١ إقلاع الحاسب ومحمل النظام Bootloader

أحد أهم الأساسيات في برمجة نظام تشغيل هي كتابة محملٌ له ، هذا المحمل يعمل على نسخ نواة النظام من أحد الأقراص الى الذاكرة الرئيسية ثم ينقل التنفيذ الى النواة ، وهكذا تنتهي دورة عمل المحمل ويبدأ نظام التشغيل متمثلا في النواة بالبدء بتنفيذ الاوامر والمهمات وتلبية إحتياجات المستخدم. في هذا الفصل سندرس كيفية برمجة المحمل وماهيته وسيتم الاقلاع من قرص مرن بنظام FAT12 ، فالغرض هذه المرحلة هو دراسة أساسيات المحمل وتحميل وتنفيذ نواة مبسطة .

١.١ إقلاع الحاسب

إقلاع الحاسب (Boot-Strapping) هي أول خطوة يقوم بها الجهاز عند وصله بالكهرباء لتحميل نظام التشغيل، وتبدأ هذه العملية مباشرة عند الضغط على مفتاح التشغيل في الحاسب ، حيث ترسل إشارة كهربائية[١] الى اللوحة الام (MotherBoard) والتي تقوم بتوجيهها الى وحدة موزد الطاقة (Power Supply Unit). بعد ذلك يأتي دور وحدة PSU لكي تقوم بمهمة تزويد الحاسب وملحقاته بالكمية المطلوبة من الطاقة، وإرسال اشارة Power Good إلى اللوحة الام وبالتحديد الى نظام ال BIOS . تدل هذه الاشارة على أنه تم تزويد الطاقة الكافية ، وفورا سيبدأ برنامج الفحص الذاتي (Power on Self Test) الذي يعرف اختصاراً ب POST بفحص أجهزة ومحلقات الحاسب (مثل الذاكرة ولوحة المفاتيح والماوس والناقل التسلسلي ...الخ) والتأكد من أنها سليمة. بعدها يقوم ال POST بنقل التحكم الى نظام ال BIOS حيث سيقوم ال POST بتحميل ال BIOS الى نهاية الذاكرة 0xFFFF0 و سيقوم أيضا بوضع تعليمة قفز (jump) في أول عنوان في الذاكرة الى نهاية الذاكرة ، كذلك من مهام ال POST هي تصفير المسجلين CS:IP وهذا يعني أن أول تعليمية سينفذها المعالج هي تعليمة القفز الى نهاية الذاكرة وبالتحديد الى ال BIOS . يستلم ال BIOS التحكم ويبدأ في انشاء جدول المقاطعات (Interrupt Vector Table) وتوفير العديد من المقاطعات ،ويقوم بالمزيد من عمليات الفحص والاختبار للحاسب ، وبعد ذلك يبدأ في مهمة البحث عن نظام تشغيل في الاجهزة الموجودة بناءا على ترتيبها في اعدادات ال BIOS في برنامج Setup ،وفي حالة لم يجد ال BIOS جهازا قابلا للاقلاع في كل القائمة فانه يصدر رسالة خطأ بعدم توفر نظام تشغيل ويوقف الحاسب عن العمل (Halt) ، وفي حالة توفر جهازاً قابلاً للإقلاع سيقوم ال BIOS بتحميل القطاع الأول منه (يحوي هذا القطاع على برنامج المحمل) الى الذاكرة الرئيسية وبالتحديد الى العنوان الفيزيائي 0x07c00 وسيُنقَل التنفيذ الى المحمل.

[١] هذه الإشارة تحوي على بت (bit) تدل قيمته اذا كانت 1 على أنه تم تشغيل الحاسب.

خلال هذه المهمة (اقلاع النظام) يوفر لنا نظام ال BIOS العديد من المقاطعات على جدول المقاطعات والذي يتم انشائه بدءاً من العنوان 0x0 ، هذه المقاطعات هي خدمات يوفرها لنا نظام البايوس لاداء وظيفة معينة مثل مقاطعة لطباعة حرف على الشاشة. واحدة من أهم المقاطعات التي يستخدمها نظام البايوس للبحث عن جهاز الاقلاع هي المقاطعة int 0x19 حيث تكمن وظيفتها في البحث عن هذا الجهاز ومن ثم تحميل القطاع الأول منه الى العنوان الفيزيائي 0x07c00 ونقل التنفيذ اليه . طريقة البحث والتحميل ليست بالامر المعقد حيث على هذه المقاطعة البحث في أول قطاع (من أي جهاز موجود على قائمة الاجهزة القابلة للاقلاع) عن التوقيع 0xAA55 وهي عبارة عن بايتين يجب أن تكون على آخر القطاع الاول تدل على أن هذا الجهاز قابل للاقلاع. ومن الجدير بالذكر أن المقاطعات التي يوفرها لنا نظام البايوس يمكن استخدامها فقط اذا كان المعالج يعمل في النمط الحقيقي Real Mode أما إذا تم تغيير نمط المعالج الى النمط المحمي Protected Mode – كما سنرى ذلك لاحقا– فانه لن يمكن الاستفادة من هذه المقاطعات بل سيتسبب استخدامها في حدوث استثناءات (Exception) توقف عمل الحاسب.

٢.١ محمل النظام Bootloader

محمل النظام هو برنامج وظيفته الرئيسية هي تحميل نواة نظام التشغيل ونقل التنفيذ اليها.هذا المحمل يجب ان تتوفر فيه الشروط الاتية :

١. حجم البرنامج يجب أن يكون 512 بايت بالضبط.

٢. أن يتواجد على القطاع الأول في القرص : القطاع رقم 1 ، الرأس 0 ، المسار 0 ، وأن يحمل التوقيع المعروف.

٣. أن يحوي شفرة تحميل النواة ونقل التنفيذ اليها.

٤. أن يكون البرنامج object code خالي من أي أضافات (header,symbol table,...etc) وهو ما يعرف أيضا بـــ Flat Binary .

الشرط الأول يُقيد وظيفة المحمل وقدرته على توفير خصائص متقدمة[٢]، حيث أن هذا الحجم لا يكفي لكي يبحث المحمل عن نواة النظام وتمهيد الطريق لها للبدء بتنفيذها ، وبسبب أن النواة ستكون 32-bit فانه يجب تجهيز العديد من الأشياء بدءاً من جداول الواصفات (العامة والخاصة) وتفعيل البوابة A20 وانتهاءاً بتغيير نمط المعالج الى النمط المحمي والقفز الى النواة للمباشرة في تنفيذها . كل ذلك يحتاج الى حجم أكبر من الحجم المشروط لذلك عادة ما يلجأ مبرمجوا المحملات الى تجزئيها على مرحلتين وهو ما يسمى ب Multi-Stage Boot Loader . الشرط الثاني للمحمل وهو أن يتواجد على أول قطاع في القرص وهو يحمل العنوان الفيزيائي التالي:

- القطاع رقم 1

[٢] مثل خاصية ال Safe Mode

- المسار رقم 0
- الرأس رقم 0

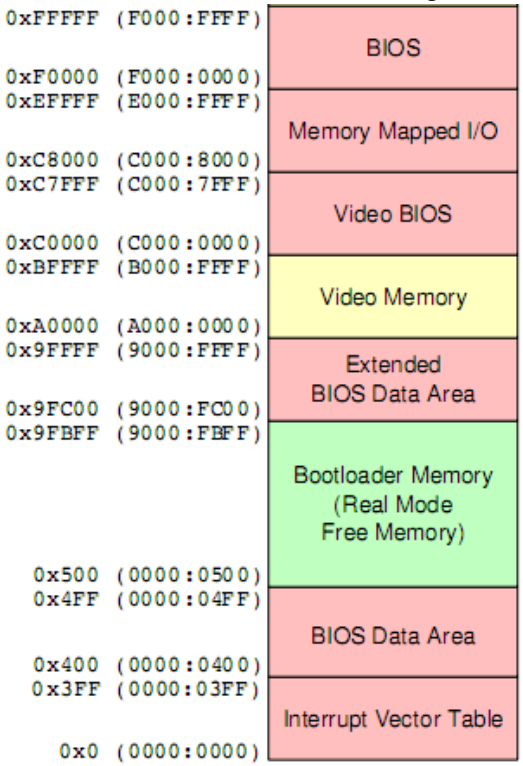
وتحقيقُ هذا الشرط ليس بالأمر المعقد خصوصا مع توفر العديد من الادوات التي تساعد على نسخ مقطع من قرص ما الى مقطع في قرص آخر ، أما الشق الثاني من الشرط فهو متعلق بتمييز القطاع الاول كقطاع قابل للاقلاع من غيره ، حيث حتى يكون القطاع قابلا للاقلاع فانه يجب أن يحمل التوقيع 0xAA55 في البايت رقم 510 و 511 . وبدون هذا التوقيع فان البايوس (وتحديدا مقاطعة رقم 0x19) لن تتعرف على هذا القطاع كقطاع قابلٌ للإقلاع. أما الشرط الثالث فهو شرط اختياري وليس اجباري ، فمن الممكن أن تكون وظيفة المحمل هي عرض رسالة ترحيب فقط ! ولكن في أغلب الحالات الواقعية يجب أن تُحمِّل النواة وتُنفِّذ عن طريق هذا المحمل. وقد أسلفنا وذكرنا أن تحميل نواة 32-bit يختلف عن تحميل نواة 16-bit ، حيث في الاولى يجب تجهيز الطريق أمام النواة وتفعيل بعض الخصائص لذلك وَجب تقسيم مهمة محمل النظام الى مرحليتن – كماسنرى ذلك – ، أما في حالة كانت النواة 16-bit فانه يمكن تحميلها بمرحلة واحدة فقط . والشرط الاخير يتعلق بصيغة الملف التنفيذي للمحمل، حيث أغلب المترجمات تخرج صيغ تنفيذية تحوي على الكثير من المعلومات المضافة من قبله (كصيغ ELF,PE,COFF,...etc) وهذا ما يجعل عملية تنفيذ المحمل وتشغيله من قبل البايوس مستحيلة ، فالبايوس عندما يقرأ محمل النظام الى الذاكرة فانه ينقل التنفيذ الى أول بايت فيه والذي يجب ان يكون قابل للتنفيذ وليس معلومات أو هيدر عن الملف – كما في حالة الصيغ السابق ذكرها– . لذلك يجب أن تكون صيغة المحمل هي عبارة عن الصيغة الثنائية المقابلة للأوامر الموجودة فيه بدون أي اضافات أي Object Code او Flat Binary.

ويجدر بنا الحديث عن لغة برمجة محمِّل النظام، فغالبا تستخدم لغة التجميع (Assembly 16-bit) لأسباب كثيرة ، منها أن الحاسب عندما يبدأ العمل فان المعالج يكون في النمط الحقيقي تحقيقا لأغراض التوفقية (Backward Compatibility) مع الأجهزة السابقة ، أيضا استخدام لغة التجميع 16-bit يجعل من الممكن استدعاء مقاطعات وخدمات البايوس – قبل الانتقال الى بيئة 32-bit – ، أخيراً لا حاجة لملفات وقت التشغيل run-time library ، حيث أن لغة التجميع ماهي الا مختصرات للغة الآلة Machine Language. كل هذا لا يجعل عملية كتابة محمِّل النظام بلغة السي مستحيلا ! فهناك كمٌّ كبير من المحملات تستخدم لغة السي والتجميع في آن واحد (مثل GRUB,NTLDR,LILO...etc) ، لكن قبل برمجة مثل هذه المحملات يجب برمجة بعض ملفات ال run-time لتوفير بيئة لكي تعمل برامج السي عليها ، أيضا يجب كتابة loader لكي يقرأ الصيغة الناتجة من برنامج السي ويبدأ التنفيذ من دالة ال main .

٣.١ مخطط الذاكرة

أثناء مرحلة الإقلاع وعندما يُنقل التنفيذ الى محمل النظام فان الذاكرة الرئيسية تأخذ الشكل ١.١ .وأول ١٠٢٤ بايت تستخدم من قبل جدول المقاطعات الذي يحوي عنوان دالة التنفيذ لكل مقاطعة للبايوس ،

شكل ١.١: مخطط محتويات الذاكرة الرئيسية



يليها منطقة بيانات البايوس ثم مساحة ذاكرة خالية تحوي العنوان 0x07c00 وهو العنوان الذي ينقل البابوس التنفيذ اليه (عنوان برنامج محمل النظام) ، ويليها منطقة بيانات بايوس الموسعة وذاكرة الفيديو والتي بمجرد الكتابة عليها تظهر الأحرف على الشاشة (Memory Mapped) ويليها البايوس على ذاكرة الفيديو ومناطق محجوزة من الذاكرة لبعض أجهزة الإدخال والإخراج ومن ثم البايوس والذي يبدأ من العنوان 0xf0000 وهو موجود على ذاكرة الروم (Memory Mapped) .

٤.١ برمجة محمل النظام

المثال ١.١ يوضح أصغر محمل للنظام يمكن كتابته وتنفيذه ، باستخدام المجمع NASM[٣] وهو مجمع متعدد المنصات ويوفر ميزة انتاج ملفات ثنائية object code .

[٣] راجع الملحق **??** لمعرفة كيفية استخدام المجمع لترجمة المحمل وكيفية نسخه الى floppy disk or CD ليتم القلاع منه سواءاً كان على جهاز فعلي أو على جهاز تخيلي (Virtual Machine) .

Example ١.١: Smallest Bootloader

```
١
٢ ;Simple Bootloader do nothing.
٣
٤ bits 16       ; 16—bit real mode.
٥
٦ start:        ; label are pointer.
٧     cli     ; clear interrupt.
٨     hlt     ; halt the system.
٩
١٠     times 510—($—$$)  db    0   ; append zeros.
١١
١٢     ; $ is the address of first instruction (should be 0x07c00).
١٣     ; $$ is the address of current line.
١٤     ;  $—$$ means how many byte between start and current.
١٥
١٦     ; if cli and hlt take 4 byte then time directive will fill
١٧     ; 510—4 = 506 zero's.
١٨
١٩     ; finally the boot signature 0xaa55
٢٠     db    0x55  ; first byte of a boot signature.
٢١     db    0xaa  ; second byte of a  boot signature.
```

وعندما يبدأ الجهاز بالعمل فان البايوس يقوم بنسخ هذا المحمل الى العنوان 0x0000:0x7c00 ويبدأ بتنفيذه ، وفي هذا المثال فان المحمل هذا الذي يعمل في النمط الحقيقي (real mode) لا يقوم بشيء ذو فائدة حيث يبدأ بتنفيذ الامر cli الذي يوقف عمل المقاطعات ، يليها الامر hlt الذي يوقف عمل المعالج وبالتالي يتوقف النظام عن العمل ، وبدون هذا الأمر فان المعالج سيستمر في تنفيذ أوامر لا معنى لها (garbage) والتي ستؤدي الى سقوط (Crash) النظام . وبسبب أن حجم المحمل يجب أن يكون 512 بايت وأن آخر بايتين فيه يجب أن تكونا التوقيع الخاص بالمحمل فانه يجب أن تكون أول 510 بايت ذات قيمة واخر بايتين هما 0xaa55 ، لذلك تم استخدام الموجه times لكي يتم ملئ المتبقي من أول 510 بايت بالقيمة صفر (ويمكن استخدام أي قيمة اخرى) وبعد ذلك تم كتابة التوقيع الخاص بالمحمل وذلك حتى يتم التعرف عليه من قِبل البايوس.

١.٤.١ عرض رسالة ترحيبية

طالما ما زلنا نعمل في النمط الحقيقي فان ذلك يمكننا من استخدام مقاطعات البايوس ، وفي المثال ١.٢ تم عرض رسالة باستخدام مقاطعة البايوس int 0x10 الدالة 0xe .

Example ١.٢: Welcom to OS World

```
١
٢ ;Hello Bootloader
٣
٤ bits 16       ; 16—bit real mode.
٥ org  0x0      ; this number will added to all addresses (relocating).
٦
٧ start:
٨     jmp main  ; jump over data and function to entry point.
٩
١٠
١١
١٢ ; ****************
١٣ ; data
١٤ ; ****************
١٥
١٦ hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam",0xa,0xd,0
١٧
١٨ ; *******************************************
١٩ ; puts16: prints string using BIOS interrupt
٢٠ ;   input:
٢١ ;       es: pointer to data segment.
٢٢ ;       si: point to the string
٢٣ ; *******************************************
٢٤
٢٥ puts16:
٢٦
٢٧     lodsb      ; read character from ds:si to al ,and increment si if
           df=0.
٢٨
٢٩     cmp al,0    ; check end of string ?
٣٠     je end_puts16 ; yes jump to end.
٣١
٣٢     mov ah,0xe    ; print character routine number.
٣٣     int 0x10    ; call BIOS.
٣٤
٣٥     jmp puts16    ; continue prints until 0 is found.
٣٦
٣٧   end_puts16:
٣٨
```

```
٣٩      ret
٤٠
٤١
٤٢ ; ************************************
٤٣ ;   entry point of bootloader.
٤٤ ; ************************************
٤٥
٤٦ main:
٤٧
٤٨      ;————————————————
٤٩      ; intit registers
٥٠      ;————————————————
٥١
٥٢      ; because bootloader are loaded at 0x07c00 we can refrence this
           location with many different combination of segment:offset
           addressing.
٥٣
٥٤      ; So we will use either 0x0000:0x7c000 or 0x:07c0:0x0000 , and in
             this example we use 0x07c0 for segment and 0x0 for offset.
٥٥
٥٦      mov ax,0x07c0
٥٧      mov ds,ax
٥٨      mov es,ax
٥٩
٦٠      mov si,hello_msg
٦١      call puts16
٦٢
٦٣      cli     ; clear interrupt.
٦٤      hlt     ; halt the system.
٦٥
٦٦      times 510—($—$$)  db    0   ; append zeros.
٦٧
٦٨      ; finally the boot signature 0xaa55
٦٩      db    0x55
٧٠      db    0xaa
```

الشيء الملاحظ في المثال السابق هو أن مقطع الكود code segment ومقطع البيانات data segment متواجدان في نفس المكان على الذاكرة (داخل ال 512 بايت) لذلك يجب تعديل قيم مسجلات المقاطع للاشارة الى المكان الصحيح. و بداية نذكر أن البايوس عندما ينقل التنفيذ الى برنامج محمل النظام الذي قمنا بكتابته فانه في حقيقة الأمر يقوم بعملية far jump والتي ينتج منها تصحيح قيم ال cs:ip لذلك لا داعي

للقلق حول هذين المسجلين، لكن يجب تعديل قيم مسجلات المقاطع الاخرى مثل ds,es,ss,fs,gs. وكما نعلم أن العنوان الفيزيائي لمحمل النظام هو 0x07c00 يمكن الوصول اليه بأكثر من 4000 طريقة مختلفة ، لكن سوف نقتصر على استخدام العنوان 0x07c0:0x0 أو العنوان 0x0:0x7c00 نظراً لان هذه هي القيم الفعلية التي تستخدمها البايوس.
وفي حالة استخدام العنونة الاولى فان مسجلات المقاطع يجب أن تحوي القيمة 0x07c0 (كما في المثال اعلاه) أما بقية العنوانين (سواءا للمتغيرات وال label) فانها يجب أن تبدأ من القيمة 0x0، وكما هو معروف ان المجمعات عندما تبدأ في عملية ترجمة الملف الى ملف ثنائي فانها تبدأ بترقيم العناوين بدءاً من العنوان 0x0 لذلك كانت وظيفة الموجه org هي عمل اعادة تعيين (relocating) للعناوين بالقيمة التي تم كتابتها ، وفي المثال أعلاه كانت القيمة هي 0x0 ، أما في حالة استخدام الطريقة الثانية للوصول الى مكان محمل النظام فان مسجلات المقاطع يجب أن تحوي القيمة 0x0 بينما المسجلات الاخرى يجب أن تبدأ قيمها من العنوان 0x7c00 ، وهذا لا يمكن بالوضع الطبيعي لان المجمعات ستبدأ من العنوان 0x0 لذلك يجب استخدام الموجه org وتحديد قيمة ال relocate بالقيمة 0x7c00 .

٢.٤.١ معلومات قطاع الاقلاع

إضافة الى محمل النظام فان قطاع الإقلاع boot sector يجب أن يحوي كذلك على معلومات تساعد في وصف نظام الملفات المستخدم ووصف القرص الذي سيتم الاقلاع منه ، هذه المعلومات تحوي معرف OEM وتحوي بيانات BIOS Parameter Block (تختصر ب BPB) ويجب أن تبدأ كل هذه البيانات من البايت رقم 3[٤]. وسوف يتم استخدام هذه البيانات بكثرة أثناء تطوير محمل النظام كذلك أحد فوائد هذه البيانات هو تعرف أنظمة التشغيل على نظام الملفات المستخدم في القرص.

Example ١.٣: Bios Parameter Block

```
١
٢ OEM_ID                db          "eqraOS  "     ; Name of your OS, Must
      be 8 byte! no more no less.
٣
٤ bytes_per_sector      dw          0x200     ; 512 byte per sector.
٥ sectors_per_cluster   db          0x1           ; 1 sector per cluster.
٦ reserved_sectors      dw          0x1           ; boot sector is
      reserved.
٧ total_fats            db          0x2           ; two fats.
٨ root_directory        dw          0xe0          ; root dir has 224
      entries.
٩ total_sectors         dw          0xb40         ; 2880 sectors in the
      volume.
```

[٤]لهذا السبب فان أول تعليمة في المحمل ستكون تعليمة القفز الى الشفرة التنفيذية، وبدون القفز فان المعالج سيبدأ بتنفيذ هذه البيانات باعتبار انها تعليمات وهذا ما يؤدي في الاخر الى سقوط النظام.

```
media_descriptor     db        0xf0          ; 1.44 floppy disk.
sectors_per_fat      dw        0x9           ; 9 sector per fat.
sectors_per_track    dw        0x12          ; 18 sector per track.
number_of_heads      dw        0x2           ; 2 heads per platter.
hidden_sectors       dd        0x0           ; no hidden sector.
total_sectors_large  dd        0x0

; Extended BPB.

drive_number         db        0x0
flags                db        0x0
signature            db        0x29          ; must be 0x28 or 0x29.
volume_id            dd        0x0           ; serial number written
    when foramt the disk.
volume_label         db        "MOS FLOPPY " ; 11 byte.
system_id            db        "fat12   "    ;  8 byte.
```

المثال ١.٤ يوضح شفرة المحمل بعد اضافة بيانات OEM and BPB.

Example ١.٤: BPB example

```

;Hello Bootloader

bits 16       ; 16—bit real mode.
org  0x0      ; this number will added to all addresses (relocating).

start:
     jmp main  ; jump over data and function to entry point.


;*****************************************
; OEM Id and BIOS Parameter Block (BPB)
;*****************************************

; must begin at byte 3(4th byte), if not we should add nop
    instruction.

OEM_ID               db        "eqraOS  "    ; Name of your OS, Must
    be 8 byte! no more no less.

```

```
١٩ bytes_per_sector     dw        0x200    ; 512 byte per sector.
٢٠ sectors_per_cluster  db        0x1          ; 1 sector per cluster.
٢١ reserved_sectors     dw        0x1          ; boot sector is
       reserved.
٢٢ total_fats           db        0x2          ; two fats.
٢٣ root_directory       dw        0xe0         ; root dir has 224
       entries.
٢٤ total_sectors        dw        0xb40        ; 2880 sectors in the
       volume.
٢٥ media_descriptor     db        0xf0         ; 1.44 floppy disk.
٢٦ sectors_per_fat      dw        0x9          ; 9 sector per fat.
٢٧ sectors_per_track    dw        0x12         ; 18 sector per track.
٢٨ number_of_heads      dw        0x2          ; 2 heads per platter.
٢٩ hidden_sectors       dd        0x0          ; no hidden sector.
٣٠ total_sectors_large  dd        0x0
٣١
٣٢ ; Extended BPB.
٣٣
٣٤ drive_number         db        0x0
٣٥ flags                db        0x0
٣٦ signature            db        0x29         ; must be 0x28 or 0x29.
٣٧ volume_id            dd        0x0          ; serial number written
       when foramt the disk.
٣٨ volume_label         db        "MOS FLOPPY " ; 11 byte.
٣٩ system_id            db        "fat12   "    ;  8 byte.
٤٠
٤١
٤٢ ; ****************
٤٣ ; data
٤٤ ; ****************
٤٥
٤٦ hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam",0xa,0xd,0
٤٧
٤٨ ; ********************************************
٤٩ ; puts16: prints string using BIOS interrupt
٥٠ ;   input:
٥١ ;       es: pointer to data segment.
٥٢ ;       si: point to the string
٥٣ ; ********************************************
٥٤
٥٥ puts16:
```

```
٥٦
٥٧      lodsb      ; read character from ds:si to al ,and increment si if
            df=0.
٥٨
٥٩      cmp al,0     ; check end of string ?
٦٠      je end_puts16 ; yes jump to end.
٦١
٦٢      mov ah,0xe     ; print character routine number.
٦٣      int 0x10     ; call BIOS.
٦٤
٦٥      jmp puts16     ; continue prints until 0 is found.
٦٦
٦٧    end_puts16:
٦٨
٦٩      ret
٧٠
٧١
٧٢  ; ************************************
٧٣  ;   entry point of bootloader.
٧٤  ; ************************************
٧٥
٧٦  main:
٧٧
٧٨      ;-----------------------
٧٩      ; intit registers
٨٠      ;-----------------------
٨١
٨٢      ; because bootloader are loaded at 0x07c00 we can refrence this
            location with many different combination
٨٣      ; of segment:offset addressing.
٨٤
٨٥      ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
٨٦      ; and in this example we use 0x07c0 for segment and 0x0 for
            offset.
٨٧
٨٨      mov ax,0x07c0
٨٩      mov ds,ax
٩٠      mov es,ax
٩١
٩٢      mov si,hello_msg
٩٣      call puts16
```

```
٩٤
٩٥      cli      ; clear interrupt.
٩٦      hlt      ; halt the system.
٩٧
٩٨      times 510-($-$$)  db    0    ; append zeros.
٩٩
١٠٠     ; finally the boot signature 0xaa55
١٠١     db    0x55
١٠٢     db    0xaa
```

و المخرج ١.٥ يوضح الشفرة السابقة في حالة عرضها بأي محرر سادس عشر Hex Editor حيث كما نلاحظ أن بيانات المحمل متداخلة مع الشفرة التنفيذية (تعليمات المعالج) لذلك يجب أن يتم القفز فوق هذه البيانات حتى لا تُنَفذ كتعليمات خاطئة ، كذلك يجب التأكد من آخر بايتين وأنها تحمل التوقيع الصحيح.

Example ١.٥: Hex value of bootloader

```
Offset(h) 00 01 02 03 04 05 06 07

00000000  E9 72 00 65 71 72 61 4F  ér.eqraO
00000008  53 20 20 00 02 01 01 00  S  .....
00000010  02 E0 00 40 0B F0 09 00  .à.@ð...
00000018  12 00 02 00 00 00 00 00  ........
00000020  00 00 00 00 00 00 29 00  ......).
00000028  00 00 00 4D 4F 53 20 46  ...MOS F
00000030  4C 4F 50 50 59 20 66 61  LOPPY fa
00000038  74 31 32 20 20 20 57 65  t12   We
00000040  6C 63 6F 6D 65 20 74 6F  lcome to
00000048  20 65 71 72 61 4F 53 2C   eqraOS,
00000050  20 43 6F 64 65 64 20 62   Coded b
00000058  79 20 41 68 6D 61 64 20  y Ahmad
00000060  45 73 73 61 6D 0A 0D 00  Essam...
00000068  AC 3C 00 74 07 B4 0E CD  ¬<.t.´.Í
00000070  10 E9 F4 FF C3 B8 C0 07  .Ăéôÿ¸À.
00000078  8E D8 8E C0 BE 3E 00 E8  .Ø.À¾>.è
00000080  E6 FF FA F4 00 00 00 00  æÿúô....
00000088  00 00 00 00 00 00 00 00  ........
                  ...
                  ...
000001F0  00 00 00 00 00 00 00 00  ........
```

```
000001F8   00 00 00 00 00 00 55 AA   ......Uª
```

ويمكن الاستفادة من هذه المحررات والتعديل المباشر في قيم الهيكس للملف الثنائي[٥]، فمثلا يمكن حذف التوقيع واستبداله بأي رقم ومحاولة الإقلاع من القرص ! بالتأكيد لا يمكن الاقلاع بسبب أن البايوس لن يتعرف على القرص بأنه قابل للإقلاع ، كذلك كمثال يمكن عمل حلقة لا نهائية وطباعة الجملة الترحيبية في كل تكرار ، ويجب أولا اعادة تجميع الملف الثنائي باستخدام أي من برامج ال Disassembler وإدخال تعليمة قفز بعد استدعاء دالة طباعة السلسلة الى ما قبلها.

Example ١.٦: Complete Example

```
١ ;Hello Bootloader
٢
٣ bits 16       ; 16—bit real mode.
٤ org  0x0      ; this number will added to all addresses (relocating).
٥
٦ start:
٧     jmp main  ; jump over data and function to entry point.
٨
٩
١٠ ;*****************************************
١١ ; OEM Id and BIOS Parameter Block (BPB)
١٢ ;*****************************************
١٣
١٤ ; must begin at byte 3(4th byte), if not we should add nop
      instruction.
١٥
١٦ OEM_ID               db        "eqraOS  "    ; Name of your OS, Must
      be 8 byte! no more no less.
١٧
١٨ bytes_per_sector     dw        0x200    ; 512 byte per sector.
١٩ sectors_per_cluster  db        0x1          ; 1 sector per cluster.
٢٠ reserved_sectors     dw        0x1          ; boot sector is
      reserved.
٢١ total_fats           db        0x2          ; two fats.
٢٢ root_directory       dw        0xe0         ; root dir has 224
      entries.
٢٣ total_sectors        dw        0xb40        ; 2880 sectors in the
      volume.
٢٤ media_descriptor     db        0xf0         ; 1.44 floppy disk.
```

[٥] في حالة لم نتمكن من الوصول الى ملف المصدر source code.

```
٢٥ sectors_per_fat      dw        0x9          ; 9 sector per fat.
٢٦ sectors_per_track    dw        0x12         ; 18 sector per track.
٢٧ number_of_heads      dw        0x2          ; 2 heads per platter.
٢٨ hidden_sectors       dd        0x0          ; no hidden sector.
٢٩ total_sectors_large  dd        0x0
٣٠
٣١ ; Extended BPB.
٣٢
٣٣ drive_number         db        0x0
٣٤ flags                db        0x0
٣٥ signature            db        0x29         ; must be 0x28 or 0x29.
٣٦ volume_id            dd        0x0          ; serial number written
       when foramt the disk.
٣٧ volume_label         db        "MOS FLOPPY " ; 11 byte.
٣٨ system_id            db        "fat12   "    ;  8 byte.
٣٩
٤٠
٤١ ; ****************
٤٢ ; data
٤٣ ; ****************
٤٤
٤٥ hello_msg   db    "Welcome to eqraOS, Coded by Ahmad Essam",0xa,0xd,0
٤٦
٤٧ ; ********************************************
٤٨ ; puts16: prints string using BIOS interrupt
٤٩ ;   input:
٥٠ ;       es: pointer to data segment.
٥١ ;       si: point to the string
٥٢ ; ********************************************
٥٣
٥٤ puts16:
٥٥
٥٦     lodsb     ; read character from ds:si to al ,and increment si if
           df=0.
٥٧
٥٨     cmp al,0    ; check end of string ?
٥٩     je end_puts16 ; yes jump to end.
٦٠
٦١     mov ah,0xe    ; print character routine number.
٦٢     int 0x10    ; call BIOS.
٦٣
```

```
٦٤      jmp puts16    ; continue prints until 0 is found.
٦٥
٦٦    end_puts16:
٦٧
٦٨      ret
٦٩
٧٠
٧١  ; ************************************
٧٢  ;   entry point of bootloader.
٧٣  ; ************************************
٧٤
٧٥  main:
٧٦
٧٧      ;-----------------------
٧٨      ; intit registers
٧٩      ;-----------------------
٨٠
٨١      ; because bootloader are loaded at 0x07c00 we can refrence this
            location with many different combination
٨٢      ; of segment:offset addressing.
٨٣
٨٤      ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
٨٥      ; and in this example we use 0x07c0 for segment and 0x0 for
            offset.
٨٦
٨٧      mov ax,0x07c0
٨٨      mov ds,ax
٨٩      mov es,ax
٩٠
٩١      mov si,hello_msg
٩٢      call puts16
٩٣
٩٤      cli     ; clear interrupt.
٩٥      hlt     ; halt the system.
٩٦
٩٧      times 510-($-$$)  db    0   ; append zeros.
٩٨
٩٩      ; finally the boot signature 0xaa55
١٠٠     db    0x55
١٠١     db    0xaa
```

٣.٤.١ تحميل قطاع من القرص باستخدام المقاطعة int 0x13

بعد أن تم تشغيل محمل النظام لعرض رسالة ترحيبة ، فان مهمة المحمل الفعلية هي تحميل وتنفيذ المرحلة الثانية له حيث كما ذكرنا سابقا أن برمجة محمل النظام ستكون على مرحلتين وذلك بسبب القيود على حجم المرحلة الاولى ، وتكمن وظيفة المرحلة الاولى في البحث عن المرحلة الثانية من محمل النظام ونقل التنفيذ اليها ، وبعدها يأتي دور المرحلة الثانية في البحث عن نواة النظام ونقل التحكم اليها. وسنتناول الان كيفية تحميل مقطع من القرص المرن الى الذاكرة الرئيسية ونقل التحكم اليها باستخدام مقاطعة البايوس int 0x13 .

إعادة القرص المرن

عند تكرار القراءة من القرص المرن فانه يجب في كل مرة أن نعيد مكان القراءة والكتابة الى أول مقطع sector في القرص وذلك لكي نضمن عدم حدوث مشاكل، وتستخدم الدالة 0x0 من المقاطعة int 0x13 لهذا الغرض.
المدخلات :

- المسجل ah : 0x0.
- المسجل dl : رقم محرك القرص المرن وهو 0x0.

النتيجة:

- المسجل ah : الحالة.
- CF : 0x1 اذا حدث خطأ ، 0x0 اذا تمت العملية بنجاح.

مثال:

Example ١.٧: Reset Floppy Drive

```
١ reset_floppy:
٢
٣     mov ah,0x0    ; reset floppy routine number.
٤     mov dl,0x0    ; drive number
٥
٦     int 0x13    ; call BIOS
٧
٨     jc reset_floppy   ; try again if error occur.
```

قراءة المقاطع sectors

أثناء العمل في النمط الحقيقي فاننا سنستخدم مقاطعة البايوس int 0x13 الدالة 0x2 لقراءة المقاطع (sectors) من القرص المرن الى الذاكرة الرئيسية RAM .
المدخلات :

- المسجل ah: الدالة 0x2
- المسجل al: عدد المقاطع التي يجب قرائتها.
- المسجل ch: رقم الاسطوانة (Cylinder) ، بايت واحد.
- المسجل cl: رقم المقطع ، من البت 0 – 5 ، أما اخر بتين يستخدمان مع القرص الصلب hard disk.
- المسجل dh: رقم الرأس.
- المسجل dl : رقم محرك القرص المرن وهو 0x0.
- العنوان es:bx : مؤشر الى المساحة التي سيتم قراءة المقاطع اليها.

النتيجة:

- المسجل ah : الحالة.
- المسجل al: عدد المقاطع التي تم قرائتها.
- CF : 0x1 اذا حدث خطأ ، 0x0 اذا تمت العملية بنجاح.

مثال:

Example ١.٨: Read Floppy Disk Sectors

```
١
٢ read_sectors:
٣
٤   reset_floppy:
٥
٦     mov ah,0x0    ; reset floppy routine number.
٧     mov dl,0x0    ; drive number
٨
٩     int 0x13    ; call BIOS
١٠
١١    jc reset_floppy   ; try again if error occur.
```

```
١٢
١٣      ; init buffer.
١٤      mov ax,0x1000
١٥      mov es,ax
١٦      xor bx,bx
١٧
١٨    read:
١٩
٢٠      mov ah,0x2     ; routine number.
٢١      mov al,1     ; how many sectors?
٢٢      mov ch,1     ; cylinder or track number.
٢٣      mov cl,2     ; sector number "fisrt sector is 1 not 0",now we read
            the second sector.
٢٤      mov dh,0     ; head number "starting with 0".
٢٥      mov dl,0     ; drive number ,floppy drive always zero.
٢٦
٢٧      int 0x13     ; call BIOS.
٢٨      jc read      ; if error, try again.
٢٩
٣٠      jmp 0x1000:0x0  ; jump to execute the second sector.
```

٥.١ مقدمة الى نظام FAT12

نظام الملفات هو برنامج يساعد في حفظ الملفات على القرص بحيث ينشئ لنا مفهوم الملف وخصائصه والعديد من البيانات المتعلقة به من تاريخ الانشاء والوقت ، كذلك يحتفظ بقائمة بجميع الملفات وأماكن تواجدها في القرص ، أيضاً أحد أهم فوائد أنظمة الملفات هي متابعة الأماكن الغير المستخدمة في القرص والأماكن التي تضررت بسبب أو لآخر bad sectors ، كذلك أنظمة الملفات الجيدة تقوم بعمل تجميع الملفات المبعثرة على القرص Defragmentation حتى تستفيد من المساحات الصغيرة التي ظهرت بسبب حذف ملف موجود أو تخزين ملف ذو حجم أقل من المساحة الخالية. وبدون أنظمة الملفات فان التعامل مع القرص سيكون مستحيلا ! حيث لن نعرف ماهي المساحات الغير مستخدمة من الاخرى ولن نستطيع ان نقوم بقراءة ملف طلبه المستخدم لعرضه على الشاشة !
وبشكل عام فان نظام الملفات يتكون من:

- برنامج للقراءة والكتابة من القرص وسنطلق عليه اسم المحرك (Driver).
- وجود هيكلة بيانات Data Structure معينة على القرص،يتعامل معها درايفر نظام الملفات.

وحيث أن برمجة برنامج القراءة والكتابة تعتمد كلياُ على هيكلة نظام الملفات على القرص ، فاننا سنبدأ بالحديث عنها أولا وسوف نأخذ نظام FAT12 على قرص مرن كمثال ، نظراً لبساطة هذا النظام وخلوه من التعقيدات.

١.٥.١ قيود نظام FAT12

يعتبر نظام FAT12 من أقدم أنظمة الملفات ظهوراً وقد انتشر استخدامه في الاقراص المرنة منذ أواخر السبعينات ، ويعيب نظام FAT12 :

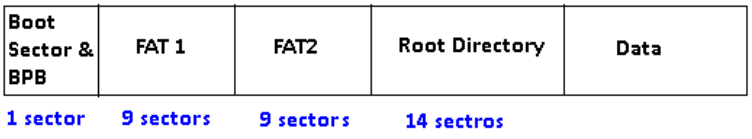
- عدم دعمه للمجلدات الهرمية ويدعم فقط مجلد واحد يسمى الجذر Root Directory.
- طول العنقود (Cluster) هو 12 بت ، بمعنى أن عدد الكلسترات هي 2^{12}.
- أسماء الملفات لا تزيد عن 12 بت.
- يستوعب كحد أقصى 4077 ملف فقط.
- حجم القرص يحفظ في 16 بت ، ولذا فانه لا يدعم الاقراص التي حجمها يزيد عن 32 MB.
- يستخدم العلامة 0x01 لتمييز التقسيمات على القرص (Partitions).

وكما ذكرنا أننا سنستخدم هذا النظام في هذه المرحلة نظراً لبساطته ، وعلى الرغم من أنه قد تلاشى استخدامه في هذا الزمن الا انه يعتبر أساس جيد للأنظمة المتقدمة لذا وجب دراسته.

٢.٥.١ هيكلة نظام FAT12 على القرص

عند تهئية القرص المرن[٦] (Format) بنظام FAT12 فان تركيبة القرص تكون على الشكل ٢.١ :

شكل ٢.١: هيكلة نظام FAT12 على القرص



[٦] سواءاً كانت التهئية من قبل درايفر نظام الملفات الذي سنقوم ببرمجته أو كانت من قبل نظام الشتغيل المستخدم أثناء عملية التطوير ، فمثلا في ويندوز يمكن إعادة تهئية القرص المرن بنظام FAT12 .

وأول مقطع هو مقطع الاقلاع (Boot Sector) ويحوي شفرة محمل النظام (المرحلة الاولى) بالاضافة الى بيانات ومعلومات BPB and OEM id ، هذا المقطع عنوانه الفيزيائي على القرص هو : المقطع 1 المسار 0 الرأس 0 وهذا العنوان هو الذي يجب تمرير الى مقاطعة البايوس int 0x13 التي تقوم بالقراءة من القرص كذلك في حالة ما أردنا التعامل المباشر مع متحكم القرص المرن. ونظراً لصعوبة هذه العنونة والتي تعرف ب Absolute Sector فان أنظمة الملفات تتعامل مع نظام عنونة مختلف للوصول الى محتويات القرص ، فبدلاً من ذكر كل من المقطع والمسار والرأس للوصول الى مقطع ما فان هذه العنونة تستخدم فقط رقم للمقطع . نظام العنونة الذي تستخدمه أنظمة الملفات يسمى بالعنونة المنطقية (Logical Sector Addressing) ويختصر ب LBA هو نظام بسيط يعتمد على ترقيم المقاطع بشكل متسلسل بدئاً من مقطع الاقلاع (Boot Sector) والذي يأخذ العنوان 0 ، والمقطع الثاني 1 وهكذا هلم جرا حتى نصل الى آخر مقطع في القرص. وبما أنه يجب استخدام العنونة الحقيقة بدلا من المنطقية لحظة القراءة من القرص (تذكر مقاطعة البايوس int 0x13 والمسجلات التي يجب ادخال قيمها) فانه يجب ايجاد طريقة للتحويل من العنونة الحقيقة الى المنطقية –سنناقش الموضوع لاحقا–. ننتقل الى المقطع التالي لمقطع الإقلاع وهو مقطع (أو عدة مقاطع) يمكن أن يحجزها المبرمج لاداء أي وظيفة يريدها وتسمى المقاطع المحجوزة الاضافية Extra Reserved Sectors ، والمقصود بمحجوزة أي انه لا يوجد لها وجود في دليل FAT ، ومقطع الإقلاع هو مقطع محجوز دائما لذلك كانت قيمة المتغير reserved sectors في معلومات BPB هي واحد ، وفي حالة ما أردت حجز مقاطع أخرى كل ما عليك هو زيادة هذه القيمة بعدد المقاطع المرغوبة ، وللوصول الى محتويات هذا المقطع الاضافي(ان كان له وجود) فان العنوان الحقيقي له هو المقطع 2 المسار 0 الرأس 0 ، أما العنوان المنطقي له هو المقطع 1. وبشكل عام فانه في الغالب لا يتم استخدام مقاطع اضافية سوى مقطع الاقلاع . المقطع الثالث هو جدول FAT ، وهو جدول يحوي سجلات بطول 12 بت عن كل كلستر (Cluster) في القرص ، بيانات هذا السجل توضح ما اذا كان الكلستر قيد الاستخدام أم لا ، وهل هو آخر كلستر للملف أم لا وإذا كان ليس باخر فانه يوضح لنا الكلستر التالي للملف ، ويوضح الشكل التالي تركيبة هذا الجدول

اذاً هذا وظيفة هذا الجدول هي معرفة الكاسترات الخالية من غيرها كذلك الوظيفة الاخرى هي معرفة جميع الكلسترات لملف ما ويتم ذلك بالنظر الى قيمة السجل (قيمة ال 12 بت) ، والقيم هي :

- القيمة 0x00: تدل على أن الكلستر خالي.
- القيمة 0x01 : تدل على أن الكلستر محجوز.
- القيم من 0x02 الى 0xfef : تدل على عنوان الكلستر التالي (بمعنى آخر أن الكلستر محجوز وتوجد كلسترات متبقية للملف).
- القيم من 0xff0 الى 0xff6: قيم محجوزة.
- القيمة 0xff6 : تدل على Bad Cluster.
- القيم من 0xff8 الى 0xfff: تدل على أن هذا الكلستر هو الاخير للملف.

ويمكن النظر الى جدول FAT بأنه مصفوفة من القيم أعلاه ، وعندما نريد تحميل ملف فاننا سنأتي بعنوان أول كلستر له من جدول Root Directory (سنأتي عليها لاحقا) وبعدها نستخدم عنوان الكلستر ك index الى جدول FAT ونقرأ القيمة المقابله للكلستر ، فاذا كانت القيمة بين 0x02 الى 0xfef فانها تدل على الكلستر التالي للملف ، ومن ثم سنستخدم هذه القيمة أيضا ك index ونقرأ القيمة الجديدة ، ونستمر على هذا الحال الى أن نقرأ قيمة تدل على نهاية الملف. هذا الجدول FAT يبدأ من المقطع المنطقي 1[٧] وطوله 9 مقاطع أي أن نهاية هذا الجدول تكون في المقطع تكون في آخر المقطع 10، ولمعرفة العنوان الحقيقي للمقطع فانه يمكن استخدام بعض المعادلات للتحويل ، والقسم التالي سيوضح ذلك بالاضافة الى شرح مبسط عن هيكلة القرص المرن وكيفية حفظه للبيانات . وبعد جدول FAT توجد نسخة أخرى من هذا الجدول وتستخدم كنسخة احتياطية backup وهي بنفس حجم وخصائص النسخة الاولى ، وبعدها يأتي دليل الجذر Root Directory وهو مصفوفة من 224 سجل كل سجل بطول 32 بايت ، وظيفية هذا الدليل هي حفظ أسماء الملفات الموجودة على القرص المرن بالاضافة الى العديد من المعلومات التي تخص وقت الانشاء والتعديل وحجم الملف وعنوان أول كلستر للملف ، عنوان الكلستر هو أهم معلومة لكي نستطيع تحميل الملف كاملا ، حيث كما ذكرنا أن هذا العنوان سيعمل ك index في جدول FAT وبعدها سنحدد ما اذا كانت توجد كلسترات أخرى يجب تحميلها أم أن الملف يتكون من كلستر واحد. والجدول التالي يوضح محتويات السجل الواحد في دليل ال root directory بداءاً من البايت الاول الى الاخير:

- البايتات 0–7: اسم الملف(وفي حالة كان الحجم أقل من 8 بايت يجب استخدام حرف المسافة لتعبئة المتبقي).
- البايتات 8–10: امتداد الملف(يجب استخدام المسافة أيضا لتعبئة المتبقي).
- البايت 11: خصائص الملف وهي :
  - البت 0: القراءة فقط.
  - البت 1: مخفي.
  - البت 2: ملف نظام.
  - البت3: اسم القرص Volume Label.
  - البت 4: الملف هو مجلد فرعي.
  - البت 5: أرشيف.
  - البت 6: جهاز.
  - البت 7: غير مستخدم.
- البايت 12: غير مستخدم.
- البايت 13: وقت الانشاء بوحدة MS.

[٧] بافتراض الوضع الغالب وهو عدم وجود مقاطع إضافية باستثناء مقطع الإقلاع

- البايتات 14–15: وقت الانشاء بالترتيب التالي:
  - البتات 0–4: الثواني (0–29).
  - البتات 5–10: الدقائق (0–59).
  - البتات 11–15: الساعات (0–23).
- البايتات 16–17: سنة الانشاء بالترتيب التالي:
  - البتات 0–4: السنة (0=1980; 127=2107).
  - البتات 5–8: الشهر (1=يناير; 12=ديسمبر).
  - البتات 9–15: الساعة (0–23).
- البايتات 18–19: تاريخ آخر استخدام (تتبع نفس الترتيب السابق).
- البايتات 20–21: EA index.
- البايتات 22–23: وقت آخر تعديل (تتبع نفس ترتيب البايتات 14–15).
- البايتات 24–25: تاريخ آخر تعديل (تتبع نفس ترتيب البايتات 16–17).
- البايتات 26–27: عنوان أول كلستر للملف.
- البايتات 28–29: حجم الملف.

ويجب ملاحظة أن حجم السجلات هو ثابت Fixed Lenght Record فمثلا اسم الملف يجب ان يكون بطول 8 بايت وفي حالة زاد على ذلك فان هذا سوف يحدث ضرراً على هذا الدليل ، أيضا في حالة كان الاسم بحجم أقل من المطلوب فانه يجب تكلمة العدد الناقص من الحروف بحرف المسافة Space.

٣.٥.١ هيكلة القرص المرن

يتكون القرص المرن من قرص Platter (أو عدة أقراص) مقسمة الى مسارات (Tracks) وكل من هذه المسارات يتكون من العديد من القطاعات ويوجد عادة رأسين للقراءة والكتابة على كل قرص. وفي الاقراص المرنة ذات الحجم 1.44 MB يوجد 80 مساراً (من المسار 0 الى المسار 79) وكل مسار يتكون من 18 قطاع ، وبالتالي فان عدد القطاعات الكلية هي $2 * 18 * 80$ وتساوي 2880 قطاعاً.
ولتخزين بيانات على القرص فانه يجب تحديد العنوان الحقيقي والذي يتكون من عنوان القطاع والمسار والرأس ، وأول قطاع في القرص (قطاع الاقلاع) يأخذ العنوان: القطاع 1 المسار 0 الرأس 0 ، والقطاع الثاني يأخذ العنوان: القطاع 2 المسار 0 الرأس 0 ، وهكذا يستمر نظام التخزين في القرص المرن الى أن يصل الى العنوان 18 المسار 0 الرأس 0 وهو عنوان آخر قطاع على المسار الاول والرأس الاول ، وسيتم حفظ البيانات التالية في الرأس الثاني على العنوان: القطاع 1 المسار 0 الرأس 1 ويستمر الى أن يصل الى آخر

قطاع في هذا المسار على الرأس الثاني، وبعدها سيتم حفظ البيانات التالية في الرأس الاول المسار الثاني ... ،وهكذا. والصورة التالية توضح شكل القرص المرن بعد عمل تهئية (Format) له.

٤.٥.١ القراءة و الكتابة من نظام FAT12

حتى نتمكن من التعامل مع القرص المرن (قراءة وكتابة القطاعات) فانه يلزمنا برمجة درايفر لنظام FAT12 والذي سيعمل كوسيط بين المستخدم وبين القرص المرن، بمعنى أن أي طلب لقراءة ملف ما يجب أن تذهب أولا الى نظام FAT12 حيث سيقرر ما اذا كان الملف موجوداً أم لا (عن طريق البحث في دليل Root directory) وفي حالة كان موجوداً سيعود لنا بجميع خصائص الملف ورقم أول كلستر له لكي نتمكن من تحميل الملف كاملاً ، ونفس المبدأ في حالة طلب المستخدم كتابة ملف على القرص فان درايفر نظام FAT12 سيبحث في جدول FAT عن مساحة خالية مناسبة للملف وذلك باتباع أحد الخورازميات المعروفة وبعدها سيتم حفظ الملف وكتابة البيانات المتعلقة به في دليل Root directory .
وسنأخذ مثال على الموضوع وذلك ببرمجة المرحلة الثانية من محمل النظام Second Stage Bootloader وستقتصر وظيفته حالياً في طباعة رسالة ترحيبة دلالة على أنه تم تحميل وتنفيذ المرحلة الثانية بنجاح ، وفي الأقسام التالية سنبدأ في تطوير المرحلة الثانية وتجهيز مرحلة الانتقال الى بيئة 32 بت.
مهمة المرحلة الاولى ستتغير عن ما سبق ، حيث الان يجب على المرحلة الاولى أن تقوم بالبحث عن المرحلة الثانية من محمل النظام ونقل التنفيذ اليها ، ويتم هذا وفق الخطوات التالية:

١. تحميل جدول Root Directory من القرص الى الذاكرة ومن ثم البحث عن ملف المرحلة الثانية وأخذ رقم أول كلستر له.

٢. تحميل جدول FAT من القرص الى الذاكرة ومن ثم تحميل جميع الكلسترات للملف.

٣. نقل التنفيذ الى أول بايت في المرحلة الثانية من محمل النظام.

إنشاء المرحلة الثانية من محمل النظام

بداية سنقوم بإنشاء المرحلة الثانية من محمل النظام ونسخها الى القرص المرن ، ونظراً لان تطوير نظامنا الخاص يجب ان يتم تحت نظام آخر فان هذا النظام الآخر غالبا ما يحوي درايفر لنظام ملفات FAT12 حيث يتكفل بعملية كتابة البيانات الى جدول Root Directory بالاضافة الى البحث عن كلسترات خالية في جدول FAT دون أي تدخل من قبل مطور النظام الجديد، لذلك في هذه المرحلة من التطوير سنتجاهل جزئية الكتابة في نظام FAT12 ونترك المهمة لنظام التشغيل الذي نعتمد عليه في عملية تطوير النظام الجديد ، وبهذا سيكون الدرايفر الذي سننشئه في هذا الفصل ما هو الا جزء من الدرايفر الكامل الذي سيتم تكلمته في الفصل الخامس بمشيئة الله.والشفرة التالية توضح مثال للمرحلة الثانية من المحمل لعرض رسالة بسيطة.

Example ١.٩: Hello Stage2

```
١  ; Second Stage Bootloader.
٢  ; loaded by stage1.bin at address 0x050:0x0 (0x00500).
٣
٤
٥ bits 16   ; 16—bit real mode.
٦ org  0x0    ; offset to zero.
٧
٨ start:  jmp stage2
٩
١٠
١١ ; data and variable
١٢ hello_msg db    "Welcome to eqraOS Stage2",0xa,0xd,0
١٣
١٤ ; include files:
١٥ %include "stdio.inc"    ; standard i/o routines.
١٦
١٧
١٨
١٩
٢٠ ; ************************************
٢١ ; entry point of stage2 bootloader.
٢٢ ; ************************************
٢٣
٢٤ stage2:
٢٥
٢٦     push cs
٢٧     pop ds    ; ds = cs.
٢٨
٢٩     mov si,hello_msg
٣٠     call puts16
٣١
٣٢     cli     ; clear interrupt.
٣٣     hlt     ; halt the system.
```

وسيتم تسمية الملف بالاسم stage2.asm أما الملف الناتج من عملية التجميع سيكون بالاسم stage2.sys ويمكن تسميته بأي اسم اخر بشرط أن لا يزيد الاسم عن 8 حروف والامتداد عن 3 حروف ، وفي حالة كان طول الاسم أقل فان درايفر FAT12 سيقوم باضافة مسافات Spaces حتى لا يتضرر جدول Root Directory. ويمكننا أن نفرق بين اسماء الملفات الداخلية (وهي التي يتم اضافة مسافات عليها ويستخدمها نظام FAT12)

والأسماء الخارجية (وهي التي ينشئها المستخدم).

تحميل ال Root Directory الى الذاكرة

جدول Root Directory يحوي أسماء كل الملفات و أماكن تواجدها على القرص لذا يجب تحميله أولا والبحث عن ملف المرحلة الثانية (ذو الاسم الخارجي stage2.sys) وعند البحث يجب البحث بالاسم الداخلي الذي يستخدمه نظام الملفات لذلك يجب أن نبحث عن الملف "stage2 sys" ، ونأتي برقم الكلستر الأول للملف.

وقبل تحميل هذا الجدول فانه يجب علينا أولاً معرفة عنوان أول قطاع فيه وحساب عدد القطاعات التي يشغلها هذا الجدول ، كذلك يجب تحديد المساحة الخالية (Buffer) لكي يتم نقل هذا الجدول اليها. والشفرة التالية توضح كيفية عمل ذلك.

Example ١.١٠: Load Root directory

```
;-------------------------------
; Compute Root Directory Size
;-------------------------------

  xor cx,cx
  mov ax,32          ; every root entry size are 32 byte.
  mul word[root_directory]  ; dx:ax = 32*224 bytes
  div word[bytes_per_sector]
  xchg ax,cx           ; cx = number of sectors to load.

;-----------------------------------------
; Get start sector of root directory
;-----------------------------------------

  mov al,byte[total_fats]      ; there are 2 fats.
  mul word[sectors_per_fat]    ; 9*2 sectors
  add ax,word[reserved_sectors] ; ax = start sector of root
     directory.

  mov word[data_region],ax
  add word[data_region],cx  ; data_region = start sector of data.


;-------------------------------------------------------
; Load Root Dir at 0x07c0:0x0200 above bootloader.
;-------------------------------------------------------
```

```
٢٦
٢٧      mov bx,0x0200      ; es:bs = 0x07c0:0x0200.
٢٨      call read_sectors
```

بعد تحميل هذا الجدول يجب البحث فيه عن اسم ملف المرحلة الثانية من محمل النظام ومن ثم حفظ رقم أول كلستر له في حالة كان الملف موجوداً ، أما اذا كان الملف غير موجود فنصدر رسالة خطأ ونوقف النظام عن العمل. والشفرة التالية توضح ذلك.

Example ١.١١: Find Stage2 Bootloader

```
١    ;--------------------
٢    ; Find stage2.sys
٣    ;--------------------
٤
٥      mov di,0x0200       ; di point to first entry in root dir.
٦      mov cx,word[root_directory] ; loop 224 time.
٧
٨    find_stage2:
٩
١٠     mov si,kernel_loader_name
١١     push cx
١٢     push di
١٣     mov cx,11           ; file name are 11 char long.
١٤
١٥     rep cmpsb
١٦     pop di
١٧     je find_successfully
١٨
١٩     mov di,32         ; point to next entry.
٢٠     pop cx
٢١
٢٢     loop find_stage2
٢٣
٢٤     ; no found ?
٢٥     jmp find_fail
٢٦
٢٧   find_successfully:
٢٨   ;-----------------------
٢٩   ; Get first Cluster.
٣٠   ;-----------------------
٣١
```

```
٣٢      mov ax,word[di+26]     ; 27 byte in the di entry are cluster
            number.
٣٣      mov word[cluster_number],ax
```

تحميل جدول FAT الى الذاكرة

جدول FAT يوضح حالة كل الكلسترات الموجودة على القرص سواءا كانت خالية أم معطوبة أم انها مستخدمة ، ويجب تحميل هذا الجدول الى الذاكرة لكي نستطيع عن طريق رقم الكلستر الذي تحصلنا عليه من جدول Root Directory أن نحمل جميع كلسترات الملف. وبنفس الطريقة التي قمنا بها لتحميل جدول Root Directory سيتم بها تحميل جدول FAT حيث يجب تحدد عنوان أول قطاع للجدول و عدد القطاعات التي يشغلها الجدول ، وكذلك المساحة الخالية في الذاكرة لكي يتم حفظ الحدول بها . والشفرة التالية توضح ذلك.

Example ١.١٢: Load FAT Table

```
١   ;-------------------------
٢   ; Compute FAT size
٣   ;-------------------------
٤
٥     xor cx,cx
٦     xor ax,ax
٧     xor dx,dx
٨
٩     mov al,byte[total_fats]      ; there are 2 fats.
١٠    mul word[sectors_per_fat]    ; 9*2 sectors
١١    xchg ax,cx
١٢
١٣  ;-----------------------------
١٤  ; Get start sector of FAT
١٥  ;-----------------------------
١٦
١٧    add ax,word[reserved_sectors]
١٨
١٩  ;---------------------------------------------------------------
٢٠  ; Load FAT at 0x07c0:0x0200
٢١  ; Overwrite Root dir with FAT, no need to Root Dir now.
٢٢  ;---------------------------------------------------------------
٢٣
٢٤    mov bx,0x0200
```

```
٢٥      call read_sectors
```

تحميل كلسترات الملف

وحدة القراءة والكتابة للقرص المرن هي بالقطاع Sector لكن نظام الملفات FAT12 يتعامل مع مجموعة من القطاعات ككتلة واحدة Cluster، وكلما كبر حجم الكلستر زادت المساحات الخالية بداخله Internel Fragmentation لذلك يجب اختيار حجم ملائم ، وفي تنفيذ نظام FAT12 على قرص مرن أخترنا أن كل كلستر يقابل قطاع واحد فقط من القرص المرن. المشكلة التي ستواجهنا هي كيفية قراءة كلستر من القرص ، فالقرص المرن لا يقرأ اي قطاع الا بتحديد العنوان المطلق له Absolute Address ولذلك يجب تحويل رقم الكلستر الى عنوان مطلق وتحويل عنوان LBA أيضا الى عنوان مطلق.
التحويل من رقم Cluster الى عنوان LBA يتم كالاتي:

Example ١.١٣: Convert Cluster number to LBA

```
١ ; ************************************************
٢ ; cluster_to_lba: convert cluster number to LBA
٣ ;   input:
٤ ;       ax: Cluster number.
٥ ; output:
٦ ;       ax: lba number.
٧ ; ************************************************
٨ cluster_to_lba:
٩
١٠      ; lba = (cluster — 2)* sectors_per_cluster
١١      ; the first cluster is always 2.
١٢
١٣      sub ax,2
١٤
١٥      xor cx,cx
١٦      mov cl, byte[sectors_per_cluster]
١٧      mul cx
١٨
١٩      add ax,word[data_region]    ; cluster start from data area.
٢٠      ret
```

حيث يتم طرح العدد 2 من رقم الكلستر وهذا بسبب أن أول رقم كلستر في نظام FAT12 هو 2 – كما سنرى ذلك لاحقا–.
وللتحويل من عنوان LBA الى عنوان Absolute Address :

Example ١.١٤: Convert LBA to CHS

```
; ******************************************
; lba_to_chs: Convert LBA to CHS.
;   input:
;       ax: LBA.
; output:
;       absolute_sector
;       absolute_track
;       absolute_head
; ******************************************
lba_to_chs:

    ; absolute_sector =  (lba % sectors_per_track) + 1
    ; absolute_track  =  (lba / sectors_per_track) / number_of_heads
    ; absolute_head   =  (lba / sectors_per_track) % number_of_heads

    xor dx,dx
    div word[sectors_per_track]
    inc dl
    mov byte[absolute_sector],dl

    xor dx,dx
    div word[number_of_heads]
    mov byte[absolute_track],al
    mov byte[absolute_head],dl

    ret
```

ولتحميل كلستر من القرص يجب أولا الحصول على رقمه من جدول Root Directory وبعد ذلك نقوم بتحويل هذا الرقم الى عنوان LBA وبعدها نقوم بتحويل عنوان LBA الى عنوان مطلق Abolsute Address ومن ثم استخدام مقاطعة البايوس `int 0x13` لقراءة القطاعات من القرص، والشفرة التالية توضح ذلك.

Example ١.١٥: Load Cluster

```
;-------------------------------------
; Load all clusters(stage2.sys)
; At address 0x050:0x0
;-------------------------------------

    xor bx,bx
```

```
٧      mov ax,0x0050
٨      mov es,ax
٩
١٠    load_cluster:
١١
١٢     mov ax,word[cluster_number]    ; ax = cluster number
١٣     call cluster_to_lba        ; convert cluster number to LBA
          addressing.
١٤
١٥     xor cx,cx
١٦     mov cl,byte[sectors_per_cluster]  ; cx = 1 sector
١٧
١٨     call read_sectors_bios         ; load cluster.
```

ودالة قراءة القطاعات من القرص تستخدم مقاطعة البايوس int 0x13 وهي تعمل فقط في النمط الحقيقي ويجب استبدالها لاحقا عند التحويل الى النمط المحمي بدالة اخرى 32-bit.

Example ١.١٦: Read Sectors Routine

```
١ ; ******************************************
٢ ; read_sectors_bios: load sector from floppy disk
٣ ;   input:
٤ ;       es:bx : Buffer to load sector.
٥ ;       ax:   first sector number ,LBA.
٦ ;       cx:   number of sectors.
٧ ; ******************************************
٨ read_sectors_bios:
٩
١٠   begin:
١١     mov di,5    ; try 5 times to load any sector.
١٢
١٣   load_sector:
١٤
١٥     push ax
١٦     push bx
١٧     push cx
١٨
١٩     call lba_to_chs
٢٠
٢١     mov ah,0x2            ; load sector routine number.
٢٢     mov al,0x1            ; 1 sector to read.
```

```
٢٣      mov ch,byte[absolute_track]   ; absolute track number.
٢٤      mov cl,byte[absolute_sector]  ; absolute sector number.
٢٥      mov dh,byte[absolute_head]    ; absolute head number.
٢٦      mov dl,byte[drive_number]   ; floppy drive number.
٢٧
٢٨      int 0x13            ; call BIOS.
٢٩
٣٠      jnc continue  ; if no error jmp.
٣١
٣٢      ; reset the floppy and try read again.
٣٣
٣٤      mov ah,0x0          ; reset routine number.
٣٥      mov dl,0x0          ; floppy drive number.
٣٦      int 0x13          ; call BIOS.
٣٧
٣٨      pop cx
٣٩      pop bx
٤٠      pop ax
٤١
٤٢      dec di
٤٣      jne load_sector
٤٤
٤٥      ; error.
٤٦      int 0x18
٤٧
٤٨    continue:
٤٩
٥٠      mov si,progress_msg
٥١      call puts16
٥٢
٥٣      pop cx
٥٤      pop bx
٥٥      pop ax
٥٦
٥٧      add ax,1            ; next sector
٥٨      add bx,word[bytes_per_sector] ; point to next empty block in
           buffer.
٥٩
٦٠
٦١      loop begin    ; cx time
٦٢
```

```
٦٣      ret
```

ولتحميل بقية كلسترات الملف يجب أخذ رقم أول كلستر للملف والذهاب به الى جدول FAT وقراءة القيمة المقابلة له والتي ستدل على ما اذا كان هذا آخر كلستر أم أن هنالك كلسترات اخرى يجب تحميلها.ويلزم الأخذ بالاعتبار بنية جدول FAT وانه يتكون من سجلات بطول 12 بت وتعادل بايت ونصف ، أي أنه اذا كان رقم الكلستر هو 0 فاننا يجب أن نقرأ السجل الاول من جدول FAT وبسبب انه لا يمكن قراءة 12 بت فسوف تتم قراءة 16 بت (السجل الاول بالاضافة الى نصف السجل الثاني) وعمل mask لاخر 4 بت (لازالة ما تم قرائته من السجل الثاني). وفي حالة كان رقم الكلستر هو 1 فيجب قراءة السجل الثاني من جدول FAT والذي يبدأ من البت 12–23 وبسبب أنه لا يمكن قراءة 12 بت سنقوم بقراءة 16 بت أي من البت 8–23 وازالة أول 4 بت.

وباختصار، لقراءة القيمة المقابلة لرقم كلستر ما فيجب أولا تطبيق القانون :
$cluster = cluster + (cluster/2)$

وقراءة 16 بت ، وفي حالة ما اذا كان رقم الكلستر هو رقم زوجي فيجب عمل Mask لاخر 4 بت ، أما اذا كان رقم الكلستر فردي فيجب ازالة أول 4 بت . والشفرة التالية توضح كيفية تحميل جميع كلسترات المرحلة الثانية من محمل النظام الى الذاكرة ونقل التنفيذ اليها .

Example ١.١٧: Read FAT entry

```
١   read_cluster_fat_entry:
٢
٣     mov ax,word[cluster_number]
٤
٥     ; Every FAT entry are 12—bit long( byte and half one).
٦     ; so we must map the cluster number to this entry.
٧     ; to read cluster 0 we need to read fat[0].
٨     ; cluster 1 —> fat[1].
٩     ; cluster 2 —> fat[3],...etc.
١٠
١١    mov cx,ax    ; cx = cluster number.
١٢    shr cx,1     ; divide cx by 2.
١٣    add cx,ax    ; cx = ax + (ax/2).
١٤    mov di,cx
١٥    add di,0x0200
١٦    mov dx,word[di] ; read 16—bit form FAT.
١٧
١٨
١٩    ; Now, because FAT entry are 12—bit long, we should remove 4
          bits.
```

```
٢٠      ; if the cluster number are even, we must mask the last four
           bits.
٢١      ; if it odd, we must do four right shift.
٢٢
٢٣      test ax,1
٢٤      jne odd_cluster
٢٥
٢٦    even_cluster:
٢٧
٢٨      and dx,0x0fff
٢٩      jmp next_cluster
٣٠
٣١    odd_cluster:
٣٢
٣٣      shr dx,4
٣٤
٣٥
٣٦    next_cluster:
٣٧      mov word[cluster_number],dx   ; next cluster to load.
٣٨
٣٩      cmp dx,0x0ff0          ; check end of file, last cluster?
٤٠      jb load_cluster          ; no, load the next cluster.
٤١
٤٢
٤٣      ; yes jmp to end
٤٤      jmp end_of_first_stage
٤٥
٤٦    find_fail:
٤٧
٤٨      mov si,fail_msg
٤٩      call puts16
٥٠
٥١      mov ah,0x0
٥٢      int 0x16    ; wait keypress.
٥٣      int 0x19    ; warm boot.
٥٤
٥٥
٥٦    end_of_first_stage:
٥٧
٥٨      ; jump to stage2 and begin execute.
٥٩      push 0x050      ; segment number.
```

```
٦٠      push 0x0       ; offset number.
٦١
٦٢      retf       ; cs:ip = 0x050:0x0
٦٣
٦٤      times 510—($—$$)  db     0    ; append zeros.
٦٥
٦٦      ; finally the boot signature 0xaa55
٦٧      db     0x55
٦٨      db     0xaa
```