

# الأسمبلي

## 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 )  
دو ز الحاجة الى تفسير وهي تخزن بصورة ثنائية [ تركيبية من الأصفار والواحد ] في الذاكرة على شكل تعليمات ووسائط تأخذ كل واحد منها عادة مقدار ٨بت = ١ بايت )  
وكان هذا النوع من البرمجة صعب جداً عندها طور المبرمجون أو لغة برمجة وهذه اللغة فكرتها بسيطة جداً حيث أنه بدل أن تكتب رموز الآله يتم كتابة كلمات مختصره تدل على نوع العمليه مثال ( MOV,ADD,CMP ) ثم ببرنامج بسيط يتم تحويل هذه الشفرة الى لغة الآله باستخدام تخطيط واحد-الى-واحد ياً ن كل سطر أو عبارة في الأسمبلي تحول الى تعليمة واحدة مقابله في لغة الآله (مثال بدل كتابة ٠١١٠٠٠٠٠٠٠٠٠٠١٠١ يتم كتابة mov al, ) يعرف البرنامج الذي يقوم بعملية التحويل بالأسمبلي Assembler ، علماً بأن هناك عدة أنواع من الأسمبلي كل نوع يختص بتقنية معينة وبعائلة معينه من المعالجات

ونحن هنا بصدد تعلم البرمجة بالأسمبلي للمعالجات المبنية على تقنية IBM-PC والمنتجة من شركة إنتل وهي العائلة ٨٠x٨٦ ويرمز لها اختصاراً ٨٦x وهي تضم :  
( ٨٠٨٦ / ٨٠٨٨ / ٨٠١٨٦ / ٨٠٢٨٦ ) لمعالجات لا ١٦ بت و ( ٨٠٢٨٦ / ٨٠٣٨٦ / ٨٠٤٨٦ )  
( ٨٠٥٨٦ = بنتنيوم ١ / ٨٠٦٨٦ = بنتنيوم ٢ / ٨٠٧٨٦ = بنتنيوم ٣ / ٨٠٨٨٦ = بنتنيوم ٤ )  
لمعالجات لا ٣٢ بت وسوف أتطرق في دروس متقدمة الى المعالج أنتيوم ٦٤ بت المبنى بتقنية جديده كلياً لمن يرغب بمعرفة مسبقة لهذا المعالج الجديد كذلك سوف أتطرق بأذن الله الى الكرو سد 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

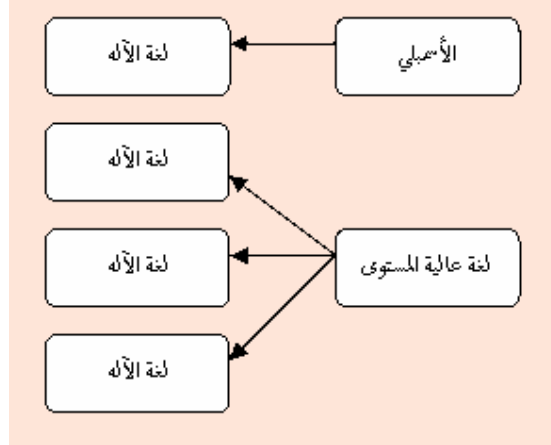
الأسمبلي هو برنامج يقوم بتحويل التعليمات المكتوبة بالأسمبلي الى لغة الآله .

## 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 الا برمجتها بالأسمبلي مثل الدرايفات (سواقات) الأجهزة ، كذلك فإن الأسمبلي يعطيط برامج سريعة جداً ، وبالطبع فإن بناء برنامج متطور بالأسمبلي أشبه بحفر حفرة بواسطة الملعقة فالبرغم أنك تحفر لا أنك أنت لحيك قليله ولكن من المحبذ جداً برمجة بعض الدلائل والأجزاء من البرلمج بالأسمبلي وبقيه البرنامج بواسطة لغة عالية المستوى مثل السي++ .

## العلاقة بين الأسمبلي واللغات الأخرى :

تعتبر كل من لغة الآله و الأسمبلي لغتين منخفضتي المستوى Low-Level Language اختصاراً LLL لأنها تكتب تعليمه تعليمه ( بعض الناس يعتبر لغة السي لغة منخفضة المستوى ، وهذا الكلام أرجح الى الخطأ منه الى الصواب ) بينما تعتبر باقي اللغات High-Level Language اختصاراً HLL وفي هذه اللغات تختفي تقنية تخطيط واحد-الى-واحد وتفسر التعليمه الواحدة الى عدد كبير من تعليمات لغة الآله



مخطط يوضح العلاقة بين لغة الأسمبلي ولغة الآله ولغة عالية المستوى ولغة الآله

## تطبيقات لغة الأسمبلي :

تتطلب كتابة البرامج بلغة الأسمبلي معرفة بالعتاد وعناية خاصة مع الأهتمام بأدق وأقل التفاصيل ، في أيام البرمجة القديمة كان المبرمجون يكتبون برامجهم بلغة الأسمبلي لأن ذاكرة الرام وقتها كانت صغيرة (أقل من ٦٤ كيلوبايت) وهم بحاجة الى برامج أصغر وأسرع خصوصاً أن معالجتهم أيضاً كانت بطيئة ، مع تطور الحاسوب وتوسع سعة ذاكرة الرام وزيادة سرعته أصبحت البرامج أكثر طولاً وتعقيداً ، هذا التعقيد أدى الى استخدام اللغات البرمجية عالية المستوى HLL مثل السي والكوبول والبيسك والباسكال والفورترن ، مرة أخرى تطور الحاسوب فأدى الى استخدام اللغات العليا الموجهة الهدف OOP مثل السي++ والجافا والتي مكنت من كتابة برامج قوامها آلاف الأسطر والتعليمات المعقدة والمتداخلة .

من الصعب أن تلاقى برامج كبيرة مكتوبة كاملة بلغة الأسمبلي لأن كتابتها صعب والأهم من ذلك تطويرها وصيانتها أصعب بكثير ، بدل ذلك يقوم المبرمجين ببرمجة مقاطع مبرمجة مثل بلغة الأسمبلي لأستخدمها في تنفيذ أسرع أو الوصول الى العتاد عن طريقها وباقي البرنامج بواسطة لغة عالية المستوى .

يفض المبرمجين لغة السي++ كلفة قياسية للبرمجة بلغة عليا لأن لها قدرة عالية وقوية جداً وموجهة الهدف مع القدرة على كتابة مقاطع السي فيها وهي لغة أقل أنخفاضاً وأكثر مرونة مع استخدام الأسمبلي كعنصر مهم في الوصول الى العتاد وبرمجة الجزئيات المحتاجة للسرعة .

لا يستخدم المبرمجون شفرة الأسمبلي وسط شفرة لغة عالية المستوى عادة وإنما يستخدمونها عن طريق واجه على شكل دالة أو كائن وتحتوي هذه الدالة أو هذا الكائن على شفرة الأسمبلي المطلوبة ، وقد تستخدم روتيناً فرعياً أو دالة في لغة عالية المستوى وأنت لاتعلم بأنك باستدعاء هذه الدالة أو الروتين الفرعي قد استدعيت شفرة مكتوبة بلغة الأسمبلي .

## لغة الآله :

لغة الآله كما ذكرنا سابقاً هي اللغة التي تستطيع الآله أو المعالج التعامل معها مباشرة ، في العائلة ٨٦X كل معالج يحتوي يستطيع تنفيذ تعليمات المعالج الذي قبله ويملك مجموعة تعليمات موسعة وأضافة لا تستطيع المعالجات التي قبله تنفيذها وأما المعالج الذي يأتي بعد هذا المعالج فإنه يدعم التعليمات الموسعة للمعالج الذي قبله بينما يحتوي هو أيضاً على تعليمات جديدة وموسعة ، بأختصار اذا صممت برنامج لمعالج ما فإن المعالجات ما قبل هذا المعالج لن تستطيع تشغيله بينما المعالج نفسه والمعالجات التي بعده (من نفس العائلة) تستطيع تشغيل البرنامج .

لقد حافظت شركة إنتل على التوافقية في العائلة ٨٦X ابتداء من المعالج ٨٠٨٦ وصولاً الى باتنيوم ٨٠٨٦٤ ولكن الحفاظ على التوافقية يفرض قيود على تصميم المعالج وأستخدام تقنيات قديمة ، ومؤخراً قرر ت شركة إنتل إيقاف عائلة المعالج ٨٦X عند باتنيوم ٤ وقامت بإنشاء معالج جديد (غير متوافق مع العائلة ٨٦X ) مبني على تقنية ٦٤ بت وهو المعالج اتانيوم الجديد .

## مثال على لغة الآله :

التعليمية ١٠١١٠٠٠٠٠٠٠٠٠١٠١ هي تعليمة بلغة الآله ومعناها أنقل الرقم ٥ الى المسجل ah يقابلها بلغة الأسمبلي ah mov , لا ٨ بت الأولى من التعليمة تشكل **شفرة التعليمة** operation code=OP-code وهي تعني نقل قيمة بطول ٨ بت الى المسجل AL ، لجزء الثاني من لا ١٦ بت الأخرى تشكل الرقم ٥ ثنائياً

## كيان الحاسوب الصلب :

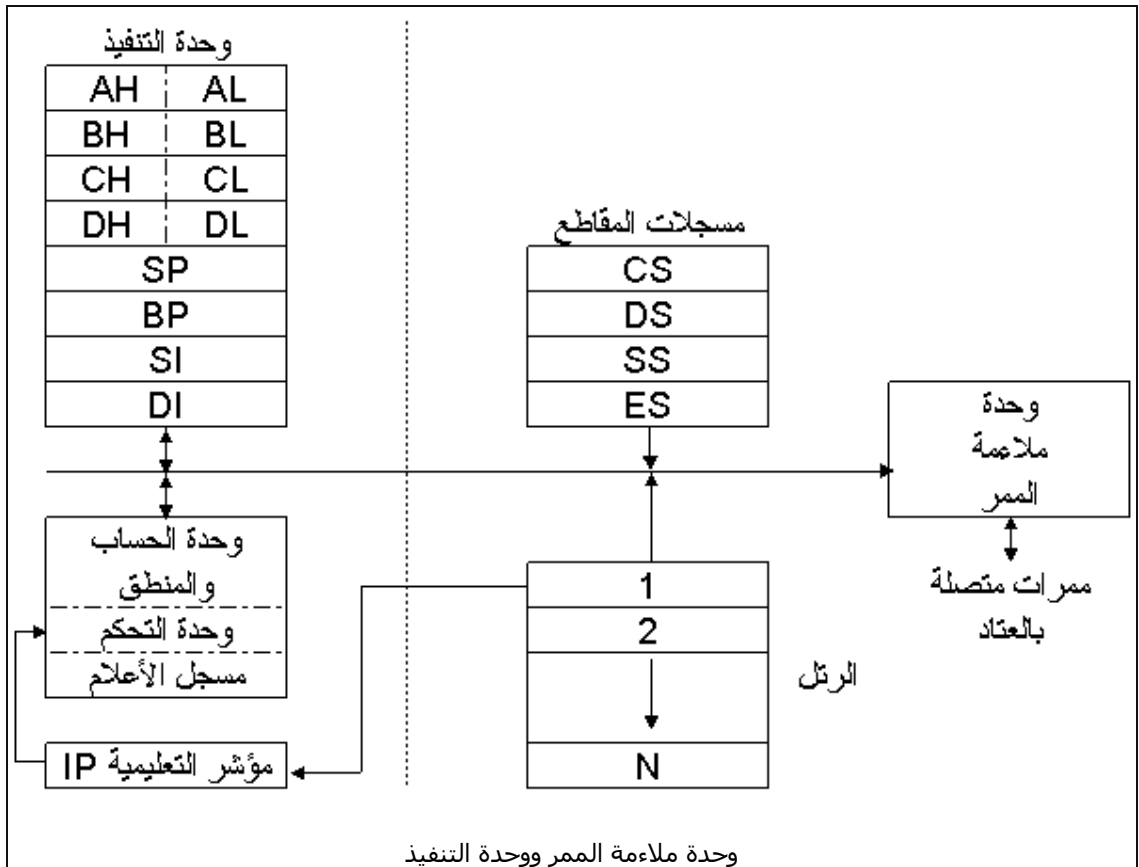
يتألف الحاسوب بشكل أساسي من اللوحة الأم Mother Board والمعالج Microprocessor وذاكرة القراءة-فقط Memory ROM=Read-Only وذاكرة الوصل العشوائي=الرام RAM=Random-Access Memory ووحدة التغذية Supply Power والمنافذ التوسعية Expansion Slots

## المعالج :

يمثل المعالج عقل الحاسوب وهي الوحدة المسؤله عن القيام بأدارة الحاسوب والقيام بالعمليات الرياضية والمنطقية ونحن هنا كما أوضحت ندرس معالجات إنتل من العائلة ٨٦X 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

أختصاراً CU ومجموعة من المسجلات. تتألف وحدة ملاءمة الممر من وحدة التحكم بالممر Unit Bus Control ومسجلات المقاطع Segment Registers ورتل=كيو التعليمات Instruction Queue (الرتل أو الكيو هو نوع من إدارة الذاكرة تكون فيه المعلومة الداخلة أولاً خارجة 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 عنونة الذاكرة. تضع وحدة ملاءمة الممر تضع التعليمات في رتلها المخصص لها في وحدة التنفيذ بعد أن تقوم بجليها من الذاكرة. يخصص رتل التعليمات لوضع التعليمات فيه بعد جليها 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 حوث عمليتي الجلب fetching التنفيذ execution في وقت واحد الأمر الذي يزيد بدوره 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 - شوائجلا ROM الحديثة يمكن إعادة الكتابة عليها بطرق مختلفة) ،تحتوي هذه الذاكرة على برنامجلا BIOS=Basic Input Output

System أو نظام الإدخال والأخراج الأساسي، ولا يمكن للمعالج القراءة من هذه الذاكرة مباشرة ولكن أو لا شيء يفعل المعالج عند تشغيله في عملية الأستنهاض هي تحميل البيانات الموجودة في الـ ROM ونقلها في الـ RAM أو بالأحرى إلى القسم الأخير من الـ RAM ذات العنوان الأكبر. تجلى فائدة BIOS في القيام بعملية الفحص الذاتي عند الأستنهاض POST=Power On Self Test بالإضافة إلى تحميل برنامج محمل نظام التشغيل بالإضافة إلى توفير دوائر ومقاطع قياسية في ذاكرة الـ ROM مع أن تستخدمها البرامج للرسم على الشاشة مثلاً أو التعامل مع لوحة المفاتيح أو القراءة والكتابة من وإلى القرص الصلب.

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

هذه الذاكرة مهمة جداً حيث أنها لا يمكن أن يعمل إلا إذا حملت إلى هذه الذاكرة كذلك فهي تستخدم لحفظ المتغيرات وحفظ برامج النظام الأساسية ومنها جداول المقاطعات والمقاطع نفسها والروتينات الفورية..... الخ ولا يتم استخدام القرص الصلب لحفظ مثل هذه المعلومات لأن وقت الوصول فيه أطول بكثير من ذاكرة الـ RAM (ولأن القرص الصلب يستخدم في توفير ذاكرة افتراضية عن طريق القيام بعمليات مبادلة للصفحات مع ذاكرة الـ RAM)، وبالطبع فإن هذه الذاكرة يمكن الكتابة إليها أو القراءة منها عن طريق عناوينها. فباستخدام العنوان يمكننا أن نصل إلى مكان محدد في الذاكرة لنعمل عليه كل عمليات التحرير المطلوبة.

قد تتساءل ما علاقة تمثيل البيانات والعد الثنائي بالأسميلي؟ حسناً كما وضحت من قبل فإن الاسميلي هي لغة قريبة جداً من لغة الآلة 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 كتواليات من الشحنات الكهربائية تأخذ قيمتين الأولى وهي وجود الشحنة ويرمز لها بـ ON أو صحيح TRUE أو '1' والأخرى وهي غياب الشحنة ويرمز لها بـ OFF أو خطأ FALSE أو '0'، ووجود الشحنة يكون عادة بين ٤,٥ إلى ٥,٥ فولت (المعالجات الحديثة بين ٢,٥ إلى ٣,٥ فولت) وغياب الشحنة يكون بين ٠,٥+ فولت و ٠,٥- فولت.

وحدة الذاكرة الأساسية في الذاكرة والوحدة التي سنتعامل معها كثيراً هي :

اسم الوحدة	الأحتمالات	ما يعادلها
البت	أحتمالين 0 و 1	1 بت
البايت	256 احتمال	8 بت
الكلمة	65536 احتمال	16 بت = 2 بايت
الكلمة المضاعفة	4,294,967,296 احتمال	32 بت = 4 بايت = 2 كلمة
الكلمة الرابعة	18,446,744,073,709,551,616 احتمال	64 بت = 8 بايت = 4 كلمات = 2 كلمة مضاعفة

بالطبع هذه الوحدة الأساسية والصغيرة أما مضاعفاتها فهي :

الوحدة	الرمز	مايعادلها
الكيلوبايت	KB	١٠٢٤ بايت
الميغابايت	MB	١٠٢٤ كيلوبايت
الجيجابايت	GB	١٠٢٤ ميغابايت
التيرابايت	TB	١٠٢٤ جيجابايت
البيتابايت	PB	١٠٢٤ تيرابايت
الأكسابايت	EB	١٠٢٤ بيتابايت
الزيتابايت	ZB	١٠٢٤ أكسابايت
اليوتابايت	YB	١٠٢٤ يوتابايت

تخزين الأرقام بدون إشارة يأخذ النطاق الآتي لكل وحدة من الوحدة الأساسية باستخدام طريقة بدون الإشارة unsigned:

الوحدة	المجال	
	من	الى
البايت	.	٢٥٦
الكلمة	.	٦٥٥٣٦
الكلمة المضاعفة	.	٤,٢٩٤,٩٦٧,٢٩٦
الكلمة الرباعية	.	١٨,٤٤٦,٧٤٤,٠٧٢,٧٠٩,٥٥١,٦١٦

### الأسكي كود ASCII:

يتم في الحاسوب وبقيّة توحيد استخدام الرموز استخدام شفرة الأسكي كود (حالياً) يعمل على تبني شفرة unicode وهي تسمح بتعدد اللغات في مستند واحد حيث يتم تمثيل كل حرف باستخدام كلمة واحدة=٢بايت) كلمة ASCII هي اختصار لـ :  
 Interchange Information Code For Standard American National  
 ويتم استخدام هذا الكود الموحد لتسهيل تناقل البيانات ويمثل كل رمز فيه بعدد ثنائي بطول ١ بايت=٨بت=٢٥٦ احتمال .  
اضغط هنا لمشاهدة الجدول الأسكي

### طريقة كتابة الأرقام في الأسمبلر :

لكتابه عدد ثنائي يوضع في آخر الرمز (b) لدلالة على أنه باينري مثال :  
 Binary=١٠١٠١٠١٠١ B أما العدد العشري فلايحتاج الى إضافة وأما العدد  
 لأساس ٨ فيكتب مع المرمز (Q) في نهايته =Octal١٢٧٦Q أو الرمز (O) في  
 نهايته =Octal١٢٧٦O أما العدد السداسي عشر فيكتب بوضع H في نهايته  
 =hexadecimal٠CDH٩AB مع مراعاة وضع ٠ اذا كان العدد يبدأ بحرف كما  
 المثال .

يجب أن تعرف الفرق بين تخزين الرقم كرقم أو تخزينه كنص فتخزين الرقم  
 ٢٠١ مثلاً كرقم سسياخذ بايت واحد وهو جاهز للقيام بعمليات رياضية  
 ومنطقيه عليه أما تخزينه كنص فسياخذ ثلاثة بايت في البايت الأول سيزخن

الرقم الخاص بالأسكي كود للرمز '٣' وكما قلت يخزن كرقم يدل على الرمز أما البايث الثاني فسيخزن رقم الأسكي كود للرمز '٠' أما البايث الثالث فيأخذ القيمة الخاصة بالرمز '١' في الأسكي كوداً يرمز بالرقم ١٠٣ بطريقة "١٠٣" وليس ١٠٢ وهذه الطريقة ليست جاهز للجمع أو الطرح ولكنها ممتاز للطباعة على الشاشة ويمكن تحويل النص الى رقم والعكس .

### الأعداد و الأشارات :

يتم تخزين الأعداد و الأشارات كالتالي :  
العدد موجب إذا كانت البت الأخيرة صفر وقيمة الرقم هي باقي البتات أ ي لو أخذنا رقماً من بايث واحد فإن البت رقم ٧ (الثامن - الترقيم يبدأ من الصفر) يجب أن تكون صفراً ليكون العدد موجب أما البتات من ٠ الى ٦ ( السبعة الأولى ) فتشكل قيمة الرقم  
أما إذا كان العدد سالب فإن البت الأخيرة تساو و واحد أما قيمة الرقم فتساو و سالب المكمل الثنائي للعدد أ ي لو أخذنا رقم مخزن في واحد بايث مثال = ١١١٠١١٠ بما أن البت السابع = ١ فإن الرقم سالب / نأخذ الآن المكمل الثنائي للعدد وهو ٠٠٠٠١٠١٠ / القيمة تساو و -٠٠٠٠١٠١٠ أ ي سالب عشرة .

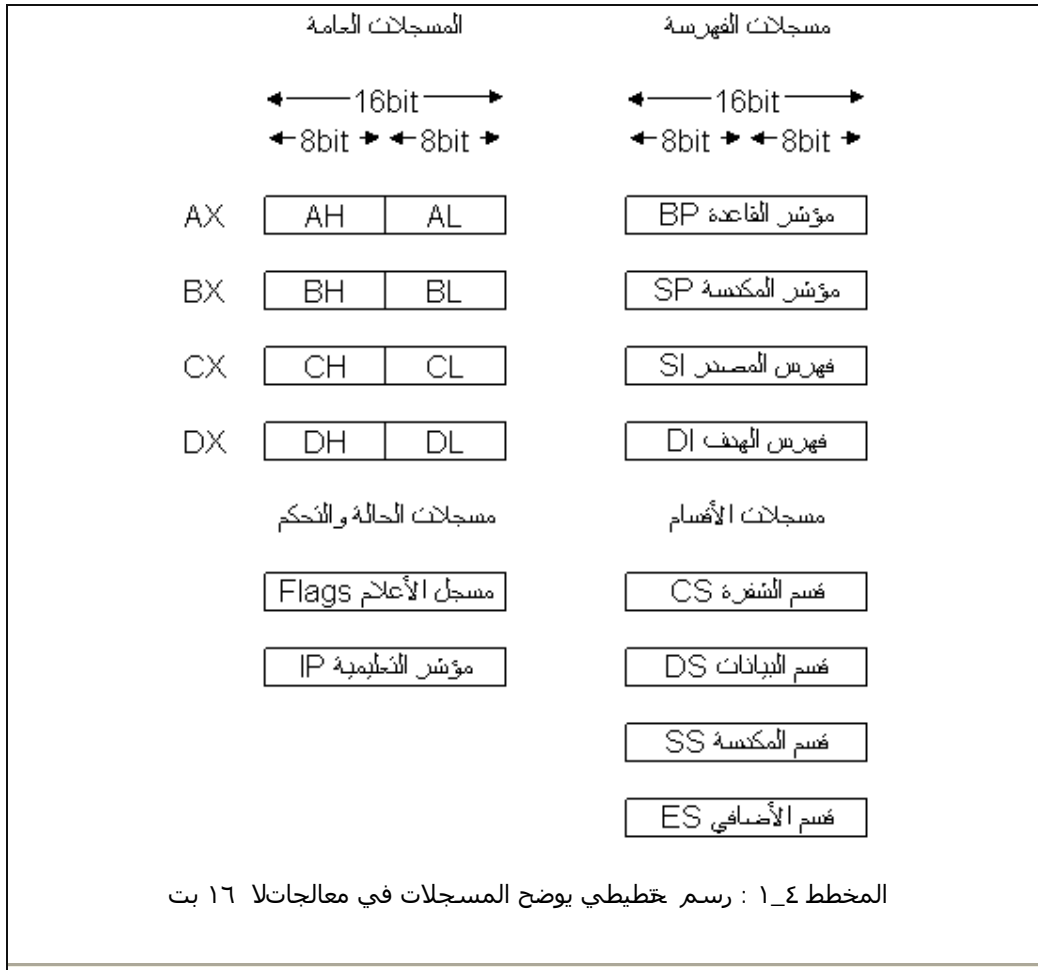
ويتخزين الرقم بالأشارة يختلف النطاق الذي تأخذ كل وحده :

المجال		أسم الوحدة
من	الى	
١٢٨-	١٢٧+	البايث
٣٢,٧٦٨-	٣٢,٧٦٧+	الكلمة
٢,١٤٧,٤٨٣,٦٤٨-	٢,١٤٧,٤٨٣,٦٤٧+	الكلمة المضاعفة
٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠٨-	٩,٢٢٣,٣٧٢,٠٣٦,٨٥٤,٧٧٥,٨٠٧+	الكلمة الرباعية

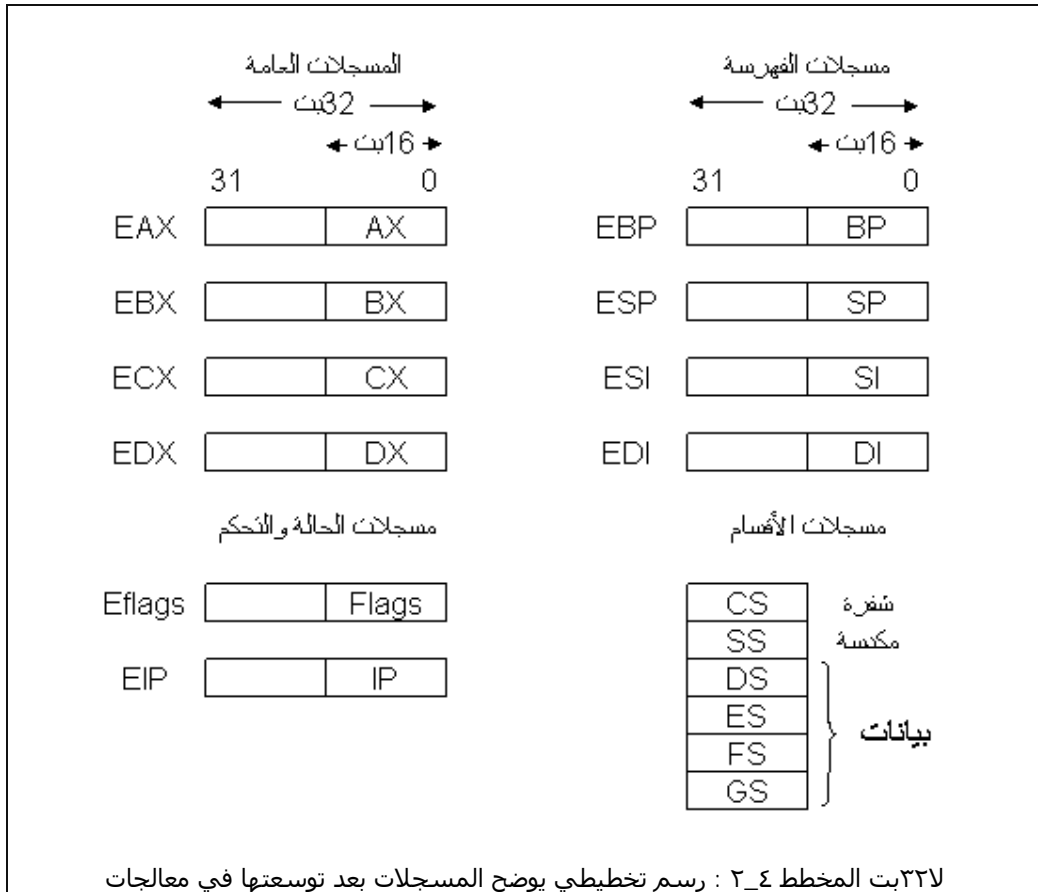
## المسجلات REGISTERS :

الكمبيوتر يحتاج في تعاملاته الى ذاكرة سريعة جداً و متصلة 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 Registers Data ) وهناك مسجلات الأقسام Segment Registers والمسجلات الدليلية Registers Index ومسجلات التأشير Pointer Registers بالإضافة الى مسجلات الحالة والتحكم Status and Control Registers.







### [١] مسجلات الأغراض العامة - General-Purpose Registers :

وهي عبارة عن كل من المسجلات AX و BX و CX و DX ، طول كل منها ١٦ بت أي كلمة ٢ بايت اليسار ي فيها يعرف بالعلوي (High) أما الباي اليمين فيها فيعرف بالمنخفض (Low) فمثلاً المسجل AX يتألف من مسجلين العلوي وهو AH والمنخفض وهو AL ، أن تعديل المسجل الجزئي سوف يؤثر في المسجل الأمل لأنه جزء منها وأيضاً التعديل في المسجل الأمل سوف يؤثر في المسجل الجزء ، تم توسيع المسجلات في المعالجات لا ٣٢ بت مع بقاء المسجلات نفسها ولكن كل منها أصبح جزء من مسجل موسع بطول ٣٢ بت وهي EAX,EBX,ECX,EDX ، أي أن المسجل EAX هو بطول ٣٢ بت وكجزء منه هناك المسجل AX بطول ١٦ بت والذي يتألف هو الآخر من مسجلين هما AL و AH بطول ٨ بت لكل منهما.

### المسجل AX (مسجل المراكم - Register Accumulator) :

هذا المسجل كان من أهم المسجلات في معالجات ال٨ بت القديمة جداً حيث كانت كل العمليات الرياضية والمنطقية تجري من خلاله ولذلك كان يسمى بمسجل المراكم لتراكم النواتج فيه لكن معالجات ال١٦ بت وسعت المرونة وجعلت كل مسجلات الأغراض العامة تستطيع أن تجري من خلالها العمليات الرياضية والمنطقية إلا أن المسجل AX مازال المفضل لأجرائها حيث أن استخدام المسجل AX أو أحد أجزاءه يؤدي مع بعض التعليمات إلى توليد شفره أقل اختصاراً (الفرق بايت واحد فقط لكل تعليمة) ، يمكن استخدام المسجل AX كمسجلين هما AL و AH حيث تعرف ال٨ بتات الأولى التي في اليسار بالمسجل AL لوال٨ بتات الأخيرة التي في اليمين بالمسجل AH ، أما بالنسبة ل EAX وهو المسجل الموسع ل AX فهو بطول ٣٢ بت ويعتبر

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 ٨ بت هما BL و BH وهو ضمن مسجل أوسع هو EBX بطول ٣٢ بت.

### 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 ٨ بت هما CL و CH وهو ضمن مسجل أوسع هو ECX بطول ٣٢ بت.

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

يفضل استخدام هذا المسجل لتخزين المعطيات في عمليات الدخل والخرج والمقاطعات وبالطبع فإنه يمكن استخدامه كباقي المسجلات في العمليات الرياضية والمنطقية وكما المسجلات الأخرى ينقسم هذا المسجل الى قسمين بطول ٨ بت هما DL و DH وهو ضمن مسجل أوسع هو EDX بطول ٣٢ بت.

## [٢] مسجلات الأقسام Segment Registers :

كانت العنوان الحقيقية في معالجات ال ١٦ بت تتم باستخدام خطوط عرض ٢٠ بت وهي تكفي لعنوان ١ ميجابايت من الرام فقط ولصعوبة التأشير للرام باستخدام مسجلات من ١٦ بت نشأت فكرة الأقسام والعنوان المنطقية وقد قسمت الرام لمقاطع كل منها بطول ٦٤ كيلوبايت (الحد الأقصى الذي يمكن عونه ب ١٦ بت) وهذه الأقسام لا يبدأ كل واحد فيها بعد الآخر وإنما هي متداخلة حيث يبدأ كل ١٦ بت قسم جديد وللتأشير على موقع ما يلزمنا عنوان المقطع والذي بطول ١٦ بت بالإضافة الى قيمة الأزرحة من بداية هذا المقطع وهي بطول ١٦ بت أيضاً لذلك لجأ مضموا المعالج على وضع مسجلات خاصة بالأقسام الشائعة في البرنامج وهي قسم الشفرة Code Segment وقسم البيانات Data segment وقسم المكده Stack Segment وقسم المقطع الإضافي Extra Segment والرغم أنه في معالجات ال ٣٢ بت يمكن العنوان باستخدام ٣٢ بت ذاكرة حقيقية أ ي مايساو ي ٤ جيجابايت من الرام إلا أن طريقة الأقسام مازالت موجودة حتى يتم خزن عناوين كثيرة باستخدام ٢ بايت بدل ٤ بايت داخل المقطع الواحد مع وجود عنوان المقطع واحد فقط مخزن في المسجل المناسب (ملاحظة : تم في معالجات ال ٣٢ بت إضافة مسجلين أقسام جديدين بطول ١٦ بت إضافة الى مسجلات الأقسام السابقة والمسجلين هما FS و GS هنا ن 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 المكده -CS Stack Segment :

يحمل هذا المسجل عنوان بداية قسم المكده في البرنامج .

**قسم الإضافي - Extra Segment - ES :**  
يحمل هذا المسجل عنواناً ز بداية قسم إضافي يمكن أن يستعمل هذا القسم الإضافي كقسم بيانات آخر.

**[٣] مسجلات التأشير Pointer Registers :**  
تحتوي مسجلات التأشير وهي بطول ١٦ بت على عنواناً ز من ١٦ بت وهي تستخدم بشكل خاص مع العمليات الخاصة بالمكدسه وعادة تشكل العناوين التي بها الأزاحة بالنسبة لمسجل قسم المكدسه SS ومسجلات التأشير هي مسجلان مؤشر القاعدة Base Pointer BP و مسجل مؤشر المكدسه Pointer SP Stack .

**مسجل مؤشر القاعدة -BP - Base Pointer :**  
يعمل هذا المسجل على تسهيل الوصول الى الوسيطات (البارمترا ت) والتي تحتوي على عناوين ومعطيات والتي دفعت PUSH بشكل مؤقت الى المكدسه عند استدعاء روتينات فرعية من البرامج مع وسيطات ممرة ، وسع هذا المسجل في معالجات ٣٢٦ بت وأصبح جزء من مسجل أوسع بطول ٣٢٦ بت هو EBP.

**مسجل مؤشر المكدسه -SP - Stack Pointer :**  
يحتوي المسجل SP على كلمة الذاكرة الحالية التي ستعالج في المكدس ، وسع هذا المسجل في معالجات ٣٢٦ بت ليصبح جزء من مسجل أوسع بطول ٣٢٦ بت هو ESP . هذا المسجل يعدل آلياً بواسطة المعالج مع عملية دفع PUSH أو سحب POP في المكدس ليشير دوماً الى قمة المكدس .

**[٤] المسجلات الدليلية Index Registers :**  
هي مسجلات بطول ١٦ بت تستخدم في عنوانة بيئات مقطع البيانات وكذلك في عمليات التأشير الى السلاسل النصية Strings ، وهناك مسجلات دليلية هما SI و DI وعادة ما يستخدمان معاً دائماً بغيه تنفيذ عملية ما .

**المسجل الدليلي المصدر -SI - Source Index :**  
يستخدم هذا المسجل في التأشير على النص المصدر وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في عنوانة بيئات مقطع البيانات ، وسع هذا المسجل في معالجات ٣٢٦ بت ليصبح جزء من مسجل أوسع بطول ٣٢٦ بت هو ESI .

**المسجل الدليلي الهدي -DI - Destination Index :**  
يستخدم هذا المسجل في التأشير على النص الهدف وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في عنوانة بيئات مقطع البيانات ، وسع هذا المسجل في معالجات ٣٢٦ بت ليصبح جزء من مسجل أوسع بطول ٣٢٦ بت هو EDI .

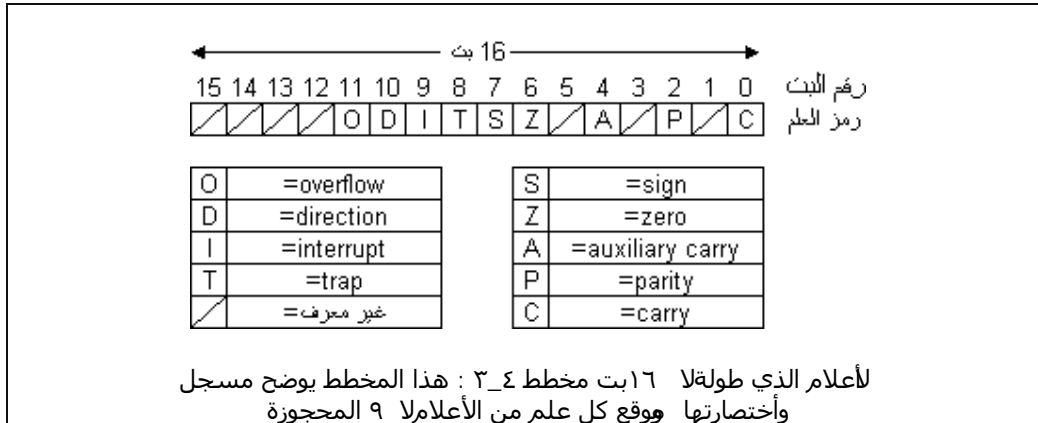
**[٥] مسجلات الحالة والتحكم Control Registers Status and :**  
تتكون هذه المسجلات من مسجلين كل بطول ١٦ بت هما مسجل الأعلام Flags Register ومسجل مؤشر التعليمية IP - Instruction Pointer .

**مسجل مؤشر التعليمية - IP - Instruction Pointer :**  
يحتوي المسجل IP على أزاحة التعليمية التالية التي ستنفذ ، أياً ز المسجل عبارة عن مؤشر الى التعليمية التالية الموجودة في مقطع الشفرة CS-Code Segment لمنفذ حالياً ، وسع هذا المسجل في معالجات ٣٢٦ بت ليصبح جزء من مسجل أوسع بطول ٣٢٦ بت هو EIP.

وما تعليمات القفز والتكرار الى تعديل للمسجل IP .

### مسجل الأعلام Flags Register :

وهو مسجل بطول ١٦ بت يحتوي على أعلام طول كل منها ١ بت فقط وتستخدم لعكس حالة آخر عملية أو للتحكم بعمليات معينة وكل علم له موقع معين فيلا ١٦ بت علماً بأن ٩ 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 واحداً ي SET أو صفراً ي CLEAR .



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 بالمقاطع أو استدعاء البرامج الفرعية أو بعض الأوامر بهدف التحكم بشئ ما وهذه الأعلام هي :

#### ١. علم الاتجاه (DF=direction) :

يؤثر في التعليمات التي تقوم بنقل البيانات مثل MOV,CMPS,SCAS. عندما يكون العلم UP=١ يأخذ الانتقال اتجاه الطبيعي أما عندما يكون DOWN=٠ يأخذ انتقال البيانات اتجاه معاكساً ( قيمة العلم DF عند بداية البرنامج =١). لضبط العلم بواحد نستخدم التعليمات std ولضبطه بvفر cld .

#### ٢. علم المقاطعة (if=intrreupt 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 حدو مقاطعات خارجية أو لا ، إذا كانت قيمة العلم=١ فهذا يعني مفعّل enable ويمكن إجراء المقاطعات أما إذا كان صفر فإنه غير-مفعّل disabled ولايمكن إجراء المقاطعات ( قيمة العلم IF عند بداية البرنامج =١). لضبط العلم بواحد نستخدم sti ضبطه بvفر CLI .

### ٢. علم المصيدة (tf=trap flag) :

يتيح هذا العلم وضع المعالج في نمط الخطوة الواحدة في الوقت الواحد ( single step mode) مما يسمح لبرامج فحص الأخطاء كالديبجر بتتبع البرنامج ، إذا كانت قيمة العلم واحد=ON فإن النمط يعمل أما إذا كان صفر=off فإن النمط لايعمل ( قيمة العلم TF عند بداية البرنامج = ٠ ) .

### #أعلام الحالة Status Flags :

هذه الأعلام تضبط ألياً 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 :

### ١. علم الحمل (flag cf=carry) :

يضبط هذا العلم=١ إذا كان نتيجة آخر عملية كبيرة جداً على الهدف أو المقصد (في الأعداد التي بدو ن إشارة فقط ) ، مثال هذا البرنامج :

```
mov ah,٢٠٠  
add ah,١٠٠
```

بما أن المسجل AH هو ٨بت فإن أقصى قيمة يتحملها هي ٢٥٦ وبما أن القيمة في المسجل هي ٢٠٠ ثم أضفنا لها ١٠٠ فإن الجواب أكبر من الهدف (ah هنا) لذلك العلم CF سوف يضبط=١ بعد عملية الجمع  
ضبط لعلم برمجياً stc ولتصفيره clc

### ٢. علم الفيض (OF=overflow flag) :

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

```
mov ah,-١٠٠  
add ah,-٥٠
```

بما أن أصغر قيمة يتحملها المسجل AH هي -١٢٨ لكن ناتج العملية -١٥٠ فإن علم الفيض يضبط=١

### ٣. 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 ١ وموجب إذا صفر ) .

### ٤. علم الصفر (zf=zero flag) :

يضبط هذا العلم=١ إذا كانت نتيجة آخر عملية رياضية أو منطقية تساوي صفر .

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

يضبط العلم=١ إذا تسببت آخر عملية رياضية أو منطقية حمل من البت الثالثة الى البت الرابعة أو استلاف من البت الرابع الى البت الثالثه . هذا العلم لا توجد له فائدة واضحة وهو قليل الاستخدام برمجياً .

### ٦. علم الأزواجية - التحقق (pf=parity flag) :

ببساطة يضبط = 1 هذا العلم اذا كان عدد الواحد في ناتج آخر عملية رياضية أو منطقية زوجياً وبصفر اذا كان فردياً ، مثال لو كان جوا ب آخر عملية = 10100100 فان العلم سوف يضبط = 1 لأن عدد البتات التي تحتوي وحيد = 3 وهو عدد زوجي أما اذا كان الجوا ب مثلاً = 11100000 فان العلم يصفر لأن عدد البتات التي تحوي وحيد = 3 وهو عدد فرد . وكما علم الحمل المساعد AF فان أستعماله قليل برمجياً ويستخدم عادة من قبل نظام التشغيل لأدارة الذاكرة وكذلك برامج الأتصال لتحقق من سلامة البيانات المرسله .

لاحظ أن كل من الأعلام [علم المصيدة (flag tf=trap) /علم الفيض ( OF=overflow /علم الإشارة (sf=sign flag) /علم الصفر (flag zf=zero) /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 مااستحتاجه هو قراءة القيم التي بها .

### تعليمات وأوامر الأسمبلي :

تتكون التعليمه الواحدة في الأسمبلي من تمثيل بسيط بالأحرف الأنجليزيه يقابله بالأرقام تعليمه لغة آله ، تتكون كل تعليمه من مهايي : أولاً جزء الأمر وهو أمر يدل على نوع العمليه المطلوبه مثل ADD (لجمع) ، الجزء الثاني هو الوسائط علماً بأن بعض التعليمات لاأخذ وسائط والجزء الآخر وسيطه واحده فقط والبعض الآخر أكثر من ذلك ، تحدد هذه الوسائط الشئ الذي سيعمل عليه الأمر ، فالأمر ADD لوحد عقيم لايدل على شئ لكن الأمر ADD AX ه يدل على جمع الرقم ه مع القيمة الموجوده في المسجل AX ويوضح المثال التالي بعض الأوامر

```
clc      ; وسائط فقط أمر بدو ن
dec ax   ; وسيطه واحده فقط
mov cx,dx ; وسيطتين
```

لاحظ أن نأ ي نص في شفرة الأسمبلي يأتي بعد الفاصله المنقوطة هو مجرد تعليق

### الوسائط ممكن تكون عدة أنواع :

1. معلومه فوريه (مباشرة) (أ ي ثابتة) مثال : 10 / 30 / 'a'
2. مسجل (سوف يتم شرح المسجلات بالتفصيل في الدروس القادمه ) مثال : AX / EAX / BL
3. موقع ذاكره (يتم تحديده عن طريق العنوا ن) مثال : [200] / [] / [100bx]
4. متغير (وهو نفس السابق لكن بدل أن تحفظ أو تحسب العنوا ن يدوياً يقوم الأسمبلي ب استبدال المتغير برقم يدل على عنوا له ) مثال : count / VAR / INTVAL / 1

### مدخل الى الديوغر Debugge :

ها قد وصلنا الى واحد من أقوى البرامج المبيتة في النظام فيواسطة البيغ تستطيع عمل أشياء عجيبه وغريبه ، حسناً شغل الدوس وعند محث الأوامر أطيح debug ثم أنتر وستظهر لك علامه '- ' ليل على أستعداد الديوغر على أستقبال أوامرك .



الآن دعنا نكتب هذا البرنامج الصغير

```
mov ax,2 ; مباشرة الى المسجل أ -أكس نقل العدد ٢ كـمـلـومـة
mov bx,3 ; المسجل بي-اكس نقل العدد ٣ كـمـلـومـة مباشرة الى
add ax,bx ; أ -أكس / أ -أكس + بي- جمع أ -أكس مع بي-اكس مع وضع الجواب في
أكس
```

كيف تقوم بأدخال هذا الكود :

١. عند المحث '-' أدخل a100 أ ي أننا سنبدأ نكتب الكود من العنوان ز ١٠٠ ثم اضغط أنتر بالطبع
٢. الآن أدخل كل تعليميه ثم اضغط أنتر ومع نهاية التعليمية الأخيرة اضغط أنتر مرتين

```
-a100
1896:0100 mov ax,2
1896:0103 mov bx,3
1896:0106 add ax,bx
1896:0108
-r
AX=0000 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1896 ES=1896 SS=1896 CS=1896 IP=0100 NV UP EI PL NZ NA PO NC
1896:0100 B80200 MOV AX,0002
-t
AX=0002 BX=0000 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1896 ES=1896 SS=1896 CS=1896 IP=0103 NV UP EI PL NZ NA PO NC
1896:0103 BB0300 MOV BX,0003
-t
AX=0002 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1896 ES=1896 SS=1896 CS=1896 IP=0106 NV UP EI PL NZ NA PO NC
1896:0106 0108 ADD AX,BX
-t
AX=0005 BX=0003 CX=0000 DX=0000 SP=FFEE BP=0000 SI=0000 DI=0000
DS=1896 ES=1896 SS=1896 CS=1896 IP=0108 NV UP EI PL NZ NA PE NC
1896:0108 012ECD21 ADD [21CD],BP DS:21CD=D823
-
```

الآن قم بأدخال الرمز R ثم أنتر لترى حالة المسجلات  
لاحظ أن المسجل AX يساوي صفر وسترى أيضاً ظهور التعليميه MOV ax,0002 وهي  
التعليميه التي عليها الدور في التنفيذ وليس المعلومة المنفذه ، الآن قم بطباعة الرمز  
T ثم أنتر لتنفيذ التعليميه التي عليها الدور هنا هي MOV AX,0002 ستري الآن أن  
المسجل AX أصبح يساوي ٢ وهذا ما نتوقعه بالضبط وسترى أيضاً التعليميع التي عليها  
دور التنفيذ وهي MOV BX,0002 أدخل الرمز T ثم أنتر لتنفيذها لترى أن المسجل BX  
أصبح يساوي ٣ وسترى أيضاً التعليميه التي عليها الدور في التنفيذ وهي ADD AX,BX  
قم بأدخال الرمز T لتنفيذها ولاحظ كيف أن المسجل AX أصبحت قيمته مجموع العددين  
٢+٣ وهو خمسة بينما بقي المسجل BX يساوي ٣ .

الآن بعدما عرفت كيف تكتب كود بسيط أخرج من الديبجر بالضغط على Q ثم أدخل مرة

أخرى بكتابة الأمر Debug حتى تصفر المسجلات مرة أخرى أدخل التعليميه ١٠٠A ثم  
 جرب تكتب كود من عندك ومع كل نهاية تعليميه أضغط أنتر وفي نهاية التعليميه الأخيرة  
 أضغط أنتر مرتين  
 ( ملاحظة لترى شفرتك بلغة الآله والأسمبلي أدخل الرمز لا ثم أنتر مباشرة بعد إدخال  
 الكود وقبل إدخال الرمز R )  
 أضغط R ثم أنتر لترى المسجلات قبل تنفيذ أ ب عملية ولترى التعليميه التي عليها  
 الدور في التنفيذ أضغط T ثم أنتر لتنفذ التعليميه وترى النتائج والتعليميه التي بعدها  
 وهكذا ولاتنسى إذا أردت أن تدخل كود جديد الخروج والعودة مرة أخرى الى الديوغر  
 لتصفر المسجلات والذاكره

### (العمليات الحسابيه المتقدمه و تنقسم الى ضرب و قسمه :-

#### الضرب :- Multiplication

يتم ضرب أى رقمين فى لغة الأسمبلى فى ثلاث أنظم ألى الآن :-  
**• الأول : لا ٨-بت (البايت :-)**

يتم وضع إحدى قيم الضرب فى AL و الآخر فى أى ريجستر آخر له نفس الحجم أو مكان  
 فى الذاكره بجانب كود الضرب الذى هو فى حالتنا هنا [MUL] للأرقام الغير محددة الإشاره  
 . و [IMUL] للأرقام المحددة الإشاره . ووضع النتيجة فى لا . AX  
 مثال : للحصول على حاصل ضرب ١٢٠ \* ٣٠ نقوم بالآتى :-

```
MOV CL,٣٠D
MOV AL,١٢٠D
MUL CL
```

ب ١٠٠٠ بالنظام السلاسد لتخزين الناتج فى الذاكره المعنونه ; AX [١٠٠٠H] عشر

#### **• الثانى : لا ١٦-بت (الكلمه :-)**

يتم وضع إحدى القيم فى AX و الآخر فى أى ريجستر آخر له نفس الحجم أو مكان فى  
 الذاكره بجانب كود الضرب و موضع النتيجة فى . DX:AX وهذا معناه أن الجزء لا Low من  
 ناتج القسمه سيخزن فى لا AX و لجزءلا High سيوضع فى لا . DX  
 مثال : للحصول على حاصل ضرب ٢٠٠٠٠ \* ٢٣٦٥ سنقوم بالآتى:-

```
MOV CX,٢٣٦٥D
MOV AX,٢٠٠٠٠D MUL CX
```

المكان فى الذاكره المعنونه ب ١٠٠٠ تخزين الجزء الأول فى ; AX [١٠٠٠H]  
 المكان فى الذاكره الذى يلى المكان تخزين الجزء الثانى فى ; DX [١٠٠١H]  
 السابق

#### **• الثالث : لا ٣٢ بت :-**

يتم وضع إحدى القيم فى EAX و الآخر فى أى ريجستر آخر له نفس الحجم أو مكان فى  
 الذاكره بجانب كود الضرب و موضع النتيجة (٦٤ بت) فى . EDX:EAX وهذا معناه أن الجزء لا  
 Low . ناتج القسمه سيخزن فى لا EAX والجزءلا High سيوضع فى لا . EDX



## القسمه :- Division

أيضاً تتم فى نفس النظم السابقه :-

**• الأول :** لا ٨ بت (بايت :- )

يتم وضع المقسوم فى AX و المقسوم عليه فى أى ريجستر حجمه بايت مثل  
BL,CL,DL,... أو مكان فى الذاكرة لها حجم بايت مثل BYTE PTR [١٢٣٤H] ,  
[BX] و يوضع ناتج القسمه فى AL و باقى القسمه فى AH باستخدام [DIV]  
لاحظ :- أنه فى حالة قسمة الأرقام المحددة بالإشاره باستخدام [IDIV] تكون إشارة ناتج  
القسمه هى الإشاره العاديه فى هذه الحالات . وتكون إشارة خارج القسمه Remender  
دائماً موجبه و صحيحه Integer.

لاحظ :- كل القيم هنا ٨-بت إذاً , لابد من تحويل القيمه لا ٨-بت للمقسوم إلى ١٦-بت  
ليمكن نقلها إلى AX ويتم ذلك فى حالة القيم غير محددة بالإشاره مسجلا AH ليكون  
كله أصفار و تكون قيمة المقسوم ١٦-بت . أما فى حالة القيم محددة بالإشاره يتم ذلك عن  
طريق كود [CBW] المسئول عن تحويل البايت (٨-بت) إلى كلمه (١٦-بت) لل AX فقط.  
مثال :- للحصول على ناتج قسمة (١٦+) على (٥-) يمكننا عمل الأتى :-

```
MOV AL,١٦D  
CBW  
MOV BL,٥D  
NEG BL  
IDIV BL  
MOV PYTE PTR[١٠٠٠H] , AL  
MOV PYTE PTR[٢٠٠٠H] , AH
```

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

**باقى النظم كما سبق من الممكن أن تقوم بإستنتاجها.**

## (٢) التحكم فى مسار البرنامج :-

ينقسم التحكم فى مسار البرنامج إلى عدة طرق و تقنيات و تكون مثل مثيلاتها فى اللغات  
العليه و لكن أكثر تفصيلاً . وستتعلم سوياً بعض من هذه التقنيات :-

**• أولاً : جمل الشرط :- IF Statements**

إننى أعتبر جمل الشرط من قواعد البرمجه عموماً و من أهم قواعد التحكم فى مسار  
البرامج خصوصاً.

تقوم جمل الشرط عموماً على وجود شرط أو إختيار لو تحقق إذاً سيقوم البرنامج بعمل  
بعض المهام لن يكون فاعلها لولم يتحقق الشرط أو فشل الإختيار , كما تحب أن تطلق  
عليه.

إذاً , يجب علينا معرفة كيفية إختيار قيمه (والتى يترتب عليها الشرط فيما بعد. )

**الإختيار :-**

يكون بإستخدام دالتين أو كودين و هما [CMP] و [TEST]

:- [CMP] - يجب أن تذكر القيمتين اللتان سوف يتم لهما إختيار بعدها . و عندما تتم هذه

الجملة يتم عمل عملية طرح فى الأصل و لكن مع عدم تغيير فى أو ل مكان (الرجستر أو الذاكرة). (Distination) لذلك التغيير يظهر فى ريجستر الأعلام (FLAG REGISTER) فقط و يكون القفز (JUMP) بعد هذا الإختبار شىء طبيعى على أساس إختبار بتات هذا الرجستر الخاص. أو أى عبارته تأتى مع القفز كما تم ذكرهم فى الدرس السابق.

مثال :- سنقوم إختبار تساوى لقيمتين و ليكن ٥ و ٥ مثلاً . لو تحقق الشرط سيقفز البرنامج إلى النهايه و لن يكمل

```
-----  
MOV AX,٥  
MOV BX,٤  
CMP AX,BX      ;=AX-BX  
JE END
```

.....  
سوف تنفذ فى حالة عدم التساوى - طبعاً بعض الجمل البرمجييه التى ;  
هتتنفذ .

.....  
تقوم هنا بوقف البرنامج مثلاً أو القفز إلى مكان آخر لأنه من غير ذلك سوف ; HLT  
ينفذ الأوامر التاليه أيضاً يتم  
END:  
الكمبيوتر إجتنب . إزاي ٥ تساوى ;

.....  
تنفذ فى حالة التساوى - طبعاً مش بعض الجمل البرمجييه التى سوف ;  
هتتنفذ .

-----  
طبعاً يوجد جمل تقوم بالقفز إذا كان عدم التساوى أكبر من مثلاً أو القيمة الأولى أصغر من الأخرى وهكذا . يوجد الكثير و الكثير كما تحب أ ن تكتب برامجك.

-----  
:- [TEST] تقوم بالعمل السابق نفسه و لكن بدلاً من عمل عملية طرح , تقوم بعمل Logic AND و هى-بينى و بينكم-تستخدم فى إختبار بت واحده و هذه العمليه تسمى Masking .

مثال :- تعالو لنرى ماذا ستفعل هذه الداله :-

```
-----  
MOV AX,٥  
NEG AX  
ADD AX,٢  
TEST AX,١٠٠٠٠٠٠٠B  
JZ END
```

كما هو متوقع القيمة سالبه ;  
HLT  
END:  
إطلاقاً القيمة موجبه . كما هو ليس متوقع ;

-----  
المثال السابق يوضح إستخدام الأمر [TEST] فى إختبار آخر بت فى القيمة الموجوده فى لا AX وهذه البت توضح إشارة الرقم كما لابدأ ن نعرف. فقامت بعمل  
TEST(AND) WITH ١٠٠٠٠٠٠٠B , IF the most seignificant bit in AX is (1) then zero flag will reset , else it will be set by the processor.

## الدورا ت-: loops

-من أهم قدرات الحاسب الألى هى عمل نفس الشىء مرآ ز عدده و لا تختلف معه عدد هذه المرآ ت و هى من أهم جوانب البرمجه أصلاً . فلا يمكن لبرنامج مفيد ألا يحتوى على دوره أو عدة دورآ ت لذا كان من المهمأ ز ندرجها هنا و تكون بعد التحكم فى مسار البرامج لما لها من إرتباط وثيق بما سوف نتحدث عليه.

-هناك ثلاث أنواع من الدورآ ت تتم بثلاث تقنيات مختلفه . سيتم ذكر الأنواع أولاً ثم تتبعها طرق عملها و تقنياتها-:

### الأنواع-:

#### •الدورا ت المستمره (الى مالانهايه-: )

وهى الدورآ ت التى تظل عامله إلى أ ز يتم غلق البرنامج من جانب نظام التشغيل أو غلق الجهاز نفسه . و طبعأ أنت تسأل نفسك هل لهذا النوع فائده فى البرمجه . أرد عليك بنعم كما سترى-:

-التارخ و الساعه , هل هما يتوقفان !!؟ بالطبع لا .  
-من الممكن إستخدامها فى ترك البرنامج عاملاً وهذا فى اللغات التى تعتمد على الأوامر المتواليه . Sequential programming  
-أيضأ من الممكن أ ز تستخدم فى عمل الفيروسات . تكفى دوره صغيره غير منتهيه لإستغلال معظم وقت المعالج من غير فائده و التأثير على باقى البرامج الأخرى فى النظم متعددة البرامج . Multiprogramming and Time-sharing Operating systems

#### •الدورا ت محددة المرآ ت-:

فى هذه الدورآ ت يكون عدد مرآ ت تنفيذ دوره محدد و هى من أكثر الأنواع إنتشارأ و لها إستخدامات عدده تتنوع بين العمليات الحسابيه والمنطقيه و باقى العمليات الأخرى .  
ب. أمثلتها ( FOR...NEXT) LOOP

#### •الدورا ت المحدده بشرط معين-:

يكون هناك شرط يتم إختبارها فى أو ل دوره أو فى آخرها . ويتوقف تنفيذ المره التاليه فى دوره على إختبار هذا الشرط , وهنا يتم إستخدام إحدى الشروط و طرق إختبارها السابقه.

### التقنيات و طرق العمل-:

#### •الطريقه البدويه -:

عن طريق تحديد عنوان ز فى أو ل أمر فى دوره ثم فى نهاية أوامر دوره يتم إدراج أمر قفز للعنوان الذى وضعناه سابقأ و يكون القفز مشروط أو غير مشروط ( Conditional / Unconditional Jump) .

مثال -:

```
-----  
;   
MOV AL,0  
XOR CL,CL ;CL = 0  
START_LOOP : بداية الدوره;  
ADD CL,AL  
DEC AL
```

```

CMP AL,00H ; والصفـر AL فـر نـ القيمـه المـوجـوبـه فـى لا
JNE START_LOOP
يساوى صفـر AL يـتم القـفـظ لو لم يـكون؛
MOV BYTE PTR [0100H],AL ; النـهائـى فـى الـذاكرـه تـخزينـ الناتـج
;

```

فـى المـثال السـابق تم إـستـخـدام الـدورـه المـشـروطـه START\_LOOP فـى جـمـع الأـرقـام مـن ١ إلى ٥ و تـخزينـهم فـى الـذاكرـه.

### • الدورـا تـ بإـستـخـدام الأـمـر :- LOOP

هـذا الأـمـر له صـور عـده مـنـها إـمـشـروط و مـنـها الغـير مـشـروط كـما لـابدأ نـرى:-  
 :- [LOOP] يـتم كـتابـة أـسم أو عـنـوا نـ أو لـ جـمـله/أـمـر فـى الـدورـه , و يـتم وـضع عـدد مـرـا تـ الـدورـه فـى CX , و تـقـوم بـتـقـليل لا CX بـواحد و تـختـبر قـيمـته لو كـانت صـفر تـنـهـى الـدورـه و لو غـير ذـلك يـتم القـفـز ألى العـنـوا نـ المـوجـود أـمـامـها.  
 مـثال :-

```

;
MOV CX,٥
XOR AX,AX
START_LOOP :
ADD AX,CX
LOOP START_LOOP
MOV [0100H],AX
;

```

نفس المـثال السـابق.

Conditional LOOP:-

- LOOPZ
- LOOPNZ
- LOOPE
- LOOPNE

وهـذه الأنـواع تـسـتـخـدم بـعد , CMP/TEST Instructions و على حـسـب الشـرط يـتم القـفـز , و يـكون إـسـتـخـدامـها مـثـل . LOOP

### • الدورـا تـ بإـسـتـخـدام :- REP

وهـى تـسـتـخـدم مـع نـقل البـيـانات String Transfer Instructions .

### (١) التـعـامل مـع الـسـلاسل النـصـيـه :- Strings

الـسـلاسل النـصـيـه هـى مـن أـهم البـيـانات الـتى تـتـعـامل مـعـها فـى نـظـم الحـاسـب عـمـوماً و فـى لغـات البرمـجـه خـصـوصاً . مـن المـمـكـن التـعـامل مـعـها على أنـها بـعض الأـحـرف المـتـتـابـعـه و التـعـامل مـعـها كـما سـبق الـدورـا تـ و التـكرار إـعـتـمـاداً على أ نـأى سـلـسـله حـرفـيـه تـتـكون مـن عـدة أـحـرف مـعـلـومـة الطـول و مـتـتـابـعـه . و لـكن الـأـسـمـى الـى تـصلـه بـعض الـدـلـا الـتى مـتـعلـق بـهـا عـلـينا هـذا التـعـامل و تـقـليل التـعـقـيد و طـول البرامـج نـسـبياً.

• أهم هـه الـدـلـا مـا يـلى :-

[LODS],[STOS],[MOVS],[INS],[OUTS],[SCAS],[CMPS]

قـبل البـدء خـذ إـنتـباهـك مـن الأـتـى :-

- DI ( Data Index/Destination Index ) is combined with the ES (Extra Segment) to reach the data place in memory which the destination of data of the current string transfer instruction.
- SI ( Stack Index/Source Index ) is combined with the DS (Data Segment) to reach the data place in memory which the source of data of the current string transfer instruction.
- D (Direction Flag) which indicates the direction of the flow of the strings in the memory .
  - At D=0 then Auto-Increment will occur to DI,SI According to the size of the current data (Increment with 1 if byte, 2 if word, 4 if double word).
  - At D=1 then Auto-Decrement will occur to DI,SI According to same previous characteristics.
  - [STD] Will Set the Direction Flag (D=1).
  - [CLD] Will Clear the Direction Flag (D=0).
- You can't move a Immediate Value to Any Segment Register , So you Should First Move The Desired Value To AX and Hence, Move it From AX to THE Segment.

سنتناول هذه اللوالبالتصلي مع شرح أمثله :-

### [LODS]

هذه الداله يمكنها التعامل مع أحجام من البيانات مختلفه مثلاً [LODSB] لنقل بايت ,  
[LODSW] لنقل كلمه [LODSD] , لنقل كلمتان . DOUBLE WORD  
:- [LODSB] -تنقل إلى AL محتويات المكان في الذاكره المعنون ب SI :DS] ثم تقوم  
بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بواحد مما يعنى أنها تقوم بالتالى :  
AL=DS:[SI] , SI = SI +- 1 .

:- [LODSW] -تنقل إلى AX محتويات المكان في الذاكره المعنون ب SI :DS] ثم تقوم  
بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بإثنين مما يعنى أنها تقوم بالتالى :  
AX=DS:[SI] , SI = SI +- 2 .

:- [LODSD] -تنقل إلى EAX محتويات المكان في الذاكره المعنون ب SI :DS] ثم تقوم  
بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بأربعة مما يعنى أنها تقوم بالتالى :  
EAX=DS:[SI] , SI = SI +- 4 .

من الجدير بالذكر:-

هناك داله للتكرار في التعامل مع السلاسل النصيه تسمى [REP] هذه الداله تعمل  
تكرار مثل LOOP بالضبط وتكرر عدد مرات ما في CX ولها أيضاً حالات خاصه كتكرار  
بشروط مثل . [REBE]/[REBNE] ويمكن وضع أسم الداله أمامها فقط مثل [REP] :-  
[STOSW] ولاحظ أن هذا لم يفع مع كل الدوال المذكوره أعلاه مالمدا [LODS]

مثال:-

```

;-----
CLD
MOV AX,0000H
MOV DS,AX      ; نحدد المقطع الذى نتعامل معه الأو ل؛
MOV CX,1000H   ; تحديد عدد مرآ ٲ النقل التى سوف تتم؛
MOV SI,0       ; البدأ من أو ل مكان داخل المقطع؛
MOV AH,2H      ; التجهيز للطبع؛
START:
LODSB
MOV DL,AL      ; تجهيز الحرف الذى سوف يتم طبعه؛
INT 21H

REP START

END :
;-----

```

فى المثال : تم طبع أو ل ١٠٠٠ حرف فى الذاكرة و طبعاً الموضوع مش عايز تفسير أكثر من كده.

### [STOS]

هذه الداله تتعامل أيضاً مع الأنواع المذكوره أعلاه و بنفس الطريقه .مثل:-  
 :- [STOSB] - نقل محتويات لا AL إلى المكان فى الذاكرة المعنون ب [DI]:ES ثم تقوم  
 بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بواحد مما يعنى أنها تقوم بالتالى :  
 $ES:[DI]=AL$  ,  $DI = DI +- ١$  .

:- [STOSW] - نقل محتويات لا AX إلى المكان فى الذاكرة المعنون ب [DI]:ES ثم تقوم  
 بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بإثنان مما يعنى أنها تقوم بالتالى :  
 $ES:[DI]=AX$  ,  $DI = DI +- ٢$  .

:- [STOSD] - نقل محتويات لا EAX إلى الكان فى الذاكرة المعنون ب [DI]:ES ثم تقوم  
 بزيادة/نقصان(على حسب علم الإتجاه D) لا SI بأربعة مما يعنى أنها تقوم بالتالى :  
 $ES:[DI]=EAX$  ,  $DI = DI +- ٤$  .

مثال:-

```

;-----
CLD
MOV AX,0B800H
MOV ES,AX      ; نحدد المقطع الذى نتعامل معه الأو ل؛
MOV DI,0       ; البدأ من أو ل مكان داخل المقطع؛
MOV CX,20*80   ; مرآ ٲ النقل التى سوف تتم تحديد عدد؛
MOV AX,0720H   ; نسخها إلى الذاكرة تحميل البيانات التى ستتم؛

REP STOSW      ; تكرار عملية النسخ؛

END:
;-----

```

هذا المثال يقوم بمسح الشاشة (٢٠ سطر \* ٨٠ عمود) عن طريق نقل قيمه ٠٧٢٠ إلى كل الأماكن في الذاكرة التي تبدأ عند (B٨٠٠٠H:٠٠٠٠H) وهذا المقطع هو مقطع الشاشة و هذا طبعاً في نظام الدوس فقط . ومن الجدير بالذكر هنا أن القيمة ٠٧٢٠ هي عباره عن جزأين , الجزء الأول ٠٧ هو كود اللون الأبيض على الخلفيه السوداء في نظام الدوس و الجزء الآخر ٢٠ H هو الأسكى كود للمسافه الفارغه .  
**الخلاصه :** أننا نطبع في ذاكره الشاشة المسافه الفارغه في كل أماكن الشاشة و ذلك لمسح الشاشة بالكامل و هذا المثال مكافئ لأمر الدوس. CLS

## [MOVS]

هذه الداله الوحيدده في جميع المعالجات من ٨٠٨٨ إلى بنتيوم ٣ التى تسمح بعمليات النقل من مكان فى الذاكرة إلى مكان آخر فى الذاكرة.  
 خذ حنرك من الآتى :-

لا نستطيع عمل أى عمليات نقل أو غيره بحيث تستلزم من المعالج النظر إلى الذاكرة مرتين . يعنى ...

This is Not Allowed : ( MOV [١٠٠٠H],[٢٠٠٠H] ) ( ADD [٠٩٠٣H],[SI] )  
 .....

هذه الداله تعمل مثل سابقتها من حيث مساحات البيانات كما يلى :-

:- [MOVSB] -تستخدم فى نقل بايت من محتويات المكان فى الذاكرة المعنون ب [ES:[DI]] إلى المكان المعنون ب [DS:[SI]] ويتم زيادة لا DI,SI بواحد أو نقصانهم بناءً على علم الإتجاه. D FLAG  
 مما يعنى أنها تقوم بالآتى.  $DI = DI + 2$  ;  $SI = SI - 2$  ;  $ES:[DI] = DS:[SI]$  :

:- [MOVSW] -تستخدم فى نقل كلمه من محتويات المكان فى الذاكرة المعنون ب [ES:[DI]] إلى المكان المعنون ب [DS:[SI]] ويتم زيادة لا DI,SI بواحد أو نقصانهم بناءً على علم الإتجاه. D FLAG  
 مما يعنى أنها تقوم بالآتى.  $DI = DI + 2$  ;  $SI = SI - 2$  ;  $ES:[DI] = DS:[SI]$  :

:- [MOVSD] -تستخدم فى نقل كلمتان من محتويات المكان فى الذاكرة المعنون ب [ES:[DI]] إلى المكان المعنون ب [DS:[SI]] ويتم زيادة لا DI,SI بواحد أو نقصانهم بناءً على علم الإتجاه. D FLAG  
 مما يعنى أنها تقوم بالآتى.  $DI = DI + 2$  ;  $SI = SI - 2$  ;  $ES:[DI] = DS:[SI]$  :

**لاحظ :** أنها تعمل بتكرار مع الأمر REP.

مثال :-

```
CLD
MOV AX, ٠١٠٠٠H
MOV DS,AX ;Source Segment
MOV AX, ٠٢٠٠٠H
MOV ES,AX ;Distination Segment
MOV DI, *
```

```
MOV SI, 0
MOV CX, 64 * 1024D
```

```
REP MOVSW
```

```
END :
```

كما توقعتم , المثال يقوم بنسخ المقطع ١٠٠٠٠ بالكامل إلى المقطع ٢٠٠٠٠ عن طريق تكرار نقل الكلمات عدد ٦٤ KWORDS و هو مساحة المقطع الواحد كما تعلمون.

### [INS]

ألم تتساءل أنه من المستحسن أن تكون هناك دالة تتعامل مع اللولابافه و تكون مهمتها إدخال أي بيانات من وحدة الإدخال المعروفة (الشاشة / الهارد ديسك مثلاً) ؟!! هذه الدالة تقوم بهذا العمل بتنظيم أكبر و مرونة أكثر و تتيح لك أيضاً التكرار عن طريق استخدام REP Instruction

### شرح كيفية العمل :

- يتم وضع عنوان وسيلة الإدخال التي سوف تستخدمها في DX
- تجهيز عنوان المكان في الذاكرة الذي سوف يستقبل البيانات في [ES:[DI]
- تجهيز علم الإتجاه على حسب متطلبات البرنامج .
- إختبار هل توجد بيانات حاضرة في وسيلة الإدخال أولاً قبل طلب هذه البيانات .
- مكافئه للآتى  $DI = DI + 1$  ;  $ES:[DI] = [DX]$  :

الداله تمكنك من نقل بايت عن طريق [INSB] وإدخال كلمه باستخدام [INSW] وإدخال كلمتين باستخدام [INSD] .  
مثال :-

```
MOV DI, 0
CLD
MOV AX, 1000H
MOV ES, AX
MOV DX, 2ACH ; وسيلة الإدخال المستخدمه عنوانها
MOV CX, 100
```

```
REP INSB
```

```
END :
```

طبعا سهل!!!

### [OUTS]

نفس طريقة الإدخال و لكن للإخراج و هنا نستخدم لعنوان المكان في الذاكرة موجود في [SI:[DS]] كما توقعتم.



مثال :-

```
MOV SI,0  
CLD  
MOV AX,1000H  
MOV DS,AX  
MOV DX,2ACH  
MOV CX,512
```

وسيلة الإدخال المستخدمه وليكن عنوان الهازرد ديسك عنوان ;

```
REP OUTB
```

```
END :
```

المثال يمثل نقل بلوك بيانات من الذاكرة [SI]:[DS] إلى الهازرد ديسك.

### [SCAS]

هذه الداله تقوم بمقارنة محتويات المكان فى الذاكرة الذى عنوانه موجود فى [DI]:[ES] و محتويات الـ AL عند استخدام [SCASB] ، وقارنها بمحتويات الـ AX عند استخدام [SCASW] ، وقارنها بمحتويات الـ EAX عند استخدام [SCASD] وطبعاً هناك ذيلها أو نقصان فى DI كما تعرفنا سابقاً.

مثال :-

```
CLD  
MOV DI,0  
MOV CX,2000  
XOR AX,AX  
MOV ES,AX
```

```
REBNE SCASW
```

```
END :
```

على المكان الفارغ أو القصود البحث عنه أو سيحتوى فى حالتنا على 2000 سيحتوى ;  
DI البحث بعد إنتهاء

فى المثال نبحث فى أو ل 2000 مكان فى الذاكرة عن أو ل مكان فارغ أو يحتوى على  
H 0000

### [CMPS]

هذه الداله فعلاً عظيمه لأنها تقارن مكانان فى الذاكرة فى أمر واحد و تتوقف مساحة هذا المكان على استخدام الأمر. [CMPSB] ، [CMPSW] ، [CMPSD]

•الأو ل يوضع فى. [DI]:[ES]

•الثانى يوضع فى. [SI]:[DS]

مثال

```
CLD
MOV DI,0
MOV SI,0
MOV CX,2000
XOR AX,AX
MOV ES,AX
MOV AX,1000H
MOV DS,AX

REBE CMPSW

END :
```

التعامل مع الإجراءات :- Procedures

• ما هى الإجراءات ؟ Procedures

الإجراءات ت هى مجموعه من سطور الكود و الأوامر البرمجييه و الجمل البرمجييه تكون أجزاء فى داخل البرنامج نفسه أو خارج البرنامج وتعمل كبرنامج صغير لها متغيراتها الخاصه و لها مدخلاتها ومخرجاتها الخاصه وأهم شىء : لها عملها الخاص التى تبنى من أجله.

• مميزا ت الإجراءات :-

- الإجراءات ت من أكثر إمكانيات اللغات البرمجييه تأثيراً فى نظام البرنامج و قوتها إستقرارها.
- توفر الكود و الوقت على المبرمج نفسه و تعطى له إحساس بقيمة ما يفعل.
- وهى بدورها تساعد فى إنتشار البرامج و خصوصاً الكبيره و الضخمه منها و تبدل ل أجزاءها بين أكثر من شخص أو مبرمج.
- يتم بناء البرامج والتطبيقات الكبرى عليها.
- تساعد فى سرعة تصحيح الأخطاء . Error Handling
- أساس عمل البرمجه الكائنيه(OOP Object Orianted Programming)

• كيفية عمل الإجراءات ت...؟

يتم عمل نداء للإجراء المراد الوصول إليه عن طريق [CALL Procedure\_name] من داخل الإجراء الرئيسى Main Procedure أو من أى إجراء آخر . هذا بعد تجهيز مدخلات الإجراء إذا كان يعمل على مدخلات . فيتم أنتقال العمل إلى الإجراء الفرعى (لأنه يتفرع من إجراء آخر) . وبعد إنتهاء عمله بالأمر [RET] يرجع الإجراء ثانياً إلى الجملة البرمجييه التى تلى الأمر [CALL] فى الإجراء الذى قام بالنداء أولاً.

خذ إنتباهك من الآتى :-

## الإجراء الرئيسى :- Main procedure

هو الإجراء الذى يتم بدأ البرنامج به , بمعنى أنه لا إجراء يعمل مع بدأ البرنامج بالعمل . و ينتهى البرنامج بإنهاء هذا الإجراء كما هو معتاد .  
يتم وضع إسم الإجراء الرئيسى بجانب أمر الإنتهاء END فى أسفل البرنامج.

### كيفية إنشاء الإجراء ت...؟

- يتم إنشاء إجراء جديد عن طريق كتابة أسم الإجراء الجديد يليه كتابة الأمر PROC.
- أكتب بالداخل جمل الإجراء أيّاً كانت على حسب إختيارك و على حسب عمل الإجراء .
- ينتهى الإجراء بكتابة أسم الأجراء يليه الأمر RET.

### ملاحظات على الإجراء ت:-

- لا يمكن كتابة إجراء داخل إجراء آخر . وإلا فلن يعمل صحيحاً وسوف يعطى المترجم Assembler رسالة خطأ . Error لذلك فإنه يجب كتابة الإجراء الجديد أما بعد إنتهاء إجراء أو قبل بدأ إجراء.
- لا يجب أن يكون الإجراء فى نفس البرنامج و لكن يمكن إضافته إلى البرنامج و هو يكون داخل ملف آخر فى الهارد أو فى الذاكره مثل إجراء ت النظام التى تعمل لها نداء عن طريق INT وهى فى هذه الحالة يطلق عليها . Software Interrupts ولكن الأخرى التى فى ملف آخر تسمى Macros or Modules ومن الممكن أن تكون فى حالتها المصدريه أو فى حالتها القابله للتشغيل مثل (DLL'S Dynamic link libraries)
- إلى هنا ينتهى حديثنا عن الإجراء ت . وسنستعرض برنامجاً كاملاً مفتوح المصدر يحتوى على إجراء ت عدة . فهيا بنا

## هذا البرنامج يقوم بالآتى :-

- ١ . يقوم بطلب البيانات المطلوب تشفيرها من المستخدم.
  - ٢ . يقوم بعمل اختبار لها , للتأكد من صحة البيانات.
  - ٣ . يشفر البيانات عن طريق عمل XOR لكل حرف على حده من هذه البيانات .
  - ٤ . يقوم بعرض البيانات فى صورتها المشفرة.
  - ٥ . ويقوم بعرض البيانات نفسها ثانياً.
- كما إستنتجتم فأن البرنامج يقوم بعمل تشفير لبيانات من المستخدم ويعرض له البيانات مشفرة .

## البرنامج

```

;-----
;App_name : DECODER.EXE
;Author : mrezzuz
;web site : http://www.t.t.t.t.com
;Date : 20/8/2003
;-----
Title decoder
.MODEL small
.STACK 100H

.DATA

    LF EQU 0AH
    CR EQU 0DH
    MSG DB 100H DUP ( ) ;Encode up to 1 kbytes
    HR db LF,CR,'-----
-----$'

    CODE_MSG db 'The code : $'
    MSG_MSG db 'The message : $'
    WELCOM DB CR,LF,'Type in a new message for working:',CR,LF,$'
    AGAIN DB CR,LF,'Play again ',\h,' ?[Y For agree]',LF,CR,$'
    CODE_VAL DB 9BH

.CODE

    MAIN PROC
    MOV AX,@DATA
    MOV DS,AX
    ;-----

START:
    XOR AX,AX
    XOR BX,BX
    XOR CX,CX
    XOR DX,DX
    LEA DX,HR
    MOV AH,9
    INT 21H
    LEA DX,WELCOM ;print the welcom message
    INT 21H
;start to read the user input
    LEA BX,MSG

READ_BYTE :
    MOV AH,1 ;input one char
    INT 21H
    CMP AL,CR ;compare end of user inputs
    JE END_READ
    CMP AL,0AH ;compare backspace
    JE BACK
    MOV [BX],AL ;store input char into msg
    INC CX ;increment counter
    INC BX ;increment offset
    JMP READ_BYTE

BACK:
    MOV DL,22 ;Print space and backspace to clear the backed char.
    MOV AH,2

```

```

    INT 21H
    MOV DL,^
    INT 21H
    DEC CX          ;if backspace then dec counter and offset
    DEC BX
    JMP READ_BYTE

END_READ:
;Not operate until user input at least one char or until cx > 00h
and else ask for again
    CMP CX,00H
    JLE ask_again
    CALL DEC_INC    ;calls xor function TO ENCODE.
    MOV AH,2 ;print new line
    MOV DL,LF
    INT 21H
    MOV DL,CR
    INT 21H
    LEA DX,CODE_MSG
    MOV AH,9
    INT 21H
    CALL PRINT_MSG ;calls msg printting which was stored in msg
    CALL DEC_INC   ;call xor TO DECODE .
    LEA DX,MSG_MSG
    MOV AH,9
    INT 21H
    CALL PRINT_MSG

ask_again:
    LEA DX,AGAIN
    MOV AH,9
    INT 21H
    MOV AH,1 ;scan what's user replay . if Y or y then again else
exit
    INT 21H
    CMP AL,'Y'
    JE START
    CMP AL,'y'
    JE START

END_MAIN: ;Exit to dos
    MOV AH,1CH
    INT 21H

    MAIN ENDP
;
PRINT_MSG PROC
;The procedure prints the bytes sequancly from the msg address CX
times.
;CX must > 00h
;The procedure uses BX as an offset
    PUSH CX ;store CX
    MOV AH,2 ;print new line
    MOV DL,LF
    INT 21H
    MOV DL,CR
    INT 21H
    LEA BX,MSG
START\ :
    MOV DL,[BX] ;start print first char of msg address

```

```

    INC BX    ;increment the offset
    INT 21H
    LOOP START\
    MOV AH,2 ;print new line
    MOV DL,LF
    INT 21H
    MOV DL,CR
    INT 21H
    POP CX   ;return CX
    RET
PRINT_MSG ENDP
;
DEC_INC PROC
;The procedure XOR the byte from address msg CX times .
;CX must > 00H
;The procedure uses BX as an offset
    PUSH CX
    LEA BX,MSG
STARTY:
    MOV AH,CODE_VAL    ;moves the code value into AH
    XOR [BX],AH        ;XOR with offset BX
    INC BX
    LOOP STARTY
    POP CX
    RET ;return
DEC_INC ENDP
END MAIN    ;start the program control from MAIN procedure

```