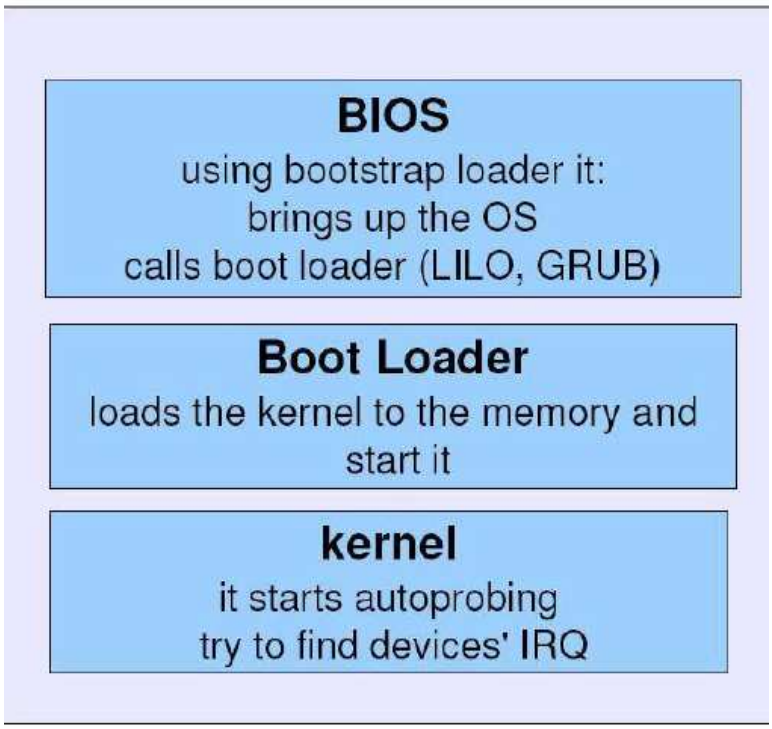

إقلاع الحاسب كمدخل لبناء نظام تشغيل

فـادي عمـروش

جميع الحقوق محفوظة©
الجمعة، 18 آذار، 2005
www.aleposoft.com

العالم و المتعلم شريكان في الأجـــر
حديث شريف

خلاصة توضح سير العمل و النتائج التي توصلت لها

بداية لا بد من شكر الدكتور حسن الصالح على تفانيه في العطاء لنا كطلاب في كلية الهندسة المعلوماتية حيث لم يبخل علينا أبدا بملاحظاته القيمة و توسيع آفاقنا بالتفكير بمواضيع متقدمة تفيدنا في المستقبل .

و لقد تم تتويج ذلك من خلال وظيفة حلقة البحث في مجال برمجة أنظمة التشغيل التي يخشى الكثير الخوض فيها .و بالنسبة لي اخترت الخوض في احد أهم مجالات برمجة نظم التشغيل و هو البحث في برامج إقلاع الحاسب .و سأقدم بهذه العجالة على ملخص لمنهج العمل الذي سرت عليه و النتائج التي توصلت لها .

عند البحث تبين أن بناء أي نظام تشغيل يتطلب برنامجين رئيسين و هما نواة نظام التشغيل أي نظام التشغيل بحد ذاته و برنامج الإقلاع الذي يقوم بإقلاع الحاسب و تحميل نظام التشغيل إلى الذاكرة و تشغيله .انطلقت في العمل في جعل الحاسب قادر على الإقلاع من قرص مرن و بالتالي تشغيل نظامي الخاص بي و الذي كان بداية عبارة عن حلقة لا نهائية لا تقوم بشيء مفيد و بعد ذلك شرعت في بناء نظام تشغيل بسيط يقوم بطباعة حرف واحد على الشاشة و بعد ذلك وضع هذا النظام على قرص مرن و الإقلاع منه .ثم قمت بتطوير نظام التشغيل ذلك بحيث يقوم بطباعة جملة HELLO WORLD على الشاشة .ثم فكرت بجعله أكثر تفاعلا بحيث يقوم نظام التشغيل بقراءة أحرف من لوحة المفاتيح و طباعتها على الشاشة .الخطوة الأخيرة كانت في كتابة برنامج إقلاع للحاسب مستقل و يقوم ذلك البرنامج بتحميل نظام التشغيل إلى الذاكرة , و بعد ذلك بحثت في برامج الإقلاع الجاهزة و المستخدمة في نظم التشغيل المختلفة في وقتنا الحاضر .

و لكي تكون الفائدة عامة قمت بكتابة البحث بالتفصيل و تقسيمه لفصول و ملحق و توضيح جميع الخطوات التي قمت بها سواء بالتعليمات اللازمة و الأدوات المستخدمة و الصور التوضيحية وصولا للشفرة المصدرية للبرامج , كما قمن بإرفاق قرص مرن جاهز للإقلاع يحتوي البرامج و جاهز لوضعه ضمن الحاسب و إعادة الإقلاع ليعمل برنامج الإقلاع و يحمل نظام تشغيل طباعة Hello World .

بقي أن أشير أن المواقع العربية فقيرة جدا و لا تتطرق لمثل هكذا مواضيع مهمة مما شجعني على كتابة بحثي بالتفصيل على شكل فصول و ترتيبها و طبعا قمت بالاستفادة من مقالات و مواقع إنكليزية كثيرة على الانترنت و جربت الكثير من الأمثلة و قمت بترجمة و إعداد المفيد منها .أتمنى أن أكون وفقت في صياغتي ليستفيد أكبر عدد ممكن من المهتمين .

فادي عمروش

و الله ولي التوفيق

الفهرس

خلاصة توضح سير العمل و النتائج التي توصلت لها 3

الفهرس 5

1.قطاع الإقلاع **THE BOOT SECTOR**: **6**

2.إنشاء قرص إقلاع. **11**

3.استخدام الأسمبلر **NASM**: **14**

4.طباعة العبارة **"HELLO, WORLD!"** **18**

5.أكثر من طباعة جملة **20**

6.محمل الإقلاع **BOOT LOADER** **22**

7.الشفرة المصدرية للبرامج **26**

ملحق..... 36

حقوق الملكية 38

① قطاع الإقلاع The Boot Sector:

في هذه الفصل سنتعرف على محتويات قطاع الإقلاع و بعد ذلك سنتعلم كيف نكتب برنـــامج الإقلاع الخاص بنا .

عندما يقوم جهاز الحاسب بالإقلاع من قرص مرن تقوم وحدة الدخل و الخرج للنظام BIOS بقراءة القرص و تحميل القطاع الأول إلى الذاكرة عند العنوان 0000:7C00 . يدعى القطاع الأول بسجل إقلاع دوس DOS Boot Record (DBR) . تقوم البيوس BIOS بالقفز إلى العنوان 0x7C00 و تقوم بتنفيذ التعليمات الموجودة هناك و تكون هذه التعليمات عبارة عن برنامج تحميل الإقلاع boot loader و الذي سيقوم بتحميل نظام التشغيل OS إلى الذاكرة و تبدأ عندئذ عملية إقلاع نظام التشغيل .

بداية سنقوم بإلقاء نظرة إلى داخل سجل الإقلاع , و سيساعدنا في ذلك أداة المنقح DEBUG و الذي يستخدم لعرض محتويات الذاكرة و الأقراص و في حالتنا سنقوم بعـــرض محتويـــات سجل الإقلاع لقرص مرن .

يمكن تشغيل المنقح من محرر الأوامر دوس DOS أو مـــن قائمـــة ابـــدأ Start و اختيـــار تشغيلRUN ثم كتابة DEBUG .

يقوم الأمر d بعرض جزء من محتويات الذاكرة RAM .

```
-d
136E:0100  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0110  00 00 00 00 00 00 00 00-00 00 00 00 34 00 5D 13   ............4.].
136E:0120  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0130  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0140  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0150  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0160  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
136E:0170  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................

-
```

من أجل الحصول على المساعدة و الشرح لأمر ما يمكنك كتابة ? .

```
-?
assemble      A [address]
compare       C range address
dump          D [range]
```

```
enter         E address [list]
fill          F range list
go            G [=address] [addresses]
hex           H value1 value2
input         I port
load          L [address] [drive] [firstsector] [number]
move          M range address
name          N [pathname] [arglist]
output        O port byte
proceed       P [=address] [number]
quit          Q
register      R [register]
search        S range list
trace         T [=address] [value]
unassemble    U [range]
write         W [address] [drive] [firstsector] [number]
allocate expanded memory        XA [#pages]
deallocate expanded memory      XD [handle]
map expanded memory pages       XM [Lpage] [Ppage] [handle]
display expanded memory status  XS
-
```

عليك الانتباه و الحذر عند التعامل مع المنقح حيث أنه يمكن أن يقوم بكتابة معطيات جديدة فوق أخرى موجودة في القرص الصلب و إتلافه أو أن يؤدي لضياع معلومات و حذفها من القرص الصلب .

خذ قرص مرن floppy و قم بتهيئته format ثم ضعه في محرك الأقراص المرنة .
لتحميل سجل الإقلاع Boot Record لقرصك المرن أدخل الأمر التالي:

```
-l 0 0 0 1
```

حيث يقوم هذا الأمر l (الحرف L) بتحميل القطاعات من القرص إلى جزء من الذاكرة RAM أما الأرقام الأربعة التالية فتعبر عن بداية العنوان الذي ستقوم بتحميله إلى الذاكرة حيث :

- الرقم الأول : رقم القرص الصلب حيث 0 تشير للقرص المرن الفلوبي floppy .
- الرقم الثاني : رقم القطاع من القرص حيث في حالتنا نريد القطاع الأول أي القطاع ذو الرقم 0 .
- الرقم الثالث : العنوان الذي سنبدأ منه التحميل .
- الرقم الرابع :عدد القطاعات التي نريد تحميلها .

و يصبح الأمر السابق يقوم بتحميل القطاع الأول من القرص المرن إلى الذاكرة بـــدءا مـــن العنوان 0 و بعدد قطاع واحد .

الآن قمنا بتحميل القطاع إلى الذاكرة و نريد عرض محتوياته , يتم ذلك بالأمر التالي:

```
-d 0
```

سيظهر لديك الشكل التالي حيث يتم عرض 128 بايت (0x80 in hex) من ســجل إقــلاع الفلوبي كالتالي:

```
0AF6:0000  EB 3C 90 4D 53 44 4F 53-35 2E 30 00 02 01 01 00   .<.MSDOS5.0.....
0AF6:0010  02 E0 00 40 0B F0 09 00-12 00 02 00 00 00 00 00   ...@............
0AF6:0020  00 00 00 00 00 00 29 F6-63 30 88 4E 4F 20 4E 41   ......).c0.NO NA
0AF6:0030  4D 45 20 20 20 20 46 41-54 31 32 20 20 20 33 C9   ME    FAT12   3.
0AF6:0040  8E D1 BC F0 7B 8E D9 B8-00 20 8E C0 FC BD 00 7C   ....{.... .....|
0AF6:0050  38 4E 24 7D 24 8B C1 99-E8 3C 01 72 1C 83 EB 3A   8N$}$....<.r...:
0AF6:0060  66 A1 1C 7C 26 66 3B 07-26 8A 57 FC 75 06 80 CA   f..|&f;.&.W.u...
0AF6:0070  02 88 56 02 80 C3 10 73-EB 33 C9 8A 46 10 98 F7   ..V....s.3..F...
```

كنظرة أولية لن يخبرنا هذا الشكل أي معلومات و لكن يمكن أن أعرف بعض المعلومات مثل أن نظام الملفات هو FAT12 و أن نوع القرص هو MS-DOS 5.0 بدون اسم .
الأرقام ضمن العمود اليساري تظهر عناوين الذاكرة RAM أما الأرقام الست عــشرية فـــي المنتصف فهي البايتات الموجودة ضمن محتوى الذاكرة حيث أن محتوى الذاكرة يعرض كقيم ست عشرية بالطبع .العمود الموجود في أقصى اليمين يظهر أحرف الأسكي المقابلـــة للقـــيم الست عشر و في حال عدم وجود حرف مرئي مقابل للرقم الست عشري فانه يـــتم عـــرض الرمز النقطة (.) لاحظ وجود عدة نقاط .... .
إن بعض هذه البايتات التي تراها في هذا الجزء من سجل الإقلاع هي أجزاء من التعليمات في محمل الإقلاع (boot loader) و بعض هذه البايتات تقوم بتخزين المعلومات حول القـــرص مثل رقم البايتات ضمن القطاع , رقم القطاعات في المسار track ... الخ .
دعونا نلقي نظرة إلى شفرة محمل الإقلاع boot loader .
لنكتب الأمر التالي:

```
-u 0
```

تقوم هذه التعليمة بفك تجميع unassembled أي تظهر لك التعليمات بلغة التجميع المقـــابلى للبايتات السابقة و بدءا من العنوان المحدد و في حالتنا هو الصفر و سنحصل علـــى الخـــرج التالي:

```
0AF6:0000 EB3C          JMP     003E
0AF6:0002 90            NOP
0AF6:0003 4D            DEC     BP
0AF6:0004 53            PUSH    BX
0AF6:0005 44            INC     SP
0AF6:0006 4F            DEC     DI
0AF6:0007 53            PUSH    BX
0AF6:0008 352E30        XOR     AX,302E
0AF6:000B 0002          ADD     [BP+SI],AL
0AF6:000D 0101          ADD     [BX+DI],AX
0AF6:000F 0002          ADD     [BP+SI],AL
0AF6:0011 E000          LOOPNZ  0013
0AF6:0013 40            INC     AX
0AF6:0014 0BF0          OR      SI,AX
0AF6:0016 0900          OR      [BX+SI],AX
0AF6:0018 1200          ADC     AL,[BX+SI]
0AF6:001A 0200          ADD     AL,[BX+SI]
0AF6:001C 0000          ADD     [BX+SI],AL
0AF6:001E 0000          ADD     [BX+SI],AL
```

تقوم التعليمة الأولى003 JMP بالقفز إلى العنوان 0x3E .

أما التعليمات التالية فهي معلومات القرص التي ذكرناها قبل قليل و لكن هـــذه التعليمـــات لا توافقها تماما و إنما يقوم المنقح بفعل ما بوسعه بتحويل البايتات إلى تعليمات أسمبلي و يفسر البايتات بالشكل السابق .

إذا إن التعليمة الأولى تقوم بالقفز إلى العنوان الذي يبدأ عنده برنامج الإقلاع boot program و الذي يبدأ عادة بالعنوان 0x3E دعونا نطلع على هذه التعليمات بكتابـــة الأمر :

```
-u 3E
```

هنا يمكنك رؤية بداية الشفرة التي ستقوم بتحميل نظام التشغيل DOS أو Windows .

هذا الكود لنظام التشغيل MS-DOS يقوم بالبحث علـــى القـــرص الـــصلب عـــن الملفـــات IO.SYS MSDOS.SYS , حيث تحتوي هذه الملفات على شفرة نظام التـــشغيل , تقـــوم شفرة محمل الإقلاع boot loader بتحميل هذه الملفات إلى الذاكرة ة البدء بتنفيذها و إما إذا لم يجد محمل الإقلاع هذه الملفات فإنه يقوم بعرض رسالة الخطأ الشائعة .

```
Invalid system disk
Disk I/O error
Replace the disk, and then press any key
```

هذه الرسالة تكتب في سجل إقلاع الدوس بدءا من العنوان 180 كما يبن الشكل:

```
-d 180
0AFC:0180  18 01 27 0D 0A 49 6E 76-61 6C 69 64 20 73 79 73   ..'..Invalid sys
0AFC:0190  74 65 6D 20 64 69 73 6B-FF 0D 0A 44 69 73 6B 20   tem disk...Disk
0AFC:01A0  49 2F 4F 20 65 72 72 6F-72 FF 0D 0A 52 65 70 6C   I/O error...Repl
0AFC:01B0  61 63 65 20 74 68 65 20-64 69 73 6B 2C 20 61 6E   ace the disk, an
0AFC:01C0  64 20 74 68 65 6E 20 70-72 65 73 73 20 61 6E 79   d then press any
0AFC:01D0  20 6B 65 79 0D 0A 00 00-49 4F 20 20 20 20 20 20    key....IO
0AFC:01E0  53 59 53 4D 53 44 4F 53-20 20 20 53 59 53 7F 01   SYSMSDOS   SYS..
0AFC:01F0  00 41 BB 00 07 60 66 6A-00 E9 3B FF 00 00 55 AA   .A...`fj..;...U.
```

يظهر الشكل السابق نهاية سجل الإقلاع , هناك ملاحظة هامة جدا و هي أن سجل الإقلاع محدد بالحجم 512 بايت و بالتالي إذا ما تم تحميله إلى الذاكرة بدءا من العنوان 0 سيتوضع البايت الأخير عن العنوان 0x1FF كما أن آخر بايتين ضمن سجل الإقلاع هما 0x1FF , 0x1FE و يحتويان دوما القيميتن 0x55 و 0xAA على الترتيب يجب أن دوما وضع هاتين القيمتين ضمن أخر بايتين و إلا سوف لن تقوم البيوس بتحميل القطاع و البدء بتنفيذه .

في الخلاصة نقول إن سجل الإقلاع لدوس يبدأ بتعليمة قفز فوق المعطيات التي تليه , هذه ال60 بايت من المعطيات تبدأ عند العنوان 0x02 وتنتهي عند العنوان 0x3D و تستأنف شفرة الإقلاع عند العنوان 0x3E و تكمل عملها حتى العنوان 0x1FD حيث تنتهي ب بايتين لهما قيمتين محددتين هما 0x55 , 0xAA .

② إنشاء قرص إقلاع.

في هذا الفصل سنتعلم كيف نقوم بصنع برنامج إقلاع لقرص مرن و سننطلق في ذلك بتعديل سجل الإقلاع لدوس DOS Boot Record .

إن هدفنا هو استبدال شفرة محمل الإقلاع بدون تغيير المعطيات الأخرى في قطاع الإقـــلاع , لأنه إذا قمنا بتغيير المعطيات إلى شيء آخر غير متاح فسوف لن يعتبر نظام التـــشغيل دوس DOS أو ويندوز Windows القرص الصلب قرصا مقبولا , و سيعطس النظام رسالة خطأ تفيد أن يجب تهيئة القرص المرن .

بداية سنحافظ على التعليمة الأولى الخاصة بالقفز jmp 0x3E لأننا نحتاج للقفز فوق معطيات سجل الإقلاع .

للقيام بهذا و البدء بالتعديل بدءا من العنوان 0x3E , دعونا نقوم بتشغيل المـــنقح و تحميـــل القطاع الأول من القرص المرن إلى الذاكرة بدا من العنوان 0 .

اكتب التعليمة التالي:

```
-u 3E
```

لنبدأ بتعديل المعطيات و ذلك بالأمر :

```
-a 3E
```

يقوم المحث prompt بالتغيير إلى العنوان الذي نريد كالتالي :

```
jmp 3E
```

يتم تنفيذ التعليمة إلى لغة الآلة و توضع في الذاكرة و يشير المحث إلى الموقع الذاكري التالي المتاح بعد إدخال التعليمة , اضغط Enter مرة أخرى لإنهاء شفرة الاسمبلي و يصبح الشكل النهائي للكود كالتالي :

```
-a 3E
0AFC:003E jmp 3E
0AFC:0040
-
```

الآن يمكنك عرض التعليمات التي أدخلتها كالتالي :

```
-u 3E
```

و يكون بالشكل التالي:

```
-u 3e
136E:003E EBFE          JMP     003E
136E:0040 8ED1          MOV     SS,CX
136E:0042 BCF07B        MOV     SP,7BF0
136E:0045 8ED9          MOV     DS,CX
136E:0047 B80020        MOV     AX,2000
136E:004A 8EC0          MOV     ES,AX
136E:004C FC            CLD
136E:004D BD007C        MOV     BP,7C00
136E:0050 384E24        CMP     [BP+24],CL
136E:0053 7D24          JGE     0079
136E:0055 8BC1          MOV     AX,CX
136E:0057 99            CWD
136E:0058 E83C01        CALL    0197
136E:005B 721C          JB      0079
136E:005D 83EB3A        SUB     BX,+3A
```

يمكنك ملاحظة أن التعليمة الأولى تغيرت إلى تعليمة القفز التي وضعناها و بالتالي سنحصل على حلقة لانهائية أي أن برنامج إقلاعنا سيعلق ببساطة لأنه سيتوقف مكانه مستمرا بالقفز عند نفس العنوان , بقي أن نحفظ هذه المعطيات الجديدة على القرص الصلب لان إغلاق المنقح لن يؤدي لآي عملية حفظ و يتم الحفظ كما ذكرنا بالكتابة إلى القطاع الأول من القرص المرن و ببساطة بالأمر التالي :

```
-w 0 0 0 1
```

إن أمر الكتابة write يشبه أمر التحميل load حيث يكتب المعطيات الموجودة في الذاكرة في العنوان 0 للقرص رقم 0 بدءا من القطاع رقم 0 و بطول قطاع واحد 1 .عليك الحذر عند استخدام هذا الأمر و خصوصا مع الأقراص الصلبة غير المرنة لأنه يؤدي لكتابة معطيات فوق أخرى موجودة و قد يسبب ضياع معلومات لك .

الآن يمكنك أن تقلع من القرص الصلب , عندما تقوم بإعادة إقلاع الحاسب تقوم البيوس بتحميل القطاع الأول من القرص إلى الذاكرة و تبدأ التنفيذ عند بداية القطاع و ستجد عندئذ تعليمة القفز إلى العنوان 0x3E و تنفيذ التعليمة هناك .

الآن قم بتنفيذ ما سبق و ستجد أن بعد أعادة الإقلاع سيعلق الحاسب أي لن يفعل شيء و هو المطلوب .

طبعا قد لا تكون راضيا عن النتائج فماذا نستفيد من تعليق الحاسب , لا تتسرع فهذه مجرد بداية لا بد منها تريك أنك يمكنك الاستغناء عن نظام تشغيلك و إعادة إقلاع حاسبك كما تريد,

إذا أردنا فعل شيء مفيد فعلينا الاستعانة بتوابع بيوس BIOS و علينا الانتباه أن هناك نوعين من الاستدعاءات منها خاص ببيوس و الآخر خاص ب دوس و طبعا لا يمكن الاستعانة بتلك الخاصة ب(دوسDOS) لأنه بالأصل لا يوجد نظام تشغيل دوس يعمل بالأصل .

تكون قيم المسجلات بالدلالة التالية:

```
AH = 0x0E
AL = ASCII code of the character to be printed
BL = color/style of character
```

أعد الآن التعليمات في هذا الفصل و بدل من تعليمة القفز التي كتبناها سابقا ادخل التعليمـــات التالية:

```
-a 3E
0AF6:003E mov ah, 0e
0AF6:0040 mov al, 48
0AF6:0042 mov bl, 07
0AF6:0044 int 10
0AF6:0046 jmp 46
0AF6:0048
-
```

تقوم التعليمات السابقة بإسناد القيمة صفر للمسجل AH و القيمة 0x48 للمسجل AH حيث تعبر القيمة عن الحرف H , القيمة 7 للمسجل BL و هي تعبر عن لون الخط الأبيض علـــى خلفية سوداء , قم بحفظ قطاع الإقلاع كما سبق (-w 0 0 0 1) , أعد الإقلاع مرة أخرى من القرص المرن , هذه المرة ستجد الحرف H مطبوعا على الشاشة قبل تعليق الحاسب .

③ استخدام الأسمبلر NASM:

سنتعلم في هذا الفصل كيفية استخدام الاسمبلي لكتابة برامجنا , كنا في الفصول السابقة نستخدم أداة المنقح لكتابة برامج الاسمبلي , و لكن كتابة برامج كبيرة يعتبر أمر متعب باستخدام المنقح DEBUG و للحصول على طريقة سريعة لكتابة برامج الاسمبلي سننطلق باستخدام برنامج الاسمبلي "Netwide Assembler" (NASM). و يمكنك تحميل البرنامج من الموقع http://nasm.sourceforge.net/index.php كما يمكنك الحصول عليه من السيدية المرفقة .

سنستخدم هذا البرنامج لكتابة نفس التعليمات التي قمنا بكتابتها في الفصل الثاني .هذا البرنامج ذي الاسم h.asm موجود و يمكنك الإطلاع عليه كاملا في الملحق و السيدية المرفقة , بداية هذه الترويسة للبرنامج .

```
; -------------------------------------------------------
;Simple boot program that prints the letter 'H'
; Disclaimer: I am not responsible for any results of the
use of the contents
;of this file
;-------------------------------------------------------
```

تبدأ التعليمات في هذه البرنامج بشكل مشابه لما عرضناه في الفصل السابق , أولا تأتي التعليمة التي تقوم بالقفز فوق معطيات سجل الإقلاع و ذلك بالقفز للعنوان begin و بعد القفز فوق هذه المعطيات و هي حوالي 20 بايت من المعطيات لها البنية التالية (حيث قمت باستخلاصها من قرصي المرن بواسطة المنقح و يمكنك التأكد من ذلك و عليك الانتباه أن الأرقام المؤلفة من أكثر من بايت واحد ستكون معروضة بشكل معكوس byte swapped بسبب بنية أنتل و بحيث يخزن البايت الأقل أهمية في المكان ذو العنوان الذاكري الأقل و العكس بالعكس .

```
------------------------------------------------------------------- ;
 ;data portion of the "DOS BOOT RECORD"
------------------------------------------------------------------- ;
brINT13Flag     DB      90H             ; 0002h - 0EH for INT13 AH=42 READ
brOEM           DB      'MSDOS5.0'      ; 0003h - OEM name & DOS version (8 chars(
brBPS           DW      512             ; 000Bh - Bytes/sector
brSPC           DB      1               ; 000Dh - Sectors/cluster
brResCount      DW      1               ; 000Eh - Reserved (boot) sectors
brFATs          DB      2               ; 0010h - FAT copies
brRootEntries   DW      0E0H          0011 ;h - Root directory entries
brSectorCount   DW      2880          0013 ;h - Sectors in volume, < 32MB
brMedia         DB      240           0015 ;h - Media descriptor
brSPF           DW      9               ; 0016h - Sectors per FAT
brSPH           DW      18              ; 0018h - Sectors per track
brHPC           DW       2            001 ;Ah - Number of Heads
brHidden        DD      0               ; 001Ch - Hidden sectors
brSectors       DD       0    0020 ;       h - Total number of sectors
               DB       0               ; 0024h - Physical drive no.
               DB       0               ; 0025h - Reserved (FAT32(
               DB      29H             ; 0026h - Extended boot record sig
brSerialNum     DD      404418EAH       ; 0027h - Volume serial number (random(
brLabel         DB      'Joels disk '   ; 002Bh - Volume label  (11 chars(
brFSID          DB      'FAT12   '      ; 0036h - File System ID (8 chars(
----------------------------------------------------------------------;
```

لننتقل الآن إلى الشفرة المكتوبة بعد اللائحة Begin حيث ستجد أنها تشبه تماما ما عرضـــناه في الفصل السابق , فهو يقوم ببساطة بطباعة الحرف H على الشاشة ثم يدخل فـــي حلقـــة لا نهائية , و في نهاية الشفرة نقوم بالتأكد أن حجم هذا الكود هو 512 بايت و الذي يقابل حجـــم قطاع واحد و من ثم يأتي السطر ذو الكلمة "times" و الذي يضيف أصفار إلى نهاية الملف بحيث يكون حجم الكود 510 بايت و بعد ذلك نضيف البايتين المعـــروفين 0x55, 0xAA و اللذان يحددان ان القطاع هو قطاع إقلاع إلى نهاية القطاع .

```
------------------------------------------- ;
  ;Boot program code begins here
------------------------------------------- ;
 ;boot code begins at 0x003E
begin:
        mov     ah, 0x0e        ;Function to print a character to the screen
        mov     al, 'H'        ; Which character to print
        mov     bl, 7           ;color/style to use for the character
        int     0x10            ;print the character

hang:
        jmp     hang            ;just loop forever.

---------------------------------------------;

size    equ      - $entry
%if size+2 > 512
%  error "code is too large for boot sector"
%endif
        times    - 512)size - 2) db 0

        db      0x55, 0xAA              2;byte boot signature
```

الآن لديك الملف كامل في القرص المرفق , كذلك لديك برنامج NASM قم بفك ضغطه مثلا إلى قرص السي و ضع داخل مجلده الملف المراد ترجمته ثم قم بترجمة الملف بواسطة الاسمبلر كالتالي :

```
nasm h.asm -o h.bin
```

سيولد لك ملف ثنائي h.bin تأكد أن حجمه هو 512 بايت من خلال خصائص الملف ,

قم الآن بنسخ الملف إلى قرصك المرن و بعد ذلك شغل المنقح و اكتب التالي:

```
debug
-n c:\nasm\h.bin
-l 0
```

ستقوم الشفرة السابقة بتحميل الملف إلى الذاكرة بدءا من العنوان 0 ويمكنك التأكد من محتوياته و انه تحمل بشكل صحيح من خلال التعليمتين (d) dump و (u) unassemble . حيث عند تنفيذ التعليمة d ستلاحظ أن الملف قد أضيف له أصفار حتى عنوان البايتين 0x1FE and 0x1FF حيث وضع فيهما قيمتين ثابتتين و طبعا هي توجد عند نهاية القطاع .

```
-d
136E:0180  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:0190  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01A0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01B0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01C0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01D0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01E0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00................
136E:01F0  00 00 00 00 00 00 00 00-00 00 00 00 00 00 55 AA  ..............U.
-
```

يمكنك أن تلاحظ أيضا إن المسجل CX سيحتوي بعد ذلك عدد البايتات المحملة من الملف و طبعا يمكنك عرض محتويات المسجل بالأمر R :

```
-r
AX=0000  BX=0000  CX=0200  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=136E  ES=136E  SS=136E  CS=136E  IP=0100   NV UP EI PL NZ NA PO NC
136E:0100 0000          ADD     [BX+SI],AL                      DS:0000=EB
-
```

لاحظ أن قيمة المسجل هي 200 بالست عشري و تقابل 512 بالعشري أي إن عملنـــا بـــدون أخطاء حتى الآن .

أصبح كل شيء جاهز و بقي أن تحمل هذه التعليمات للقرص المرن في مكانها الصحيح :

```
-w 0 0 0 1
```

اعد الإقلاع و سترى نفس النتيجة و هي طباعة الحرف h,في الفصل القادم ســنقوم بطباعــة "Hello, World" .

④ طباعة العبارة "Hello, World!" .

الآن دعونا ننتقل للإثارة و المتعة و الغوص في برامج الإقلاع و كتابة نظام تشغيل خاص بنا و ككل المبتدئين في لغات البرمجة لا بد أن بدأت حياتك البرمجية ببرنامج "Hello, World" دعونا نقوم بالشيء بالنسبة لنظام التشغيل و ليكن نظامنا ذو الاسم "Hello, World" operating system . سنقوم بإنشاء تابع يقوم بطباعة نص string و من خلاله سوف نعرض جملتنا المنشودة. سيكون من الممل أن نقوم بطبع الجملة من خلال طبع الأحرف بشكل متتالي لذلك سنقوم بإنشاء تابع يقوم بطباعة سلسلة نصية تنتهي بالصفر على الشاشة و يكون ذلك من خلال حلقة بسيطة كل الأحرف ضمن السلسلة النصية في نفس الوقت .

```
; ---------------------------------------------
; Print a null-terminated string on the screen
; ---------------------------------------------
putstr:
     lodsb           ; AL = [DS:SI]
     or al, al       ; Set zero flag if al=0
     jz putstrd      ; jump to putstrd if zero flag is set
     mov ah, 0x0e    ; video function 0Eh (print char)
     mov bx, 0x0007  ; color
     int 0x10
     jmp putstr
putstrd:
     retn
```

دعونا نلقي نظرة سريعة على عمل التابع السابق.نقوم بداية بتحميل العنوان للحرف الأول للسلسلة النصية إلى المسجل SI و من ثم نستدعي ببساطة التابع الفرعي putstr .

و يمكنك تشكيل السلسلة النصية كالتالي:

```
msg  db 'Hello, World!', 0
```

و يتم الإشارة لنهاية السلسلة ب الصفر و يبقى أمر طباعتها ببساطة كالتالي:

```
mov si, msg     ; Load address of message
call putstr     ; Print the message
```

بقي أمر واحد عليك إعداده قبل أن يعمل برنامجنا . إن العنوان ل MSG المحمل إلى المسجل SI هو عبارة عن إزاحة عن البداية من بداية القطعة segment التي يؤشر لها بالمسجل DS, لذلك قبل أن تستطيع استخدام العنوان MSG يجب أن تقوم بإعداد قطعة المعطيات الحالية current data segment , دعونا الآن نقوم باستخدام العنونة من أسفل الذاكرة RAM و لتحديد قطعة المعطيات بحيث تبدأ من الأسفل نقوم بإسناد القيمة 0 إلى المسجل DS و تقوم التعليمات التالية بذلك :

```
xor  ax, ax     ; Zero out ax
mov  ds, ax     ; Set data segment to base of RAM
```

لم يبقى سوى إعادة الخطوات السابقة للحصول على برنامج إقلاع يطبع الجملة الشهيرة HELLO WORLD . إذا قم بنسخ الملف helowrld.asm إلى مسار NASMعلى سطح القرص Cفي حالتنا , قم بترجمته كما سبق .

5 أكثر من طباعة جملة .

طبعا إن طباعة جملة على الشاشة أمر جيد و لكن أي نظام تشغيل يحتاج لشيء من التفاعل مع المستخدم و ليس مجرد عرض المعلومات , دعونا نضفي شيئا من التفاعلية على نظام تشغيلنا الوليد و ذلك بجعله يقرأ دخل من لوحة المفاتيح . سنقوم بقراءة الدخل من لوحة المفاتيح اعتمادا على مقاطعات بيوس BIOS و ليس دوس DOS و من أجل ذلك سنستخدم التابع function 0 مع المقاطعة 16 (interrupt 0x16) كالتالي:

```
xor  ah, ah     ; we want function zero
int  0x16       ; wait for a keypress
```

يقوم هذا التابع بجعل الحاسب ينتظر حتى أن يتم ضغط مفتاح من لوحة المفاتيح و يخزن شفرةالمسح للمفتاح المضغوط في المسجل AH و شفرة الاسكي ASCII code في المسجل AL. سنقوم بهذا الوقت باضافة شيء من التفاعلية و ذلك في الملف المرفق lesson5.asm.

```
---------------------------------------- ;
  ;Boot program code begins here
---------------------------------------------- ;
 ;boot code begins at 0x003E
begin:
     xor  ax, ax          ;zero out ax
     mov  ds, ax          ;set  data  segment  to  base  of
RAM
     mov  si, msg         ;load address of our message
     call putstr          ;print the message

loop1:
     xor  ah, ah          ;function 0
     int  0x16            ;get a key from the keyboard

     mov  si, charmsg     ;load address of message
     call putstr          ;print the message

     mov  ah, 0x0e        ;function print character
```

```
     mov  bl, 0x07        ;white on black
     int  0x10

     mov  si, newline     ;print a newline
     call putstr

     jmp  loop1           ;just loop forever.

------------------------------------------- ;
```

أما المعطيات كالتالي:

```
 ;data for our program

msg  db   'Press a key'.
newline db     13,10,0
charmsg db     'Character: ',0
```

و التابع من أجل الطباعة كما سبق مع تعديل بسيط :

```
-------------------------------------------
 ;Print a null-terminated string on the screen
---------------------------------------------- ;
putstr:
      push  ax
putstrl:
      lodsb                    ;AL = [DS:SI[
      or    al, al             ;Set zero flag if al=0
      jz    putstrd            ;jump to putstrd if zero flag is set
      mov   ah, 0x0e           ;video function 0Eh (print char(
      mov   bx, 0x0007         ;color
      int   0x10
      jmp   putstrl
putstrd:
      pop   ax
      retn
---------------------------------------------;
```

⑥ محمل الإقلاع Boot Loader .

كل ما كتبناه في الفصول السابقة تم وضعه ضمن القطاع الأول بحيث أصبح برنـــامج الإقلاع موجود ضمن القطاع الأول من القرص , و لكن كما نعلم انه من المستحيل أن يقتصر نظام تشغيلنا على 512 بايت فقط فما هو الحل ؟ الحل هو بكتابة برنامج إقلاع يقوم ببـــساطة بتحميل ملف تنفيذي من القرص و يبدأ بتنفيذه و يدعى هذا البرنامج بمحمل الإقـــلاع Boot Loader , و يكون الملف الذي سنحمله من القرص بالحجم الذي نريده و لن يقتصر حجمـــه على قطاع واحد . أنها لفكرة جيدة أن نعرج قليلا على نظام الملفات FAT حيث ســـنفترض أننا نستخدم هذا النظام من الملفات في بحثنا هذا , دعونا الآن نلقي نظرة سريعة إلى عمليـــة تحميل الإقلاع boot loading process .

يحتوي القرص المرن في حالتنا هذه على سجل الإقلاع DOS (القطاع الأول مـــن القـــرص الذي نعمل عليه), جدول توضع الملفات File Allocation Table (FAT), مجلد الجـــذر the Root Directory , و من ثم المعطيات المحتواة في الملفات علـــى القـــرص المـــرن . بالنسبة للقرص الصلب hard disk فالأمر أكثر تعقيدا , حيث يحتوي على ســـجل الإقـــلاع الأساسي Master Boot Record و مبدأ تعدد الأجزاء multiple partitions .

لنفترض أننا قمنا بكتابة نظام تشغيل و قمنا بترجمته إلى ملف ســـميناه LOADER.BIN و وضعناه على القرص الصلب , إن مهمة محمل الإقلاع هو تحميله كالتالي:

1. تتم قراءة سجل إقلاع دوس DOS Boot Record (DBR) لتحديد حجم الأشـــياء التالية : DBR, FAT, Root Directory و بذلك يتحدد موقع كـــل مـــنهم علـــى القرص .

2. تتم قراءة مجلد الجذر Root Directory إلى الذاكرة .

3. يبحث ضمن المجلد الجذر Root Directory عن الملف ذو الاسم فإذا وجد يمكننـــا عندئذ البحث ضمن مدخل المجلد directory entry للبحث عن أول كلستر cluster

(وحدة توضع الملف file allocation unit)للملف , أما إذا لم يوجد سنعطي رسالة خطأ .

4. تتم قراءة جدول توضع الملفات من القرص إلى الذاكرة .

5. نبدأ من الكلستر cluster الأول للملف و نستخدم نظام الملفات FAT لوضع جميع الكلسترات التي تخص الملف,و تتم قراءتهم من القرص إلى الذاكرة عند موقع محدد .

6. نقفز إلى ذلك العنوان و نبدأ تنفيذ نظام التشغيل .

يجب أن تتم جميع عمليات القراءة من القرص بواسطة استدعاءات بيوس BIOS , يمكنك الإطلاع على توابع البيوس المستخدمة من أجل قراءة القطاعات من القرص , على كل حال ستجد محمل إقلاع يتعامل مع نظام الملفات FAT و هو بالاسم BOOT12.ASM و يمكنك ترجمته و تنفيذه كما تعلمنا سابقا .

هناك عدة إعدادات قابلة للتعديل ضمن هذا المحمل loader , هذا المحمل loader يفترض استخدام نظام الملفات FAT12 و هو النظام الذي تتم به تهيئة القرص المرن . و لذلك من اجل نظام آخر عليك استخدام محمل loader آخر , تتلخص الأشياء التي يمكنك تعديلها بمواقع نظام التشغيل و بنى المعطيات المختلفة لنظام FAT التي سيتم تحميلها إلى الذاكرة , طبعا يمكنك تعديل اسم نظام التشغيل الذي سيقوم المحمل loader بتحميله .

افتراضيا يقوم المحمل بتحميل الملف المسمى LOADER.BIN الموجود على مجلد الجذر إلى الذاكرة بدءا من العنوان 0x1000:0000 و يمكنك تعديل العنوان من خلال : %define IMAGE_SEG و بذلك يمكنك ترجمة نظام تشغيلك و نسخه إلى القرص المرن بالاسم LOADER.BIN . دعونا نعتبر أن نظام تشغيلنا هو ذلك النظام الذي يقوم بطباعة جملة Hello World على الشاشة و المكتوب في الفصل الرابع و سنشغل نظامنا بمحمل الإقلاع هذا , لا يمكننا نفس الملف الذي برمجناه حرفيا في الفصل الرابع و انما علينا القيام ببعض التعديلات الصغيرة , أحد التعديلات التي تلفت النظر أنه يجب الانتباه أن الملف سيتم تحميله إلى موقع مختلف ضمن الذاكرة و هو 0x1000:0000 عوضا عن 0000:7C00, تعديل آخر أنه يمكننا التخلص و حذف معطيات سجل الإقلاع دوس DOS Boot Record data.

سنبدأ شفرتنا بتحديد قطع المعطيات و المكدس data and stack segments و مؤشـــر المكدس stack pointer , و سنقوم بذلك على النحو التالي :

تخزن قطعة الشفرة الحالية ضمن المسجل CS و تجمع المعطيات الستاتيكية إلى الملـــف التنفيذي هنا , لذلك سنستخدمه كقطعة معطيات بالإضافة لدوره , و مبدئيا سنستخدم هـــذا كذلك في قطعة المكدس و من المحتمل أن نغيره مستقبلا .

```
        mov   ax, cs       ; Get the current segment
        mov   ds, ax       ; The data is in this segment
        cli                ; disable interrupts while changing stack
        mov   ss, ax       ; We'll use this segment for the stack too
        mov   sp, 0xfffe   ; Start the stack at the top of the segment
        sti                ; Reenable interrupts
```

أخيرا يمكننا التخلص من بعض الأسطر في نهاية البرنامج التي تقوم توقيع قطاع الإقلاع و تتأكد أن حجم الملف هو 512 بايت فلسنا بحاجة لذلك الآن و ستجد نظام تشغيلنا تحـــت الاسم lesson6.asm في القرص المرفق , قم بترجمة البرنامج و نسخه للقرص المـــرن باسم آخر

```
nasm lesson6.asm -o lesson6.bin
copy lesson6.bin a:\LOADER.BIN
```

و يصبح برنامج Hellowrld كالتالي:

```
-----------------------------------------------------------------------;
 ;Hello World Operating System;
 ;Disclaimer: I am not responsible for any results of the use of the contents
 ;of this file
-----------------------------------------------------------------------;

------------------------------------------- ;
  ;Here is the operating system entry point
------------------------------------------- ;
begin:
        mov     ax, cs          ;Get the current segment
        mov     ds, ax          ;The data is in this segment
        cli                     ;disable interrupts while changing stack
        mov     ss, ax          ;We'll use this segment for the stack too
        mov     sp, 0xfffe      ;Start the stack at the top of the segment
        sti                     ;Reenable interrupts

        mov     si, msg         ;load address of our message
        call    putstr          ;print the message

hang:
        jmp     hang            ;just loop forever.

------------------------------------------- ;
 ;data for our program

msg     db      'Hello, World!', 0

------------------------------------------- ;
 ;Print a null-terminated string on the screen
------------------------------------------- ;
```

```
putstr:
        lodsb           ;AL = [DS:SI[
        or al, al       ;Set zero flag if al=0
        jz putstrd      ;jump to putstrd if zero flag is set
        mov ah, 0x0e    ;video function 0Eh (print char(
        mov bx, 0x0007  ;color
        int 0x10
        jmp putstr
putstrd:
        retn
```

الآن كل ما هو مطلوب منك نسخ ملف نظام تشغيلك ذو الاسم Loader.bin مع برنامج الإقلاع boot12.bin إلى القرص المرن و من ثم نسخ برنامج الإقلاع boot12.bin إلى قطاع الإقلاع على القرص المرن كما تعلمنا سابقا .

أعد الإقلاع و استمتع بالجملة Helo,World! .

⑦ الشفرة المصدرية للبرامج .

h.asm:

```
-----------------------------------------------------------------------;
 ;Simple boot program that prints the letter 'H'
  ;and then hangs
 ;Joel Gompert 2001
;
 ;Disclaimer: I am not responsible for any results of the use of the contents
   ;of this file
-----------------------------------------------------------------------;
        org 0x7c00      ;This is where BIOS loads the bootloader


 ;Execution begins here
entry:
        jmp short begin ; jump over the DOS boot record data


-------------------------------------------------------------------- ;
 ;data portion of the "DOS BOOT RECORD"
-------------------------------------------------------------------- ;
brINT13Flag     DB      90H             ; 0002h - 0EH for INT13 AH=42 READ
brOEM           DB      'MSDOS5.0'      ; 0003h - OEM name & DOS version (8 chars(
brBPS           DW      512             ; 000Bh - Bytes/sector
brSPC           DB      1               ; 000Dh - Sectors/cluster
brResCount      DW      1               ; 000Eh - Reserved (boot) sectors
brFATs          DB      2               ; 0010h - FAT copies
brRootEntries   DW      0E0H          0011 ;h - Root directory entries
brSectorCount   DW      2880          0013 ;h - Sectors in volume, < 32MB
brMedia         DB      240           0015 ;h - Media descriptor
brSPF           DW      9               ; 0016h - Sectors per FAT
brSPH           DW      18              ; 0018h - Sectors per track
brHPC           DW      2             001 ;Ah - Number of Heads
brHidden        DD      0               ; 001Ch - Hidden sectors
brSectors       DD      0     0020 ;        h - Total number of sectors
               DB      0               ; 0024h - Physical drive no.
               DB      0               ; 0025h - Reserved (FAT32(
               DB      29H             ; 0026h - Extended boot record sig
brSerialNum     DD      404418EAH       ; 0027h - Volume serial number (random(
brLabel         DB      'Joels disk '   ; 002Bh - Volume label  (11 chars(
brFSID          DB      'FAT12   '      ; 0036h - File System ID (8 chars(
-------------------------------------------------------------------------;

------------------------------------------- ;
  ;Boot program code begins here
------------------------------------------- ;
 ;boot code begins at 0x003E
begin:
        mov     ah, 0x0e        ;Function to print a character to the screen
        mov     al, 'H'        ; Which character to print
        mov     bl, 7           ;color/style to use for the character
        int     0x10            ;print the character

hang:
        jmp     hang            ;just loop forever.

---------------------------------------------;
```

```
size    equ      - $entry
%if size+2 > 512
%  error "code is too large for boot sector"
%endif
       times    - 512)size - 2) db 0

       db      0x55, 0xAA              2;byte boot signature
```

helowrld.asm:

```
-----------------------------------------------------------------------;
 ;Hello World Operating System Boot Program
;
 ;Joel Gompert 2001
;
 ;Disclaimer: I am not responsible for any results of the use of the contents
   ;of this file
-----------------------------------------------------------------------;
       org 0x7c00      ;This is where BIOS loads the bootloader


 ;Execution begins here
entry:
       jmp short begin ; jump over the DOS boot record data


-------------------------------------------------------------------- ;
 ;data portion of the "DOS BOOT RECORD"
-------------------------------------------------------------------- ;
brINT13Flag     DB      90H             ; 0002h - 0EH for INT13 AH=42 READ
brOEM           DB      'MSDOS5.0'      ; 0003h - OEM name & DOS version (8 chars(
brBPS           DW      512             ; 000Bh - Bytes/sector
brSPC           DB      1               ; 000Dh - Sectors/cluster
brResCount      DW      1               ; 000Eh - Reserved (boot) sectors
brFATs          DB      2               ; 0010h - FAT copies
brRootEntries   DW      0E0H          0011 ;h - Root directory entries
brSectorCount   DW      2880          0013 ;h - Sectors in volume, < 32MB
brMedia         DB      240           0015 ;h - Media descriptor
brSPF           DW      9               ; 0016h - Sectors per FAT
brSPH           DW      18              ; 0018h - Sectors per track
brHPC           DW      2             001 ;Ah - Number of Heads
brHidden        DD      0               ; 001Ch - Hidden sectors
brSectors       DD      0     0020 ;        h - Total number of sectors
               DB      0               ; 0024h - Physical drive no.
               DB      0               ; 0025h - Reserved (FAT32(
               DB      29H             ; 0026h - Extended boot record sig
brSerialNum     DD      404418EAH       ; 0027h - Volume serial number (random(
brLabel         DB      'Joels disk '   ; 002Bh - Volume label  (11 chars(
brFSID          DB      'FAT12   '      ; 0036h - File System ID (8 chars(
-----------------------------------------------------------------------;


------------------------------------------- ;
  ;Boot program code begins here
------------------------------------------- ;
 ;boot code begins at 0x003E
begin:
       xor     ax, ax          ;zero out ax
       mov     ds, ax          ;set data segment to base of RAM
       mov     si, msg         ;load address of our message
       call    putstr          ;print the message

hang:
       jmp     hang            ;just loop forever.

------------------------------------------- ;
 ;data for our program

msg     db      'Hello, World!', 0

------------------------------------------- ;
 ;Print a null-terminated string on the screen
------------------------------------------- ;
putstr:
```

```
        lodsb           ;AL = [DS:SI[
        or al, al       ;Set zero flag if al=0
        jz putstrd      ;jump to putstrd if zero flag is set
        mov ah, 0x0e    ;video function 0Eh (print char(
        mov bx, 0x0007  ;color
        int 0x10
        jmp putstr
putstrd:
        retn
--------------------------------------------;

size    equ     - $entry
%if size+2 > 512
%  error "code is too large for boot sector"
%endif
        times   - 512)size - 2) db 0

        db      0x55, 0xAA              2;byte boot signature
```

lesson5:

```
-----------------------------------------------------------------------;
 ;Interactivity Example Boot Program
;
 ;Joel Gompert 2001
;
 ;Disclaimer: I am not responsible for any results of the use of the contents
   ;of this file
-----------------------------------------------------------------------;
        org 0x7c00      ;This is where BIOS loads the bootloader


 ;Execution begins here
entry:
        jmp short begin ; jump over the DOS boot record data


-------------------------------------------------------------------- ;
 ;data portion of the "DOS BOOT RECORD"
-------------------------------------------------------------------- ;
brINT13Flag     DB      90H             ; 0002h - 0EH for INT13 AH=42 READ
brOEM           DB      'MSDOS5.0'      ; 0003h - OEM name & DOS version (8 chars(
brBPS           DW      512             ; 000Bh - Bytes/sector
brSPC           DB      1               ; 000Dh - Sectors/cluster
brResCount      DW      1               ; 000Eh - Reserved (boot) sectors
brFATs          DB      2               ; 0010h - FAT copies
brRootEntries   DW      0E0H          0011 ;h - Root directory entries
brSectorCount   DW      2880          0013 ;h - Sectors in volume, < 32MB
brMedia         DB      240           0015 ;h - Media descriptor
brSPF           DW      9               ; 0016h - Sectors per FAT
brSPH           DW      18              ; 0018h - Sectors per track
brHPC           DW      2             001 ;Ah - Number of Heads
brHidden        DD      0               ; 001Ch - Hidden sectors
brSectors       DD      0     0020 ;        h - Total number of sectors
               DB      0               ; 0024h - Physical drive no.
               DB      0               ; 0025h - Reserved (FAT32(
               DB      29H             ; 0026h - Extended boot record sig
brSerialNum     DD      404418EAH       ; 0027h - Volume serial number (random(
brLabel         DB      'Joels disk '   ; 002Bh - Volume label  (11 chars(
brFSID          DB      'FAT12   '      ; 0036h - File System ID (8 chars(
-----------------------------------------------------------------------;


------------------------------------------ ;
  ;Boot program code begins here
------------------------------------------ ;
 ;boot code begins at 0x003E
begin:
        xor     ax, ax          ;zero out ax
        mov     ds, ax          ;set data segment to base of RAM
        mov     si, msg         ;load address of our message
        call    putstr          ;print the message
```

```
loop1:
        xor     ah, ah          ;function 0
        int     0x16            ;get a key from the keyboard

        mov     si, charmsg     ;load address of message
        call    putstr          ;print the message

        mov     ah, 0x0e        ;function print character
        mov     bl, 0x07        ;white on black
        int     0x10

        mov     si, newline     ;print a newline
        call    putstr

        jmp     loop1           ;just loop forever.

-------------------------------------------- ;
 ;data for our program

msg     db      'Press a key'.
newline db      13,10,0
charmsg db      'Character: ',0

-------------------------------------------- ;
 ;Print a null-terminated string on the screen
-------------------------------------------- ;
putstr:
        push    ax
putstrl:
        lodsb                   ;AL = [DS:SI[
        or      al, al          ;Set zero flag if al=0
        jz      putstrd         ;jump to putstrd if zero flag is set
        mov     ah, 0x0e        ;video function 0Eh (print char(
        mov     bx, 0x0007      ;color
        int     0x10
        jmp     putstrl
putstrd:
        pop     ax
        retn
---------------------------------------------;

size    equ      - $entry
%if size+2 > 512
%  error "code is too large for boot sector"
%endif
        times    - 512)size - 2) db 0

        db      0x55, 0xAA              2;byte boot signature
```

lesson6:

```
; ---------------------------------------------------------------------
 ;Hello World Operating System
;
 ;Joel Gompert 2001
;
 ;Disclaimer: I am not responsible for any results of the use of the contents
   ;of this file
-------------------------------------------------------------------------;

-------------------------------------------- ;
  ;Here is the operating system entry point
-------------------------------------------- ;
begin:
        mov     ax, cs          ;Get the current segment
        mov     ds, ax          ;The data is in this segment
        cli                     ;disable interrupts while changing stack
        mov     ss, ax          ;We'll use this segment for the stack too
        mov     sp, 0xfffe      ;Start the stack at the top of the segment
        sti                     ;Reenable interrupts

        mov     si, msg         ;load address of our message
```

```
        call    putstr          ;print the message

hang:
        jmp     hang            ;just loop forever.

------------------------------------------- ;
 ;data for our program

msg     db      'Hello, World!', 0

------------------------------------------- ;
 ;Print a null-terminated string on the screen
------------------------------------------- ;
putstr:
        lodsb           ;AL = [DS:SI[
        or al, al       ;Set zero flag if al=0
        jz putstrd      ;jump to putstrd if zero flag is set
        mov ah, 0x0e    ;video function 0Eh (print char(
        mov bx, 0x0007  ;color
        int 0x10
        jmp putstr
putstrd:
        retn
```

Boot12.asm:

```
 ;boot12.asm  FAT12 bootstrap for real mode image or loader
 ;Version 1.0, Jul 5, 1999
 ;Sample code
 ;by John S. Fine  johnfine@erols.com
 ;I do not place any restrictions on your use of this source code
 ;I do not provide any warranty of the correctness of this source code
                                                                                ;
;
 ;Documentation:
;
 ;I)    BASIC features
 ;II)   Compiling and installing
 ;III)  Detailed features and limits
 ;IV)   Customization
_____________________________________________________________________________;
;
 ;I)    BASIC features
;
    ;This boot sector will load and start a real mode image from a file in the
 ;root directory of a FAT12 formatted floppy or partition.
;
    ;Inputs:
 ;DL = drive number
;
    ;Outputs:
 ;The boot record is left in memory at 7C00 and the drive number is patched
 ;into the boot record at 7C24.
 ;SS = DS = 0
 ;BP = 7C00
                                                                                ;
;
 ;II)   Compiling and installing
;
  ;To compile, use NASM
;
  ;nasm boot12.asm -o boot12.bin
;
  ;Then you must copy the first three bytes of BOOT12.BIN to the first three
  ;bytes of the volume and copy bytes 0x3E through 0x1FF of BOOT12.BIN to
  ;bytes 0x3E through 0x1FF of the volume.  Bytes 0x3 through 0x3D of the
  ;volume should be set by a FAT12 format program and should not be modified
  ;when copying boot12.bin to the volume.
;
  ;If you use my PARTCOPY program to install BOOT12.BIN on A:, the
  ;commands are:
;
```

```
  ;partcopy boot12.bin 0 3 -f0
  ;partcopy boot12.bin 3e 1c2 -f0 3e
;
  ;PARTCOPY can also install to a partition on a hard drive.  Please read
  ;partcopy documentation and use it carefully.  Careless use could overwrite
  ;important parts of your hard drive.
;
  ;You can find PARTCOPY and links to NASM on my web page at
  ;http://www.erols.com/johnfine/
                                                                            ;
;
 ;III)  Detailed features and limits
;
   ;Most of the limits are stable characteristics of the volume.  If you are
 ;using boot12 in a personal project, you should check the limits before
 ;installing boot12.  If you are using boot12 in a project for general
 ;distribution, you should include an installation program which checks the
 ;limits automatically.
;
 ;CPU:  Supports any 8088+ CPU.
;
 ;Volume format:  Supports only FAT12.
;
 ;Sector size:  Supports only 512 bytes per sector.
;
 ;Drive/Partition:  Supports whole drive or any partition of any drive number
 ;supported by INT 13h.
;
 ;Diskette parameter table:  This code does not patch the diskette parameter
 ;table.  If you boot this code from a diskette that has more sectors per
 ;track than the default initialized by the BIOS then the failure to patch
 ;that table may be a problem.  Because this code splits at track boundaries
 ;a diskette with fewer sectors per track should not be a problem.
;
 ;File position:  The file name may be anywhere in the root directory and the
 ;file may be any collection of clusters on the volume.  There are no
 ;contiguity requirements.  (But see track limit.(
;
 ;Track boundaries:  Transfers are split on track boundaries.  Many BIOS's
 ;require that the caller split floppy transfers on track boundaries.
;
64 ;Kb boundaries:  Transfers are split on 64Kb boundaries.  Many BIOS's
 ;require that the caller split floppy transfers on track boundaries.
;
 ;Cluster boundaries:  Transfers are merged across cluster boundaries whenever
 ;possible.  On some systems, this significantly reduces load time.
;
 ;Cluster 2 limit:  Cluster 2 must start before sector 65536 of the volume.
 ;This is very likely because only the reserved sectors (usually 1) and
 ;the FAT's (two of up to 12 sectors each) and the root directory (usually
 ;either 15 or 32 sectors) precede cluster 2.
;
 ;Track limit:  The entire image file must reside before track 32768 of the
 ;entire volume.  This is true on most media up to 1GB in size.  If it is a
 ;problem it is easy to fix (see boot16.asm).  I didn't expect many people
 ;to put FAT12 partitions beyond the first GB of a large hard drive.
;
 ;Memory boundaries:  The FAT, Root directory, and Image must all be loaded
 ;starting at addresses that are multiples of 512 bytes (32 paragraphs.(
;
 ;Memory use:  The FAT and Root directory must each fit entirely in the
 ;first 64Kb of RAM.  They may overlap.
;
 ;Root directory size:  As released, it supports up to 928 entries in the
 ;root directory.  If ROOT_SEG were changed to 0x7E0 it would support up
 ;to 1040.  Most FAT12 volumes have either 240 or 512 root directory
 ;entries.
                                                                            ;
;
 ;IV)   Customization
;
   ;The memory usage can be customized by changing the _SEG variables (see
 ;directly below.(
;
   ;The file name to be loaded and the message displayed in case of error
 ;may be customized (see end of this file.(
```

```
;
   ;The ouput values may be customized.  For example, many loaders expect the
 ;bootsector to leave the drive number in DL.  You could add "mov dl,[drive"[
 ;at the label "eof.":
;
   ;Some limits (like maximum track) may be removed.  See boot16.asm for
 ;comparison.
;
   ;Change whatever else you like.  The above are just likely possibilities.
                                                                                ;


 ;Change the _SEG values to customize memory use during the boot.
 ;When planning memory use, remember:
;
  (* ;Each of ROOT_SEG, FAT_SEG, and IMAGE_SEG must be divisible by 0x20
;
  (* ;None of ROOT, FAT or IMAGE should overlap the boot code itself, or
     ;its stack.  That means: avoid paragraphs 0x7B0 to 0x7DF.
;
  (* ;The FAT area must not overlap the IMAGE area.  Either may overlap
     ;the ROOT area;  But, if they do then the root will not remain in
     ;memory for possible reuse by the next stage.
;
  (* ;The FAT area and the root area must each fit within the first 64Kb
     ;excluding BIOS area (paragraphs 0x60 to 0xFFF.(
;
  (* ;A FAT12 FAT can be up to 6Kb (0x180 paragraphs.(
;
  (* ;A FAT12 root directory is typically either 0x1E0 or 0x400 paragraphs
     ;long, but larger sizes are possible.
;
  (* ;The code will be two bytes shorter when FAT_SEG is 0x800 than when it
     ;is another value.  (If you reach the point of caring about two bytes.(
;
%define ROOT_SEG       0x60
%define FAT_SEG               0x800
%define IMAGE_SEG      0x1000

%if ROOT_SEG & 31
%  error "ROOT_SEG must be divisible by 0x20"
%endif
%if ROOT_SEG > 0xC00
%  error "Root directory must fit within first 64Kb"
%endif
%if FAT_SEG & 31
%  error "FAT_SEG must be divisible by 0x20"
%endif
%if FAT_SEG > 0xE80
%  error "FAT must fit within first 64Kb"
%endif
%if IMAGE_SEG & 31
%  error "IMAGE_SEG must be divisible by 0x20"
%endif

 ;The following %define directives declare the parts of the FAT12 "DOS BOOT
 ;RECORD" that are used by this code, based on BP being set to 7C00.
;
%define sc_p_clu       bp+0Dh         ;byte  Sectors per cluster
%define sc_b4_fat      bp+0Eh         ;word  Sectors (in partition) before FAT
%define fats           bp+10h         ;byte  Number of FATs
%define dir_ent               bp+11h         ;word  Number of root directory entries
%define sc_p_fat       bp+16h         ;word  Sectors per FAT
%define sc_p_trk       bp+18h         ;word  Sectors per track
%define heads          bp+1Ah         ;word  Number of heads
%define sc_b4_prt      bp+1Ch         ;dword Sectors before partition
%define drive          bp+24h         ;byte  Drive number

        org     0x7C00

entry:
        jmp     short begin

------------------------------------------------- ;
 ;data portion of the "DOS BOOT RECORD"
------------------------------------------------------------------- ;
```

```
brINT13Flag     DB      90H             ; 0002h - 0EH for INT13 AH=42 READ
brOEM           DB      'MSDOS5.0'      ; 0003h - OEM ID - Windows 95B
brBPS           DW      512             ; 000Bh - Bytes per sector
brSPC           DB      1               ; 000Dh - Sector per cluster
brSc_b4_fat    DW      1               ; 000Eh - Reserved sectors
brFATs          DB      2               ; 0010h - FAT copies
brRootEntries   DW      0E0H          0011 ;h - Root directory entries
brSectorCount   DW      2880          0013 ;h - Sectors in volume, < 32MB
brMedia         DB      240           0015 ;h - Media descriptor
brSPF           DW      9               ; 0016h - Sectors per FAT
brSc_p_trk     DW      18              ; 0018h - Sectors per head/track
brHPC           DW      2             001 ;Ah - Heads per cylinder
brSc_b4_prt    DD      0               ; 001Ch - Hidden sectors
brSectors       DD      0     0020 ;        h - Total number of sectors
brDrive        DB      0               ; 0024h - Physical drive no.
               DB      0               ; 0025h - Reserved (FAT32(
               DB      29H             ; 0026h - Extended boot record sig (FAT32(
brSerialNum     DD      404418EAH       ; 0027h - Volume serial number
brLabel         DB      'Joels disk '   ; 002Bh - Volume label
brFSID          DB      'FAT12   '      ; 0036h - File System ID
------------------------------------------------------------------------;


begin:
        xor     ax, ax
        mov     ds, ax
        mov     ss, ax
        mov     sp, 0x7C00
        mov     bp, sp
        mov     ]drive], dl     ;Drive number

        mov     al, [fats]      ;Number of FATs
        mul     word [sc_p_fat]         ; * Sectors per FAT
        add     ax, [sc_b4_fat]         ; + Sectors before FAT
                                ;AX = Sector of Root directory

        mov     si, [dir_ent]   ;Max root directory entries
        mov     cl, 4
        dec     si
        shr     si, cl
        inc     si              ;SI = Length of root in sectors

        mov     di, ROOT_SEG/32         ;Buffer (paragraph / 32(
        call    read_16         ;Read root directory
        push    ax              ;Sector of cluster two
%define sc_clu2 bp-2            ;Later access to the word just pushed is via bp

        mov     dx, [dir_ent]   ;Number of directory entries
        push    ds
        pop     es
        mov     di, ROOT_SEG*16

search:
        dec     dx              ;Any more directory entries?
        js      error           ;No
        mov     si, filename    ;Name we are searching for
        mov     cx, 11           11;characters long
        lea     ax, [di+0x20]   ;Precompute next entry address
        push    ax
        repe cmpsb              ;Compare
        pop     di
        jnz     search          ;Repeat until match

        push word [di-6]        ;Starting cluster number

        mov     ax, [sc_b4_fat]         ;Sector number of FAT
        mov     si, [sc_p_fat] ;Length of FAT
        mov     di, FAT_SEG/32 ;Buffer (paragraph / 32(
        call    read_16         ;Read FAT

next:
        pop     bx              ;Cluster number
        mov     si, bx          ;First cluster in this sequence
        mov     ax, bx          ;Last cluster in this sequence

:0.
```

```
        cmp     bx, 0xFF8       ;End of file?
        jae     2.               ;Yes
        inc     ax              ;Last cluster plus one in sequence

        ;Look in FAT for next cluster
        mov     di, bx          ;Cluster number
        rcr     bx, 1            1.5;byte entry per cluster
                                ;bx = 0x8000 + cluster/2
                                ;c-bit set for odd clusters

        mov     bx, [bx+di+FAT_SEG*16-0x8000[
        jnc     1.
        shr     bx, 1
        shr     bx, 1
        shr     bx, 1
        shr     bx, 1
:1.     and     bh, 0xF

        cmp     ax, bx          ;Is the next one contiguous?
        je      0.              ;Yes: look further ahead
:2.     sub     ax, si          ;How many contiguous in this sequence?
        jz      eof             ;None, must be done.

        push    bx              ;Save next (eof or discontiguous) cluster

        mov     bl, [sc_p_clu] ;Sectors per cluster
        mov     bh, 0             ;as a word
        mul     bx              ;Length of sequence in sectors
:3.     mov     di, IMAGE_SEG/32 ;Destination (paragraph / 32(
        add      ,[1+3.]ax      ; Precompute next destination
        xchg    ax, si          ;AX = starting cluster ;SI = length in sectors
        dec     ax
        dec     ax              ;Starting cluster minus two
        mul     bx               * ;sectors per cluster
        add     ax, [sc_clu2]  ; + sector number of cluster two
        adc     dl, dh          ;Allow 24-bit result

        call    read_32         ;Read it
        jmp     short next      ;Look for more

eof:
        jmp     IMAGE_SEG:0

error:  mov     si, errmsg      ;Same message for all detected errors
        mov     ax, 0xE0D       ;Start message with CR
        mov     bx, 7
:1.     int     10h
        lodsb
        test    al, al
        jnz     1.
        xor     ah, ah
        int     16h             ;Wait for a key
        int     19h             ;Try to reboot

read_16:
        xor     dx, dx

read_32:
;
 ;Input:
    ;dx:ax = sector within partition
    ;si    = sector count
    ;di    = destination segment / 32
;
 ;The sector number is converted from a partition-relative to a whole-disk
) ;LBN) value, and then converted to CHS form, and then the sectors are read
 ;into (di*32):0.
;
 ;Output:
    ;dx:ax  updated (sector count added(
    ;di     updated (sector count added(
    ;si = 0
    ;bp, ds preserved
    ;bx, cx, es modified

:1.     push    dx                      );high) relative sector
```

```
        push    ax                      );low) relative sector

        add     ax, [sc_b4_prt]                 ;Convert to LBN
        adc     dx, [sc_b4_prt+2[

        mov     bx, [sc_p_trk]          ;Sectors per track
        div     bx                      ;AX = track ;DX = sector-1
        sub     bx, dx                  ;Sectors remaining, this track
        cmp     bx, si                  ;More than we want?
        jbe     2.                      ;No
        mov     bx, si                  ;Yes: Transfer just what we want
:2.     inc     dx                      ;Sector number
        mov     cx, dx                  ;CL = sector ;CH = 0
        cwd                             );This supports up to 32767 tracks
        div     word [heads]            ;Track number / Number of heads
        mov     dh, dl                  ;DH = head

        xchg    ch, al                  ;CH = (low) cylinder  ;AL=0
        ror     ah, 1                   ;rotate (high) cylinder
        ror     ah, 1
        add     cl, ah                  ;CL = combine: sector, (high) cylinder

        sub     ax, di
        and     ax, byte 0x7F           ;AX = sectors to next 64Kb boundary
        jz      3.                      ;On a 64Kb boundary already
        cmp     ax, bx                  ;More than we want?
        jbe     4.                      ;No
:3.     xchg    ax, bx                  ;Yes: Transfer just what we want
:4.     push    ax                      ;Save length
        mov     bx, di                  ;Compute destination seg
        push    cx
        mov     cl, 5
        shl     bx, cl
        pop     cx
        mov     es, bx
        xor     bx, bx                  ;ES:BX = address
        mov     dl, [drive]             ;DL = Drive number
        mov     ah, 2                   ;AH = Read command
        int     13h                     ;Do it
        jc      error
        pop     bx                      ;Length
        pop     ax                      );low) relative sector
        pop     dx                      );high) relative sector
        add     ax, bx                  ;Update relative sector
        adc     dl, dh
        add     di, bx                  ;Update destination
        sub     si, bx                  ;Update count
        jnz     1.                      ;Read some more
        ret

errmsg  db      ",10Error Executing FAT12 bootsector",13
        db      ",10Press any key to reboot",13,10,0

size    equ      - $entry
%if size+11+2 > 512
%  error "code is too large for boot sector"
%endif
        times    - 512)size - 11 - 2) db 0

filename db     "LOADER  BIN"           ; 11byte name
        db      0x55, 0xAA                2;byte boot signature
```

ملحق

يتحدث الملحق عن أهم آلية الإقلاع بشكل عام و برنامجي الإقلاع LILO و GRUB المستخدمين مع نظام التشغيل LINUX و للأمانة العلمية قمت باقتباسه من موقع الخوارزمي و لم أقم بتأليفه .

www.alepposoft.com/booting/appendex.zip

المراجع

http://cse.unl.edu/~jgompert/OS/
http://www.nondot.org/sabre/os/articles
http://www.nondot.org/sabre/os/files/Booting
http://my.execpc.com/~geezer/osd/boot/index.htm
http://www.alkhawarzmi.com

حقوق الملكية

جميع الحقوق محفوظة لكل عباد الله الناطقين باللغة العربية

أسوة بقول رسول الله صلى الله عليه و آله و سلم

"لأن يمشي أحدكم في قضاء حاجة أفضل من أن يعتكف في مسجدي هذا شهرين" رواه الحاكم.

"الخلق كلهم عيال الله وأحب الخلق إلى الله انفعهم لعياله"

رواه الطبراني في الكبير والأوسط

و تحت رخصة المصادر المفتوحة GNU

http://www.alepposoft.com/info/showthread.php?t=2126

و يمكنك نشرها و توزيعها كما تريد بشرط الإشارة لكاتبها و تزويدي بكل التحديثات عليها و نشر أي وثيقة تعتمد على الوثيقة السابقة تحت نفس الترخيص السابق .

سأكون ممنونا إذا قمت بالإشارة لموقع http://www.Alepposoft.com/ مصدر للمقالة عند الاقتباس .أتمنى أن يستفيد منها اكبر عدد ممكن و أرحب بكل الاقتراحات من اجل إصدار نسخة مطورة .السعر مجاني سأكون مسرورا لو تذكرتني ببطاقة بريدية .

يمكنك الحصول على الأدوات اللازمة من الرابط :

www.alepposoft.com/booting/tools.zip

يمكنك الحصول على البرامج و ترجمتها من الرابط :

www.alepposoft.com/booting/programs.zip

فادي عمروش
هندسة معلوماتية – قسم شبكات
الموقع: http://www.alepposoft.com/info
البريد الالكتروني: alfady@scs-net.org
موبايل : 093676049
سورية – حلب