الأسمبلي

ماهي الأسمبلي :

في قديم الزمان أيام بدايات الكمبيوتر كانت برمجة الكمبيوتر تتم بواسطة **لغة الأله** Machine Language أختصاراً ML *(لغة الآله هي اللغه التي تفهمها الآله مباشرة دو ن الحاجة الى تفسير وهي تخزن بصورة ثنائية [تركيبة من الأصفار والوحايد] 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 ل لغة برمجة وهذه اللغه فكرتها بسيطة جداً حيث أنه بدل أ ن 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 بأستخدام تخطيط واحد-الى-واحد أ ي أ ن كل سطر أو عبارة في الأسمبلي تحول الى تعليمة واحدة مقابله في لغة الآله (مثال بدل كتابة ٠١١٠٠٠٠٠٠٠٠١٠١ يتم كتابة mov al, ٥) يعرف البرنامج الذي يقوم بعملية التحويل **بالأسمبلر** Assembler ، علماً بأن هناك عدة أنواع من الأسمبلر كل نوع يختص بتقنية معينة وبعائلة معينه من المعالجات
ونحن هنا بصدد تعلم البرمجة بالأسمبلي للمعالجات المبنية على تقنية IBM-PC والمنتجة من شركة أنتل وهي العائلة **٨٦×٨٠** ويرمز لها أختصاراً X٨٦ وهي تضم : (٨٠٨٦ / ٨٠٨٨ / ٨٠١٨٦ / ٨٠٢٨٦) معالجاتلا ١٦ بت و (٨٠٣٨٦ / ٨٠٤٨٦ / ٨٠٥٨٦=بنتنيوم١ / ٨٠٦٨٦=بنتنيوم٢ / ٨٠٧٨٦=بنتنيوم٣ / ٨٠٨٨٦=بنتنيوم٤)
معالجاتلا ٣٢ بت وسوف أتطرق في دروسي متقدمة الى المعالج أتنيوم ٦٤ بت المبني بتقنية جديده كلياً لمن يرغب بمعرفة مسبقة لهذا المعالج الجديد كذلك سوف أتطرق بأذ ن الله الى **الكروسي أسمبلر** 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 .

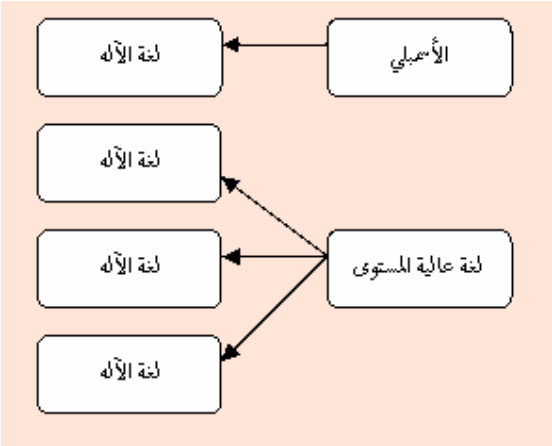
تعريف الأسمبلر
الأسمبلر هو برنامج يقوم بتحويل التعليمات المكتوبة بلأسمبلي الى لغة الآله .

لماذا أريد أستخدام الأسمبلي :

بتعلمك لغة الأسمبلي فأنك تكشف النقاب عن الأسرار المختفيه وراء الكمبيوتر وتصبح قادراً على الفهم تماماً كيف يعمل المعالج وكيف يعمل البرنامج وبذلك تزيد خبرتك كمبرمج وبالطبع فأن 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) الأجهزه ، كذلك فأن الأسمبلي يعطيك برامج سريعة جداً ، وبالطبع فأن بناء برنامج متطور بالأسمبلي أشبه بحفر حفرة بواسطة الملعقه فالبرغم أنك تحفر الا أ نك أنت تجيد قليلاً ولكن من المحبذ جداً برمجة بعض الدوال و الأجزاء من البرامج بالأسمبلي وبقية 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 أختصاراً LLL لأنها تكتب تعليمه تعليمه (بعض الناس يعتبر لغة السي لغة منخفضة المستوى ، وهذا الكلام أرجح الى الخطأ منه الى الصوا ب) بينما تعتبر باقي اللغات High-Level Language أختصاراً HLL وفي هذه اللغات تختفي تقنية تخطيط واحد-الى-واحد وتفسر التعليمة الواحدة الى عدد كبير من تعيلمات 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 بأد ق وأقل التفاصيل ، في أيام البرمجة القديمة كان المبرمجون يكتبون برامجهم بلغة الأسمبلي لأن ذاكرة الرام وقتها كانت صغيرة (أقل من ٦٤كيلوبايت) وهم بحاجة الى برامج أصغر وأسرع خصوصاً أ ن معالجتهم أيضاً كانت بطيئة ، مع تطور الحاسوب وتوسع سعة ذاكرة الرام وزيادة سرعته أصبحت البرامج أكثر طولاً وتعقيداً ، هذا التعقيد أدى الى أستخدام اللغات البرمجية عالية المستوى HLL مثل السي والكوبول والبيسك والباسكال والفورترن ، مرة أخرى تطور الحاسوب فأدى الى أستخدام اللغات العليا الموجهة الهدف OOP مثل السي++ والجافا والتي مكنت من كتابة برامج قوامها آلاف الأسطر والتعليمات المعقده والمتداخله .

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

يفض المبرمجين لغة السي++ كلغة 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 أستخدام الأسمبلي كعنصر مهم في الوصول الى العتاد وبرمجة الجزئيات المحتاجة للسرعة .

لا يستخدم المبرمجون شفرة الأسمبلي وسط شفرة لغة عالية المستوى عادة وأنما 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 أستدعيت شفرة مكتوبة بلغة الأسمبلي .

لغة الآله :

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

لفد حافظت شركة أنتل على التوافقية في العائلة ٨٦X أبتداء من المعالج ٨٠٨٦ وصولاً الى بانتنيوم ٤=٨٠٨٨٦ ولكن الحفاظ على التوافقية يفرض قيود على تصميم المعالج وأستخدام تقنيات قديمة ، ومؤخراً قرر ت شركة أنتل ايقاف عائلة المعالج ٨٦X عند بانتنيوم٤ وقامت بأنشاء معالج جديد (غير متوافق مع العائلة ٨٦X) مبني على تقنية ٦٤لا بت وهو المعالج اتانيوم الجديد .

مثال على لغة الآله :

التعليمية ١٠١١٠٠٠٠٠٠٠٠١٠١ هي تعليمة بلغة الآله ومعناها أنقل الرقم ٥ الى المسجل ah يقابلها بلغة الأسمبلي ,omov ah ٥ 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 من١لا ١٦ ـتبلا ٨بت الأخرى تشكل الرقم ٥ ثنائياً

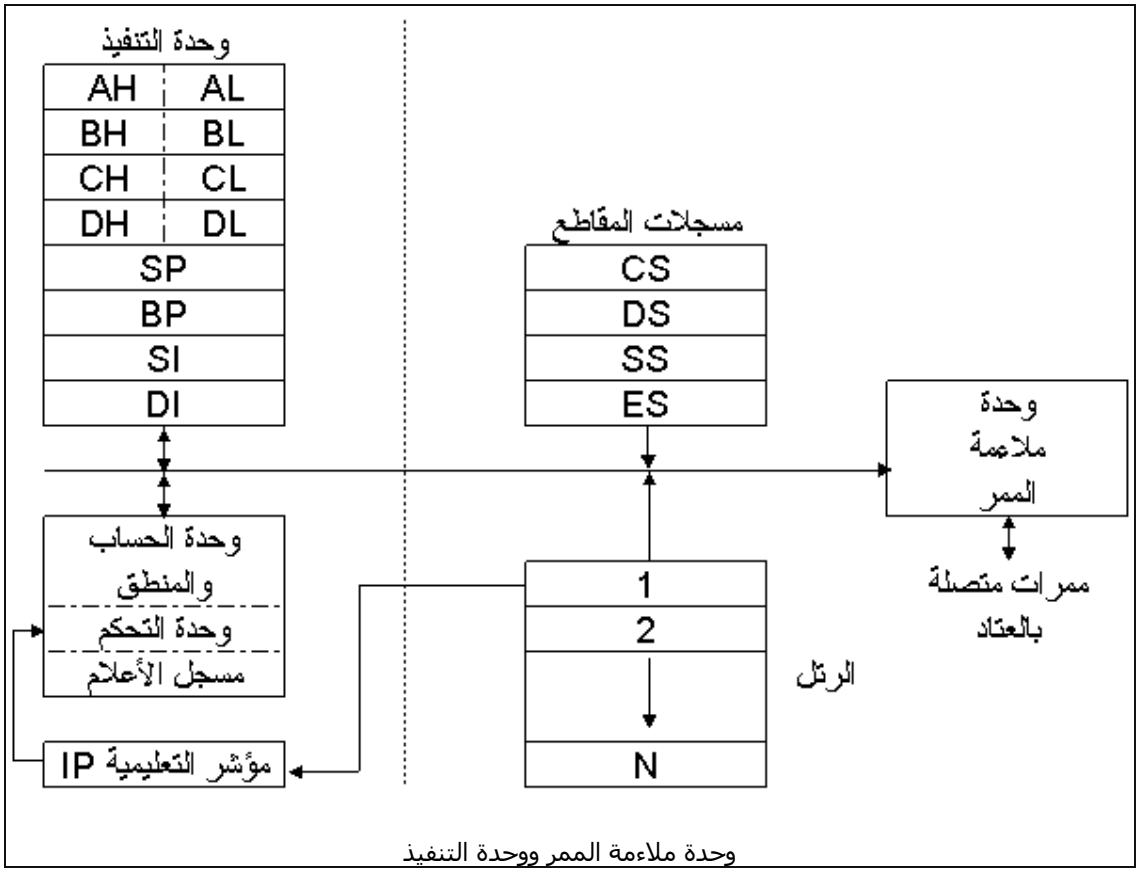
كيان الحاسوب الصلب :

يتألف الحاسوب بشكل أساسي من اللوحة الأم Mother Board والمعالج Microprocessor وذاكرة القرأة-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 ندر س معالجات أنتل من العائلة ٨٦X لأنها العائلة الأشهر والأكثر أستخداماً بين الناس .

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

أختصاراً CU ومجموعة من المسجلات. تتألف وحدة ملائمة الممر من وحدة التحكم بالممر Unit Bus Control ومسجلات المقاطع Segment Registers ورتل=كيو التعليمات Instruction Queue (الرتل أو الكيو هو نوع من أدارة الذاكرة تكون فيه المعلومة الداخلة أولا خارجة أولاً FIFO=First In First Out). وتقوم وحدة ملاءمة الممر بعمليات التحكم بالممر ونقل المعطيات بين كل من وحدة التنفيذ والذاكرة وأجهزة الأدخال 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 أ ن 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 الأدخال أو الأخراج . وخلافاً للطريقة التسلسلية في المعالجة فأن هذه العملية تحقق حدو ث عمليتي الجلب fetching التنفيذ execution في وقت واحد الأمر الذي يزيد بدورة من سرعة المعالج .



وحدة ملاءمة الممر ووحدة التنفيذ

ذاكرة القرأة-فقط ROM = Read-Only Memory :

وهي عبارة عن شريحة دائرة متكاملة IC تحوي على ذاكرة فيها بيانات غير قابلة لأعادة الكتابة عليها (أفتراضياً - شرائحلا ROM الحديثة يمكن إعادة الكتابة عليها بطرق مختلفة) ،حتتوي هذه الذاكرة على برنامجلا BIOS=Basic Input Output

System أو نظام الأدخال والأخراج الأساسي، ولايمكن للمعالج القرأة من هذه الذاكرة مباشرة ولكن أو ل شئ يفعل المعالج عند تشغيله في عملية الأستنهاض هي تحميل البيانات الموجودة في الروم ونقلها في الرام أو بالأحرى الى القسم الأخير من الرام ذا العنوا ن الأكبر . تجلى فائدةلا BIOS في القيام بعملية الفحص الذاتي عند الأستنهاض POST=Power On Self Test بالأضافة الى تحميل برنامج محمل نظام ال شغيل بالأضافة ا لى توفير دولا ومقاطعات قيلا ية في ذ اكرة الرلمستظ ع أن تستخدمها البرامج للرسم على الشاشة مثلاً أو التعامل مع لوحة المفاتيح أو القرأة والكتابة من والى القرص الصلب .

ذاكرة الوصول العشوائي Random Access Memory = RAM :
هذه الذاكرة مهمة جداً حيث أ ن أ ي برنامج لايمكن أ ن يعمل الا اذا حمل الى هذه الذاكرة كذلك فهي تستخدم لحفظ المتغيرا ت وحفظ برامج النظام الأساسية ومنها جدولا المقاطعات والمقاطعات أنفسها والرويتنات الفوية ..... الخ ولايتم أستخدام القرص الصلب لحفظ مثل هذه المعلومات لأن وقت الوصول فيه أبطأ بكثير من ذاكرة الرام (ولو أ ن القرص الصلب يستخدم في توفير ذاكرة أفتراضية عن طريق القيام بعمليات مبادلة للصفحات مع ذاكرة الرام) ، وبالطبع فأن هذه الذاكرة يمكن الكتابة اليها أو القرأة منها عن طريق عنونتها .
فبأستخدام العنوا ن يمكننا أ ن نصل الى مكان محدد في الذاكرة لنعمل عليه كل عمليات التحرير المطلوبة

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

العد الثنائي :
يتم تمثيل الشفرا ت والبيانات في ذاكرة الكمبيوتر كتواليف من الشحنات الكهربائية تأخذ قيمتين الأولى وهي وجود الشحنة ويرمز لها ب ON أو صحيح TRUE أو ١′ والأخرى وهي غياب الشحنه ويركز لها ب OFF أو خطأ FALSE أو ٠′ ، ووجود الشحنة يكون عادة بين ٤,٥ الى ٥,٥ فولت (المعالجات الحديثه بين ٢,٥ الى ٣,٥ فولت) وغياب الشحنة يكون بين +٠,٥ فولت و -٠,٥فولت .

وحدا ت الذاكره الأساسيه في الذاكره والوحدا ت التي سنتعامل معها كثيراً هي :

أسم الوحدة		الأحتمالات	مايعادلها
البت	Bit	أحتمالين ٠ و ١	١بت
البايت	Byte	٢٥٦ أحتمال	٨بت
الكلمة	Word	٦٥٥٣٦ أحتمال	١٦بت=٢بايت
الكلمة المضاعفة	DoubleWord=Dword	٤،٢٩٤،٩٦٧،٢٩٦ أحتمال	٣٢بت=٤بايت=٢كلمة
الكلمة الرابعية	Quafword=Qword	١٨،٤٤٦،٧٤٤،٠٧٣،٧٠٩،٥٥١،٦١٦ أحتمال	٦٤بت=٨بايت=٤كلمات=٢كلمة مضاعفة

بالطبع هذه الوحدا ت الأساسية والصغيره أما مضاعفاتها فهي :

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

تخزين الأرقام بدو ن أشارة يأخذ النطاق الآتي لكل وحده من الوحدا ت الأساسية بأستخدام طريقة بدو ن الأشارة unsigned:

أسم الوحدة	المجال	
	من	الى
البايت	٠	٢٥٦
الكلمة	٠	٦٥٥٣٦
الكلمة المضاعفة	٠	٤،٢٩٤،٩٦٧،٢٩٦
الكلمة الرباعية	٠	١٨،٤٤٦،٧٤٤،٠٧٣،٧٠٩،٥٥١،٦١٦

الأسكي كود ASCII:

يتم في الحاسوب وبقية توحيد أستخدام الرموز أستخدام شفرة الآسكي كود (حالياً عمل على تبني شفرةلا unicode وهي تسمح بتعدد اللغات في مستند واحد حيث يتم تمثيل كل حرف بأستخدام كلمة واحده=٢بايت) كلمة ASCII هي أختصار ل : Interchange Information Code For Standard American National ويتم أستخدام هذا الكود الموحد لتسهيل تناقل البيانات ويمثل كل رمز فيه بعدد ثنائي بطول ١بايت=٨بت=٢٥٦أحتمال .
إضغط هنا لمشاهدة الجدو ل آسكي

طريقة كتابة الأرقام في الأسمبلر :

لكتابة عدد ثنائي يوضع في آخر الرمز (b) لدلالة على أنه باينري مثال : B١١٠١٠٠١٠Binary= أما العدد العشري فلايحتاج الى أضافة وأما العدد لأساس ٨ فيكتب مع المرمز (Q) في نهايته =Q١٢٧٦Octal أو الرمز (O) في نهايته =O١٢٧٦Octal أما العدد السداسي عشر فيكتب بوضع H في نهايته =CDH٩AB٠hexadecimal مع مراعاة وضع ٠ اذا كان العدد يبدأ بحرف كما المثال .

يجب أ ن تعرف الفرق بين تخزين الرقم كرقم أو تخزينه كنص فتخزين الرقم ٢٠١ مثلاً كرقم سسيأخذ بايت واحد وهو جاهز للقيام بعمليات رياضية ومنطقيه عليه أما تخزينه كنص فسيأخذ ثلاثة بايت في البايت الأو ل سيخزن

الرقم الخاص بالأسكي كود للرمز '٣' وكما قلت يخزن كرقم يدل على الرمز أما البايت الثاني فسيخزن رقم الآسكي كود للرمز '٠' أما البايت الثالث فيأخذ القيمه الخاصة بالرمز '١' في الآسكي كود أ ي أ ن الرقم خزن بطريقة "١٠٣" وليس ١٠٢ وهذه الطريقة ليست جاهز للجمع أو الطرح ولكنها ممتاز للطباعه على الشاشه ويمكن تحويل النص الى رقم والعكس .

الأعداد ذ ي الأشارة :
يتم تخزين الأعداد ذ ي الأشارة كالتالي :
العدد موجب أذا كانت البت الأخيره صفر وقيمة الرقم هي باقي البتات أ ي لو أخذنا رقماً من بايت واحد فأن البت رقم ٧ (الثامنه - الترقيم يبدأ من الصفر) يجب أ ن تكون صفراً ليكون العدد موجب أما البتات من ٠ الى ٦ (السبعه الأولى) فتشكل قيمة الرقم
أما أذا كان العدد سالب فأن البت الأخيرة تساو ي واحد أما قيمة الرقم فتساو ي سالب المكمل الثنائي للعدد أ ي لو أخذنا رقم مخزن في واحد بايت مثال = ١١١١٠١١٠ بما أ ن البت السابعه=١ فأن الرقم سالب / نأخذ الآن المكمل الثنائي للعدد وهو ٠٠٠٠١٠١٠ / القيمة تساو ي -٠٠٠٠١٠١٠ أ ي سالب عشرة .

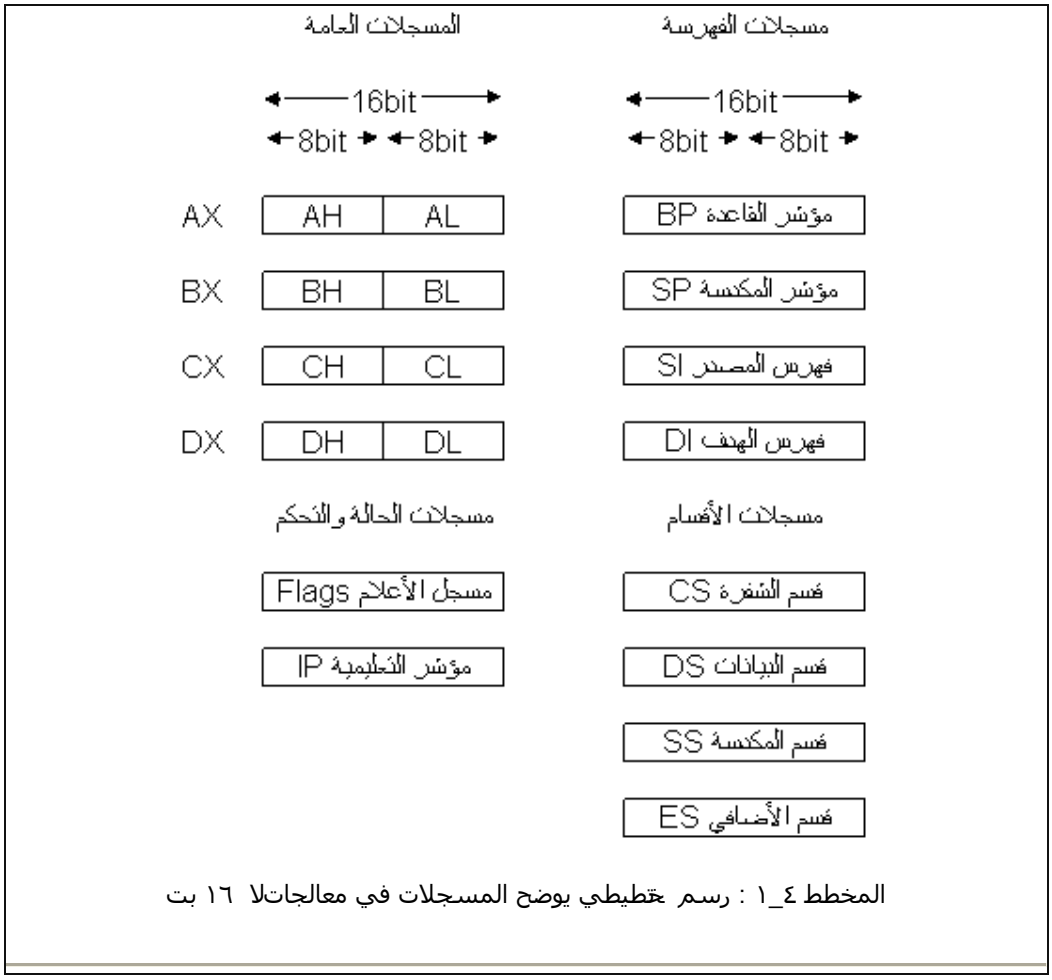
وبتخزين الرقم بالأشارة يختلف النطاق الذي تأخذ كل وحده :

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

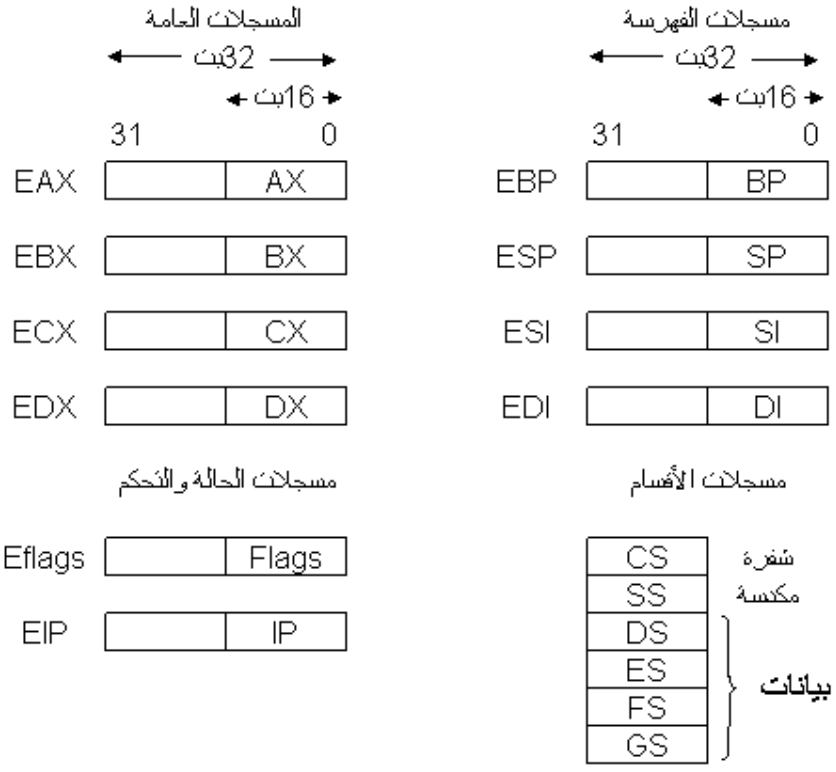
المسجلات REGISTERS :

الكمبيزتر يحتاج في تعاملاته الى ذاكرة سريعة جداً ومتصلة بالمعالج مباشرة حتى يمكن له أ ن يخزن فيها المعلومات المطلوبة لعملية حسابية معينة أو عداد لحلقة معينة ،هذه الذاكرة تعرف بالمسجلات REGISTERS وهي ذاكرة سريعة جداً تفيد المعالج في أجراء العمليات بسرعه وكفاءة أكبر

هناك خمسة أنواع أو تصنيفات للمسجلات (تختلف هذه التصنيفات قليلاً من مرجع لآخر) وهي مسجلات الأغرا ض العامة General-Purpose Registers (تعرف في بعض المراجع بمسجلات المعطيات Registers Data) و هناك مسجلات الأقسام Segment Registers والمسجلات الدليلية Registers Index ومسجلات التأشير Pointer Registers بالأضافة الى مسجلات الحالة والتحكم Status and Control Registers.



المخطط ٤_١ : رسم تخطيطي يوضح المسجلات في معالجاتلا ١٦ بت



لا٣٢بت المخطط ٤_٢ : رسم تخطيطي يوضح المسجلات بعد توسعتها في معالجات

[١] مسجلات الأغرا ض العامة General-Purpose Registers :

وهي عبارة عن كل من المسجلات AX و BX و CX و DX ،طول كل منها ١٦بت أ ي كلمة أ ي ٢بايت البايت اليسار ي فيهما يعرف بالعلوي (High) أما البايت الأيمن فيهما فيعرف بالمنخفض(Low) فمثلاً المسجل AX يتألف من مسجلين العلوي وهو AH والمنخفض وهو AL ، أ ن 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 معالجاتلا ٣٢ بت مع بقاء المسجلات نفسها ولكن كل منها أصبح جزء من مسجل موسع بطول ٣٢بت وهي EAX,EBX,ECX,EDX ، أ ي أ ن المسجل EAX هو بطول ٣٢بت وكجزء منه هناك المسجل AX بطول ١٦بت والذي يتألف هو الآخر من مسجلين هما AL و AH بطول ٨بت لكل منهما.

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

هذا المسجل كان من أهم المسجلات في معالجاتلا ٨بت القديمة جداً حيث كانت كل العمليات الرياضية والمنطقيه تجري من خلاله ولذلك كان يسمى بمسجل المركم لمراكم النواتج فيه لكن معالجاتلا ١٦ بت وسعت المرونة وجعلت كل مسجلات الأغرا ض العامة تستطيع أ ن تجرى من خلالها العمليات الرياضيه والمنطقية الا أ ن المسجل AX مازلا المفضل لأجراءها حيث أ ن أستخدام الله جل AX أو أحد أجزاءه يؤد ي مع بعض التعليمات الى توليد شفره أقل أختصاراً (الفرق بايت واحد فقط لكل تعليمية) ، يمكن استخدام المسجل AX كمسجلين هما AL و AH حيث تعرفلا ٨بتات الأولى التي في اليسار بالمسجل AL لا ٨بتات الأخيره التي في اليمين بالمسجل AH ، أما بالنسبة ل EAX وهو المسجل الموسع ل AX فهو بطول ٣٢بت ويعتبر

المسجل AX كجزء منه.

المسجل BX (**مسجل القاعدة**-Base Register) :
هو المسجل الوحيد من بين مسجلات الأغرا ض العامة الذي يمكن أستخدامه كدليل(INDEX) ، يمكن أستخدام 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 ٣٢بت.

المسجل CX (**مسجل العداد**-Counter Register) :
يستخد عادة كعداد ويستخدم هذا المسجل بشكل خاص مع تعليمة التكرار LOOP حيث يعمل كعداد لها وبالطبع يمكن استخدامة في العمليات الرياضية والمنطقيه ، وكما المسجلات الأخرى ينقسم هذا المسجل الى قسمين بطول ٨ بت هما CL و CH وهو ضمن مسجل أوسع هو ECX بطول ٣٢بت.

المسجل DX (**مسجل المعطيات**-Data Register) :
يفضل أستخدام هذا المسجل لتخزين المعطيات في عمليات الدخل والخرج والمقاطعات وبالطبع فأنه يمكن أستخدامه 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 ٣٢بت.

[٢]مسجلات الأقسام Segment Registers :
كانت العنونة الحقيقية في معالجاتلا ١٦بت تتم بأستخدام خطوط عرض ٢٠بت وهي تكفي لعنونة ١ميجابايت من الرام فقط ولصعوبة التأشير للرام بأستخدام مسجلات من ١٦بت نشأت فكرة الأقسام والعنونة المنطقية وقد قسمت الرام لمقاطع كل منها بطول ٦٤كيلوبايت(الحد الأقصى الذي يمكن عونته ب ١٦بت) وهذه الأقسام لايبدأ كل واحد فيها بعد الآخر وأنما هي متداخلة حيث يبدأ كل ١٦بت قسم جديد وللتأشير على موقع ما يلزمنا عنوا ن المقطع والذي بطول ١٦بت بالأضافة الى قيمة الأزاحة من بداية هذا المقطع وهي بطول ١٦بت أيضاً لذلك لجأ مصمموا المعالج على وضع مسجلات خاصة بالأقسام الشائعة في البرنامج وهي قسم الشفرة Code Segment وقسم البيانات Data segment وقسم المكدسه Stack Segment وقسم المقطع الأضافي Extra Segment والرغم أنه في معالجاتلا ٣٢بت يمكن العنونة بأستخدام ٣٢بت ذاكره حقيقية أ ي مايساو ي ٤جيجابايت من الرام الا أ ن طريقة الأقسام مازالت موجودة حتى يتم خزن عنواين كثيرة بأستخدام ٢بايت بدل ٤بايت داخل المقطع الواحد مع وجود عنوا ن مقطع واحد فقط مخزن قي المسجل المناسب .(ملاحظة : تم في معالجات لا٣٢بت أضافة مسجلين أقسام جديدين بطول ١٦ بت أضافة الى مسجلات الأقسام السابقة والمسجلين هما FS و GS هذا ن القسمان يمكن أستخدمهما كما المسجل ES كمسجل قسم بيانات أضافي)

قسم الشفرة Code Segment -CS:
يحمل هذا المسجل عنوا ن بداية القسم الخاص بالشفرة في البرنامج .

قسم البيانات Data Segment -DS :
يحمل هذا المسجل عنوا ن بداية قسم البيانات في البرنامج .

قسم المكدسة Stack Segment -CS :
يحمل هذا المسجل عنوا ن بداية قسم المكدسه في البرنامج .

قسم الأضافي Extra Segment -ES :
يحمل هذا المسجل عنوا ن بداية قسم أضافي يمكن أ ن يستعمل هذا القسم الأضافي كقسم بيانات آخر.

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

مسجل مؤشر القاعدة Base Pointer -BP :
يعمل هذا المسجل على تسهيل الوصول الى الوسيطات(البارمترا ت) والتي تحتوي على عنواين ومعطيات والتي دفعت PUSH بشكل مؤقت الى المكدسه عند أستدعاء روتيمات فرعية من البرامج مع وسيطات ممرة ، وسع هذا المسجل في معالجات لا٣٢بت وأصبح جزء من مسجل أوسع بطول ٣٢بت هو EBP.

مسجل مؤشر المكدسه Stack Pointer -SP :
يحتوي المسجل SP على كلمة الذاكرة الحاليه التي ستعالج في المكدس ، وسع هذا لمسجل في معالجاتلا ٣٢بت ليصبح جزء من مسجل أوسع بطول ٣٢بت 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 ١٦بت تستخدم في عنونة بيتات مقطع البيانات وكذلك في عمليات التأشير الى السلاسل النصية Strings ، وهناك مسجلات دليليان هما SI و DI وعادة مايستخدمان معاً دائماً بغيه تنفيذ عملية ما .

المسجل الدليلي المصدر Source Index -SI :
يستخدم هذا المسجل في التأشير على النص المصدر وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في في عنونة بيتات مقطع البيانات ،وسع هذا لمسجل في معالجاتلا ٣٢بت ليصبح جزء من مسجل أوسع بطول ٣٢بت هو ESI .

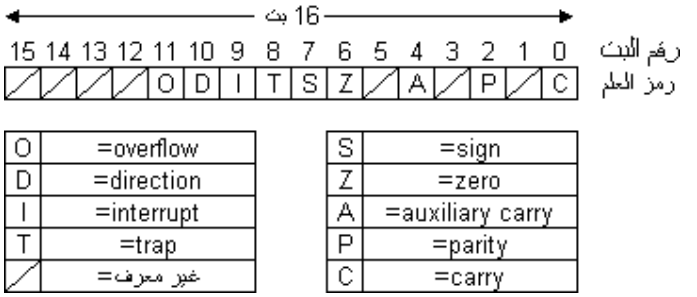
المسجل الدليلي الهدفي Destination Index -DI :
يستخدم هذا المسجل في التأشير على النص الهدف وذلك لأجراء العمليات التي تتعامل مع نصوص وكذلك يستخدم في في عنونة بيتات مقطع البيانات ،وسع هذا لمسجل في معالجاتلا ٣٢بت ليصبح جزء من مسجل أوسع بطول ٣٢بت هو EDI .

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

مسجل مؤشر التعليمية Instruction Pointer - IP :
يحتوي المسجل IP على أزاحة التعليمية التالية التي ستنفذ ، أ يأ ن المسجل عبارة عن مؤشر الى التعليمية التالية الموجودة في مقطع الشفرة CS-Code Segment لمنفذ حالياً ،وسع هذا المسجل في معالجاتلا ٣٢بت ليصبح جزء من مسجل أوسع بطول ٣٢بت هو EIP.

وما تعليمات القفز والتكرار الى تعديل للمسجل IP .

مسجل الأعلام Flags Register :
وهو مسجل بطول ١٦بت يحتوي على أعلام طول كل منها ١بت فقط وتستخدم لعكس حالة آخر عملية أو للتحكم بعمليات معينه وكل علم له موقع معين فيلا ١٦ بت علماً بأن ٩ فقط منها محجوزة والباقي غير معرفة وليس عليك حفظ مواقع هذه الأعلام لأنك ستستعامل معها بالرمز الذي يرمز لها وكل علم له رمز مكون من حرف واحد فقط يدل عليه ويتبعه أحياناً الحرف F أختصار FLAG للتوضيح أنه علم ، كل علم أما يكون واحد أ يـ SET أو صفر أ يـ CLEAR .



لأعلام الذي طولهلا ١٦بت مخطط ٤_٣ : هذا المخطط يوضح مسجل
وأختصارتها ووقع كل علم من الأعلاملا ٩ المحجوزة

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

١. **علم الأتجاه** (DF=direction flag) :
يؤثر في التعليمات التي تقوم بنقل البيانات مثل MOVS,CMPS,SCAS. عندما يكون العلم ١=UP يأخذ الأنتقال أتجاه الطبيعي أما عندما يكون ٠=DOWN يأخذ أنتقال البيانات أتجاه معاكساً (قيمة العلم DF عند بداية البرنامج =١). لضبط العلم بواحد نستخدم التعليمة std ولضبطه بصفر cld .

٢. **علم المقاطعه** (if=intrreupt flag) :
يحدد هذا العلم اذا ماكان بمقدور النظام أجراء مقاطعات أو لا ، ويضبط هذا العلم بواسطة أجهزة الهاردوير وكذلك وقت النظام ، تستطيع أنت ضبطه أو تصفيره اذا كنت تريد حدو ث مقاطعات خارجية أو لا ، اذا كانت قيمة العلم=١ فهذا يعني مفعل enable ويمكن أجراء المقاطعات أما اذا كان صفر فأنه غير-مفعل disabled ولايمكن أجراء المقاطعات (قيمة العلم IF عند بداية البرنامج =١). لضبط العلم بواحد نستخدم sti ضبطه بصفر CLI .

٣. **علم المصيدة** (tf=trap flag) :
يتيح هذا العلم وضع المعالج في نمط الخطوة الواحدة في الوقت الوحد (single step mode) مما يسمح لبرامج فحص الأخطاء كالديبغر بتتبع البرنامج ، اذا كانت قيمة العلم واحد=ON فأن النمط يعمل أما اذا كان صفر=off فأن النمط لايعمل (قيمة العلم TF عند بداية البرنامج =٠) .

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 :

١. **علم الحمل** (flag cf=carry) :
يضبط هذا العلم=١ اذا كان نتيجة آخر عملية كبيرة جداً على الهدف أو المقصد (في الأعداد التي بدو ن أشارة فقط) ، مثال هذا البرنامج :

```
mov ah,٢٠٠
add ah,١٠٠
```

بما أ ن المسجل AH هو ٨بت فأن أقصى قيمة يتحملها هي ٢٥٦ وبما أ ن القيمة في المسجل هي ٢٠٠ ثم أضفنا لها ١٠٠ فأن الجوا ب أكبر من الهدف (aH هنا) لذلك العلم CF سوف يضبط=١ بعد عملية الجمع
لضبط لعلم برمجياً stc ولتصفيره clc

٢. **علم الفيض** (OF=overflow flag) :
هو نفس علم الحمل لكن مع العمليات ذ ي الأشارة أ ي أنه يضبط اذا كان ناتج آخر عملية أكبر أو أصغر من حدود الهدف ، مثال :

```
mov ah,-١٠٠
add ah,-٥٠
```

بما أ ن أصغر قيمة يتحملها المسجل AH هي -١٢٨ لكن ناتج العملية -١٥٠ فأن علم الفيض يضبط=١

٣. **علم الأشارة** (sf=sign flag) :
يضبط هذا العلم اذا كان ناتج آخر عملية رياضية أو منطقية سالب ويصفر اذا موجب (في الواقع أ ن العلم نسخة من البت الأخيرة للجوا ب(بت الأشارة) - كما ذكرنا سابقاً فأن العدد سالب اذا البت الأخيرة ١ وموجب اذا صفر) .

٤. **علم الصفر**(zf=zero flag) :
يضبط هذا العلم=١ اذا كانت نتيجة آخر عملية رياضية أو منطقية تساو ي صفر .

٥. **علم الحمل المساعد**(af=auxiliary carry flag) :
يضبط العلم=١ اذا تسببت آخر عملية رياضية أو منطقية حمل من البت الثالثة الى البت الرابعة أو أستلاف من البت الرابعه الى البت الثالثه . هذا العلم لاتوجد له فائدة واضحةوهو قليل الأستخدام برمجياً .

٦. **علم الأزدواجية - التحقق** (pf=parity flag) :

ببساطة يضبط=١ هذا العلم اذا كان عدد الوحايد في ناتج آخر عملية رياضية أو منطقية زوجياً وبصفر اذا كان فردياً ، مثال لو كان جوا ب آخر عملية = ٠٠١٠٠٠٠١٠ فأن العلم سوف يضبط=١ لأن عدد البتات التي تحتوي وحايد =٢ وهو عدد زوجي أما اذا كان الجوا ب مثلاً = ١١١٠٠٠٠٠ فأن العلم يصفر لأن عدد البتات التي تحوي وحايد=٣ وهو عدد فرد ي . وكما علم الحمل المساعد AF فأن أستعمالة قليل برمجياً ويستخدم عادة من قبل نظام التشغيل لأدارة الذاكرة وكذلك برامج الأتصال لتحقق من سلامة البيانات المرسلة .

لاحظ أ ن كل من الأعلام [علم المصيدة (flag tf=trap)/علم الفيض (OF=overflow flag)/علم الأشارة (sf=sign flag)/علم الصفر(flag zf=zero)/علم الحمل المساعد(af=auxiliary carry flag)/علم الأزدواجية - التحقق (pf=parity flag)] لايوجد لهم تعليمات مباشرة لضبطهم أو تصفيرهم وتحتاج أ ن تستخدم طريق فيها أنحناءة بسيطة لتعديل قيم هذه الأعلام سوف يتم شرحها في دروسـ قادمة علماً بأن البرامج العادية لن تحتاج لتعديل قيم هذه الأعلام وكل ماستحتاجه هو قرآة القيم التي بها .

-

تعليمات وأوامر الأسمبلي :

تتكون التعليمه الواحدة في الأسمبلي من تمثيل بسيط بالأحرف الأنجليزيه يقابله بالأرقام تعليمة لغة آله ، تتكون كل تعليمة من مما يلي : أولاً جزء الأمر وهو أمر يدل على نوع العملية المطلوبة مثل ADD (للجمع) ، الجزء الثاني هو الوسائط علماً بأن بعض التعليمات لايأخذ وسائط والجزء الآخر وسيطة واحدة فقط والبعض الآخر أكثر من ذلك ، تحدد هذه الوسائط الشئ الذي سيعمل عليه الأمر ،
فالأمر ADD لوحده عقيم لايدل على شئ لكن الأمر ADD AX,٥ يدل على جمع الرقم ٥ مع القيمة الموجودة في المسجل AX ويوضح المثال التالي بعض الأوامر

```
clc          ; وسائط فقط أمر بدو ن
dec ax       ; وسيطة واحدة فقط
mov cx,dx ; وسيطتين
```

لاحظ أ ن أ ي نص في شفرة الأسمبلي يأتي بعد الفاصلة المنقوطة هو مجرد تعليق

الوسائط ممكن تكون عدة أنواع :

١. معلومة فورية (مباشرة) (أ ي ثابته) مثال : ١٠ / ٣٠ / 'a'
٢. مسجل (سوف يتم شرح المسجلات بالتفصيل في الدروسـ القادمه) مثال : AX / EAX / BL
٣. موقع ذاكره (يتم تحديده عن طريق العنوا ن) مثال : [٢٠٠] / [] / [١٠٠bx]
٤. متغير (وهو نفس السابق لكن بدل أ ن تحفظ أو تحسب العنوا ن يدوياً يقوم الأسمبلر ب بأستبدلا المتغير برقم يدل علىع نوانه) مثال : VAR / count١ / STR / INTVAL /١

مدخل الى الديبغر Debugge :

ها قد وصلنا الى واحد من أقوى البرامج المبيته في النظام فبواسطة الديبغ تستطيع عمل أشياء عجيبه وغريبه ، حسناً شغل الدوسـ وعند محث الأوامر أطبع debug ثم أنتر وستظهر لك علامه '-' ليل على أستعداد الديبغر على أستقبال أوامرك .

الأن دعنا نكتب هذا البرنامج الصغير

```
mov ax,٢   ; مباشرة الى المسجل أ  يـأكس نقل العدد ٢ كمعلومة
mov bx,٣   ; المسجل بي-اكس نقل العدد ٣ كمعلومة مباشرة الى
add ax,bx  ; أ  يـاكس / أ  يـاكس=أ  يـأكس + بي- جمع أ  يـاكي مع بي-اكس مع وضع الجوا ب في
أكس
```

كيف تقوم بأدخال هذا الكود :
١. عند المحث '-' أدخل ١٠٠a أ ي أننا سنبدأ نكتب الكود من العنوا ن ١٠٠ ثم أضغط أنتر بالطبع
٢. الآن أدخل كل تعليميه ثم أضغط أنتر ومع نهاية التعليمية الأخيرة أضغط 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 01D8          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
-
```

الأن قم بأدخال الرمز R ثم أنتر لترى حالة المسجلات
لاحظ أ ن المسجل AX يساو ي صفر وسترى أيضاً ظهور التعليميه ٠٠٠٢MOV ax, وهي التعليميه التي عليها الدور في التنفيذ وليس المعلومة المنفذه ، الآن قم بطباعة الرمز T ثم أنتر لتنفيذ التعليميه التي عليها الدور هنا هي ٠٠٠٢MOV AX, سترى الآن أ ن المسجل AX أصبح يساو ي ٢ وهذا مانتوقعه بالضبط وسترى أيضاً التعليميع التي عليها دور التنفيذ وهي ٠٠٠٣MOV BX, أدخل الرمز T ثم أنتر لتنفيذها لترى أ ن المسجل BX أصبح يساو ي ٣ وسترى أيضاً التعليميه التي عليها الدور في التنفيذ وهي ADD AX,BX قم بأدخال الرمز T لتنفيذها ولاحظ كيف أ ن المسجل AX أصبحت قيمته مجموع العددين ٣+٢ وهو خمسة بينما بقى المسجل BX يساو ي ٣ .

الآن بعدما عرفت كيف تكتب كود بسيط أخرج من الديبغر 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 U ثم أنتر مباشرة بعد إدخال الكود وقبل إدخال الرمز R)
أضغط R ثم أنتر لترى المسجلات قبل تنفيذ أ ي عملية ولترى التعليميه التي عليها الدور في التنفيذ أضغط T ثم أنتر لتنفذ التعليميه وترى النتائج والتعليميه التي بعدها وهكذا ولاتنسى أذا أرد ت أ ن تدخل كود جديد الخروج والعودة مرة أخرى الى الديبغر لتصفر المسجلات والذاكره

(العمليات الحسابيه المتقدمه و تنقسم إلى ضرب و قسمه :-

الضرب:- Multiplication

يتم ضرب أى رقمين فى لغة الأسمبلى فى ثلاث أنظم ألى الآن :-

- **الأو ل : لا٨-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 الإشارة . و [IMUL] للأرقام المحددة الإشارة . ووضع النتيجه فى لا AX .
مثال : للحصول على حاصل ضرب ١٢٠ * ٣٠ نقوم بالأتى :-

```
;---------------------------------
MOV CL,٣٠D
MOV AL,١٢٠D
MUL CL
MOV [١٠٠٠H] , AX      ; لتخزين الناتج فى الذاكره المعنونه ب١٠٠٠ بالنظام السادس عشر
;---------------------------------
```

- **الثانى : لا١٦-بت (الكلمه**-:)

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

```
;---------------------------------
MOV CX,٢٣٦٥D
MOV AX,٢٠٠٠٠D MUL CX
MOV [١٠٠٠H],AX      ; تخزين الجزء الأو ل فى المكان فى الذاكره المعنون ب ١٠٠٠
MOV [١٠٠١H],DX      ; تخزين الجزء الثانى فى المكان فى الذاكره الذى يلى المكان السابق
;---------------------------------
```

- **الثالث : لا٣٢ بت** :-

يتم وضع إحدى القيم فى EAX و ألآخرفى أى ريجستر آخر له نفس الحجم أو مكان فى الذاكره بجانب كود الضرب و توضع النتيجه(٦٤ بت) فى . EDX:EAX وهذا معناه أ ن الجزء لا Low من ناتج القسمه سيخزن فى لا EAX والجزء لا High سيوضع فى لا . EDX

القسمه-: Division

أيضاً تتم فى نفس النظم السابقه-:

•**الأو ل** : **لا٨بت** (**بايت** -: (

يتم و ضع المقسوم فى AX و المقسوم عليه فى أى ريجستر حجمه بايت مثل BL,CL,DL,...أو مكان فى الذاكره لها حجم بايت مثل BYTE PTR [١٢٣٤H] , BYTE PTR [BX]و يوضع ناتج القسمه فى AL و باقى القسمه فى AH بإستخدام. [DIV]

لاحظ -:أنه فى حالة قسمة الأرقام المحددة الإشارة بإستخدام[IDIV] تكون إشارة ناتج القسمه هى الإشارة العاديه فى هذه الحالات . وتكون إشارة خارج القسمه Remender دائماً موجبه و صحيحه.Integer

لاحظ-:كل القيم هنا ٨-بت إذاً , لابد من تحويل القيمهلا ٨-بت للمقسوم إلى ١٦-بت ليمكن نقلها ألى AX ويتم ذلك فى حالة القيم غير محددة الإشارة بمسحلا AH ليكون كله أصفار و تكون قيمة المقسوم ١٦-بت . أما فى حالة القيم محددة الإشارة 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 الإشارة السالبه عن طريق .[NEG] إذا القيم جاهزه لعملية القسمه ويتم حفظ الناتج فى الذاكره بعدها.

باقى النظم كما سبق من الممكن أ ن 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 . وسنتعلم سوياً بعض من هذه التقنيات -:

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

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

تقوم جمل الشرط عموما على و جود شرط أو إختبار لو تحقق إذاً سيقوم البرنامج بعمل بعض المهام لن يكون فاعلها لولم يتحقق الشرط أو فشل الإختبار , كما تحب أ ن تطلق عليه.

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

الإختبار -:

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

-: [CMP]-**١**يجب أ ن تذكر القيمتين اللتان سوف يتم لهما إختبار بعدها . و عندما تتم هذه

الجمله يتم عمل عملية طرح فى الأصل و لكن مع عدم تغيير فى أو ل مكان (الرجستر أو الذاكره.(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 عبارة تأتى مع القفز كما تم ذكرهم فى الدر سـ السابق.

مثال -: سنقوم إختبار تساوى لقيمتين و ليكن ٥و٤ 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 البرمجيه التى;
. هتتنفذ
.....
HLT          ; تقوم هنا بوقف البرنامج مثلا أو القفز إلى مكان آخر لأنه من غير ذلك سوف
ينفيذ الأوامر التاليه أيضاً يتم
END:
;٤ الكمبيوتر إتجنن . إزاى ٥ تساوى .
.....
.....      تنفذ فى حالة التساوى - طبعاً مش  بعض الجمل البرمجيه التى سوف;
. هتتنفذ
.....
;---------------------------------
```

طبعا يوجد جُمل تقوم بالقفز إذا كان عدم التساوى أكبر من مثلاً أو القيمه الأولى أصغر من الأخرى وهكذا . يوجد الكثير و الكثير كما تحب أ ن تكتب برامجك.

-: [TEST]-**٢**تقوم بالعمل السابق نفسه و لكن بدلاً من عمل عملية طرح , تقوم بعمل Logic ANDو 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 القيمه سالبه
HLT
END:
;إطلاقاً القيمه موجبه . كما هو ليس متوقع.
;---------------------------------
```

المثال السابق يوضح إستخدام الأمر [TEST] فى إختبار آخر بت فى القيمه الموجودة فى AXوهذه البت توضح إشارة الرقم كما لابد أ ن نعرف.فقمت بعمل لا

TEST(AND) WITH ١٠٠٠٠٠٠٠B , IF the most segnificant bit in AX is (١) then zero flag will reset , else it will be set by the processor.

(٣)الدورا ت:- loops

-من أهم قدرا ت الحاسب الألى هى عمل نفس الشىء مرا ت عده و لا تختلف معه عدد هذه المرا ت و هى من أهم جوانب البرمجه أصلاً . فلا يمكن لبرنامج مفيد ألا يحتوى على دورة أو عدة دورا ت لذا كان من المهم أ ن ندرجها هنا و تكون بعد التحكم فى مسار البرامج لما لها من إرتباط وثيق بما سوف نتحدث عليه.

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

الأنواع-:

- **الدورا ت المستمره (إلى مالانهايه)-:**

وهى الدورا ت التى تظل عامله إلى أ ن يتم غلق البرنامج من جانب نظام التشغيل أو غلق الجهاز نفسه . و طبعاً 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 . Squencial programming
-أيضاً من الممكن أ ن تستخدم فى عمل الفيروسات . تكفى دورة صغيره غير منتهيه لإستغلال معظم و قت المعالج من غير فائده و التأثير على باقى البرامج الأخرى فى النظم متعددة البرامج . Multiprogramming and Time-sharing Operating systems

---

- **الدورا ت محددة المرا ت-:**

فى هذه الدورا ت يكون عدد مرا ت تنفيذ الدورة محدد و هى من أكثر الأنواع إنتشاراً و لها إستخدامات عده تتنوع بين العمليات الحسابيه والمنطقيه و باقى العمليات الأخرى.
مثلتها الأ . (LOOP FOR...NEXT)

- **الدورا ت المحدده بشرط معين-:**

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

التقنيات و طرق العمل-:

- **الطريقه البدويه** -:

عن طريق تحديد عنوا ن فى أو ل أمر فى الدورة ثم فى نهاية أوامر الدورة يتم إدراج أمر قفز للعنوا ن الذى وضعناه سابقاً و يكون القفز مشروط أو غير مشروط / (Conditional Unconditional Jump) .

مثال -:

```
;---------------------------------
MOV AL,٥
XOR CL,CL ;CL = ٠
START_LOOP : ;بداية الدورة
ADD CL,AL
DEC AL
```

```
CMP AL,٠٠H ; قارن القيمه الموجوده فى AL والصفر.
JNE START_LOOP
;يساوى صفر AL يتم القفظ لو لم يكون
MOV BYTE PTR [٠١٠٠H],AL ; تخزين الناتج فى الذاكره النهائى .
;---------------------------------
```

فى المثال السابق تم إستخدام الدورة المشروطه START_LOOP فى جمع الأرقام من ١ إلى ٥ و تخزينهم فى الذاكره.

- **الدورات باستخدام الأمر** -: LOOP

هذا الأمر له صور عده منها المشروط و منها الغير مشروط كما لابد أن نرى-:
-:[LOOP] يتم كتابة أسم أو عنوان أو لجمله/أمر فى الدورة , ويتم وضع عدد مرات الدورة فى CX , وتقوم بتقليل CX بواحد وتختبر قيمته لو كانت صفر تنهى الدورة و لو غير ذلك يتم القفز ألى العنوان الموجود أمامها.

مثال -:

```
;---------------------------------
MOV CX,٥
XOR AX,AX
START_LOOP :
ADD AX,CX
LOOP START_LOOP
MOV [٠١٠٠H],AX
;---------------------------------
```

نفس المثال السابق.

Conditional LOOP:-

- LOOPZ
- LOOPNZ
- LOOPE
- LOOPNE

وهذه الأنواع تستخدم بعد CMP/TEST Instructions , وعلى حسب الشرط يتم القفز , ويكون إستخدامها مثل. LOOP

- **الدورات باستخدام**-: REP

وهى تستخدم مع نقل البيانات String Transfer Istructions .

(١)**التعامل مع السلاسل النصيه** -:Strings

السلاسل النصيه هى من أهم البيانات التى نتعامل معها فى نظم الحاسب عموماً و فى لغات البرمجه خصوصاً . من الممكن التعامل معها على أنها بعض الأحرف المتتابعه و التعامل معها كما سبق بالدورات و التكرار إعتماداً على أنها سلسله حرفية تتكون من عدة أحرف معلومة الطول ومتتابعه. ولكى الأسباب تعمل ببعض الدوال التى ستسهل علينا هذا التعامل و تقليل التعقيد و طول البرامج نسبياً.

من أهم هذه الدوال ما يلى -:

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

قبل البدء خذ إنتباهك من الأتى -:

- DI (Data Index/Distination Index) is combined with the ES (Extra Segment) to reach the data place in memory which the distenation od 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 indecates the direction of the flow of the strings in the memory .
    - At D=٠ then Auto-Increment will occur to DI,SI According to the size of the current data (Increment with ١ if byte,٢ if word,٤ if double word).
    - At D=١ then Auto-Decrement will occur to DI,SI According to same previous charactaristics.
    - [STD] Will Set the Direction Flag (D=١).
    - [CLD] Will Clear the Direction Flag (D=٠).
- 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 Form AX to THE Segment.

سنتناول هذه الدوال بالتفصيل 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 فى الذاكره المعنون ب [DS:SI] ثم تقوم بزيادة/نقصان(على حسب علم الإتجاه D لا SI بواحد مما يعنى أنها تقوم بالتالى :
١ -+ SI = SI , AL=DS:[SI] .

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

:- [LODSD] -تنقل إلى EAX محتويات الكان فى الذاكره المعنون ب [DS:SI] ثم تقوم بزيادة/نقصان(على حسب علم الإتجاه D لا SI بأربعه مما يعنى أنها تقوم بالتالى :
٤ -+ SI = SI , EAX=DS:[SI] .

<u>من الجدير بالذكر</u>:-

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

<u>مثال</u>:-

```
;--------------------------------
CLD
MOV AX,٠٠٠٠H
MOV DS,AX           ;نحدد المقطع الذى نتعامل معه الأو ل
MOV CX,١٠٠٠H        ;تحديد عدد مرا ت النقل التى سوف تتم
MOV SI,٠            ;البدأ من أو ل مكان داخل المقطع
MOV AH,٢H           ;التجهيز للطبع
START:
LODSB
MOV DL,AL           ;تجهيز الحرف الذى سوف يتم طبعه
INT ٢١H

REP START

END :
;--------------------------------
```

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

[STOS]

هذه الداله تتعامل أيضاً مع الأنواع المذكورة أعلاه و بنفس الطريقه .مثل-:

-: [STOSB] - تنقل محتوياتلا AL إلى المكان فى الذاكره المعنون ب [ES:DI] ثم تقوم بزيادة/نقصان(على حسب علم الإتجاه D لا SI بواحد مما يعنى أنها تقوم بالتالى :
ES:[DI]=AL , DI = DI +- ١ .

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

-: [STOSD] - تنقل محتوياتلا EAX إلى الكان فى الذاكره المعنون ب [ES:DI] ثم تقوم بزيادة/نقصان(على حسب علم الإتجاه D لا SI بأربعه مما يعنى أنها تقوم بالتالى :
ES:[DI]=EAX , DI = DI +- ٤ .

مثال-:

```
;--------------------------------
CLD
MOV AX ,٠B٨٠٠H
MOV ES,AX           ;نحدد المقطع الذى نتعامل معه الأو ل
MOV DI,٠            ;البدأ من أو ل مكان داخل المقطع
MOV CX,٢٠*٨٠        ;مرا ت النقل التى سوف تتم تحديد عدد
MOV AX,٠V٢٠H        ;نسخها إلى الذاكره تحميل البيانات التى ستتم

REB STOSW           ;تكرار عملية النسخ

END:
;--------------------------------
```

هذا المثال يقوم بمسح الشاشه (٢٠سطر *٨٠ عمود) عن طريق نقل القيمه ٠٧٢٠إلى كل الأماكن فى الذاكره التى تبدأ عند ([B٨٠٠H:٠٠٠٠H]) وهذا المقطع هو مقطع الشاشه و هذا طبعا فى نظام الدوس فقط . ومن الجدير بالذكر هنا أن القيمه ٠٧٢٠ هى عبارة عن جزأين , الجزء الأول ٠٧ هو كود اللون الأبيض على الخلفيه السوداء فى نظام الدوس و الجزء الآخر ٢٠ Hهو الأسكى كود للمسافه الفارغه .
الخلاصه : أننا نطبع فى ذاكره الشاشه المسافه الفارغه فى كل أماكن الشاشه و ذلك لمسح الشاشه بالكامل و هذا المثال مكافىء لأمر الدوس CLS.

[MOVS]

هذه الداله الوحيده فى جميع المعالجات من ٨٠٨٨ إلى بنتيوم ٣ التى تسمح بعمليات النقل من مكان فى الذاكره إلى مكان آخر فى الذاكره.
خذ حذرك من الآتى- :

لا نستطيع عمل أى عمليات نقل أو غيره بحيث تستلزم من المعالج النظر ألى الذاكره 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 الذاكره المعنون ب [ES:DI]إلى المكان المعنون ب [DS:SI] . وتم ذيادةلا DI,SI بواحد أو نقصانهم بناءاً على علم الإتجاه. D FLAG
مما يعنى أنها تقوم بالأتى. ٢ -+ DI = DI ; ٢-+ SI=SI ; ES:[DI] = DS:[SI] :

-: [MOVSW] -تستخدم فى نقل كلمه من محتويات المكان فى الذاكره المعنون ب [ES:DI]إلى المكان المعنون ب [DS:SI] . وتم ذيادةلا DI,SI بواحد أو نقصانهم بناءاً على علم الإتجاه. D FLAG
مما يعنى أنها تقوم بالأتى. ٢ -+ DI = DI ; ٢-+ SI=SI ; ES:[DI] = DS:[SI] :

-: [MOVSD] -تستخدم فى نقل كلمتان من محتويات المكان فى الذاكره المعنون ب [ES:DI]إلى المكان المعنون ب [DS:SI] . وتم ذيادةلا DI,SI بواحد أو نقصانهم بناءاً على علم الإتجاه. D FLAG
مما يعنى أنها تقوم بالأتى. ٢ -+ DI = DI ; ٢-+ SI=SI ; 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,٠
```

```
MOV SI,٠
MOV CX,٦٤ * ١٠٢٤D

REP MOVSW

END :
;--------------------------------
```

كما توقعتم , المثال يقوم بنسخ المقطع ١٠٠٠ بالكامل إلى المقطع ٢٠٠٠ عن طريق تكرار نقل الكلمات عدد ٦٤ KWORDS و هو مساحة المقطع الواحد كما تعلمون.

[INS]

ألم تتساءل أنه من المستحسن أن تكون هناك داله تقوم بالمهام التالية و تكون مهمتها إدخال أى بيانات من وحدة الإدخال المعروفه (الشاشه / الهارد ديسك مثلا) .!!؟ هذه الداله تقوم بهذا العمل بتنظيم أكبر و مرونه أكثر و تتيح لك أيضاً التكرار عن طريق إستخدام REP Instruction.

شرح كيفية العمل :

- يتم وضع عنوان وسيلة الإدخال التى سوف تستخدمها فى DX
- تجهيز عنوان المكان فى الذاكره الذى سوف يستقبل البيانات فى [ES:[DI
- تجهيز علم الإتجاه على حسب متطلبات البرنامج .
- إختبار هل توجد بيانات حاضره فى وسيلة الإدخال أولاً قبل طلب هذه البيانات .
- مكافئه للأتى : ES:[DI] = [DX] ; DI = DI +- ١

الداله تمكنك من نقل بايت عن طريق , [INSB] وإدخال كلمه بإستخدام , [INSW] وإدخال كلمتين بإستخدام [INSD].

مثال :-

```
;--------------------------------
MOV DI,٠
CLD
MOV AX,١٠٠٠H
MOV ES,AX
MOV DX,٣ACH          ;وسيلة الإدخال المستخدمه عنوان
MOV CX,١٠٠

REP INSB

END :
;--------------------------------
```

طبعاً سهل.!!!

[OUTS]

نفس طريقة الإدخال و لكن للإخراج و هنا نستخدم لعنوان المكان فى الذاكره موجود فى [DS:[SI كما توقعتم.

مثال :-

```
;--------------------------------
MOV SI,٠
CLD
MOV AX,١٠٠٠H
MOV DS,AX
MOV DX,٣ACH             ; وسيلة الإدخال المستخدمه وليكن عنوا ن الهارد ديسك عنوا ن
MOV CX,٥١٢

REP OUTB

END :
;--------------------------------
```

المثال يمثل نقل بلوك بيانات من الذاكره [DS:SI] إلى الهارد ديسك.

[SCAS]

هذه الداله تقوم بمقارنة محتويات المكان فى الذاكره الذى عنوانه موجود فى [ES:DI] و محتوياتلا ALعند إستخدام , [SCASB] وقارنها بمحتوياتلا AX عند إستخدام , [SCASW] وقارنها بمحتوياتلا EAX عند إستخدام. [SCASD] وطبعاً هناك ذيلها أو نقصان فى DI كماتعرفنا سابقاً.

مثال :-

```
;--------------------------------
CLD
MOV DI,٠
MOV CX,٢٠٠٠
XOR AX,AX
MOV ES,AX

REBNE SCASW

END :
;على المكان الفارغ أو القصود البحث عنه أو سيحتوى فى حالتنا على ٢٠٠٠  سيحتوى DI البحث بعد إنتهاء
;--------------------------------
```

فى المثال نبحث فى أو ل ٢٠٠٠ مكان فى الذاكره عن أو ل مكان فارغ أو يحتوى على ٠٠٠٠ H

[CMPS]

هذه الداله فعلاً عظيمه لأنها تقار ن مكانان فى الذاكره فى أمر واحد و تتوقف مساحة هذا المكان على إستخدام الأمر. [CMPSD] , [CMPSW] , [CMPSB]

- الأو ل يوضع فى.[ES:DI]

•الثانى يوضع فى.[DS:[SI

مثال

```
;--------------------------------
CLD
MOV DI,٠
MOV SI,٠
MOV CX,٢٠٠٠
XOR AX,AX
MOV ES,AX
MOV AX,١٠٠٠H
MOV DS,AX

REBE CMPSW

END :
```

التعامل مع الإجراءا ت-: Procedures

- **ما هى الإجراءا ت Procedures ؟**

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

- **مميزا ت الإجراءا ت-:**
  - الإجراءا ت من أكثر إمكانيات اللغات البرمجيه 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 تبلد ل أجزاءها بين أكثر من شخص أو مبرمج.
  - يتم بناء البرامج والتطبيقات الكبرى عليها.
  - تساعد فى سرعة تصحيح الأخطاء . Error Handling
  - أساس عمل البرمجه الكائنيهOOP) (Object Orianted Programming

- **كيفية عمل الإجراءا ت ...؟**

يتم عمل نداء للإجراء المراد الوصول إليه عن طريق [CALL Procedure_name] من داخل الإخراء 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 الجمله البرمجيه التى تلى الأمر [CALL] فى الإجراء الذى قام بالنداء أولاً.

خذ إنتباهك من الأتى -:

الإجراء الرئيسى-: Main procedure

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

- **كيفية إنشاء الإجراءا ت...؟**
  - يتم إنشاء إجراء جديد عن طريق كتابة أسم الإجراء الجديد يليه كتابة الأمر PROC.
  - أكتب بالداخل جمل الإجراء أياً كانت على حسب إختيار ك و على حسب عمل الإجراء .
  - ينتهى الإجراء بكتابة أسم الأجراء يليه الأمر.RET
- **ملاحظات على الإجراءا ت**-:
  - لا يمكن كتابة إجراء داخل أجراء آخر . وإلا فلن يعمل صحيحاً وسوف يعطى المترجم Assembler رسالة خطأ . Error لذلك فأنه يجب كتابة الإجراء الجديد أما بعد إنتهاء إجراء أو قبل بدأ إجراء.
  - لا يجب أ ن يكون الإجراء فى نفس البرنامج و لكن يمكن إضافته إلى البرنامج و هو يكون داخل ملف آخر فى الهارد أو فى الذاكره مثل إجراءا ت النظام التى نعمل لها نداء عن طريق INT وهى فى هذه الحاله يطلق عليها . Software Interrupts ولكن الأخرى التى فى ملف آخر تسمى Macros or Modules . ومن الممكن أ ن تكون فى حالتها المصدريه أو فى حالتها القابله للتشغيل مثل. DLL'S) Dynamic link liberaries)

إلى هنا ينتهى حديثنا عن الإجراءا ت . وسنستعرض برنامجاً كاملاً مفتوح المصدر يحتوى على إجراءا ت عده . فهيا بنا

هذا البرنامج يقوم بالأتى -:

١. يقوم بطلب البيانات المطلوب تشفيرها من المستخدم.
٢. يقوم بعمل أختبار 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 المشفره.
٥. ويقوم بعرض البيانات نفسها ثانياً.

كما إستنتجتم فأن البرنامج يقوم بعمل تشفير لبيانات من المستخدم ويعرض له البيانات مشفره .

البرنامج

```
;-------------------------------------------------------------
;App_name : DECODER.EXE
;Author : mrezzuz
;web site : http://www.١t.٢t.٣t.٤t.com
;Date : ٢٠/٤/٢٠٠٣
;-------------------------------------------------------------
Title decoder
.MODEL small
.STACK ١٠٠H

.DATA

    LF EQU ٠AH
   CR EQU ٠DH
   MSG DB ١٠٠٠H DUP (٠)  ;Encode up to ٤ 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 ',١h,' ?[Y For agree]',LF,CR,'$'
   CODE_VAL DB ٩BH

.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,٩
    INT ٢١H
    LEA DX,WELCOM    ;print the welcom message
    INT ٢١H
;start to read the user input
    LEA BX,MSG

READ_BYTE :
    MOV AH,١       ;input one char
    INT ٢١H
    CMP AL,CR     ;compare end of user inputs
    JE END_READ
    CMP AL,٠٨H   ;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,٣٢    ;Print space and backspace to clear the backed char.
    MOV AH,٢
```

```
    INT ٢١H
    MOV DL,٨
    INT ٢١H
    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 > ٠٠h
and else ask for again
    CMP CX,٠٠H
    JLE ask_again
    CALL DEC_INC        ;calls xor function  TO ENCODE.
    MOV AH,٢ ;print new line
    MOV DL,LF
    INT ٢١H
    MOV DL,CR
    INT ٢١H
    LEA   DX,CODE_MSG
    MOV AH,٩
    INT ٢١H
    CALL PRINT_MSG   ;calls msg printting which was stored in msg
    CALL DEC_INC        ;call xor TO DECODE .
    LEA DX,MSG_MSG
    MOV AH,٩
    INT ٢١H
    CALL PRINT_MSG

ask_again:
   LEA DX,AGAIN
   MOV AH,٩
   INT ٢١H
   MOV AH,١   ;scan what's user replay . if Y or y then again else
exit
   INT ٢١H
   CMP AL,'Y'
   JE START
   CMP AL,'y'
   JE START

END_MAIN:   ;Exit to dos
    MOV AH,٤CH
    INT ٢١H

    MAIN ENDP
;_____________________________________________________________
PRINT_MSG PROC
;The procedure prints the bytes sequancly from the msg address CX
times.
;CX must > ٠٠h
;The procedure uses BX as an offset
    PUSH CX   ;store CX
    MOV AH,٢ ;print new line
    MOV DL,LF
    INT ٢١H
    MOV DL,CR
    INT ٢١H
    LEA BX,MSG
START١:
    MOV DL,[BX]   ;start print first char of msg address
```

```
    INC BX   ;icrement the offset
    INT ٢١H
    LOOP START١
    MOV AH,٢ ;print new line
    MOV DL,LF
    INT ٢١H
    MOV DL,CR
    INT ٢١H
    POP CX   ;return CX
    RET
PRINT_MSG ENDP
;_____________________________________________________________________
DEC_INC PROC
;The procedure XOR the byte from address msg CX times .
;CX must > ٠٠H
;The procedure uses BX as an offset
   PUSH CX
   LEA BX,MSG
START٢:
   MOV AH,CODE_VAL   ;moves the code value into AH
   XOR [BX],AH   ;XOR with offset BX
   INC BX
   LOOP START٢
   POP CX
   RET  ;return
DEC_INC ENDP
   END MAIN   ;start the program control from MAIN procedure
```