

الفريق العربي للهندسة المكتبية

دورة ATRE لتعليم الأسهل للمبتدئين من الصفر الدرس الثالث -- نظام التشغيل DOS – العنونة في البرامج

مقدمة

بداية أود أن أشير إلى أن أجزاء كبيرة من هذا الدرس مأخوذة – حرفياً – من درس للأخ xacker فله الشكر... طبعاً مع القليل من التصرف.

نظام التشغيل DOS

يمثل DOS نظاماً لتشغيل الحاسوب يؤمن عملية استخدام موارد وأجهزة الحاسوب دون الخوض في تفاصيلها التقنية والفيزيائية . تشمل هذه الأجهزة لوحة المفاتيح ، الشاشة وسواقات الأقراص . وعلى هذا فإن DOS مسئول عن المعالجة الأدنى مستوى لموارد وأجهزة الحاسوب . ومن بين هذه الوظائف الكثيرة للنظام سنهتم بمجموعة الوظائف التالية :

- **إدارة الملفات** : يعتبر نظام DOS مسئولاً عن إدارة وتنظيم الفهارس directories والملفات الموجودة على القرص. فجميع برامجنا تستطيع إنشاء وتعديل الملفات عن طريق خدمات نظام DOS (هذا بالنسبة لتطبيقات 16 بت طبعاً) تاركين في ذلك عبء عملية إدارة مواقع تخزينها إلى DOS.
- **Input/output** : يريح نظام DOS المبرمج من عنا الخل / الخرج المباشر ، فهو يسمح بطلب إدخال/إخراج المعطيات عن طريق المقاطعات Interrupts .
- **تحميل البرنامج** : يؤمن DOS تنفيذ أي برنامج يريده المستخدم. فهو يقرأ البرنامج من القرص ليحمله في مكان في الذاكرة ثم يسلم التنفيذ له.
- **إدارة الذاكرة** : عندما يحمل DOS أحد البرامج لتنفيذه ، يخصص قبل ذلك حيزاً كافياً من الذاكرة لشفرة التعليمات والمعطيات ، ويسمح للبرنامج بطلب المزيد من الذاكرة أو تحريرها .
- **المقاطعات Interrupts** : يؤمن DOS عملية الاتصال بينه وبين المبرمج عن طريق المقاطعات ، كما يترك لنا حرية تغيير خدمات هذه المقاطعات ، فعلى سبيل المثال يمكننا ترك البرنامج مقيماً في الذاكرة (Terminate-Stay Resident) وجعله جزءاً من نظام المقاطعات لإنجاز وظائف معينة.

عملية الإقلاع The Boot Process

ينشأ عن إعادة وصل الكهرباء إلى الحاسوب حدوث عملية تسمى الإقلاع البارد cool boot والتي ينتج عنها دخول المعالج في حالة إعادة التهيئة التي تفرض عليه تصفير كافة مواقع الذاكرة وإجراء تدقيق لخانات التحقق في الذاكرة ، ومن ثم تحميل المسجل CS بالعنوان FFFF[0]h والمسجل IP بالقيمة صفر ، وعلى هذا فإن التعليمات الأولى التي ستجلب من الذاكرة تقع عند العنوان FFFF[0]h والتي تشكل البداية لروتينات BIOS (Basic Input Output System) في الذاكرة ROM . تبدأ بعدها روتينات BIOS بفحص المنافذ المختلفة بغية التعرف على كافة الأجهزة الموصولة بالحاسوب وتتهيئتها. تبني روتينات BIOS بعد ذلك منطقتين في الذاكرة وهما :

- 1- جدول خدمة المقاطعات Interrupt Services Table : تحتوي هذه المنطقة على عناوين برامج خدمات المقاطعات ، وذلك ابتداءً من العنوان 00000h في الذاكرة .
- 2- منطقة معطيات BIOS : وهي منطقة تبدأ عند العنوان 00400h من الذاكرة وتضم معلومات عديدة عن الأجهزة المحيطة بالحاسوب.

تبحث روتينات BIOS بعد ذلك عن قرص النظام في سواقات الأقراص لينتقل التنفيذ في حال العثور عليه إلى برنامج التحفيز الذاتي bootstrap الموجود في ذلك القرص . يحمل البرنامج الأخير الملفين io.sys و msdos.sys من القرص إلى الذاكرة وينتقل التنفيذ نقطة إلى نقطة بداية التنفيذ في io.sys . يعيد io.sys

توضعه في الذاكرة وينقل التنفيذ بدوره إلى msdod.sys . يهيئ الأخير جداول معطيات DOS الداخلية والقسم الخاص للنظام من جدول عناوين خدمات المقاطعات ثم يقرأ الملف config.sys من القرص وينفذ أوامره. أخيرا ينقل msdos.sys التنفيذ إلى command.com الذي بدوره سينفذ أوامر الملف autoexec.bat ويظهر محث الأوامر ويراقب المعطيات المدخلة من لوحة المفاتيح. هذا طبعا بالنسبة لنظام DOS القديم التقليدي.

Loading System Program

يدعم النظام DOS نوعين اثنين من البرامج التنفيذية exe و com . يتألف برنامج com من مقطع وحيد عليه أن يضم شفرة التعليمات والمعطيات والمكدس. وهو نوع مناسب للبرامج الصغيرة الخدمية والبرامج المقيمة في الذاكرة. أما برامج exe فهي تتألف من مقاطع منفصلة هي مقطع شفرة التعليمات والمعطيات والمكدس (على الأقل... قد يكون هناك مقاطع أخرى) وهي تناسب البرامج الأكثر جدية. يؤمن DOS عملية التعامل مع كلا النوعين السابقين من خلال برنامج التحميل command.com الذي يقوم بتحميل البرنامج التنفيذي إلى الذاكرة قبل تسليمه التحكم. وبالطبع سيختلف تعامله مع كلا النوعين بسبب طبيعة اختلافهما.... سنستعرض الآن الخطوط العريضة التي ينفذها برنامج التحميل عند طلب تنفيذ برنامج من النوع exe وسنؤجل الحديث عن ملفات com إلى وقت لاحق.

تشمل خطوات تنفيذ ملفات exe ما يلي :

- 1- الوصول إلى برنامج exe الموجود على القرص.
- 2- بناء منطقة تسمى بادئة مقطع البرنامج (Program Segment Prefix (PSP والتي تتألف من 256 بايت على حدود فقرة ضمن الذاكرة المتوفرة للبرنامج
- 3- تحميل البرنامج بعد منطقة PSP مباشرة.
- 4- تحميل المسجلات DS و ES بعنوان مقطع PSP
- 5- تحميل المسجل CS بعنوان مقطع شفرة التعليمات للبرنامج ، والمسجل IP بإزاحة التعليمات الأولى والتي تساوي عادة 0 في مقطع شفرة التعليمات.
- 6- تحميل المسجل SS بعنوان مقطع ال stack والمسجل SP بحجم ال stack المطلوب
- 7- تسليم التنفيذ إلى البرنامج ابتداء من نقطة بداية التنفيذ (Original Entry Point (OEP

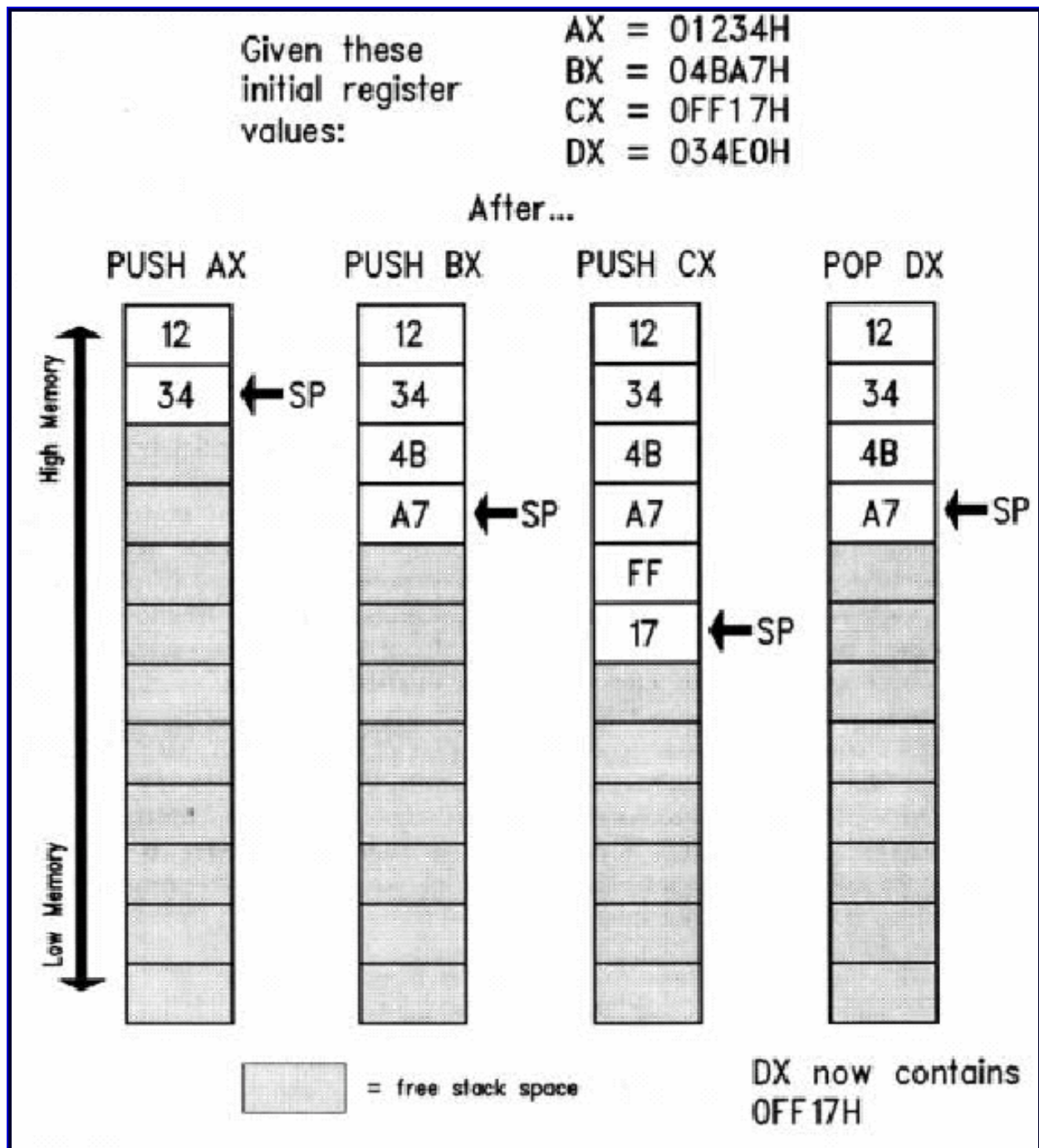
لاحظ أنه قد تم بهذه الطريقة تحميل زوجي المسجلات CS:IP و SS:SP بالقيم المناسبة الصحيحة إلا أن المسجل DS لا يحتوي على عنوان مقطع المعطيات وكذلك الأمر مع ES فكلاهما يحملان عنوان مقطع PSP وعلى هذا يمكن الاستنتاج أن على البرنامج exe أن يغير من قيمة DS و ES بنفسه. للوصول إلى مقطع المعطيات في البرنامج .

المكدس Stack

تحتاج البرامج التنفيذية إلى منطقة محجوزة في الذاكرة تدعى stack. تفيد هذه المنطقة في توفير حيز من الذاكرة لتخزين العناوين والمعطيات بشكل مؤقت ، علما أن طول عنصر المعطيات الأساسي في stack هو كلمة واحدة word. ينبغي على المبرمج تعريف ال stack في برامج exe في حينه إلا أنه لا حاجة لذلك في برامج com إذا يتولى نظام التشغيل تعريف مكدسة آليا. يهيئ DOS مسجل المقطع SS بعنوان بداية مقطع ال stack، كما يهيئ المسجل SP بحجم ال stack ليشير إلى نهايته أو قيمته الحالية.

تتعامل مجموعة من التعليمات مع ال stack بشكل مباشر كالتعليمات POP, PUSH, CALL... والتعليمات PUSH مثلا تنقص محتويات المسجل SP بمقدار 2 عند تنفيذها وتحزن قيمة ما في تلك الكلمة المحجوزة التي أصبح SP يشير إليها . أما POP فهي على العكس حيث تنقل الكلمة التي يشير إليها SP إلى موقع ما وتزيد قيمة المسجلين AX و BX إلى قمة ال stack ثم سحبها من قمة ال stack إلى نفس المسجلين.

والآن تمعن في الصورة التالية...هناك قيم مخزنة في المسجلات العامة الأربعة ، ونريد أن نرى ماذا يحدث أثناء تنفيذ بعض التعليمات المتعلقة بالمكدس...



لاحظ أننا إذا حفظنا قيم المسجلات AX,BX,CX,DX على الترتيب بواسطة التعليمات التالية :

Push ax
Push bx
Push cx
Push dx

فعلينا أن نسترجعها كالتالي:

Pop dx
Pop cx
Pop bx
Pop ax

يسمى هذا بمبدأ **LIFO (Last In First Out)** أي أن آخر قيمة تنقل لل stack هي أول قيمة تخرج منه. لاحظ أن القيم التي تم استرجاعها من ال stack تبقى موجودة فيه لكن لا مؤشر عليها المسجل SP ، مما ينبغي ذكره هنا هو وجود حالتين يجب ألا يمر فيها ال stack . الأولى هي محاولة سحبنا قيمة ما من ال stack وهو في حالة underflow أي فارغ يحتوي على القيمة FFFFh) أما الحالة الثانية فهي محاولة دفع قيمة لل stack وهو ممتلئ في حالة overflow (أي SP تحوي القيمة 0) ولذلك ينبغي توخي بعض الحذر في تعاملنا مع ال stack . يشكل ذلك مطلباً بالغ الأهمية لعمل البرنامج في حالة تعطل مفاجئ وانتقال النظام إلى حالة غير مستقرة تستدعي غالباً إعادة استنهاض أو إقلاع الحاسب من جديد.

العنونة في البرنامج : Addressing

تستطيع المعالجات x86 تنفيذ عمليات الوصول إلى كلمات الذاكرة بفعالية أكبر إذا كانت عناوينها زوجية ففي تعليمة MOV يستطيع المعالج الوصول إلى الكلمة ذات الإزاحة 0012 مثلاً بشكل كامل ونسخها مباشرة إلى المسجل AX مثلاً لأنها تملك عنواناً زوجياً أما إذا كانت الكلمة ذات عنوان فردي كعنوان الإزاحة 0013 عندئذ سينفذ المعالج خطوتين للوصول إلى هذه الكلمة التي سنفرض أن محتوياتها كما في الشكل التالي :

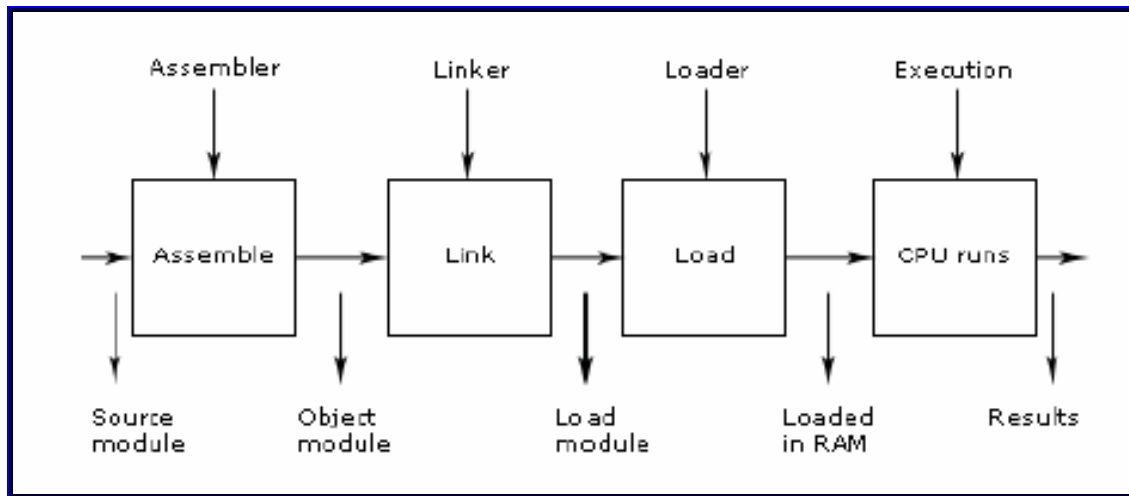
xx	23	01	xx	محتويات الذاكرة
0012	0013	0014	0015	الإزاحة

في الخطوة الأولى ينسخ المعالج محتويات الحجرة ذات الإزاحة 0013 إلى المسجل AL بعد وصوله إلى كلا الحجرتين عند الإزاحتين 0012 و 0014 ، أما في الخطوة الثانية فهو ينسخ الحجرة ذات الإزاحة 0014 إلى المسجل AH بعد وصوله إلى كلتا الحجرتين ذات الإزاحتين 0014 و 0015 وبذلك تصبح محتويات المسجل AX هي 0123h . لا تحتاج برمجة المواقع الزوجية برمجة تختلف عن الفردية / ولا إلى اهتمام بمعرفة فيما إذا كانت هذه المواقع فردية أم زوجية ففي كلا الحالتين سنحصل على نتائج صحيحة لأن عملية الوصول إلى مواقع كلمات الذاكرة تضمن قراءتها وكتابتها ونقلها بشكل صحيح.

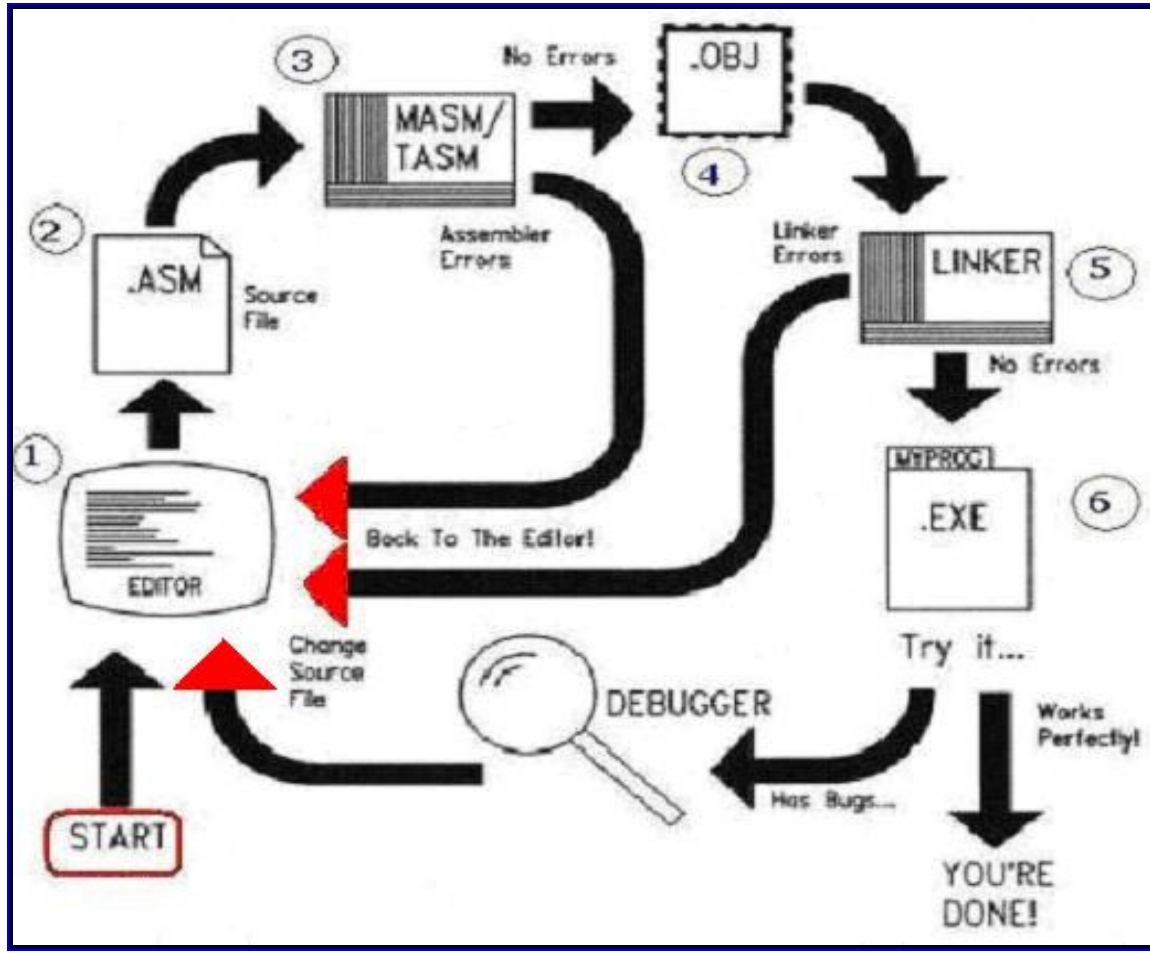
نستطيع استخدام التوجيه (directive) المسمى even في برامجنا لرصف أو توضع كافة المعطيات في الذاكرة في المواقع ذات العناوين الزوجية لضمان فعالية أكبر للبرامج.

: Assembling and linking

ما تقوم بكتابته من تعليمات يسمى source code ، وعند تجميعه بواسطة assembler ك masm أو tasm فانك تحصل على ملف وسيط يسمى object code ، ثم نقوم باستخدام احد الـ Linkers لربط ملفات الـ .obj مع بعضها البعض والحصول على ملف .exe. واحد.
الصورة التالية توضح المقصود :



والآن أنظر الى الصورة التالية فهي توضح مراحل تطوير برامج الأسمبلي :



- 1- باستخدام أي محرر (winasm , radasm , notepad , ...) نقوم بكتابة برنامج الأسمبلي
- 2- بعد حفظ المشروع سنحصل على ملف (أو أكثر) بامتداد .asm
- 3- باستخدام أي مجمع assembler (MASM , TASM , FASM , ...) نقوم بتجميع ملف أو ملفات الـ .asm
- 4- بعد التجميع ، نحصل على ملف يسمى relocatable object module بامتداد .obj. (كل ملف .asm يقابله ملف .obj واحد)
- 5- نقوم باستخدام أي linker (link , tlink , ...) لنربط جميع ملفات الـ .obj. ونحصل على الملف التنفيذي.
- 6- ها قد انتهينا...والناتج هو ملف .exe. جاهز للتشغيل.

طبعاً لاحظ أنه في كل خطوة كنا نفترض أنه لم يحدث أي خطأ...لأن حدوث خطأ يتطلب إعادة تلك الخطوة من جديد...هل ترى السهمين باللون الأحمر ؟ كل منهما يعبر عن حدوث خطأ في إحدى المراحل...

أمر آخر...إذا لم يحدث خطأ وحصلنا على ملف .exe. لكنه لم يعمل فهذا نحتاج الى استخدام debugger لتنقيح البرنامج واكتشاف موطن الخطأ فيه...وبعد اكتشاف مكان الخطأ نعود للـ سورس كود ونقوم بتغيير الكود الذي سبب الخطأ.

إذا أردت التوسع في مفهوم المجمع والرابط فهناك العديد من الدروس والمقالات التي تتحدث عن هذا... كما قلت سابقاً لا أريد لهذه الدورة أن تطول كثيراً...

مزيد من التوضيح حول الـ effective addresses

قلنا بأن العناوين في المعالجات x86 تتكون من 20 بت ، لكن معظم المسجلات registers في هذه المعالجات تتكون من 16 بت ، إذن كيف سيتم التعامل مع عناوين ذاكرة بحجم 20 بت رغم أن المسجلات حجمها الأقصى هو 16 بت ؟ الحل يكمن في استخدام مسجلين بدلا من واحد...المسجلين سويا حجمهما 32 بت أي أكبر من 20 وبالتالي فقد حلت المشكلة...

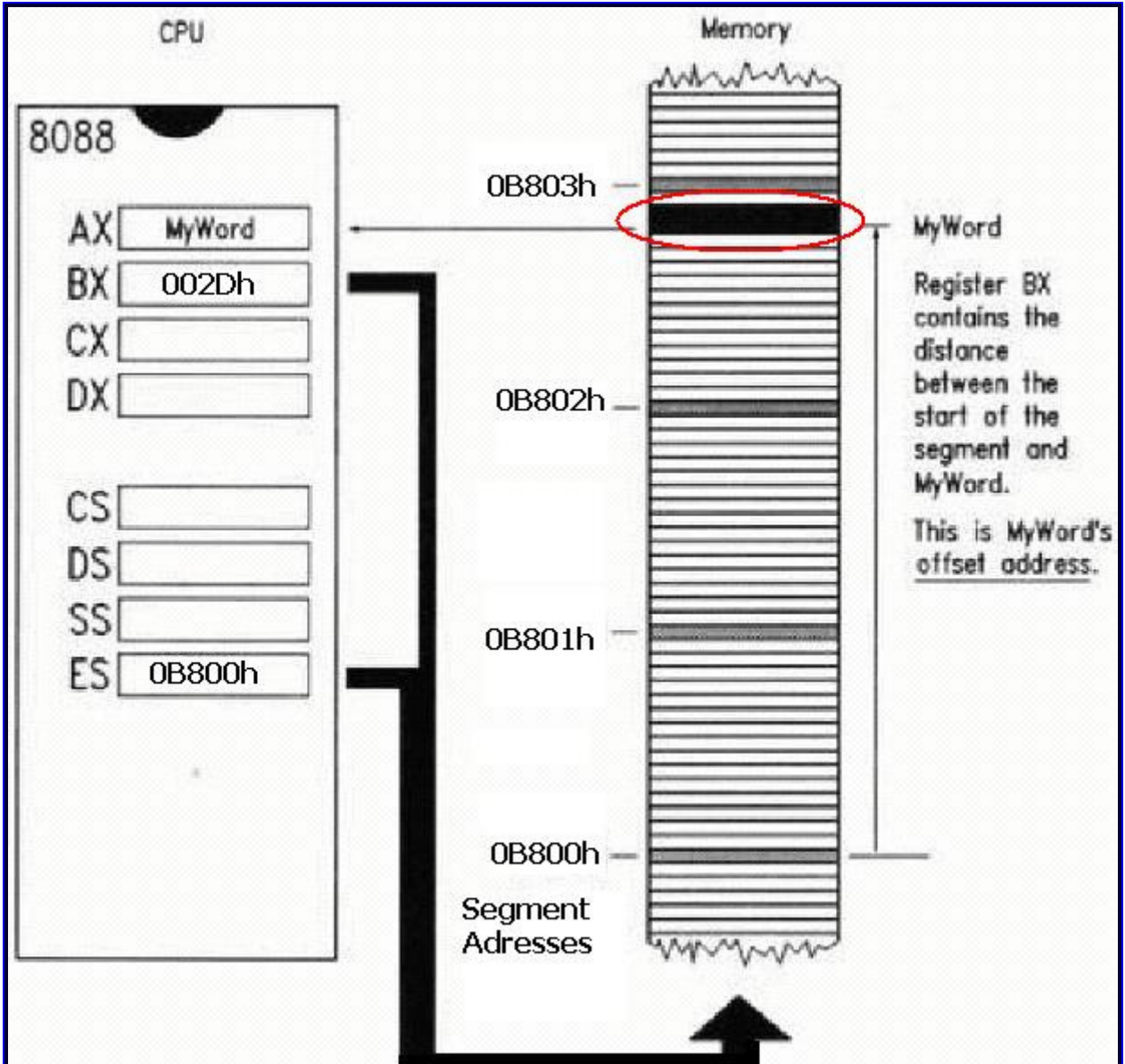
دعنا نرى ما يحدث عند تنفيذ تعليمة مثل

MOV AX, ES:[BX]

كما ترى فإننا نريد نقل بيانات معينة موجودة في الذاكرة (ولنسم البيانات هذا : MyWord) . نريد أن نحدد للمعالج العنوان الذي توجد فيه MyWord ، فكما ترى استخدمنا مسجلين لنصف هذا العنوان كالتالي ES:[BX] ، تذكر ان الصيغة العامة للـ effective addresses هي كالتالي :

Segment: offset

وهذا يعني أن جزء الـ segment من العنوان موجود في ES وجزء الـ offset من العنوان موجود في BX ونريد من المعالج أن يذهب الى هناك ويحضر الـ MyWord ويذهب بها إلى AX . الشكل التالي يوضح لنا كل ما يحدث أثناء تنفيذ التعليمة MOV AX, ES:[BX]



في اليسار ترى المعالج 8088 وترى جزءا من مسجلاته... في اليمين ترى الذاكرة (RAM) وهناك على يسارها عناوين ال Segment Addresses (ال Segment Addresses تكون مقسمة evenly الى مجموعات فالسيجمنت تبدأ كل 16 بايت أي 10h. أو دعنا نقول يمكن أن تبدأ كل 16 بايت ، فإذا تذكرنا أن 8088 يمكنه التعامل مع 1mb – أي 1048576 بايت - من الذاكرة ، وبالأخذ بالحسبان أن السيجمنت يمكن أن تبدأ كل 16 بايت ، إذن 1048576 تقسيم 16 الناتج 65536 ، أي أنه هناك 65536 مكانا مختلفا يمكن للسيجمنت أن تبدأ عنده . كما ترى فالفرق بين عنوان كل سيجمنت هو 10h . قد تقول : لكنني أرى الفرق 1h... هناك 0 في النهاية تم الاستغناء عنه من باب التسهيل ، فالعناوين التي على اليسار مثل 0B800h كانت بالأصل 0B8000h لكن الصفر الأخير سيقى صفرا لأنه كما قلت الفرق بين كل عنوان والذي يليه هو 10h إذن فهناك صفر دائما لذا استغنيانا عنه)

إن MyWord التي نريد نسخها موجودة في مكان ما بالذاكرة ، هذا المكان يبعد 02Dh عن السيجمنت 0B800h لذا فإن عنوان هذه ال MyWord هو 02D : 0B800 . لكن لاحظ أنه يمكننا استخدام سيجمنت أخرى ، ولتكن السيجمنت التي فوقها مباشرة 0B801 عندها ستختلف الإزاحة offset وستصبح 01Dh وبالتالي يمكننا أن نقول أن جميع العناوين التالية لها نفس التأثير :

0B800 : 02D

0B801 : 01D

0B802 : 00D

أعتقد أن الصورة أصبحت أوضح الآن... ملاحظة صغيرة... عن كتابة العناوين بصيغة Segment: offset يجب ألا تضيف اللاحقة h في نهاية الرقمين التي تدل على ان النظام هو الهكس...

فلنعد الى الصورة مرة أخرى . الآن أصبحنا نعلم أين توجد MyWord و أين قد خزنا عناوينها الاثنين. بعد تنفيذ التعليمة يكون المعالج قد ذهب الى العنوان المطلوب و أخذ MyWord ووضعها في AX . لاحظ أنه بإمكانك استخدام أي Segment Register آخر ك DS مثلا.

إن ال offset يمكن وضعه فقط في BX, BP, SP, SI and DI .

References:

1 <http://www.arabteam2000.com/>

2 Assembly Language: Step-by-Step , Jeff Duntemann

Arab Team for Reverse Engineering

αλλκο
July - 2006