل **Segment** ثم جمعه مع ال .. **offset** يعني

يمكن أن يكون القانون كالتالي:

$$\text{Physical Address} = \text{Segment} \times 16 + \text{offset}$$

(راجع الدرس السابق)

نأخذ مثال توضيحي ، العنوان المنطقي **F000:FFFF** ، لكي نحصل على الفيزيائي:

$$F0000 + FFFF = FFFFF$$

العنوان الفيزيائي اذن هو **FFFFD**

-----

مثال آخر **F:E2FF923**

$$293F0 + E2FF = A06EF$$

العنوان الفيزيائي اذن هو **A06EF**

الى هنا الأمر جميل للغاية ، وعرفت كيف أن تحل المشكلة باستخدام **Segment : offset**

او كي ساشغلكم قليلا الان

لدينا العنوان المنطقي:   
0007:7B90 كم هو العنوان الفيزيائي له ؟  
0008:7B80 كم هو العنوان الفيزيائي له ؟

بعد أن تحلهم أنا متأكد أنك ستكون محترار للغاية !! والسبب أن العنوانين المنطقيين لهم نفس العنوان الفيزيائي!

سازيدك جرب أيضا:

0049:7770 و 07BD:0030 و 07BF:0010 و 07C0:0000 و و و و ... هناك حوالي 1,985

عنوان منطقي لهم نفس العنوان

الفيزيائي هذا !!

الآن اعمل استراحة، اشرب قهوة كالعادة ( sbitar Imajanine )

و مرحبا بك في عالم ال **Overlapping**

( عفاوا لا اعرف ترجمة الكلمة لانني درست الاسمبلي من مراجع انجليزية )

خرج عنوان فيزيائي واحد لعدة عناوين منطقية ??? وذلك لأن الذاكرة كما ذكرنا مقسمة لمقاطع كل (منها بحجم ) 64 KB أي 65,536 بايت

ولكن بعد كل 16 بايت من بداية المقطع تكون هذه بدايه المقطع الثاني ، وبعد 16 بايت تكون بداية المقطع الثالث وهكذا ... أي المقاطع

متداخلة في بعض ، ولذلك أن ممكن أن أصل لعنوان في المقطع الثالث عن طريق المقطع الأول أو الثاني .. مثلا عنوان في المقطع 100

سوف أستطيع الوصول اليه عن طريق المقطع 99 أو 98 وما قبله بكثير ، وهكذا كل ما نكون في مقطع أكبر كلما زاد عدد العناوين المنطقية

التي أستطيع الوصول من خلالها الى ذلك العنوان

الصورة التالية تبين لك المقطع الأول يكون ( 00000 عنوان فيزيائي ) وينتهي ب 0FFFFH ، والمقطع الثاني يبدأ 00010H أي أنه يبدأ بعد بدايه

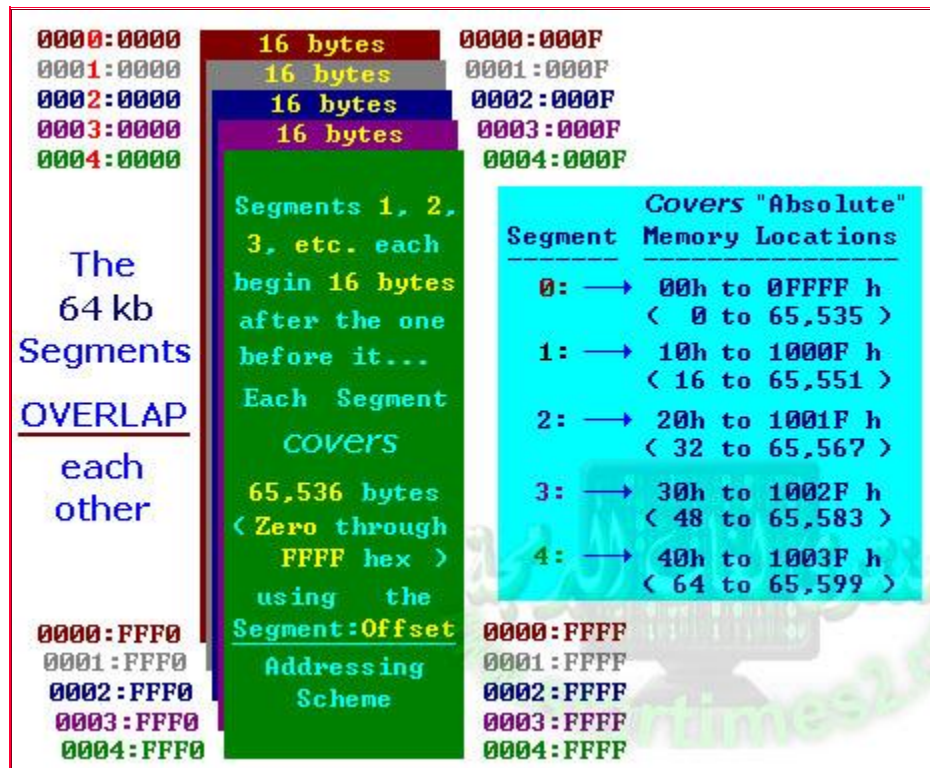
المقطع الأول ب 16 بايت .. ومن ثم يبدأ الثالث أيضا بعد 16 بايت وهكذا تبدأ جميع المقاطع ( تذكر أن حجم أي مقطع هو 64 .. (KB تسمى

المنطقة من بداية المقطع الى بداية المقطع الأخر ) التي بحجم 16 بايت ( باسم Paragraph



بدايه المقطع		نهايه المقطع		
0	0000:0000	00000H	0000:FFFF	0FFFFH
1	0001:0000	00010H	0001:FFFF	1000FH
2	0002:0000	00020H	0002:FFFF	2000FH
3	0003:0000	00030H	0003:FFFF	3000FH
4	0004:0000	00000H	0004:FFFF	4000FH
.	.....	.....	.....	.....
L	FFFF:0000	FFFF0H	FFFF:FFFF	10FFE FH
	Logical	Physical		

الصورة التالية سوف تزيل ما تبقى لديك من غموض والتباس في المفهوم:



تمعن في الصورة جيدا ، فهي موضحة لكي شيء

نعود للصورة ، لاحظ لدينا هنا أول 5مقاطع في الذاكرة (تم تجاهل الباقي ، لأنهم بنفس المفهوم  
طبعاً .. (لاحظ ال16 بايت في بداية كل مقطع

وأهم شيء هو ملاحظة التداخل في المقاطع ، المقطع الأول بدأ، وبعد 16بايت بدأ الثاني ، ثم  
الثالث ووو هكذا .. لذلك لو أردت أن أصل

لقيمة في المقطع ( 4باللون الأخضر (يمكن أن أستخدام المقطع رقم .. 4 أو 3 أو 2 أو 1 أو 0 لأن  
هذه المنقطة تقع ضمن ال64 KB الخاصة

بهم ، فهتم الآن لماذا يمكن أن يكون لعنوان فيزيائي أكثر من عنوان منطقي

لو أردنا أن نصل لمنطقة في المقطع (قبل بداية المقطع الثاني ، بكم طريقة يمكن الوصول لها؟؟  
أنظر للصورة وسترى أنه لا يوجد غير طريقة

فقط باستخدام المقطع 0 ثم Offset من 0 الى ) F قبل بدايه المقطع الثاني (

لذلك هذه المنطقة لا يمكن الوصول اليها الا من خلال المقطع 0 فقط

الصورة التاليه توضح هذه النقطة بمزيد من التوضيح ، لاحظ اللون اللبني (سماوي) في اليمين وهو  
عبارة عن الطرق الممكنة للوصول للعنوان

الموجود قبل عمليه المساواة في الصورة

0:0000	16 bytes 00h through 0Fh	0000:000F	
1:0000	16 bytes 10h through 1Fh	0001:000F =	0000:001F
2:0000	16 bytes 20h through 2Fh	0002:000F =	0001:001F 0000:002F
3:0000	16 bytes 30h through 3Fh	0003:000F =	0002:001F 0001:002F = 0000:003F
4:0000	16 bytes 40h through 4Fh	0004:000F =	0003:001F 0002:002F 0001:003F 0000:004F
and so on ....			
FFF:0000	16 bytes FFF0h through FFFFh	0FFF:000F =	0000:FFFF 0001:FFEF, etc.
1000:0000	16 bytes 10000h through 1000Fh	1000:000F =	0001:FFFF 0002:FFEF, etc.
1001:0000	16 bytes 10010h through 1001Fh	1001:000F =	0002:FFFF 0003:FFEF, etc.
1002:0000	16 bytes 10020h through 1002Fh	1002:000F =	0003:FFFF 0004:FFEF, etc.

لاحظ بعد ما بعد الفاصل في الصورة ، ستجد أن المقطع الأول أنهى عند **FFF:000**

قديمًا كانت الذاكرة **100 KB** شيء كبير جدًا ، لذلك **640KB** كان شيء خرافي .. ولم يكن هناك ذاكرة أصلاً بهذا الحجم ، لذلك لم تستغل

المناطق ما بعد هذا الحجم .. لذلك أطلق على المساحة الباقية "محموزة" أو الأفضل **Upper Memory Area** اختصاراً **UMA** هذه المنطقة

تبدأ من العنوان **A000** وتستمر إلى النهاية حوالي **384 KB** أي المجموع هو **640 + 384** يساوي **1,048,576** وهي **1 ميغا**.. (المقطع

الأخير في الذاكرة يسمى **High Memory Area** اختصاراً **HMA** ، على العموم لن نتحدث فيه لأنه حالياً أنتهى هذا الكلام .. الذاكر الآن

أصبحت بعشرات ال **Gigabyte** أيضاً موضوع **Segment : offset** فقط هو للمعالج **8086** و **8088** ( و **80286** معالجات **16-بت** والتي

تعمل في النمط الحقيقي **Real Mode**

حاليا المعالجات تدعم النظام المحمي وهو لن لن يسمح برنامج في الذاكرة الدخول في منطقه  
برنامج آخر Protected Mode

/\* \*/

اذا تعرفنا على حل مشكلة العنوانين باستخدام الزوج Segment:Offset وتعرفنا على موضوع  
ال Overlapping وذكرنا أنه يمكن أن يكون لأي

عنوان فيزيائي أكثر من عنوان منطقي والسبب كما ذكرنا هو تقسيم الذاكرة لمناطق Segments  
متداخلة مع بعضها البعض

من هنا نعود للمسجلات Segment Registers والتي ذكرناها في أول الموضوع وهي:

( CS,DS,SS,ES سوف نتجاهل الآن ال ES لأنه لا يستخدم الا نادرا .. )

كل واحد من هذه المسجلات يجب أن يحتوي على عنوان المقطع ، وبمساعده مسجل آخر نقوم  
بتخزين ال offset فيه ، نستطيع الوصول لأي

منطقة في الذاكرة ،تهيئة هذه المسجلات بقيم صحيحة ضروري جدا لكي يعمل برنامجك ، فيجب أن  
يحتوي CS على عنوان المقطع الذي

يوجد به الكود (IP) يحتوي على (offset ، أيضا DS يجب أن يحتوي على عنوان مقطع البيانات

(في حال عرفت بيانات في الجزء Data Segment

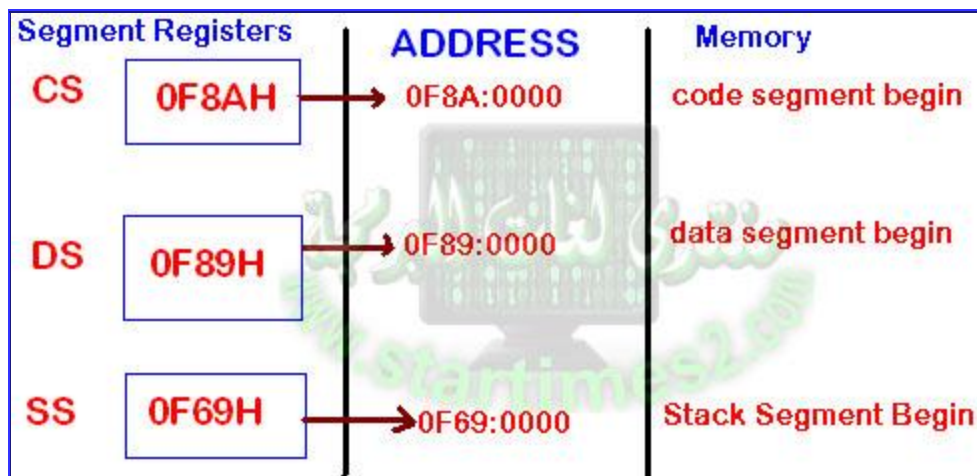
عليك أن تعرف الفرق بين Data Segment

( وهو المقطع في الذاكرة الذي يحتوي على البيانات التي تعرفها في برنامجك )

وبين DS Register وهو المسجل الذي يحتوي على عنوان Data Segment

بالنسبه ل SS يجب أيضا أن يحتوي على عنوان ال Stack و SP يحتوي على (offset ، انظر  
الصورة التالية التي توضح بالضبط الأمور التي

يجب أن تحصل في البدايه:



\*\*\*\*\*/

والمعالجات الان أصبحت تعنون 32بت فالمعالج يعتبر الذاكرة كلها منطقه واحدة flat ويستطيع الوصول لأي منطقة من خلال عنوان بطول 4 بايت

\*\*\*\*\*

\*\*\*/

انتهى هذا الدرس

بعد ان استغرق مني 8 ساعات والله

في الدرس القادم سندخل في تعلم لغة الاسمبلي بادن الله

لكن قبل ذلك سافتح موضوع للنقاش وطرح اسئلتكم حول المسائل التي بقيت غامضة





MOV AL, BL

MOV BL,60, H

عند تنفيذ الأمر الأول MOV AL, BL ستصبح قيمة المسجل AL تساوي 50H

عند تنفيذ الأمر الثاني MOV BL,60H ستصبح قيمة المسجل BL تساوي 60H

الأمر MOV لا يؤثر على الرايات يعني ان قيم الرايات لا تتغير

يجب ان يكون المصدر من نفس كبر الهدف (بايت، بايت) او ( word , word )

(راجع درس المسجلات )

مثال:

MOV AX, CX      MOV DL ,AL      MOV CH,20,H

لاحظ الجدول الآتي

السبب	الوضع	الأمر
لان op2 اكبر من op1	غير صحيح	MOV AL,BX
لا يمكن تخزين عدد من ثلاث منازل في الميزان 16 في مسجل بكون 8 بتات. يمكن تخزين قيمة بكون منزلة او منزلتين لان H78=120 أي انه يتكون من منزلتين	غير صحيح	MOV AL,123H
لانه في مسجل بكون 16 بت يمكن تخزين قيمة بكون 4 منازل على الاكثر في الميزان السادس عشر	غير صحيح	MOV BL,120
	غير صحيح	MOV DX,12345H

-----\*

تعليمة ADD

تقوم هذه التعليمة بإضافة قيمة المعامل الثاني لقيمة المعامل الأول ، وينجم عن هذه العملية تغيير في محتوى الرايات، ويتم تخصيص ناتج

الجمع للمعامل الأول.

قد يكون أحد المعاملين سجلا، كما قد يكون المعامل الثاني مرجع ذاكري (متغير أو عنوان ذاكري) أو رقم مباشر.

مثال:

MOV AX,3d

ADD AX,2d

الناتج AX = 5d

### تعليلة SUB

يعكس عملية الجمع، فإن التعليلة SUB تقوم بطرح قيمة المعامل الثاني من قيمة المعامل الاول وينجم عن هذه العملية تغيير في محتوى الرايات، ويتم تخصيص النتيجة للسجل (المعامل الأول)

### تعليلة CMP

CMP OP1,OP2

في الكثير من الاحيان نستعمل هذا الامر للمقارنة

مثلا افحص اذا كان المسجل AX يحتوي على القيمة 5

CMP AX,5

الامر هذا يعتمد على طرح op2 من op1 ولا ياتر على op1

هناك ثلاث نتائج لهذا الامر :

صفر	تساوي	SF = 0 ZF=1
موجبة	op1 اكبر من op2	SF = 0 ZF=0
سالبة	op1 اصغر من op2	SF = 1 ZF=0



## تعلیمة AND

هذه التعلیمة منطقیة وكل عملیة منطقیة قاعدة تتبعتها

(هذه التعلیمة هی عملیة ضرب بین الصفر والواحد فقط)

### قاعدة التعلیمة AND

1		1AND1
0		1AND0
0		0AND1
0		0AND0

مثال:

### AND AX,BX

1	1	0	0	0	1	1	1	0	1	0	1	0	0	0	0	AX قبل تنفيذ الامر
1	0	0	0	0	1	0	0	0	0	0	1	1	0	0	1	BX قبل تنفيذ الامر
1	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	AX بعد تنفيذ الامر

تستعمل غالبا في الكشف عن قيمة بت معين في الذاكرة

نأخذ مثلا:

مثلا لديك طابعة عندما تكشف انه ليس لديك ورق ستقوم بتعديل البت التاسع في كلمة سعتها 2 بايت الى واحد لا تأتي رسالة خطأ تخبرك بعدم وجود ورق من برنامجك ولكن تأتي بعد ان يقوم البرنامج بمساعدة نظام التشغيل بالكشف عن البت المفترض انه تغير الى 1

لنقم بوضع رقم تنائي بالمسجل BX كل بتاته صفر ما عدا البت التاسع يكون ب 1

```
MOV BX,0000000010000000b
```

نقوم بوضع الرقم المراد اختباره في AX تم نكتب امر AND

### AND AX,BX

اذا كان البت التاسع المطلوب اختباره في AX يساوي 1 فان الرقم الناتج بعد تنفيذ العملية هو نفس الرقم الموجود في BX

(راجع القاعدة فوق)

وسيكون  $Z = 0$  لان ناتج العملية ليس صفرا ( راجع الدرس السابق )الرايات ( )

اذا كانت النتيجة صفر تاخذ هذه الراية 1 والا تأخذ 0 وضع ZF Zero Flags

(كما قلت سابقا ستفهمون الدروس جيدا مع الامثلة)

اما اذا كان البت التاسع في AX مساويا للصفير فسيكون الناتج بعد تنفيذ الامر هو

0000000000000000b اي ان العملية صفرية يعني  $Z = 1$  وعندها يمكن تنفيذ اي امر من اوامر القفز) ساشرحها في الدرس القادم)

لنقل واعداد توجيه البرنامج الى نقطة معينة او رسالة معينة

-----\*

### تعليلة OR

تقوم بعملية منطقية وهذا جدولها الخاص ( نفس عمل AND الاختلاف فقد في 1 )

1		1OR1
1		1OR0
1		0OR1
0		0OR0

مثال:

OR AX,BX

0	1	0	1	1	0	1	1	1	0	0	0	1	1	0	0	AX قبل تنفيذ الامر
1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	0	BX قبل تنفيذ الامر
1	1	1	1	1	0	1	0	1	1	0	0	1	1	1	0	AX بعد تنفيذ الامر

### تعليلة XOR

تقوم بعملية منطقية و جدولها كالآتي

0		1XOR1
1		1XOR0

1	0XOR1
0	0XOR0

(اذا كان المعاملان متشابهان فالنتيجة 0 وادا كان المعاملان مختلفان فالنتيجة 1)

مثال:

**XOR AX,BX**

0	1	0	1	1	0	1	1	1	0	0	0	1	1	0	0	AX قبل تنفيذ الامر
1	1	1	0	0	0	1	0	0	1	0	0	1	0	1	0	BX قبل تنفيذ الامر
1	0	1	1	1	0	0	1	1	1	0	0	0	1	1	0	AX بعد تنفيذ الامر

تستعمل غالبا لتصفير المسجلات سنرى هذا لاحقا

-----\*

تعلیمة TEST

نفس عمل AND لكنه ياتر على الرايات فقط

$Z = 1$  اذا كانت قيم البيئات ذات الترتيب 2 و 4 و 6 و 8 مساوية للصفر

-----/

انتهى الجزء الاول من تعليمات الاسمبلي

اعيدو مشاهدة هذه الصورة من الدرس السابق وانضرو للرايات وايضا انضرو الى بعض تعليمات الاسمبلي التي درسنا اليوم

<http://img205.imageshack.us/img205/3535/reda.jpg>





**MOV BL,-4**

**IMUL BL**

**AX = 8 h = 1000 b**

تقوم بضرب محتوى المسجل **AL** مع **BL** ووضع النتيجة في المسجل **AX** في حالة مسجلات 8 خانة

أو تقوم بضرب محتوى المسجل **AX** مع قيمة ما ووضع النتيجة في المسجلين **AX,DX** في حالة مسجلات 16 خانة وهكذا

/-----/

**1700 \* 520 = 0D7D20 h = 11010111110100100000 b**

**MOV AX,1700**

**MOV BX,520**

**MUL BX**

**DX = 000D h = 0000000000001101 b**

**AX = 7D20 h = 0111110100100000 b**

/-----/

**12345678h \* 99999999 h = AEC33E18EAD65B8 h = 101011101100001100111110000110001110101011010110010110111000 b**

**MOV EAX, 12345678h**

**MOV EBX, 99999999h**

**MUL EBX**

**EDX = 0AEC33E1 = 00001010111011000011001111100001**

**EAX = 8EAD65B8 = 10001110101011010110010110111000**

التعليمتين تؤثران على الرايات **OF** , **CF**

إذا كانت النتيجة أكبر من حجم المعاملات تحمل هذه الرايات القيمة **أ** فيما عدا ذلك فتحملان القيمة

0

/-

\*\*\*\*\*

\*\*\*\*\*/-

### التعليمات DIV و IDIV

نفس الشيء مع التعليمتين السابقتين

**DIV** هي تعليمة القسمة

**IDIV** هي تعليمة القسمة مع اشارات

ساوضح بجدول:

المقسوم عليه	الباقى	الناتج	الحجم
AX	AH	AL	BYTE
DX:AX	DX	AX	WORD
EDX:EAX	EDX	EAX	DWORD

مثال :

11CCCEE44BBBAAA h / 33A33A33 = 583EF4DF h =  
101100000111101111010011011111 b

MOV EDX, 11CCCEE h

MOV EAX, 44BBBAAA h

MOV ECX, 33A33A33 h

IDIV ECX

EAX = 583EF4DF h = 101100000111101111010011011111 b

EDX = 15B96C3D h

جيد حتى الان الامور واضحة

/-

\*\*\*\*\*

\*/

### التعليمة XCHG

تقوم بتبديل محتويات المعاملن الممررين لها

مثال :

XCHG AX,BX

اذا كان AX=5 و BX=10

يصبح AX=10 و BX=5

/-

\*\*\*\*\*

\*/

### التعليمة INC

تستخدم في عمل زيادة للمعطى بمقدار 1 ، وهي تقابل عملية جمع واحد + القيمة التي بداخل المعطى  
و من تم تخزين القيمة الجديدة

مثال :

INC AX ==> AX = AX + 1

تؤثر على الرايات ZF-SF-OF-PF-AF :

/-

\*\*\*\*\*

\*/

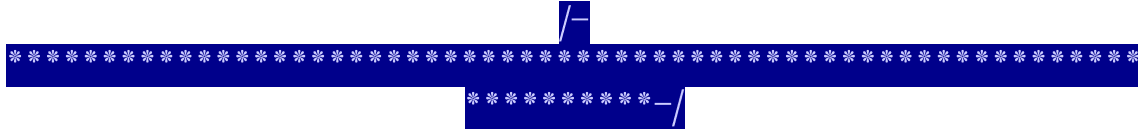
### التعليمة DEC

عكس التعليمة INC حيث تقوم هذه الاخيرة بنقص 1



`DEC AX ==> AX = AX - 1 === SUB AX,1`

تؤثر على الرايات ZF-SF-OF-PF-AF :



التعليمة NOT

هذه التعليمة مسؤولة عن عكس القيمة التي بداخل المكان الذي يذكر بعدها

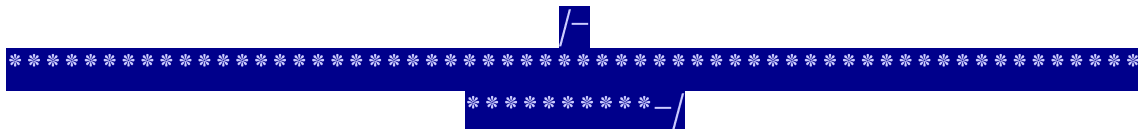
مثال :

لو أنه يوجد بالمسجل AL قيمة ثنائية على هذا الشكل 01110010 ونريد عكس هذه القيمة لتكون هكذا 10001101 ، نستخدم هذه التعليمة

`MOV AL,01110010B`

`NOT AL`

`AL = 100011010B`



التعليمة NEG

تستخدم في تغيير اشارة المعطى , بمعنى جعل القيمة سالبة وطبعا تعمل مع جميع القيم السالبة لتجعلها موجبة

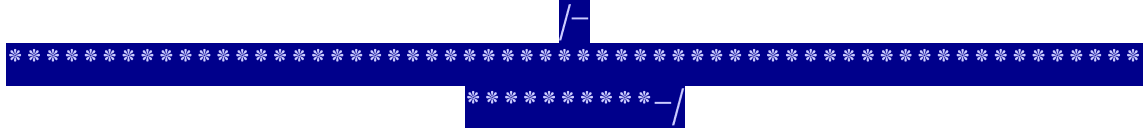
مثال :

`MOV AX,100 ==> AX = 100`

`NEG AX ==> AX = FF00 = -100`

NEG AX ==> AX = 100

تؤثر على الرايات CF-ZF-SF-OF-PF-AF :



انتهى درس اليوم

اي سؤال؟

السلام عليكم



\*/---\*/\*\*\*/---\*/

## /-/تعليمات الاسمبلي الاساسية/-/

\*/---\*/\*\*\*/---\*/

تواصل دورة الاسمبلي , اليوم معنا درس عن اوامر (الازاحة والدوران واوامر اخرى

التعليمات SHL,SHR,ROR,ROL,CALL,RET,INT :

\*\*\*\*\*  
\*/

### تعليمة SHL

تقوم هذه التعليمه بعمل ازاحة لليسار بمقدار بايت واحد تم تترك الب اقصى اليمين بصفر وهذه العملية عبارة عن مضاعفة عدد بدون اشارة

(تتم اضافة 0 من اليمين الى (LSB) ويذهب MSB الى CF)

(راجع درس العد ودرس المسجلات=)

مثال :

0	0	1	1	0	0	1	0	العدد
0	0	0	1	1	0	0	1	SHL

اذا نتج عن عملية المضاعفة رقم اكبر من سعة حيز التخزين سيتم حمل البت الزائد الى

CARRY-FLAG

مثال :

MOV AL,01001010B

SHL AL,1D

بعد تنفيذ الامر SHL ستصبح محتويات AL 10010100B ويتم وضع قيمة C-FLAG ب 1

**ملاحظة:** تنفيذ الامر SHL لمرة واحدة يقوم بضرب العدد الذي سيتم تدويره في 2

SHL AX,1 ==> AX\*2

SHL AX,2 ==> AX\*4

SHL AX,3 ==> AX\*8

\*\*\*\*\*

### تعليمية SHR

تقوم بعمل ازاحة لليمين بمقدار واحد بت تم تترك البت اقصى اليسار بصفر

**مثال:**

1	1	1	0	0	1	1	0	العدد
1	1	0	0	1	1	0	0	SHR

**مثال:**

MOV AL,01100111B

SHR AL,1D

بعد تنفيذ الامر SHL ستصبح محتويات AL 00110011B ويتم وضع قيمة C-FLAG ب 1

**ملاحظة:** تنفيذ الامر SHR لمرة واحدة يعني قسمة القيمة التي سيتم تدويرها على 2

SHR AX,1 ==> AX/2

SHR AX,1 ==> AX/2

SHR AX,1 ==> AX/2

\*\*\*\*\*

يجب الانتباه الى خطوة مهمة وهي عند استعمال SHL

لاحظ هنا SHL ECX,32

هنا انت تضيع الوقت , السبب ان هذه ستجعل قيمة ECX صفرا

لأنك ستقوم ب 32 عملية ازاحة وكل عملية تقوم بادخال 0 من اليمين في النهاية ستستبدل 32 بت التي في ECX ب 32 صفرا

يمكنك استعمالها في حالة واحدة ان كنت تريد اللعب مع كراكر مبتدأ وادخاله في متاهات

\*\*\*\*\*

### تعليمة ROR

تقوم هذه العملية بالتدوير الى اليمين

مثال :

```
MOV AL,01010111B
```

```
ROR AL,1
```

بعد التنفيذ سيصبح AL حاملا للقيمة 1010111B

مثال اخر:

```
MOV AL,00111111B
```

```
ROR AL,3
```

==> 10011111 الدورة الاولى

==> 11001111 الدورة الثانية

==> 11100111 الدورة الثالثة

CF : 1

انتبهه ايضا هنا ROR BX,16

تدوير المسجل BX ب 16 مرة يعني انه سيعود الى حالته الاولى

لاحظ ايضا ان ROR BX,17 تكافئ ROR BX,1

ونفس الشيء مع التعليمة الاتية ROL

\*\*\*\*\*

## تعلیمة ROL

تقوم هذه العملية بالتدوير الى اليسار

مثال :

```
MOV AL,00111111B
```

```
ROL AL,1
```

بعد التنفيذ سيصبح AL حاملا للقيمة 0B0111111

كما تلاحظ اخر بت (MBS) تم تدويره واصبح اول بت (LBS) وتذهب نسخة من هذا البت الى CF

\*\*\*\*\*

## تعلیمة CALL

من اسمه يبين انه امر استدعاء

وصيغته هكذا : CALL ADDRESS

يتم تنفيذ الامر الموجود على سطر العنوان 'address' وما ان يتم تنفيذ هذا الامر حتى تتابع الشفرة  
تنفيذ الامر الموجود بعد امر CALL

(سنرى هذا لاحقا بشكل اوضح )

\*\*\*\*\*

## تعلیمة RET

العودة من وظيفة او امر وسنراه ايضا بشكل اوضح لاحقا

\*\*\*\*\*

## تعلیمة INT

صيغة هذا الامر INT interrupt\_number

ويقوم باستدعاء امر افتراضي) غالبا ما تكون مبرمجة ضمن اليبوس)

مثال 10h INT :

سنراه ايضا لاحقا عندما ندرس المقاطعات

\*\*\*\*\*

انتهى درس اليوم مازال هناك جزء فقط غدا , وندخل في برمجة اول برنامج بالاسمبلي

\*\*\*\*\*

|





تفريغ الوعاء اي سحب منه الاقراص نستعمل الامر POP

\*\*\*\*\*

من الاحسن مراجعة الدرس :

" الدرس الثالث من دورة الاسبلي " <> الجزء الثاني <>

\*\*\*\*\*

تعليمية الدفع PUSH

يستعمل هذا الامر بكثره وهو لحفض المسجلات و البارامترات في المكس Stack

الامر PUSH لا ياتر على الاعلام

\*\*\*\*\*

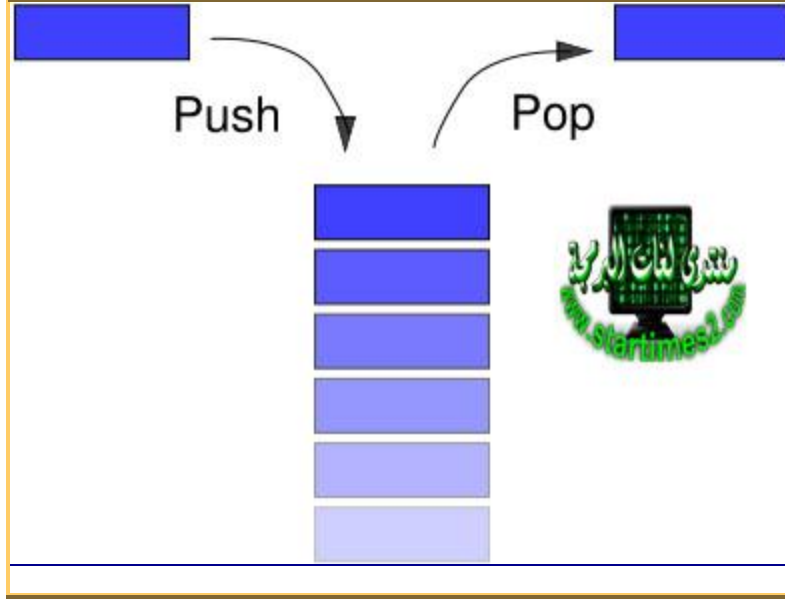
تعليمية السحب POP

يستعمل هذا الامر لاسترجاع المسجلات و البارامترات من المكس Stack

الامر POP لا ياتر على الاعلام

\*\*\*\*\*

الصورة توضح الامرين POP , PUSH



## وامر القفز

تعليمات القفز نوعين : شرطية وغير شرطية

الغير شرطية منها واحدة وهي **JMP** والتي تقوم بنقل التنفيذ إلى أي مكان تريده في البرنامج بدون أي شرط مسبق.

مثال :

```

mov eax,1
  jmp @1
add eax,2
  jmp @2
      :@1
add eax,3
      :@2

```

ستكون قيمة **eax** نتيجة هذا التنفيذ مساوية **4** وليس **3** لأن التنفيذ ينتقل عند **jmp @1**

ويقوم بتخطي مجموعة من التعليمات التي لا يعود لها أي استخدام.

أما التعليمات الشرطية فهي عديدة .. على سبيل المثال:

JE/JZ , JNE/JNZ , JG , JGE , JA , JAE , JL , JLE, JBE , .....

المستعملة بكثرة هي JE و JZ و JNZ

هذه التعليمات دائما يسبقها شرط معين مثلا تعليمة JE ليتم تنفيذها عندما تكون قيمة الـ ZF مساوية

لـ 1

وهذه الراية تتأثر بعمليات المقارنة مثلا فلو أردت تستطيع أن تقوم بمقارنة قيمة مسجل مع آخر

وترى إن كان هناك تطابق

مثال :

```
cmp eax ebx
je startimes
....
:stoune
...
startimes
...
```

في هذا المثال اذا تطابقت قيمة المسجل EAX مع قيمة المسجل EBX فسيتم تنفيذ القفزة الى startimes

اما في حالة لم يكن هناك تطابق فلن يتم تنفيذ القفزة وسيكمل البرنامج سيره.

جميع أوامر القفز السابقة تعمل بنفس المبدأ فهناك شرط إن تحقق يتم القفز وإلا فلا

|

وهذا جدول توضيحي :

التعليمة	المعنى
JC	القفز إذا كان $CF = 1$
JNC	القفز إذا كان $CF = 0$
JO	القفز إذا كان $OF = 1$
JNO	القفز إذا كان $OF = 0$
JS	القفز إذا كان $SF = 1$
JNS	القفز إذا كان $SF = 0$
JCXZ	القفز إذا كان $CX = 0000$
JZ/JE	القفز في حالة التساوي/أو إذا كان الناتج يساوي الصفر
JNL/JGE	القفز إذا كان أكبر أو يساوي/القفز إذا لم يكن أصغر
NBE/JA	القفز إذا كان فوق/القفز إذا لم يكن تحت أو يساوي
JNB/JAE	القفز إذا كان فوق أو يساوي/القفز إذا لم يكن تحت
JNAE/JB	القفز إذا كان تحت/القفز إذا لم يكن فوق أو يساوي
JNA/JBE	القفز إذا كان تحت أو يساوي/القفز إذا لم يكن فوق
JNLE/JG	القفز إذا كان أكبر/القفز إذا لم يكن أصغر أو يساوي
JNG/JLE	القفز إذا كان أصغر أو يساوي/القفز إذا لم يكن أكبر
JNZ/JNE	القفز إذا لم يكن يساوي/القفز إذا كان الناتج يساوي قيمة غير صفرية
JBO/JNB	القفز إذا كان $PF = 0$
JPE/JP	القفز إذا كان $PF = 1$

\*\*\*\*\*

انتهى درس اليوم , في الدرس القادم سنرى كيف تكتب اول برنامج بالاسمبلي

\*\*\*\*\*



السلام عليكم

وبعد ما تعرفنا على اساسيات الاسمبلي سنقوم اليوم بكتابة أول برنامج بلغة الاسمبلي

البرنامج عبارة عن برنامج بسيط يقوم بطباعة بعض الكلمات على الشاشة .

\*\*\*\*\*

\*\*\*\*

لغة الاسمبلي لديها العديد من المترجمات كمتال.... MASM , TASM , NASM :

وانا شخصيا اشتغل ب MASM + WINASM المست انا فقط بل الاغلبية

لتحميل البرنامجين من هـ

الباسورد startimes :

قم اولا بفك الضغط عن الملف سينتج لك ملفان

الاول MASM32 :

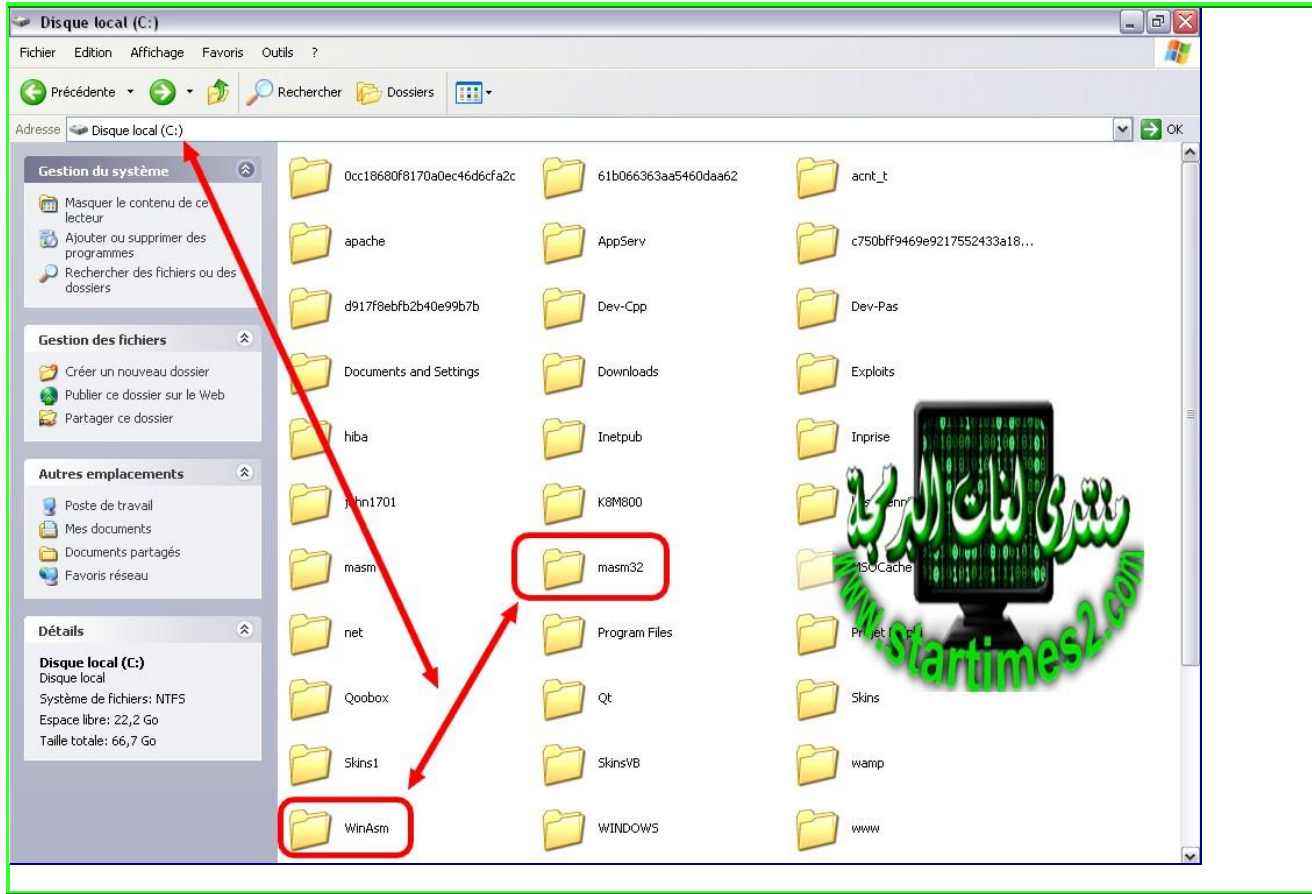
التاني WinASM :

افتح MASM32 ستجد ملف واحد وهو ملف التنصيب ، دوبلك كليك عليه تم اختر القرص C والبقية تعرفونها

ستجد مجلد باسم MASM32 اضيف الى القرص C

حسنا جيد الان قم بالرجوع الى المجلد التاني WinAsm اعمل له نسخ لصق في القرص C

شاهد الصورة:

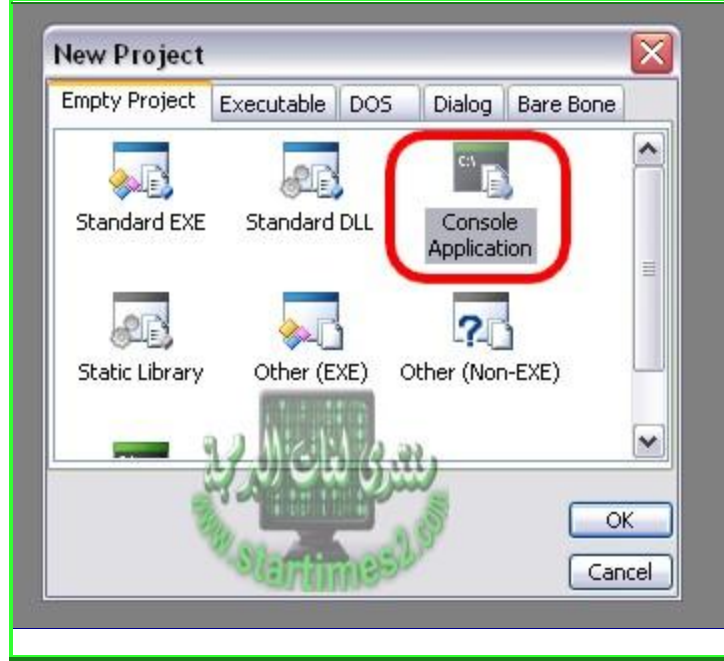


اوکي جيد ، نتابع:

الان قم بفتح المجلد , WinAsm تم دوبرل كليك على WinAsm.exe

ستفتح معك نافذة البرنامج اختر كما في اللغات الاخرى FILE تم NewProrjet

تم اختر Console Application



ستفتح معك نافذة كتابة الكود ; قم بكتابة الكود كما في الصورة

(لا اريد كتابته, لان الجميع سيعمل نسخ لصق )

```
C:\WinAsm\hhf.asm
(Select Procedure Or GoTo Top)

.386
.MODEL flat, stdcall
OPTION CASEMAP:NONE

Include windows.inc
Include kernel32.inc
Include masm32.inc
Includelib kernel32.lib
Includelib masm32.lib

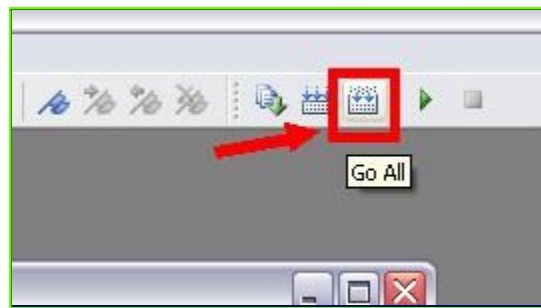
.data
HelloMsg DB "Startimes : Hello Assembly World", 0
CRLF     DB 00Ah, 00Dh, 0
ExitMsg  DB "Enter to Exit", 0

.data?
Buffer DB ?

.code
Start:
invoke StdOut, addr HelloMsg
invoke StdOut, addr CRLF
invoke StdOut, addr ExitMsg
invoke StdIn,  addr Buffer, 1
invoke ExitProcess, 0
End Start
```



الآن اضغط على هذا الزر



النتيجة





جيذا جدا

قبل ان اشرح الكود ساقوم بشرح تعريف المتغيرات في الاسبلي

الشكل العام للتعريف هو :

Variable name DB initial value

DB : تحديد بايت

DW : تحديد ( 2 بايت )

DD : تحديد ( 4 بايت )

امثلة:

X1 DB 100001b

X1 DB 'A';

|

اوكي الان لنقم بشرح الكود سطرا سطرا

\*\*\*\*\*

\*\*\*

.386

هذا السطر يخبر المجمع باستعمال تعليمات المعالج 386، وهذا السطر لن نغيره فقط احيانا  
تاستعمل 486. بدلا منه

\*\*\*\*\*

model flat, stdcall

يقوم السطر model الذي هو عبارة عن توجيه directive للمجمع يخبره فيه بان يقوم باستخدام  
نمط العنوان flat

والذي لا نملك سواه تحت بيئة 32 bit.

بالنسبة لـ stdcall فهذا يخبر المجمع ايضا بكيفية تمرير البيانات, من اليمين الى اليسار, او من  
اليسار الى اليمين

\*\*\*\*\*

option casemap: none

هذا السطر يدل Masm على ان اسماء الوظائف حساسة لحالة الاحرف اي ان وظيفة exitprocess  
تختلف عن ExitProcess

\*\*\*\*\*

Include windows.inc

يخبر MASM بان يقوم بتضمين windows.inc في ملفنا

\*\*\*\*\*

Include kernel32.inc

نفس الشيء

\*\*\*\*\*

Include masm32.inc

نفس الشيء

\*\*\*\*\*

Includelib kernel32.lib

هذا السطر يخبر **linker** بادن يقوم بربط برنامجنا مع ملف **kernel32.lib**

\*\*\*\*\*

Includelib masm32.lib

نفس الشيء

\*\*\*\*\*

.data

تعني بداية مقطع البيانات

\*\*\*\*\*

HelloMsg DB "Startimes : Hello Assembly World", 0

هنا نقوم بتعريف **String** اسمها **MsgBoxCaption** ومحتوياتها هي ما بين علامتي التنصيص

اما الصفر في النهاية فيعني ان **String** ستنتهي ب **0**

\*\*\*\*\*

CRLF DB 00Ah, 00Dh, 0

هنا نقوم بتعريف متغير باسم **CRLF** ليقوم بازال مؤشر الكتابة سطرًا للأسفل

\*\*\*\*\*

ExitMsg DB "Enter to Exit", 0

نفس شرح **HelloMsg**

\*\*\*\*\*

.data?

تعني بداية مقطع البيانات ( المتغيرات ) التي ليست لها قيمة ابتدائية

\*\*\*\*\*

Buffer DB ?

متغير ليس له قيمة ابتدائية

\*\*\*\*\*

.code

بداية مقطع الكود التنفيذي

\*\*\*\*\*

start :

Start عبارة عن ( Label يوجد بعدها نقطتان رأسيتان ) وهي تشير الى بداية الكود

\*\*\*\*\*

invoke StdOut, addr HelloMsg

هنا نقوم باستدعاء الدالة StdOut المسؤولة عن اظهار String ما

اما addr HelloMsg فهي عنوان String التي نريد عرضها اي ان هذا هو البارامتر الوحيد لهذه الدالة

\*\*\*\*\*

invoke StdOut, addr CRLF

نفس الشيء ، هنا نستدعي الـ CRLF يعني انا سننزل مؤشر الكتابة سطرا للاسفل

\*\*\*\*\*

invoke StdOut, addr ExitMsg

متل `invoke StdOut, addr HelloMsg`

\*\*\*\*\*

`invoke StdIn, addr Buffer, 1`

هنا نستدعي دالة `StdIn` وهي دالة خاصة باستقبال نص من المستخدم ، استعمالها هنا رغم اننا لا نريد استقبال نص من المستخدم

هل عرفت السبب ؟

ان لم نقم باستعمالها فستفتح نافذة الدوس للبرنامج وتعلق بعد اقل من ثانية , اما عن `!` فهو البارامتر الثاني للدالة

وهو يشير الى اننا سنستقبل محرفا واحدا من المستخدم

\*\*\*\*\*

`invoke ExitProcess, 0`

هنا نقوم باستدعاء دالة `ExitProcess` الخاصة بانهاء البرنامج ونعطيها البارامتر الوحيد وهو `0`

\*\*\*\*\*

End Start

نهاية البرنامج

\*\*\*\*\*

\*\*\*

ونهاية الدرس ، نلتقي في الدرس القادم

\*\*\*\*\*



السلام عليكم

في الدرس السابق تعرفنا على طريقة كتابة اول برنامج لك بالاسمبلي

بصراحة لم يكن علي كتابة ذلك الموضوع في هذا الوقت لكني اعرف ان الكثير من الاعضاء ينتظرون فقط متى يكتبون اول برنامج

بهذه اللغة ولهذا السبب وضعته , كي لا يطبع على الدورة الطابع الممل

اليوم ساشرح درسا جديدا في الاسمبلي عن السؤال الذي يدور بذهن العديد وهو الفرق بين الاسمبلي  
16بت و الاسمبلي 32بت

ان اكبر مشكلة تواجه المبتدئين هي معرفة الفرق بين الاسمبلي 16 و الاسمبلي 32 , واكثر خطأ  
شائع هو ان المبتدء يقوم بتحميل

مترجم 32بت مثل **masm32** ويحاول تطبيق برنامج باستخدام المقاطعات (ساشرحها في  
الدرس القادم (اي 16بت

بصيغة اخرى ، هناك نوعان من الاسمبلي هناك الاسمبلي 16بت و الاسمبلي 32بت ايضا هناك 64  
بت لكن لن اتكلم عنها لانها حديثة

ولا يمكننا تطبيقها على المعالجات التي نستعملها حاليا.

للتبسيط اكثر للمبتدئين فالامر يتعلق بالمعالج **Processeur**

المعالجات القديمة كانت من نوع 16بت اما المعالجات التي نستعملها الان فهي من نوع 32بت

وهناك معالجات ضهرت حديثا وهي معالجات 64بت. وكل نوع له مترجم خاص به

اسمبلي 16بت لها مثلا مترجم اسمه ( Masm ) ساقوم برفعه في الدرس القادم )

اسمبلي 32بت لها مترجم اسمه ( Masm32 ) لقد رفعت في الدرس السابق )

يقوم اغلبية المبتدئين بعمل برنامج 16بت في مترجم 32بت

في اغلب الاحيان يقوم المترجم بترجمة البرنامج بدون اخطاء ولكن عند التنفيذ تظهر اخطاء غريبة ولا يستطيع المبتدء فهم السبب

اما بالنسبة للمعالجات فمتلا نحن الان نستعمل معالجات 32بت سيسألني احدكم

هل يمكن ان نكتب برنامج 16بت باستعمال مترجم الاسمبلي 16بت على معالج 32بت؟

والجواب نعم اي ان برامج 16بت تشتغل على معالجات 32بت لكن العكس غير صحيح

وبصراحة انا افضل في بعض الاحيان البرمجة ب 16بت عن 32بت

|

\_\_\_\_\_\*\*\*\*\*\_\_\_\_\_

هناك سؤال ثاني يطرحه العديد من الاعضاء

وهو كيف يمكنني ان اعرف هل البرنامج 16بت ام 32بت؟

\_\_\_\_\_\*\*\*\*\*\_\_\_\_\_

|

برامج الـ 16 بت تحتوي على مقاطعات وهي التي تبدأ بالكلمة int, كذلك بعض البرامج تحتوي على الدالة in و الدالة out

للتعامل مع منافذ الجهاز بشكل مباشر.





اما الاسبلي 32بت فالذاكرة مفتوحة و نظريا تستطيع استخدام 4جيجابيت ك offset واحد دون الحاجة لتحديد ال sgment .

الاسبلي 16بت تتيح لك التحكم في الهاردوير دون قيود

الاسبلي 32بت تتيح لك برمجة النوافذ , وهذا غيرممكن في ال 16بت

بالتاكيد الاساسيات مشتركة بين ال 16 و 32بت ,مثل mov, add, div الخ من التعليمات الاساسية

ولكن الفرق شاسع بين بين البرمجة لبيئة الوندوز 32بت و بيئة ال دوس 16بت

هناك بيئة تسمى console في الوندوز مشابه لبيئة DOS

مع الفارق ان console هي مجرد برنامج و ليست System d exploitation

( وهذا احسن ما عملت مايكروسوفت في حياتها هههههه )

-----\* \* \* \* \*

بالمناسبة في الاسبلي 32بت يمكننا استعمال مكتبات اللغات الاخرى كمكتبات ++C مثلا وهذا

سنراه في الدروس القادمة

-----\* \* \* \* \*

في الدرس القادم سنرى المقاطعات ( DOS & BIOS )

وسنرى كيف نعمل برامج تتعامل مع الملفات ( انشاء /قراءة /كتابة /حذف /بحث )

في البيئة 16بت باستعمال المقاطعات وفي البيئة 32بت باستعمال دوال API

وستتضح لكم قوة الاسبلي

-----\* \* \* \* \*

ارجو من الاعضاء من يتابع الدورة الرد على الموضوع فلا يمكنني اكمال الدورة بلا متتبعين

السلام عليكم

اليوم لدينا درس جديد في الاسبلي 16 بت عن المقاطعات مع مثال تطبيقي على المقاطعة . 21h

المقاطعات Interrupts هي مجموعة من المهام يتم استدعائها لتنفيذ امر ما , وهي نوعان  
مقاطعات برمجية : Software interrupts تتعامل مع القرص وبعض الاجهزة المعينة ( طابعة  
..... )

مقاطعات صلبة : Hardware interrupts تتعامل مع اجهزة اخرى مثل BIOS

حاليا سنعمل على المقاطعات البرمجية وبالتحديد المقاطعة 21h

كل مقاطعة واحدة لديها العديد من الخيارات ( مهام مختلفة في نفس السياق ) يصل عددها الى 256  
وضيفة ممكن ان تقوم بها

كمثال نأخذ ما يهمنا في هذا الدرس:

المقاطعة 21h سنستعمل منها في هذا الدرس الخيارات الآتية ( بارامترات المقاطعة )

طباعة جملة 9h :

البحث عن ملف 4eh :

تغيير اسم الملف 56h :

انشاء ملف 3Ch :

فتح ملف 3Dh :

الكتابة في الملف 40h :

اغلاق الملف 3Eh :

حذف الملف 41h :

انهاء البرنامج واعطاء السيطرة والتحكم لنظام التشغيل 4ch :

يتم استدعاء اي مقاطعة بتمرير عدد من البارامترات المعينة والتي تحدد الوظيفة المطلوبة من المقاطعة اجراؤها قبل استدعائها

هذا النوع من تمرير البارامترات ملزم بشكل معين حيث يتم دفع هذه البارامترات وتخزينها في مسجلات الاغراض العامة

وقد يتم استخدام بعض المسجلات الاخرى مثل DX لتخزين قيم معينة فيها كاسلاسل النصية..  
اخيرا يتم استدعاء المقاطعة بالشكل الاتي:

INT NUM

NUM : رقم المقاطعة

مثال في المقاطعة 21h مع بارامتر الطباعة على الشاشة 9h

```
mov dx,msg
```

```
mov ah,9h
```

```
int 21h
```

الشرح :

mov dx, msg ==> موضع محتوى msg في المسجل ( dx مسجل نقل البيانات + الاخراج والادخال )

( ==> mov ah,9h تتمرير بارامتر الطباعة الى المسجل ah )

( ==> 21 lint استدعاء المقاطعة )

\*-----\*

مثال كامل :

قبل المثال ، وبما اننا نشتغل على 16 بت فبرامج 16 بت تأخذ الشكل الاتي:

في السي بلس بلس مثلا:

```
#include
```

```
main
```

```
}
```

```
.....
```

```
.....
```

```
}
```

في الاسبملي 16 بت :

```
.model small
```

```
.stack
```

```
.data
```

```
.....
```

```
.code
```

```
start:
```

```
.....
```

```
.....
```

```
end start
```

\*-----\*

هذا الكود سيقوم بطباعة محتوى msg في الشاشة تم الخروج وارجاع التحكم للنظام

ماذا لدينا؟؟! طباعة جملة يعني البرامتر + 9h الخروج من النظام وارجاع السيطرة (4ch) وهذه  
توجد في جميع برامج الاسبلي 16 بت

```
.model small
```

```
.stack 100
```

```
.data
```

```
msg db "startimes", "&"
```

```
.code
```

```
start:
```

```
mov ax, @data
```

```
mov ds, ax
```

```
mov dx, offset msg
```

```
mov ah, 9
```

```
int 21h
```

```
mov ah, 4ch
```

```
int 21h
```

```
end start
```

شرح الكود:

```
-----
```

```
.model small
```

يعبر عن نموذج الذاكرة وبه عدة اختيارات) : سنراها لاحقا بتمعن)

Tiny صغير جدا  
Small صغير  
Medium متوسط  
Compact مدمج  
Large كبير  
Huge ضخم

`.stack 100`

يحجز مكان في المكس (مراجعة الدروس السابقة مهم)

`.data`

مقطع البيانات ، وفيه نعرف المتغيرات والبيانات التي سنستعملها في البرنامج

`msg DB "startimes", "&`

تعريف متغير

`.code`

بداية مقطع الكود

`start:`

يعبر عن بداية الكود Label

```
mov ax,@data
```

```
mov ds,ax
```

السطرين نستعملهما في اغلبية البرامج 16 بت

البيانات يجب ان تسجل في المسجل DS الذي هو مسجل مقطع البيانات

يعني اننا يجب ان نضع فيه البيانات DATA هكذا MOV DS,@data لكن هذه التعليمة خاطئة

بل يجب تمرير البيانات الى المسجل ax اولاً ثم تمريرها من ax الى ds

```
<<== mov ax,@data
```

 وضع البيانات في المسجل

```
<<== mov ds,ax
```

 الى ds نقل البيانات من

--/

```
mov dx, offset msg
```

offset نسيت ان اشرح في الدروس السابقة

لدي صورة تشرح هذه العملية ساضعها في الجزئ القادم

msg المهم هي واضحة بالشرح الشعبي نريد الجملة الموجودة في المتغير

( مسجل نقل البيانات + الاخراج والادخال dx ) نسحبها ونضعها في المسجل

لأننا نستعد لاجراج الجملة على الشاشة

(يبدو ان درس المسجلات اصبح واضحاً الان , ان رجعت وقرنته ستفهمه جيداً الان )

--/

```
mov ah,9
```

ah وضع بارامتر الاخراج في المسجل

--/

int 21h

21h استدعاء المقاطعة



حتى الان انتهى امر الطباعة وعلينا اعادة السيطرة للنظام



mov ah,4ch

ah وضع بارامتر ارجاع السيطرة للنظام في المسجل



int 21h

21h استدعاء المقاطعة



end start

انهاء البرنامج



الان بقي الترجمة والربط وتنفيذ البرنامج ، نحتاج الى مترجم 16 بت



لمن يريد جميع المقاطعات مع بارامترات كل مقاطعة + الشرح

فمن هنا

<http://www.ctyme.com/intr/int.htm>



اضغط على 21h مثلا ستظهر لك جميع ما يمكن ان تعمل هذه المقاطعة

**Int 21/AH=09h - DOS 1+ - WRITE STRING TO STANDARD OUTPUT**

المقاطعة -- int 21 البرامتر 09 -- h يوضع في المسجل -- ah هي من مقاطعات DOS

**عملها WRITE STRING TO STANDARD OUTPUT**

اضغط عليها ايضا ستفتح صفحة بها الشرح الممل ( ترجم للفرنسية من خاصية جوجل وانتبه لان الاكواد تترجم ايضا )

/-----/

عدرا ساكمل الجزئ الثاني في المساء لدي عمل يجب ان اقوم به

ارجوو ان يكون الدرس واضحا واي سؤال متعلق بالدرس يمكن طرحه هنا

السلام عليكم

تتمة الدرس السابق

رأينا في الجزء السابق تعريف للمقاطع كما اعطينا مثلا لاستعمال المقاطعة 21h والبارامتر 9h

اليوم سنعمل برنامج بسيط بالاسمبلي يقوم بالاتي :

انشاء ملف على المسار C:/ تم الكتابة عليه

ربما قد تعمل هذا البرنامج في 6 اسطر باحدى اللغات العالية المستوى لكن الاسمبلي الامر مختلف

سنكتب حوالي 70 سطر لانه في الاخير لو عملت هذا البرنامج باي لغة عالية المستوى سترجم اولاً الى الكود الذي سنكتبه بالاسمبلي تم الى

لغة الالة ) لو كان البرنامج من 70 سطر فبلغة الالة ايضا سيكون تقريبا 70 سطر لان كل تعليمة في الاسمبلي تقابلها تعليمة اخرى في

لغة الالة ، وعكس ذلك في اللغات العالية المستوى حيث ممكن ان تجد كلمة واحدة عندما تترجم للغة الالة تجدها تتكون من 20 سطر )

\*\*\*\*\*

هناك من يقول لا داعي لاتعلم الاسمبلي مادمت اتقن سي او الباسكال او الجافا...

وهذه نقطة خاطئة تماما ، بتعلمك الاسمبلي ستفتح بابا واسعا في عالم البرمجة وتصبح لك دراية واسعة في كيفية عمل البرامج

وتعاملها مع المعالجات والدواكر .... وتاهلك للدخول في عالم الهندسة العكسية واشياء عديدة

وسنرى دروسا في القادم عن الهاردوير كبرمجة الـ BIOS واسخراج برامج الـ... ROM

المهم الكلام في هذا الامر يلزمه موضوع وحده ، ساضعه غدا بادن الله

\*\*\*\*\*

تكمّل درسنا

الترتيب يكون كالآتي:

1 / انشاء ملف

2 / فتح الملف

3 / الكتابة على الملف

4 / غلق الملف

5 / حذف الملف ( ان اردنا)

\_\_\_\_\_

انشاء ملف

كما رأينا في الجزئ السابق ، من خدمات المقاطعة 21h انشاء الملفات بتمرير البارامتر 3Ch

تقوم هذه المقاطعة مع هذا البارامتر بانشاء ملف في مجلد البرنامج او في اي مسار نحدده نحن تم  
تقوم باعادة مقبض الملف file handle

الى المسجل ، ax في حالة نجاح التنفيذ وتم انشاء الملف تقوم بوضع الرقم 0 في مسجل الذاكرة  
CF ، اما ان لم تنجح العملية

فستضع الرقم 1 في مسجل الذاكرة CF دليلا على وقوع خطأ ، ويتم وضع رقم الخطأ في المسجل  
ax

(ساضع لاحه بارقام الاخطاء مع معنى كل رقم)

وكما نعلم جميعاً فعندما يقع خطأ يجب على البرنامج اخبار المستخدم بوقوع خطأ

لهذا سنستعمل تعليمة القفز ( JC تعليمات القفز تعمل مثل جملة GOTO في اللغات العالية المستوى

تعليمية JC تنفذ في حالة كان CF يساوي واحد اي وجود خطأ كما ذكرت

اما في حالة نجاح التنفيذ فسيكون CF مساوي للصفر وهنا يتم تجاهل تعليمة JC رغماً منا كتبناها في الكود لكن يتل تجاهلها

اما عن كيف نقوم بكتابة الكود فهكذا

اولاً في قسم data بتعريف msg بظهر كلمة ' Error ' على الشاشة هكذا:

```
"&"msg_error_creating DB 13,10,"*****Error creating the  
file*****",13,10
```

ثم كما دي الدرس السابق لاضهار هذه الكلمة على الشاشة يجب ان يكون الكود هكذا

```
mov ah,9h  
mov dx, offset msg_error_creating  
int 21h
```

نكتبه فوي اي مكان من الكود لكن كيف سيقفز له البرنامج؟

نقوم قبل كتابة الكود بوضع Label هكذا

```
:Error
```

```
mov ah,9h  
mov dx, offset msg_error_creating  
int 21h
```

```
*****
```

سنقوم بكتابة JC على هذا الشكل

```
JC Error يعني ان كان CF=1 فاقفز الى Error
```

ماذا بقي لدينا؟ كما هو معلوم عند وقوع خطأ يجب ان يظهر للمستخدم رسالة تشير الى وجود خطأ وهذا عملنا

بقي الخروج من البرنامج وارجاع السيطرة للنظام لهذا سنضيف قفزة اخرى **JMP** غير شرطية يعني سيففز دون شرط

وسنضع **Label** للخروج من البرنامج في اخر البرنامج يعني قبل كلمة **end start** هكذا

**:Close**

**mov ax,4c01h**  
**int 21h**

**end start**

\*\*\*\*\*

خلاصة هذه الفقرة في كود عملي مع شرحه:

```
.model small  
.stack  
.data
```

```
,"&msg_error_creating DB 13,10," ***** Error creating the  
file***** ",13,10
```

```
myfile DB "c:\myfile2.txt",0
```

```
.code  
:start  
    mov ax,@data  
    mov ds,ax  
    mov dx, offset myfile  
    xor cx,cx  
    mov ah,3ch  
    int 21h  
    jc Error_Creating
```

```
:Error_Creating
```

```
    mov ah,9h  
    mov dx, offset msg_error_creating  
    int 21h
```

```
Jmp Close
```

:Close

mov ax,4ch

int 21h

end start

شرحت كود في الجزئ السابق لدى لا داعي لاعادة شرحه

الاشياء الغريبة هي :

\_\_\_\_\_

xor cx,cx

تعليمية xor تقوم بجعل قيمة المسجل تساوي صفر

(تقوم بعمل CX - XC كما انك تعمل 5 5 - يعني مهما كانت القيمة الموجودة به فستصبح صفرا)

كما هو معلوم فهذا المسجل يستخدم كعداد لهذا اسخدمناه هنا ودوره سنراه في الكود الاتي للكتابة على الملف

حيث سنضع به عدد الاحرف التي نريد كتابتها في الملف لهذا قمنا هنا بجعل قيمته صفرا

\_\_\_\_\_

mov ah,3ch

int 21h

jc Error\_Creating

اولا وضعنا ببارامتر انشاء ملف 3ch في المسجل ah تم استدعينا المقاطة لتقوم بانشاء الملف

وفي حالة تم انشاء الملف بنجاح فلن تنفذ التعليمية jc Error\_Creating سيتم تجاهلها اما ان وقع خطأ يعني CF = 1

فستنفذ التعليمية وتقفز الى jc Error\_Creating

\_\_\_\_\_

:Error\_Creating

```
mov ah,9h
mov dx, offset msg_error_creating
int 21h
```

```
jmp Close
```

عندما يتم القفز هنا سيتم تمرير بارامتر اضهار جملة الخطأ 9h التي وضعناها في المتغير msg\_error

تم يتم وضع الجملة في المسجل dx الذي هو مسجل البيانات والادخال والايخارج

تم استدعاء المقاطعة لاضهار النص على الشاشة

تم بعدها تنفيذ القفزة jmp الى الـ Close Label

```
_____
```

```
:Close
```

```
mov ax,4ch
int 21h
```

```
end start
```

تمرير بارامتر ارجاع السيطرة للنظام وتنفيذ المقاطعة تم انتهاء البرنامج

```
_____
```

الى هنا كل شيء واضح

```
_____
```

الان ماذا ان كان الملف موجودا سابقا اي نفس الاسم وبه بيانات ، ماذا سيجري عندما ننفذ البرنامج السابق؟

بكل بساطة سيتم ارجاع حجه الى الصفر يعني محو جميع البايات المسجلة به

لهذا يجب ان نأخذ بعين الاعتبار هذه الحالة ونقوم بوضع كود يتحقق اولاً ما اذا كان الملف موجوداً أم لا

اذا كان موجوداً يقوم بتغيير اسمه ثم انشاء ملفنا الجديد

لهذا سنستعمل البرامتر 4Eh الذي يقوم بالبحث عن الملف و البارامتر 56h الذي يقوم بتغيير اسم الملف

```
mov dx, offset myfile
mov cx, 3fh
mov ah, 4eh
int 21h
```

jc Create\_file

```
mov ah, 56h
mov dx, offset old_name
mov di, offset new_name
```

```
int 21h
```

:Create\_file

....

....

ماذا وقع هنا ؟ عملية البحث ترجع قيمتان لمسجل الولاية CF اما 1 او 0

0 : تم ايجاد الملف -- لا تتفد القفزة Create\_file بل تجاهلها ونفذ كود تغتير اسم الملف --

1 : لم يتم ايجاد الملف -- اقفز الى بداية كود انشاء الملف --

(بإمكانك الخروج من البرنامج دون انشاء الملف في حالة كان موجودا باستعمال البارامتر 4 Eh تم وضع قفزة للخروج )

-----

### فتح الملف

سنستعمل نفس المقاطعة 21h مع البارامتر 3dh الذي يقوم بفتح الملف

ستقوم المقاطعة هنا بفتح الملف بعد ان نضه في dx

وهناك شيء اخر هو تحديد الوضع الذي نريد فتح الملف به (للقراءة فقط , الكتابة فقط , كلاهما)

سنستعمل نحن البارامتر 2 ويعني القراءة والكتابة , سنضعه في المسجل al



```
mov dx, offset myfile
mov al,2
mov ah,3dh
int 21h
```

```
jc error_opening
```

نفس الشيء نعرف متغير به رسالة فشل في فتح الملف

إذا نجح فتح الملف فـ  $CF=0$  تم يتم ارجاع مقبض الملف في المسجل  $ax$  ويتم تجاهل القفزة

أما إن حدث خطأ فسيحمل  $CF$  بـ 1 ويتم وضع رقم الخطأ في  $ax$  ويتم تنفيذ القفزة

الأمر واضح

### الكتابة في الملف

سنستعمل نفس المقاطعة  $21h$  مع البارامتر  $40h$  الذي يقوم بالكتابة في الملف

ستقوم المقاطعة بتحديد عدد البايتات (الأحرف) التي سيتم كتابتها في الملف بوضع عددها في المسجل  $CX$

(بايت = حرف واحد)

وتقوم بحفظها في ملف ما نقوم بتحديد المقبض الخاص به في المسجل  $BX$

بينما نقوم بتحديد السلسلة التي نريد كتابتها في الملف في المسجل  $DX$

نحن سنكتب كلمة Hello World!

تتطلب هذه الكلمة 12 بايت لأن عدد الأحرف هو + 1 فراغ يعني 12 بايت

الكود كالآتي:

```
mov dx, offset message
mov bx,handle
mov cx,12
mov ah,40h
int 21h
```

jc Error\_write

message تقوم بتعريفه في قسم data هكذا

message DB "Hello World!",0

كذلك يجب تعريف متغير مقبض الملف هكذا

? handle DW

عند التنفيذ الصحيح يتم تحميل ax بعدد البايتات المكتوبة بالملف تم تحميل CF بـ 0 يعني تجاوز القفزة

عند حدوث خطأ CF=1 وتنفيذ القفزة

-----

اغلق الملف

سنستعمل نفس المقاطعة 21h مع البارامتر 3Eh الذي يقوم باغلاق الملف

تقوم هنا بوضع البارامتر 3Eh في المسجل ah

تم وضع مقبض الملف في bx

تم استدعاء المقاطعة 21h

الكود كالآتي:

mov bx,handle

mov ah,3eh

int 21h

jc error\_close

-----

حذف ملف

سنستعمل نفس المقاطعة 21h مع البارامتر 41h الذي يقوم بحذف الملف

```
mov ah, 41h
mov dx, offset myfile
int 21h
```

### قائمة الاخطاء

- 00h (0) no error
- 01h (1) function number invalid
- 02h (2) file not found
- 03h (3) path not found
- 04h (4) too many open files (no handles available)
- 05h (5) access denied
- 06h (6) invalid handle
- 07h (7) memory control block destroyed
- 08h (8) insufficient memory
- 09h (9) memory block address invalid
- 0Ah (10) environment invalid (usually >32K in length)
- 0Bh (11) format invalid
- 0Ch (12) access code invalid
- 0Dh (13) data invalid
- 0Eh (14) reserved
- 0Eh (14) (PTS-DOS 6.51+, S/DOS 1.0+) fixup overflow
- 0Fh (15) invalid drive
- 10h (16) attempted to remove current directory
- 11h (17) not same device
- 12h (18) no more files

### مثال تطبيقي

وصلنا الى نهاية الطريق وهي برمجة البرنامج

وهذا كود بسيط قمت بكتابته كخلاصة مبسطة للدرس

model small.

stack.

data.

, "&",msg1 DB 13,10,"\*\*\*\*\*Start\*\*\*\*\*",13,10

"&",msg2 DB 13,10,"\*\*\*\*\*END\*\*\*\*\*",13,10

"&",msg\_error\_create DB 13,10,"\*\*\*\*\*Error creating the file\*\*\*\*\*",13,10

"&",msg\_error\_open DB 13,10,"\*\*\*\*\*Error opening the file\*\*\*\*\*",13,10

"&",msg\_error\_write DB 13,10,"\*\*\*\*\*Error writing the file\*\*\*\*\*",13,10

"&",msg\_error\_close DB 13,10,"\*\*\*\*\*Error closing the file\*\*\*\*\*",13,10

message DB "Hello World!",0

myfile DB "c:\myfile2.txt",0

? handle DW

code.

:start

mov ax,@data

mov ds,ax

mov ah,9h

mov dx, offset msg1

int 21h

Create the file;

mov dx, offset myfile

xor cx,cx

mov ah,3ch

int 21h

jc error\_create

Open the file;

mov dx, offset myfile

mov al,2 ; acces mode : read and write

mov ah,3dh

int 21h

jc error\_open

```

mov handle,ax
write to the file;
    mov dx, offset message
    mov bx,handle
    mov cx,12
    mov ah,40h
    int 21h
jc error_write
cmp ax,cx
jne error

close the file;
    mov bx,handle
    mov ah,3eh
    int 21h
jc error_close
;
    mov ah,9h
    mov dx, offset msg2
    int 21h
Exit to DOS;
    mov ah,4ch
    int 21h
:error_create
mov ah,9h
mov dx, offset msg_error_create
int 21h
jmp error

:error_open
mov ah,9h
mov dx, offset msg_error_open
int 21h
jmp error

:error_write
mov ah,9h
mov dx, offset msg_error_write
int 21h
jmp error

:error_close
mov ah,9h
mov dx, offset msg_error_close
int 21h

```

```
jmp error
```

```
:error
```

```
mov ax,4c01h
```

```
int 21h
```

```
end start
```

لا تعمل نسخ لصق ، سيكتب بشكل مبعثر وستنتج اخطاء عند الترجمة

حمل الكود من هنا

<http://www.mediafire.com/?dtiz2mdqyuw>

استعمل برنامج notepad ++ لترى الكود بشكل واضح

(اختر assembly من قائمة language)

ترجمة برامج 16 بت

تحتاج لبرنامجين ( linker+compilateur )

ساستعلم masm

حملهما من هنا

<http://www.mediafire.com/?ajte0n3jmm0>

قم بفك الضغط وضع المجلد masm في المسار C:

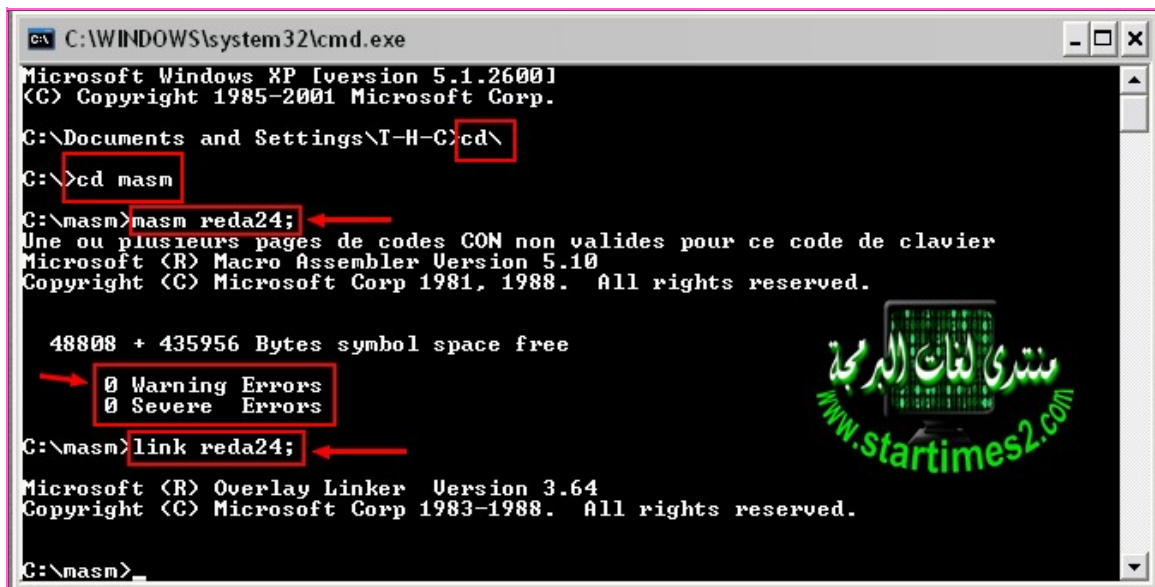
قم بكتابة كود البرنامج في ملف نصي bloc not ثم احض الملف بامتداد asm. مثلا  
reda24.asm

تم انقله الى مجلد masm

الان ادخل للدوس واكتب cd\ masm ثم امر الترجمة; masm reda24;

ان تمت العملية دون اخطاء فاكتب امر الربط; link reda24;

شاهد الصورة



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\Documents and Settings\T-H-C>cd\
C:\>cd masm
C:\>cd masm
C:\masm>masm reda24;
Microsoft (R) Macro Assembler Version 5.10
Copyright (C) Microsoft Corp 1981, 1988. All rights reserved.

48808 + 435956 Bytes symbol space free
0 Warning Errors
0 Severe Errors

C:\masm>link reda24;
Microsoft (R) Overlay Linker Version 3.64
Copyright (C) Microsoft Corp 1983-1988. All rights reserved.

C:\masm>
```

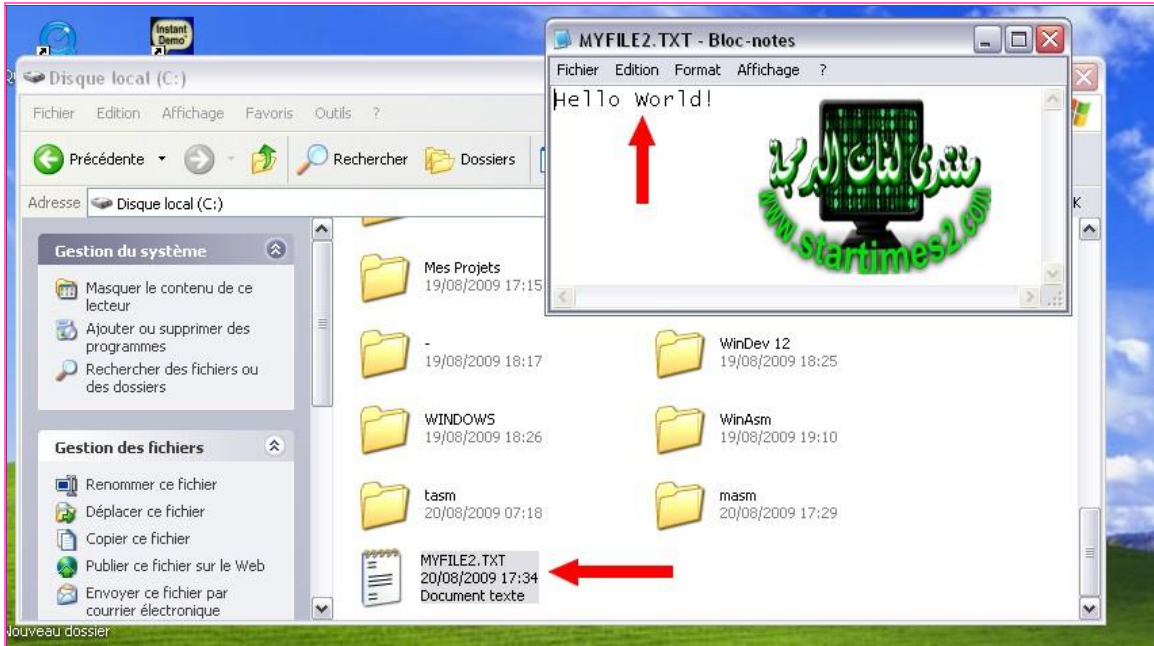
الان ارجع لمجلد masm ستجد ملف تنفيذي باسم reda24.exe وهو برنامجنا الذي عملناه

اما ان تنفده بدوبل كليك او تنفذ من الدوس بكتابة reda24 تم

النتيجة:



اذهب الى المسار C: ستجد الاتي:



انتهى درس اليوم



في الدرس القادم سنرى طريقة عمل نفس البرنامج في بيئة 32 بت باستعمال دوال API

لمن يريد قائمة القاطعات

فهذه بعضها بعد ترجمتها للعربية

\*\*\*\*\*

DOS مقاطعات

h01 : تنفيذ برنامج ما في حالة خطوة خطوة (للسماح باكتشاف الأخطاء).

h03 : خطأ في الانفصال.

h08 : ساعة تدور بمقدار 18.6 نقرة في الثانية.

h09 : قراءة من لوحة المفاتيح. المفتاح مشفر بكود خاص بهذه اللوحة ( scan code )، ويتم تحويله إلى كود أسكي بفضل المقاطعة h16 .

Bh0 : تسيير المنفذ COM2 .

Ch0 : تسيير المنفذ COM1 .

h10 : تسيير بطاقة الفيديو.

h11 : قائمة التعديلات (الذاكرة، عدد منافذ الكوم ( COM )، المعالج الثانوي Co-Processor ، ... )

h12 : حجم الذاكرة المنخفضة Low (640 كيلو بايت كحد أقصى).

h13 : تسيير مختلف الأقراص.

h14 : تسيير واجهة السلسلة (منافذ الكوم)

h15 : مقبض اللعب، أشرطة و التوب فيو ( TopView ).

h16 : تحويل كود المفتاح (الملمس) المقروء من قبل المقاطعة h09 وتحويله إلى كود أسكي.

h17 : تسيير الطابعة.

h18 : الذاكرة الميثة القاعدية ( basic rom ).

h19 : روتين تحميل الدوس.

Ah1 : تسيير الساعة الحقيقية ( real ).

Bh1 : مراقبة الضغط المشترك للمفتاحين CTRL+C .

Ch1 : عداد نقرة/نقرة بسرعة الساعة h08 أي 18.6 هرتز، قيمته تحفظ في الموضع h: 0060h0040

Dh1 : جدول تهيئة (استهلال) الفيديو .

Eh1 : جدول بارامترات الأقراص المرنة.

Fh1 : جدول الرموز البيانية ( Graphic ) .

h20 : مقاطعة الدوس: نهاية برنامج من نوع .com (استعمال النوع .exe هو السائد حاليا)

h21 : مقاطعة الدوس: دوال ومهام عامة (القرص الصلب، الساعة، الطباعة، الإخراج والإدخال من و إلى الشاشة أو الملفات، ...).

h22 : عنوان نهاية المعالجة (القرص الصلب، الساعة، ...).

h23 : مراقبة CTRL+PAUSE أو CTRL+BREAK .

h24 : خطأ قاتل في اتجاه المقاطعة.

h25 : قراءة مباشرة من قرص ما.

h26 : كتابة مباشرة على قرص معين.

h27 : برامج مستقرة في الذاكرة.

h28 : نهاية برنامج بقي مستقرا في الذاكرة.

Fh2 : مقاطعة للعديد من البرامج الفرعية (تسيير الشبكة، درايفر السيدي...).

مازال العديد العديد منها وكل واحد لها 256 بارامتر تقريبا يعني 256 وظيفة



## مقاطعات BIOS

المقاطعة	العنوان	مجال التخصص
INT 02h	0000:0008h	مقاطعة الأجهزة الخارجية لماعرف بديوس
INT 03h	0000:000Ch	مقاطعة لكتابة نقطة توقف للبرنامج
INT 04h	0000:0010h	مقاطعة راية الفيض في مسجل الحالة
INT 05h	0000:0014h	طباعة الشاشة
INT 06h	0000:0018h	مقاطعة لإختبار تنفيذ تعليمة غير مصرح لها
INT 07h	0000:001Ch	مقاطعة ضغط وتشفير التعليمات
INT 08h	0000:0020h	خدمات المؤقت
1Fh INT	0000:007Ch	مقاطعة خصائص الخط والمحارف
1Eh INT	0000:0078h	مقاطعة جدول وبارمترات القرص المرن
1Dh INT	00000074h	مقاطعة جدول وبارمترات شاشة العرض
INT 1Ch	0000:0070h	مقاطعة التوقيت مقدر باللحظة
INT 1Bh	0000:006Ch	تحديد مدة الإستجابة للوحة المفاتيح
INT 1Ah	0000:0068h	خدمات الوقت الحقيقي للساعة
INT 19h	0000:0064h	مقاطعة تحفيز قرص الإقلاع
INT 18h	0000:0060h	ROM BASIC خدمات لتحميل
INT 17h	0000:005Ch	خدمات الطباعة
INT 09h	0000:0024h	خدمات الجهاز المشغل للوحة المفاتيح
INT 0Ah	0000:0028h	مقاطعة تستخدم في التبديل بين المهام
INT 0Bh	0000:002Ch	com2 خدمة المنفذ التسلسلي
INT 0Ch	0000:0030h	com2 خدمة المنفذ
INT 16h	0000:0058h	خدمات لوحة المفاتيح
INT 15h	0000:0054h	خدمات النظام
INT 14h	0000:0050h	خدمات منافذ لإتصالات
INT 13h	0000:004Ch	خدمات القرص الصلب
INT 12h	0000:0048H	مقاطعة حجم الذاكرة
INT 11h	0000:0044h	مقاطعة تعيد الأجهزة المتصلة بالكمبيوتر
INT 10h	0000:0040h	خدمات شاشة العرض
INT 0Fh	0000:003Ch	LPT1 خدمة المنفذ
INT 0Eh	0000:0038h	خدمات القرص المرن
INT 0Dh	0000:0034h	LPT 2 خدمة المنفذ المتوازي

## السلام عليكم

\*\_\_\*\* \*\*\*\*\_\_\*

يبدو اني اكتب مواضيع دون ان يستفيد منها احد  
ان كان السبب هو صعوبتها فهذه هي البرمجة فهم الاشياء بدقة وليست البرمجة بحث عن كود تم  
نسخ لصق، السبب واضح لما تريد ان تعمل برنامج ستبحث عن الكود الذي يعمل كذا وكذا تم نسخ  
لصق ، لكن لو فهمت الاشياء عن كتب فلن تحتاج شيئا وهذا ما يسمى الابداع البرمجي

كانك تتكلم مع شخص وانت لا تعرف لغته ، لو اردت ان تقول له جملة فعليك البحث عن هذه الجملة  
بلغته او ترجمتها من لغتك للغته ، اما ان كنت تتقن لغته فان تحتاج الى شيء ستتكلم معه مباشرة  
واي شيء تستطيع قوله له ، نفس الشيء في البرمجة فمستوى الابداع يتطلب فهم الاشياء بعمق  
الاسمبلي تتكلم مع المعالج والذاكرة والهارد بشكل عام بشكل مباشر تقول له اعمل هذه واعمل هذه  
.... وكذا .... اما في اللغات الاخرى فانت مضطر للبحث عن الكود والمكتبات

قد تلحق مستوى الابداع في اللغات العليا لكن بعد سنوات عديدة

لهذا فان فهم الاشياء بعمق وفهم ما يجري وراء الكواليس ياهلك الى الوصول لمستوى الابداع  
البرمجي في وقت اشبه بالقصير

\*\_\_\*\* \*\*\*\*\_\_\*

اليوم ساتكلم عن البرمجة بالاسمبلي في بيئة 32 بت التي سنستعمل بها دوال API  
والموضوع مهم حتى لاصحاب اللغات الاخرى في التعامل مع الـ API التي تسهل عليك  
العمل

سواء كان VB او DELPHI او ++C او ..... JAVA

سنقوم اليوم بعمل نفس البرنامج الذي عملناه في الدرس السابق عن التعامل مع الملفات

بصراحة من يوم ما ظهرت 32بت سهلت البرمجة بالاسمبلي بشكل كبير

وسترون الفرق في هذا الموضوع مع الموضوع السابق

في الموضوع السابق عملنا برنامج التعامل مع الملفات على بيئة 16بت اليوم سنعمل تقريبا نفس البرنامج على 32بت

الفرق ان الاول كان للانشاء والكتابة الان سيكون للقراءة والعرض ( فقط للتنويع )

لكن الاسمبلي 16بت تبقى الاقوى في التعامل مع الهارد و هي المفضلة لدي

اما 32بت فنتفع ان كنت تريد عمل برنامج كبير دو واجهات وازرار وقواعد بيانات...

حتى لا يقع لك خلط بين 16بت و 32بت ففي 16بت نستعمل المقاطعات اما في 32بت فنستعمل الـ API

خلاصة القول : ان وجدت كلمة مثل 21h lint او 16h lint او ..... فاعلم انه 16بت

\*\_\_\*\*\*\*\*

تبدأ بتعريف معنى الـ API

الـ API ببساطة هي مجموعة كبيرة من الوظائف كل وظيفة لها امر معين تقوم به وهي تساعد كثيرا في البرمجة

لابسط الفهوم من اوله نأخذ اسهل كود في الدلفي

ShowMessage ("startimes")

من العروف فهذا السطر سيقوم بادهار رسالة لها شكل رسومي على الشاشة

انت هنا استعملت دالة من دوال الـ API وهي دالة MessageBox

في الدلفي الامر ShowMessage يقوم باستعداد هذه الدالة

او كي جيد ، لكن ماذا لو لم تكن هذه الدالة موجودة وارادت ان تظهر رسالة للمستخدم ؟

اممممم ، سنقوم بعمل ما يسمى السبلاش Splash وتقوم باسدا عائها



برنامج يعمل الاتي:

```
/*-----*/  
فتح ملف  
قراءة بيانات الملف  
غلق الملف
```

```
/*-----*/
```

سنحتاج الدوال الاتية

CreateFile , ReadFile , CloseHandle , MessageBox , GetModuleHandle  
, ExitProcess

من يبرمج بالـ ++C او الـ QT فهي مفهومة لديه

|

لكن لنقم بشرح كل دالة على حدى حتى نكون واضحين

|

CreateFile

( من اسمها يظهر عملها ) اسم على مسمى

تقوم هذه الدالة بانشاء وفتح ملف واعادة مقبض الملف

لمعرفتها اكثر زر موقع مايكروسوفت لكن ستجد الشرح بالانجليزي

: ستجدها على الشكل الاتي

HANDLE CreateFile(

LPCTSTR lpFileName, // pointer to name of the file



```
DWORD dwDesiredAccess, // access (read-write) mode
DWORD dwShareMode, // share mode
LPSECURITY_ATTRIBUTES lpSecurityAttributes, // pointer to security attributes
DWORD dwCreationDistribution, // how to create
DWORD dwFlagsAndAttributes, // file attributes
HANDLE hTemplateFile // handle to file with attributes to copy
);
```

نشرح هذه الدالة اكثر ، ما تراه تحت اسم الدالة هي بارامتراتها

ما يهمنا هو البرامتر الاول الذي يخص اسم الملف

: لنستعمل هذه الدالة في برنامجنا نكتبها بهذا الشكل

```
invoke CreateFileA, addr lpFileName, GENERIC_READ | GENERIC_WRITE, FILE_SHARE_READ |
FILE_SHARE_WRITE, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL
```

الامر **invoke** في الاسملي يستخدم لاستدعاء الدوال

نشرح قليلا برامترات هذه الدالة :

**lpFileName** : هذا خاص باسم الملف المراد انشائه

**dwDesiredAccess** : ويأخذ احد القيمتين **GENERIC\_READ-WRITE**

**dwShareMode** : بارامتر نوع الوصول للملف للكتابة فقط او القراءة او كلاهما  
سنستخدمه نحن للكتابة والقراءة في نفس الوقت

**lpSecurityAttributes** : لا نحتاجه وسنعطيه القيمة **NULL**

**dwCreationDistribution** : نستخدم في هذا البرامتر **CREATE\_NEW** او  
**OPEN\_EXISTING**

**dwFlagsAndAttributes** : هذا لتحديد خصائص الملف المراد انشائه

**hTemplateFile** : لا نحتاجه وسنعطيه القيمة **NULL**

هذه الدالة ليست فقط لانشاء ملف بل للتحقق ما اذا كان الملف موجودا ام لا

هذه الدالة ترجع قيمتين الى المسجل **EAX** اما قيمة مقبض الملف او القيمة **FFFFFFFF**

اذا تم انشاء الملف فالمسجل EAX سيكون حاملا لمقبض الملف اما ان وقع مشكل فستضع القيمة FFFFFFFF في المسجل EAX

يمكننا معرفة ما اذا تمت العملية بنجاح ام فشلت ، كيف ؟

سنقوم بمقارنة قيمة المسجل EAX مع القيمة FFFFFFFF بعد تنفيذ العملية

سيكون الامر هكذا

```
CMP EAX,-1
```

```
JE Error_Creating
```

>> ملاحظة : لمن لا يعرف انظمة العد فقد يتساؤل من اين جاء -1

<< FFFFFFFFh = -1

الكود واضح اذا كانت قيمة EAX=FFFFFFF فهذا يعني فشل العملية بالتالي سنقد امر القفز JE الى كود اضهار رسالة خطأ للمستخدم

اعرف سؤالك ، سنقول لي ما هو مقبض الملف file handle ؟

لا اعرف كيف اشرح لك ما معنى المقبض ، تخيل ان لكل نافذة مقبض ، ان اردت الوصول لها يلزمك مقبضها

نفس الشيء في برنامجنا عندما تنشأ الدالة الملف سنقوم بتخزين مقبضه حتى تتمكن الدالة الاخرى من الكتابة على هذا الملف

لكي تكتب عليه يلزمها مقبضه

```
ReadFile
```

هذه الدالة تستدم مع الدالة الاولى ، وتعمل في حالة تم ارجاع مقبض الملف ، ودورها قراءة البيانات من الملف الذي اخدت مقبضه

هذه الدالة لما الشكل الاتي

:

**BOOL ReadFile(**

```
HANDLE hFile, // handle of file to read  
LPVOID lpBuffer, // address of buffer that receives data  
DWORD nNumberOfBytesToRead, // number of bytes to read  
LPDWORD lpNumberOfBytesRead, // address of number of bytes read  
LPOVERLAPPED lpOverlapped // address of structure for data  
);
```

**hFile** : مقبض الملف المرجع من قبل الدالة **CreateFile**

**lpBuffer** : مخزن البايتات المقروءة

**nNumberOfBytesToRead** : عدد البايتات التي نريد قرائتها من الملف

**lpNumberOfBytesRead** : عدد البايتات المقروءة فعليا من الملف

**lpOverlapped** : سيحمل القيمة **NULL**

: لنستعمل هذه الدالة في برنامجنا نكتبها بهذا الشكل

```
invoke ReadFile, addr hfile, addr fbuffer, 12, addr bnum, NULL
```

**12** هو عدد البايتات المراد قرائتها ويمكن ان نستعمل متحول يحمل قيمة البايتات التي نريد قرائتها

|

عدرا فقد تعبت من الكتابة ساكمل الدرس في المساء ان شاء الله

لا اريد قراءة الموضوع والخروج فاني اتعب على الموضوع لاكتبه والله اكثر من 3 ساعات لانه ليس من السهل التبسيط وترتيب الامور

اريد ان تردو بما ان كنتو قد استفدتم ام اني اكتب مواضيعي وتذهب مع الريح

## الي الملحقات :

# x86 registers & Flags (عمل تكبير Zoom للصورة لتفاصيل أكثر)

## GPR (General Purpose Registers)

Register	Width	Sub-Registers
RAX	63 QWORD	EAX, AX, AH, AL
RCX	31 DWORD	ECX, CX, CH, CL
RDX	63 QWORD	EDX, DX, DH, DL
RBX	63 QWORD	EBX, BX, BH, BL
RSP	15	ESP, SP, SPL
RBP	63 QWORD	EBP, BP, BPL
RSI	63 QWORD	ESI, SI, SIL
RDI	63 QWORD	EDI, DI, DIL
RIP	63 QWORD	EIP, IP
RFLGS	63 QWORD	EFLAGS, FLAGS
R8	8	R8D, R8W, R8B
R9	8	R9D, R9W, R9B
R10	16	R10D, R10W, R10B
R11	16	R11D, R11W, R11B
R12	16	R12D, R12W, R12B
R13	16	R13D, R13W, R13B
R14	16	R14D, R14W, R14B
R15	16	R15D, R15W, R15B

## MXCSR REGISTER

Bit	Field	Description
31	RESERVED	
17	MM_FLUSH	Flush to Zero
15	FZ	Rounding Control
14	RC	Rounding Control
12	PM	Precision Mask
11	UM	Underflow Mask
10	OM	Overflow Mask
9	ZM	Denormalized-Zero Mask
8	DM	Denormalized-Zero Mask
7	IM	Invalid Operation Mask
6	DAZ	Denormals Are Zeros
5	PE	Precision Flag
4	UE	Underflow Flag
3	OE	Overflow Flag
2	ZE	Zero Flag
1	DE	Denormalized Flag
0	IE	Invalid Operation Flag

## EFLAGS REGISTER

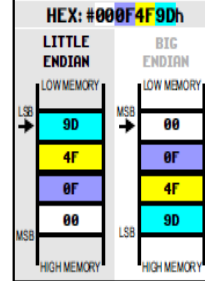
Bit	Field	Description
31	RESERVED	RESERVED FOR INTEL
30	ID	Virtual Interrupt Pending Flag
29	VP	Virtual Interrupt Pending Flag
28	IF	Interrupt Flag
27	PL	Priority Flag
26	AC	Alignment Check Flag
25	VM	Virtual 8086 Mode
24	RF	Resume Flag
23	NT	Nested Task
22	IOPL	I/O Privilege Level
21	IOPL	I/O Privilege Level
20	IOPL	I/O Privilege Level
19	IOPL	I/O Privilege Level
18	IOPL	I/O Privilege Level
17	IOPL	I/O Privilege Level
16	IOPL	I/O Privilege Level
15	IOPL	I/O Privilege Level
14	IOPL	I/O Privilege Level
13	IOPL	I/O Privilege Level
12	IOPL	I/O Privilege Level
11	IOPL	I/O Privilege Level
10	IOPL	I/O Privilege Level
9	IOPL	I/O Privilege Level
8	IOPL	I/O Privilege Level
7	IOPL	I/O Privilege Level
6	IOPL	I/O Privilege Level
5	IOPL	I/O Privilege Level
4	IOPL	I/O Privilege Level
3	IOPL	I/O Privilege Level
2	IOPL	I/O Privilege Level
1	IOPL	I/O Privilege Level
0	IOPL	I/O Privilege Level

## FLAGS

Bit	Field	Description
31	CF	Carry Flag
30	PF	Parity Flag (0=odd 1=even)
29	AF	Auxiliary Carry Flag
28	SF	Sign Flag (result is negative)
27	ZF	Zero Flag (result is zero)
26	OF	Overflow Flag (you single step mode)
25	DF	Direction Flag
24	IF	Interrupt Enable Flag
23	TF	Trap Flag
22	IF	Interrupt Enable Flag
21	IF	Interrupt Enable Flag
20	IF	Interrupt Enable Flag
19	IF	Interrupt Enable Flag
18	IF	Interrupt Enable Flag
17	IF	Interrupt Enable Flag
16	IF	Interrupt Enable Flag
15	IF	Interrupt Enable Flag
14	IF	Interrupt Enable Flag
13	IF	Interrupt Enable Flag
12	IF	Interrupt Enable Flag
11	IF	Interrupt Enable Flag
10	IF	Interrupt Enable Flag
9	IF	Interrupt Enable Flag
8	IF	Interrupt Enable Flag
7	IF	Interrupt Enable Flag
6	IF	Interrupt Enable Flag
5	IF	Interrupt Enable Flag
4	IF	Interrupt Enable Flag
3	IF	Interrupt Enable Flag
2	IF	Interrupt Enable Flag
1	IF	Interrupt Enable Flag
0	IF	Interrupt Enable Flag

## Segment Registers

Register	Width	Segment
SS	16	Stack Segment
DS	16	Data Segment
ES	16	Extra Data Segment
CS	16	Code Segment
FS	16	Extra Data Segment 2
GS	16	Extra Data Segment 3



Bit	Field	Description
15	ST(7)	Stack Top
14	ST(6)	Stack Top
13	ST(5)	Stack Top
12	ST(4)	Stack Top
11	ST(3)	Stack Top
10	ST(2)	Stack Top
9	ST(1)	Stack Top
8	ST(0)	Stack Top
7	ST(0)	Stack Top
6	ST(0)	Stack Top
5	ST(0)	Stack Top
4	ST(0)	Stack Top
3	ST(0)	Stack Top
2	ST(0)	Stack Top
1	ST(0)	Stack Top
0	ST(0)	Stack Top

BIT	BYTES	DEFINE	RESERVE	RANGE	SIGN RANGE
BIT	1b			0-1	
NIBBLE	4b			0-15	
BYTES	8b	1B	DB, RB	0-255	-128 +127
WORD	16b	2B	DW, DU, RW	0-65535	-32768 +32767
DWORD	32b	4B	DD, RD	0-2132	-2131 +2131-1
QWORD	64b	8B	DQ, RQ	0-2148	-2147 +2147-1
TWORD	80b	10B	DT, RT	0-2164	-2163 +2163-1
DQWORD, OWORD	128b	16B		0-2179	-2179 +2179-1

## Control registers

Register	Width	Description
CR0	32	Control Register 0
CR2	32	Control Register 2
CR3	32	Control Register 3
CR4	32	Control Register 4
CR8	32	Control Register 8
CR15	32	Control Register 15

## Debug registers

Register	Width	Description
DR0	32	Linear Address of Instruction
DR1	32	Linear Address of Instruction
DR2	32	Linear Address of Instruction
DR3	32	Linear Address of Instruction
DR4	32	Linear Address of Instruction
DR5	32	Linear Address of Instruction
DR6	32	Linear Address of Instruction
DR7	32	Linear Address of Instruction
MCAR	63	Register Address of Instruction
MCTR	63	Register Type of Instruction
TR12	63	Testovaci Register
FIP	31	Offset of Instruction
FOP	31	Offset of Opcode
FCS	31	Offset of Segment
FOS	15	Stack Pointer FPU
SWR	15	Stack Pointer FPU
TAG	15	Tag of Instruction

Fixed Point	Value
0000	0
1000	+0.125
2000	+0.25
3000	+0.375
4000	+0.5
5000	+0.625
6000	+0.75
7000	+0.875
7FFF	+0.99
8000	-1
9000	-1.125
0000	-0.75
0000	-0.5
0000	-0.25
E000	-0.125
FFFF	-0.01

## Floating-Point Registers (8 BYTES QWORD 64b) MMX Registers (10 BYTES TWORD)

Register	Width	Sub-Registers
ST0	79	MM0
ST1	79	MM1
ST2	79	MM2
ST3	79	MM3
ST4	79	MM4
ST5	79	MM5
ST6	79	MM6
ST7	79	MM7

## SSE2 (16 BYTES) LONG MODE 64-BIT

Register	Width	Sub-Registers
XMM0	127	
XMM1	127	
XMM2	127	
XMM3	127	
XMM4	127	
XMM5	127	
XMM6	127	
XMM7	127	
XMM8	127	
XMM9	127	
XMM10	127	
XMM11	127	
XMM12	127	
XMM13	127	
XMM14	127	
XMM15	127	

Control Word Register CWR	FPU Control Register
15	IC
14	RC
13	PC
12	EM
11	PM
10	UM
9	OM
8	ZM
7	DM
6	IM
5	IM
4	IM
3	IM
2	IM
1	IM
0	IM

Status Word Register SWR	FPU Status Register
15	B
14	C3
13	TOP
12	C2
11	C1
10	CO
9	IR
8	SF
7	PE
6	UE
5	OE
4	ZE
3	DE
2	IE
1	IE
0	IE

## ملحق جداول اكواد الآسكي ASCII Table and Description

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	Space	64	40	100	&#64;	@	96	60	140	&#96;	`
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	A	97	61	141	&#97;	a
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	B	98	62	142	&#98;	b
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	C	99	63	143	&#99;	c
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	\$	68	44	104	&#68;	D	100	64	144	&#100;	d
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	E	101	65	145	&#101;	e
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	F	102	66	146	&#102;	f
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	G	103	67	147	&#103;	g
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	(	72	48	110	&#72;	H	104	68	150	&#104;	h
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	I	105	69	151	&#105;	i
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	J	106	6A	152	&#106;	j
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	K	107	6B	153	&#107;	k
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	L	108	6C	154	&#108;	l
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	M	109	6D	155	&#109;	m
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	N	110	6E	156	&#110;	n
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	O	111	6F	157	&#111;	o
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	P	112	70	160	&#112;	p
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	Q	113	71	161	&#113;	q
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	R	114	72	162	&#114;	r
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	S	115	73	163	&#115;	s
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	T	116	74	164	&#116;	t
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	U	117	75	165	&#117;	u
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	V	118	76	166	&#118;	v
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	W	119	77	167	&#119;	w
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	X	120	78	170	&#120;	x
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	Y	121	79	171	&#121;	y
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	Z	122	7A	172	&#122;	z
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	;	91	5B	133	&#91;	[	123	7B	173	&#123;	{
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	]	125	7D	175	&#125;	}
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	^	126	7E	176	&#126;	~
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	DEL

Source: [www.LookupTables.com](http://www.LookupTables.com)

## Extended ASCII Codes

128	Ç	144	É	160	á	176	⋯	192	Ł	208	⋮	224	α	240	≡
129	ù	145	æ	161	í	177	⋯	193	ł	209	⋈	225	β	241	±
130	é	146	Æ	162	ó	178	⋯	194	ṽ	210	⋈	226	Γ	242	≥
131	â	147	ô	163	ú	179		195	ṽ	211	⋮	227	π	243	≤
132	ä	148	ö	164	ñ	180	†	196	–	212	⋮	228	Σ	244	∫
133	à	149	ò	165	Ñ	181	‡	197	+	213	ƒ	229	σ	245	∫
134	â	150	û	166	ª	182	‡	198	†	214	ƒ	230	μ	246	+
135	ç	151	ù	167	º	183	π	199	†	215	‡	231	τ	247	≈
136	ê	152	ÿ	168	¸	184	¶	200	⋮	216	‡	232	Φ	248	°
137	ë	153	Ö	169	¸	185	‡	201	ƒ	217	∫	233	⊖	249	·
138	è	154	Û	170	¬	186		202	⋮	218	∫	234	Ω	250	·
139	ï	155	◊	171	½	187	¶	203	⋈	219	■	235	δ	251	√
140	î	156	£	172	¼	188	¶	204	†	220	■	236	∞	252	∞
141	ï	157	¥	173	¡	189	¶	205	=	221	■	237	φ	253	²
142	Ä	158	£	174	«	190	¶	206	‡	222	■	238	ε	254	■
143	Å	159	f	175	»	191	¶	207	±	223	■	239	∩	255	

Source: [www.LookupTables.com](http://www.LookupTables.com)

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	\$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	□



Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	Ł	224	E0	α
129	81	ü	161	A1	í	193	C1	ł	225	E1	β
130	82	é	162	A2	ó	194	C2	Ť	226	E2	Γ
131	83	â	163	A3	ú	195	C3	ł	227	E3	Π
132	84	ä	164	A4	ñ	196	C4	—	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	†	229	E5	σ
134	86	å	166	A6	ª	198	C6	‡	230	E6	μ
135	87	ç	167	A7	º	199	C7	‡	231	E7	τ
136	88	ê	168	A8	¿	200	C8	Ł	232	E8	Φ
137	89	ë	169	A9	ƒ	201	C9	Ŧ	233	E9	Θ
138	8A	è	170	AA	Ŧ	202	CA	Ł	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	Ŧ	235	EB	δ
140	8C	î	172	AC	¼	204	CC	‡	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	=	237	ED	∞
142	8E	Ë	174	AE	«	206	CE	‡	238	EE	ε
143	8F	Ā	175	AF	»	207	CF	Ł	239	EF	Π
144	90	É	176	B0	⋯	208	DO	Ł	240	FO	≡
145	91	æ	177	B1	⋮	209	D1	Ŧ	241	F1	±
146	92	Æ	178	B2	■	210	D2	π	242	F2	≥
147	93	ô	179	B3		211	D3	Ł	243	F3	≤
148	94	ö	180	B4	†	212	D4	Ł	244	F4	[
149	95	ò	181	B5	‡	213	D5	Ŧ	245	F5	]
150	96	û	182	B6	‡	214	D6	π	246	F6	÷
151	97	ù	183	B7	π	215	D7	‡	247	F7	∞
152	98	ÿ	184	B8	ƒ	216	D8	‡	248	F8	°
153	99	ÿ	185	B9	‡	217	D9	┘	249	F9	▪
154	9A	Ü	186	BA		218	DA	Ŧ	250	FA	·
155	9B	◊	187	BB	Ŧ	219	DB	■	251	FB	√
156	9C	£	188	BC	Ł	220	DC	■	252	FC	∩
157	9D	₣	189	BD	Ł	221	DD	■	253	FD	∞
158	9E	℞	190	BE	Ł	222	DE	■	254	FE	■
159	9F	f	191	BF	Ŧ	223	DF	■	255	FF	□

## جدول الآسكي للحروف العربية

**ملحوظة:** الأرقام في قائمة Windows 1256 هي بالسداسي العشري او ( الهكسي Hex )

Name (Unicode name)	UNICODE	Buckwalter	ISO 8859-6	ASMO 449	Windows 1256	Glyph
hamza-on-the-line (Arabic letter hamza)	\u0621	'	C1	A	C1	ء
madda (Arabic letter alef with madda above)	\u0622		C2	B	C2	آ
hamza-on-'alif (Arabic letter aleph with hamza above)	\u0623	>	C3	C	C3	أ
hamza-on-waaw (Arabic letter waw with hamza above)	\u0624	&	C4	D	C4	ؤ
hamza-under-'alif (Arabic letter aleph with hamza below)	\u0625	<	C5	E	C5	إ
hamza-on-yaa' (Arabic letter yeh with hamza above)	\u0626	}	C6	F	C6	ى
bare 'alif (Arabic letter alef)	\u0627	A	C7	G	C7	ا
baa' (Arabic letter beh)	\u0628	b	C8	H	C8	ب
taa' marbuuTa (Arabic letter teh marbuta)	\u0629	p	C9	I	C9	ة
taa' (Arabic letter teh)	\u062A	t	CA	J	CA	ت
thaa' (Arabic letter theh)	\u062B	v	CB	K	CB	ث
jiim (Arabic letter jeem)	\u062C	j	CC	L	CC	ج
Haa' (Arabic letter hah)	\u062D	H	CD	M	CD	ح
khaa' (Arabic letter khah)	\u062E	x	CE	N	CE	خ
daal (Arabic letter dal)	\u062F	d	CF	O	CF	د
dhaal (Arabic letter thal)	\u0630	*	D0	P	D0	ذ

## تابع ... جدول الآسكي للحروف العربية

**ملحوظة :** الأرقام في قائمة Windows 1256 هي بالسداسي العشري او ( الهكسي Hex )

reh (Arabic letter reh)	\u0631	r	D1	Q	D1	ر
zain (Arabic letter zain)	\u0632	z	D2	R	D2	ز
seen (Arabic letter seen)	\u0633	s	D3	S	D3	س
sheen (Arabic letter sheen)	\u0634	\$	D4	T	D4	ش
sad (Arabic letter sad)	\u0635	S	D5	U	D5	ص
dad (Arabic letter dad)	\u0636	D	D6	V	D6	ض
tah (Arabic letter tah)	\u0637	T	D7	W	D8	ط
zah (Arabic letter zah)	\u0638	Z	D8	X	D9	ظ
ain (Arabic letter ain)	\u0639	E	D9	Y	DA	ع
ghain (Arabic letter ghain)	\u063A	g	DA	Z	DB	غ
tatweel (Arabic letter tatweel)	\u0640	-	E0	0x60	DC	ا
feh (Arabic letter feh)	\u0641	f	E1	a	DD	ف
qaf (Arabic letter qaf)	\u0642	q	E2	b	DE	ق
kaf (Arabic letter kaf)	\u0643	k	E3	c	DF	ك
lam (Arabic letter lam)	\u0644	l	E4	d	E1	ل
meem (Arabic letter meem)	\u0645	m	E5	e	E3	م
noon (Arabic letter noon)	\u0646	n	E6	f	E4	ن
heh (Arabic letter heh)	\u0647	h	E7	g	E5	ه
waw (Arabic letter waw)	\u0648	w	E8	h	E6	و

kaaf (Arabic letter kaf)	\u0643	k	E3	c	DF	ك
laam (Arabic letter lam)	\u0644	l	E4	d	E1	ل
miim (Arabic letter meem)	\u0645	m	E5	e	E3	م
nuun (Arabic letter noon)	\u0646	n	E6	f	E4	ن
haa' (Arabic letter heh)	\u0647	h	E7	g	E5	ه
waaw (Arabic letter waw)	\u0648	w	E8	h	E6	و
'alif maqSuura (Arabic letter alef maksura)	\u0649	Y	E9	i	EC	ى
yaa' (Arabic letter yeh)	\u064A	y	EA	j	ED	ي
fatHatayn (Arabic fathatan)	\u064B	F	EB	k	F0	؀
Dammatayn (Arabic dammatan)	\u064C	N	EC	l	F1	؁
kasratayn (Arabic kasratan)	\u064D	K	ED	m	F2	؂
fatHa (Arabic fatha)	\u064E	a	EE	n	F3	؃
Daamma (Arabic damma)	\u064F	u	EF	o	F5	؅
kasra (Arabic kasra)	\u0650	i	F0	p	F6	؆
shaddah (Arabic shadda)	\u0651	~	F1	q	F8	؈
sukuun (Arabic sukun)	\u0652	o	F2	r	FA	؉
dagger 'alif (Arabic letter superscript alef)	\u0670	`	(missing)	(missing)	(missing)	
waSla-on-alif (Arabic letter alef wasla)	\u0671	{	(missing)	(missing)	(missing)	

# ملحق جداول التعليمات لمعالجات انتل Instruction Tables

Intel Assembler 80186 and higher

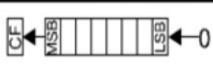





## CodeTable 1/2

© 1996-2003 by Roger Jegerlehner, Switzerland  
V 2.3 English. Also available in Spanish

TRANSFER		Code	Operation	Flags									
Name	Comment			O	D	I	T	S	Z	A	P	C	
MOV	Move (copy)	MOV Dest,Source	Dest:=Source										
XCHG	Exchange	XCHG Op1,Op2	Op1:=Op2 , Op2:=Op1										
STC	Set Carry	STC	CF:=1										1
CLC	Clear Carry	CLC	CF:=0										0
CMC	Complement Carry	CMC	CF:= ¬CF										±
STD	Set Direction	STD	DF:=1 (string op's downwards)		1								
CLD	Clear Direction	CLD	DF:=0 (string op's upwards)		0								
STI	Set Interrupt	STI	IF:=1			1							
CLI	Clear Interrupt	CLI	IF:=0			0							
PUSH	Push onto stack	PUSH Source	DEC SP, [SP]:=Source										
PUSHF	Push flags	PUSHF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL										
PUSHA	Push all general registers	PUSHA	AX, CX, DX, BX, SP, BP, SI, DI										
POP	Pop from stack	POP Dest	Dest:=[SP], INC SP										
POPF	Pop flags	POPF	O, D, I, T, S, Z, A, P, C 286+: also NT, IOPL	±	±	±	±	±	±	±	±	±	±
POPA	Pop all general registers	POPA	DI, SI, BP, SP, BX, DX, CX, AX										
CBW	Convert byte to word	CBW	AX:=AL (signed)										
CWD	Convert word to double	CWD	DX:AX:=AX (signed)	±				±	±	±	±	±	±
CWDE	Conv word extended double	CWDE <small>386</small>	EAX:=AX (signed)										
IN <i>i</i>	Input	IN Dest, Port	AL/AX/EAX := byte/word/double of specified port										
OUT <i>i</i>	Output	OUT Port, Source	Byte/word/double of specified port := AL/AX/EAX										


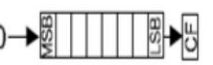
*i* for more information see instruction specifications

Flags: ±=affected by this instruction ?=undefined after this instruction

ARITHMETIC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
ADD	Add	ADD Dest,Source	Dest:=Dest+Source	±				±	±	±	±	±
ADC	Add with Carry	ADC Dest,Source	Dest:=Dest+Source+CF	±				±	±	±	±	±
SUB	Subtract	SUB Dest,Source	Dest:=Dest-Source	±				±	±	±	±	±
SBB	Subtract with borrow	SBB Dest,Source	Dest:=Dest-(Source+CF)	±				±	±	±	±	±
DIV	Divide (unsigned)	DIV Op	Op=byte: AL:=AX / Op      AH:=Rest	?				?	?	?	?	?
DIV	Divide (unsigned)	DIV Op	Op=word: AX:=DX:AX / Op      DX:=Rest	?				?	?	?	?	?
DIV <sub>386</sub>	Divide (unsigned)	DIV Op	Op=double.: EAX:=EDX:EAX / Op      EDX:=Rest	?				?	?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op=byte: AL:=AX / Op      AH:=Rest	?				?	?	?	?	?
IDIV	Signed Integer Divide	IDIV Op	Op=word: AX:=DX:AX / Op      DX:=Rest	?				?	?	?	?	?
IDIV <sub>386</sub>	Signed Integer Divide	IDIV Op	Op=double.: EAX:=EDX:EAX / Op      EDX:=Rest	?				?	?	?	?	?
MUL	Multiply (unsigned)	MUL Op	Op=byte: AX:=AL*Op      if AH=0 ♦	±				?	?	?	?	±
MUL	Multiply (unsigned)	MUL Op	Op=word: DX:AX:=AX*Op      if DX=0 ♦	±				?	?	?	?	±
MUL <sub>386</sub>	Multiply (unsigned)	MUL Op	Op=double: EDX:EAX:=EAX*Op      if EDX=0 ♦	±				?	?	?	?	±
IMUL <sub>i</sub>	Signed Integer Multiply	IMUL Op	Op=byte: AX:=AL*Op      if AL sufficient ♦	±				?	?	?	?	±
IMUL	Signed Integer Multiply	IMUL Op	Op=word: DX:AX:=AX*Op      if AX sufficient ♦	±				?	?	?	?	±
IMUL <sub>386</sub>	Signed Integer Multiply	IMUL Op	Op=double: EDX:EAX:=EAX*Op      if EAX sufficient ♦	±				?	?	?	?	±
INC	Increment	INC Op	Op:=Op+1 (Carry not affected !)	±				±	±	±	±	
DEC	Decrement	DEC Op	Op:=Op-1 (Carry not affected !)	±				±	±	±	±	
CMP	Compare	CMP Op1,Op2	Op1-Op2	±				±	±	±	±	±
SAL	Shift arithmetic left (≡ SHL)	SAL Op,Quantity		<i>i</i>				±	±	?	±	±
SAR	Shift arithmetic right	SAR Op,Quantity		<i>i</i>				±	±	?	±	±
RCL	Rotate left through Carry	RCL Op,Quantity		<i>i</i>								±
RCR	Rotate right through Carry	RCR Op,Quantity		<i>i</i>								±
ROL	Rotate left	ROL Op,Quantity		<i>i</i>								±
ROR	Rotate right	ROR Op,Quantity		<i>i</i>								±

*i* for more information see instruction specifications

♦ then CF:=0, OF:=0 else CF:=1, OF:=1

LOGIC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NEG	Negate (two-complement)	NEG Op	Op:=0-Op      if Op=0 then CF:=0 else CF:=1	±				±	±	±	±	±
NOT	Invert each bit	NOT Op	Op:=¬Op (invert each bit)									
AND	Logical and	AND Dest,Source	Dest:=Dest∧Source	0				±	±	?	±	0
OR	Logical or	OR Dest,Source	Dest:=Dest∨Source	0				±	±	?	±	0
XOR	Logical exclusive or	XOR Dest,Source	Dest:=Dest (exor) Source	0				±	±	?	±	0
SHL	Shift logical left (≡ SAL)	SHL Op,Quantity		<i>i</i>				±	±	?	±	±
SHR	Shift logical right	SHR Op,Quantity		<i>i</i>				±	±	?	±	±

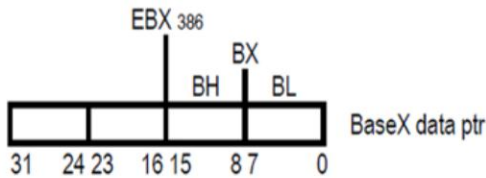
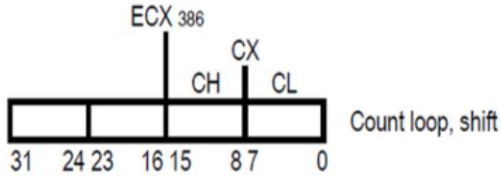
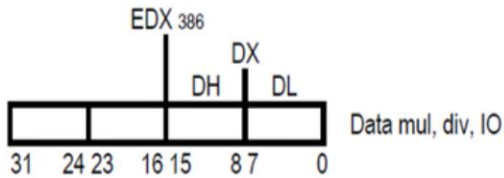
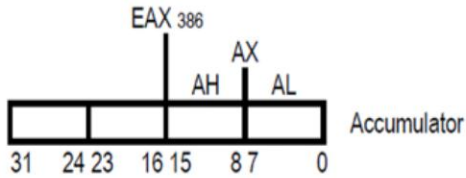
MISC				Flags								
Name	Comment	Code	Operation	O	D	I	T	S	Z	A	P	C
NOP	No operation	NOP	No operation									
LEA	Load effective address	LEA Dest,Source	Dest := address of Source									
INT	Interrupt	INT Nr	interrupts current program, runs spec. int-program			0	0					

JUMPS (flags remain unchanged)				Name	Comment	Code	Operation
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(= JZ)	JNE	Jump if not Equal	JNE Dest	(= JNZ)
JZ	Jump if Zero	JZ Dest	(= JE)	JNZ	Jump if not Zero	JNZ Dest	(= JNE)
JCXZ	Jump if CX Zero	JCXZ Dest		JECXZ	Jump if ECX Zero	JECXZ Dest	386
JP	Jump if Parity (Parity Even)	JP Dest	(= JPE)	JNP	Jump if no Parity (Parity Odd)	JNP Dest	(= JPO)
JPE	Jump if Parity Even	JPE Dest	(= JP)	JPO	Jump if Parity Odd	JPO Dest	(= JNP)

JUMPS (flags remain unchanged)				Name	Comment	Code	Operation
CALL	Call subroutine	CALL Proc		RET	Return from subroutine	RET	
JMP	Jump	JMP Dest					
JE	Jump if Equal	JE Dest	(= JZ)	JNE	Jump if not Equal	JNE Dest	(= JNZ)
JZ	Jump if Zero	JZ Dest	(= JE)	JNZ	Jump if not Zero	JNZ Dest	(= JNE)
JCXZ	Jump if CX Zero	JCXZ Dest		JECXZ	Jump if ECX Zero	JECXZ Dest	386
JP	Jump if Parity (Parity Even)	JP Dest	(= JPE)	JNP	Jump if no Parity (Parity Odd)	JNP Dest	(= JPO)
JPE	Jump if Parity Even	JPE Dest	(= JP)	JPO	Jump if Parity Odd	JPO Dest	(= JNP)

JUMPS Unsigned (Cardinal)				JUMPS Signed (Integer)			
JA	Jump if Above	JA Dest	(= JNBE)	JG	Jump if Greater	JG Dest	(= JNLE)
JAЕ	Jump if Above or Equal	JAЕ Dest	(= JNB = JNC)	JGE	Jump if Greater or Equal	JGE Dest	(= JNL)
JB	Jump if Below	JB Dest	(= JNAE = JC)	JL	Jump if Less	JL Dest	(= JNGE)
JBE	Jump if Below or Equal	JBE Dest	(= JNA)	JLE	Jump if Less or Equal	JLE Dest	(= JNG)
JNA	Jump if not Above	JNA Dest	(= JBE)	JNG	Jump if not Greater	JNG Dest	(= JLE)
JNAE	Jump if not Above or Equal	JNAE Dest	(= JB = JC)	JNGE	Jump if not Greater or Equal	JNGE Dest	(= JL)
JNB	Jump if not Below	JNB Dest	(= JAE = JNC)	JNL	Jump if not Less	JNL Dest	(= JGE)
JNBE	Jump if not Below or Equal	JNBE Dest	(= JA)	JNLE	Jump if not Less or Equal	JNLE Dest	(= JG)
JC	Jump if Carry	JC Dest		JO	Jump if Overflow	JO Dest	
JNC	Jump if no Carry	JNC Dest		JNO	Jump if no Overflow	JNO Dest	
				JS	Jump if Sign (= negative)	JS Dest	
				JNS	Jump if no Sign (= positive)	JNS Dest	

## General Registers:



Flags:     O D I T  S Z  -  A  -  P  -  C

### Control Flags (how instructions are carried out):

D: Direction 1 = string op's process down from high to low address  
 I: Interrupt whether interrupts can occur. 1= enabled  
 T: Trap single step for debugging

### Example:

```
.DOSSEG           ; Demo program
.MODEL SMALL
.STACK 1024

Two EQU 2         ; Const
.DATA
VarB DB ?        ; define Byte, any value
VarW DW 1010b    ; define Word, binary
VarW2 DW 257     ; define Word, decimal
VarD DD 0AFFFFh  ; define Doubleword, hex
S DB "Hello!",0  ; define String
.CODE
main: MOV AX,DGROUP ; resolved by linker
      MOV DS,AX     ; init datasegment reg
      MOV [VarB],42 ; init VarB
      MOV [VarD],-7 ; set VarD
      MOV BX,Offset[S] ; addr of "H" of "Hello !"
      MOV AX,[VarW]  ; get value into accumulator
      ADD AX,[VarW2] ; add VarW2 to AX
      MOV [VarW2],AX ; store AX in VarW2
      MOV AX,4C00h  ; back to system
      INT 21h
      END main
```



### Status Flags (result of operations):

C: Carry result of unsigned op. is too large or below zero. 1 = carry/borrow  
 O: Overflow result of signed op. is too large or small. 1 = overflow/underflow  
 S: Sign sign of result. Reasonable for Integer only. 1 = neg. / 0 = pos.  
 Z: Zero result of operation is zero. 1 = zero  
 A: Aux. carry similar to Carry but restricted to the low nibble only  
 P: Parity 1 = result has even number of set bits



واخيرا احمد الله الذي وفقني في جمع وترتيب الدورتين معا واخراجهما في كتاب واحد لتعميم الفائدة , اذا اردت يا أخي الكريم الاستزادة والتوسع في مجال لغة الاسبمبلي وخاصة الهندسة العكسية وفهم عمل البرمجيات والحمايات ما عليك الا التوجه لاول وافضل منتديين عربيين هما :  
العرب المتحدون للهندسة العكسية **AoRE** - والفريق العربي للهندسة العكسية **AT4RE** .

<http://www.aoreteam.com/vb/>

<http://www.at4re.com/f/>

**ولكي أسهل الامر عليك :**

**فقد اصدر اخوتنا في AoRE اسطوانتين لتعلم الهندسة العكسية :**

1- اسطوانة دروس المبتدئين في الهندسة العكسية :

<http://www.mediafire.com/?6pgfj36ciu76c>

2- اسطوانة الفك اليدوي ( كلمة سر فك الضغط AoRE2008 ) :

[http://www.4shared.com/folder/3cXiYd66/Aore\\_Manual\\_Unpack\\_Alfidai.html](http://www.4shared.com/folder/3cXiYd66/Aore_Manual_Unpack_Alfidai.html)

**كما وفق الله اخواننا في AT4RE علي اصدار كتابين لتعلم هذا العلم الفريد :**

1- كتاب مدخل الي الهندسة العكسية :

<http://www.mediafire.com/?bo20mmu79obve02>

2- كتاب الهندسة العكسية خطوة للأمام :

<http://www.mediafire.com/?ahqqa47g4a8e95q>

اضافة لبعض الدروس في الهندسة العكسية في منتدي الفريق العربي للبرمجة:

[www.arabteam2000-forum.com](http://www.arabteam2000-forum.com)

هذا كل ما لدينا علي مستوي العالم العربي من مصادر علمية في مجال الهندسة العكسية نتمني ان نري المزيد من المشاركات العربية في هذا المجال .

## المراجع :

اليك أخي الكريم روابط افضل مكتبتين للغة الاسبمبلي على النت ( ستجد فيها كتب متنوعة عن لغة الاسبمبلي ) :

**مكتبة كتب – ركن لغات البرمجة – لغة الاسبمبلي :**

<http://www.kutub.info/library/category/24>

**مكتبة البوصلة التقنية قسم لغة برمجة الاسبمبلي :**

<http://www.boosla.com/articlesList.php?Sec=Programm&menu=Assembly>

أما عن أجمل كتاب قابلني هو ( مرجع في البرمجة بلغة الاسبمبلي ) للدكتور حميد المسمري لكن عيبه انه خاص بـ اسيمبلي 16 بت وليس 32 لكنه مفيد يعطيك الاساسيات في الاسبمبلي وانا شخصيا استفدت منه كثيرا قم بتنزيله من الروابط الاتية :

<http://www.kutub.info/library/book/7528>

<http://www.boosla.com/showArticle.php?Sec=Programm&id=166>

<http://www.kutub.info/library/book/7471>

<http://forum.kku.edu.sa/showthread.php?t=13839&page=2>

وأخر دعوانا بتوفيق ربنا ان الحمد لله الذي وحده علا

أخوكم في الله : علي السيد محمد ( **الفدائي : Alfidai** )  
[Alfarahidi2@yahoo.com](mailto:Alfarahidi2@yahoo.com)

( الخرطوم – السودان )

2011