١ برمجة محمل النظام – المرحلة الثانية

بسبب القيود على حجم محمل النظام فان هذا قد أدى الى تقسيم المهمة الى مرحلتين حيث اقتصرت مهمة المرحلة الاولى على تحميل المرحلة الثانية من المحمل ، أما المرحلة الثانية stage 2 فلا قيود عليها وغالبا ما يتم تنفيذ المهمات التالية في هذه المرحلة:

- الانتقال الى النمط المحميPMode.
- تفعيل البوابة A20 لدعم ذاكرة حتى 4 جيجا بايت.
- توفير دوال للتعامل مع المقاطعات Interrupt Handler.
- تحميل النواة ونقل التنفيذ والتحكم اليها.
- توفير خصائص أثناء الإقلاع مثل Safe Mode.
- دعم الإقلاع المتعدد Multi Boot وذلك عبر ملفات التهيئة.

١.١ الانتقال الى النمط المحمي

المشكلة الرئيسية في النمط الحقيقي Real Mode هي عدم توفر حماية للذاكرة حيث يمكن لأي برنامج يعمل أن يصل لأي جزء من الذاكرة ، كذلك أقصى حجم يمكن الوصول له هو 1 ميجا من الذاكرة ، ولا يوجد دعم لتقنية Paging ولا للذاكرة الظاهرية Virtual Memory حتى تعدد البرامج لا يوجد دعم له.
كل هذه المشاكل تم حلها باضافة النمط المحمي الى المعالج ويمكن الانتقال بسهولة الى هذا النمط عن طريق تفعيل البت الاول في المسجل cr0 ، ولكن بسبب أن المعالج في هذا النمط يستخدم طريقة عنونة للذاكرة تختلف عن الطريقة المستخدمة في النمط الحقيقي فانه يجب تجهيز بعض الجداول تسمى جداول الواصفات Descriptor Table وبدون تجهيز هذه الجداول فان المعالج سيصدر استثناء General Protection Fault واختصاراً GPF والذي بدوره يؤدي الى حدوث triple fault وتوقف النظام عن العمل.
أحد هذه الجداول ويسمى جدول الواصفات العام (Global Descriptor Table) واختصاراً GDT وظيفته الاساسية هي تعريف كيفية استخدام الذاكرة ، حيث يحدد ما هو القسم الذي سينفذ كشفرة ؟ وما هو القسم الذي يجب أن يحوي بيانات ؟ ويحدد أيضا بداية ونهاية كل قسم بالاضافة الى صلاحية الوصول الى ذلك القسم.

١.١.١ جدول الواصفات العام Global Descriptor Table

عند الانتقال الى النمط المحمي PMode فان أي عملية وصول الى الذاكرة تتم عن طريق هذا الجدول GDT ، هذا الجدول يعمل على حماية الذاكرة وذلك بفحص العنوان المراد الوصول اليه والتأكد من عدم مخالفته لبيانات هذا الجدول.هذه البيانات تحدد القسم الذي يمكن أن ينفذ كشفرة (Code) والقسم الذي لا ينفذ (Data) كذلك تحدد هذه البيانات العديد من الخصائص كما سنراها الان.
وعادة يتكون جدول GDT من ثلاث واصفات Descriptors (حجم كلٌ منها هو 64 بت) وهم:

- Null Descriptor: تكون فارغة في العادة.
- Code Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي ينفذ كشفرة Code.
- Data Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي لا ينفذ ويحوي بيانات Data.

بيانات أي واصفة Descriptor تأخذ الجدول التالي:

- البتات 0–15: تحوي أول بايتين (من بت 0 –15) من حجم المقطع.
- البتات 16–39: تحوي أول ثلاث بايتات من عنوان بداية المقطع Base Address.
- البت 40: بت الوصول Access Bit (يستخدم مع الذاكرة الظاهرية Virtual Memory.
- البتات 43–41: نوع الواصفة Descriptor Type:
  - البت 41: القراءة والكتابة:
    - Data Descriptor: القيمة 0 للقراءة فقط والقيمة 1 للقراءة والكتابة.
    - Code Descriptor: القيمة 0 للتنفيذ فقط execute والقيمة 1 للقراءة والتنفيذ.
  - البت 42: Expansion direction (Data segments), conforming (Code Segments).
  - البت 43: قابلية التنفيذ:
    - 0: اذا كان المقطع عبارة عن بيانات.
    - 1: اذا كان المقطع عبارة عن شفرة.
- البت 44: Descriptor Bit:
  - 0: System descriptor.
  - 1: Code or Data Descriptor.
- البتات 45–46: مستوى الحماية Privilege Level
  - 0: Highest (Ring 0).

- 3: Lowest (Ring 3).
- البت 47: Segment is in memory (Used with Virtual Memory).
- البتات 48-51: تحوي البت 16 -19 من حجم المقطع.
- البت 52: محجوزة.
- البت 53: محجوزة.
- البت 54: نوع المقطع Segment type:
  - 0: اذا كان المقطع 16 بت.
  - 1: اذا كان المقطع 32 بت.
- البت 55: Granularity:
  - 0: None.
  - 1: Limit gets multiplied by 4K.
- البتات 56-63: تحوي البت 23 -32 من عنوان بداية المقطع Base Address.

وفي هذه المرحلة سنقوم ببناء هذا الجدول ويتكون من واصفة للكود وللبيانات Code and Data Descriptor، بحيث يمكن القراءة و الكتابة من أول بايت في الذاكرة الى آخر الذاكرة 0xffffffff.

Example ١.١: GDT

```
;**************************
; Global Descriptor Table
;**************************

begin_of_gdt:

; Null Descriptor: start at 0x0.

   dd  0x0      ; fill 8 byte with zero.
   dd  0x0

; Code Descriptor: start at 0x8.

   dw  0xffff     ; limit low.
   dw  0x0      ; base low.
   db  0x0      ; base middle.
   db  10011010b ; access byte.
```

```
١٨    db  11001111b ; granularity byte.
١٩    db  0x0     ; base high.
٢٠
٢١ ; Data Descriptor: start at 0x10.
٢٢
٢٣    dw  0xffff    ; limit low.
٢٤    dw  0x0     ; base low.
٢٥    db  0x0     ; base middle.
٢٦    db  10010010b ; access byte.
٢٧    db  11001111b ; granularity byte.
٢٨    db  0x0     ; base high.
٢٩
٣٠ end_of_gdt:
```

هذا الجدول يبدأ بالواصفة الخالية Null Descriptor وحجمها 8 بايت ومتحوياتها تكون صفراً في العادة ، أما الواصفة التالية لها فهي واصفة مقطع الشفرة Code Descriptor وتوضح المقطع من الذاكرة الذي سيتسخدم كشفرة وما هي بدايته وحجمه وصلاحيات استخدامه حيث يمكن أن نسمح فقط للبرامج التي تعمل على مستوى النواة Kernel Mode بالدخول الى هذا المقطع.وفيما يلي شرح لمحتويات هذه الواصفة ويمكنك المطابقة مع الجدول الذي يوضح الشكل العام لكل واصفة.
تبدأ واصفة الكود Code Descriptor من العنوان 0x8 وهذا العنوان مهم جدا حيث سيكون هذا العنوان هو قيمة المسجل CS ، والبتات من 0-15 تحدد حجم المقطع Segment Limit والقيمة هي 0xffff تدل على أن أكبر حجم يمكن التعامل معه هو 0xffff.
البتات من 16-39 تمثل البتات 0-23 من عنوان بداية المقطع Base Address والقيمة التي تم اختيارها هي 0x0 وبالتالي نعرف أن عنوان بداية مقطع الكود هو 0x0 وعنوان النهاية 0xffff .
البايت رقم 6 ويسمى Access Byte يحدد العديد من الخصائص وفيما يلي توضيح لمعنى كل بت موجودة فيه:

- البت 0: Access Bit ويستخدم مع الذاكرة الظاهرية لذلك اخترنا القيمة 0.
- البت 1: بت القراءة والكتابة ، وتم اختيار القيمة 1 لذا يمكن قراءة وتنفيذ أي بايت موجودة في مقطع الكود من 0x0-0xffff.
- البت 2: expansion direction لا يهم حاليا لذا القيمة هي 0.
- البت 3: تم اختيار القيمة 1 دلالة على أن هذا مقطع شفرة Code Segment.
- البت 4: تم اختيار القيمة 1 دلالة على أن هذا مقطع للشفرة او للبيانات وليس للنظام.
- البتات 5-6: مستوى الحماية وتم اختيار القيمة 0 دلالة على أن هذا المقطع يستخدم فقط في الحلقة صفر Ring0 أو ما يسمى Kernel Mode.

- البت 7: تستخدم مع الذاكرة الظاهرية لذا تم اهمالها.

البايت رقم 7 ويسمى granularity يحدد أيضا بعض الخصائص، وفيما يلي توضيح لمعنى كل بت موجودة فيه:

- البتات 0-3: تمثل البتات من 16-19 من نهاية حجم المقطع Segment Limit والقيمة هي 0xf ، وبهذا يكون أقصى عنوان للمقطع هو 0xfffff أي 1 ميجا من الذاكرة ، ولاحقاً عندما يتم تفعيل بوابة A20 سنتمكن من الوصول حتى 4 جيجا من الذاكرة.
- البتات 4-5: محجوزة للنظام لذا تم اهمالها.
- البت 6: تم اختيار القيمة 1 دلالة على هذا المقطع هو 32 بت.
- البت 7: باختيار القيمة 1 سيتم إحاطة المقطع ب 4 KB.

البايت الاخير في واصفة مقطع الكود (البايت رقم 8) يمثل البتات من 24-32 من عنوان بداية مقطع الكود والقيمة هي 0x0 وبالتالي عنوان بداية مقطع الكود الكلي هو 0x0 أي من أول بايت في الذاكرة.
إذاً واصفة مقطع الكود Code Descriptor حددت عنوان بداية مقطع الكود ونهايته وكذلك صلاحية التنفيذ وحددت بأن المقطع هو مقطع كود Code Segment.
الواصفة التالية هي واصفة مقطع البيانات Data Descriptor وتبدأ من العنوان رقم 0x10 وهي مشابهة تماما لواصفة الكود باستثناء البت رقم 43 حيث يحدد ما اذا كان المقطع كود أم بيانات.
وبعد إنشاء هذا الجدول (GDT) في الذاكرة ، يجب أن يَحمل المسجل gdtr على حجم هذا الجدول ناقصا واحد وعلى عنوان بداية الجدول، ويتم ذلك عن طرَيق إنشاء مؤشرا الى جدول GDT ومن ثم استخدام الامر lgdt (وهو أمر يعمل فقط في الحلقة صفر Ring0) ، والشفرة التالية توضح ذلك.

Example ١.٢: Load GDT into GDTR

```
١
٢ bits  16    ; real mode.
٣
٤ ;*****************************
٥ ; load_gdt: Load GDT into GDTR.
٦ ;*****************************
٧
٨ load_gdt:
٩
١٠     cli          ; clear interrupt.
١١     pusha        ; save registers
١٢     lgdt [gdt_ptr]    ; load gdt into gdtr
١٣     sti          ; enable interrupt
```

```
١٤      popa        ; restore registers.
١٥
١٦      ret
١٧
١٨
١٩ ;*************************************
٢٠ ; gdt_ptr: data structure used by gdtr
٢١ ;*************************************
٢٢
٢٣ gdt_ptr:
٢٤
٢٥      dw end_of_gdt – begin_of_gdt – 1    ; size –1
٢٦      dd begin_of_gdt             ; base of gdt
```

٢.١.١ العنونة في النمط المحمي PMode Memory Addressing

في النمط الحقيقي يستخدم المعالج عنونة Segment:Offset وذلك بأن تكون أي من مسجلات المقاطع (Segments Registers) تحوي عنوان بداية المقطع ، ومسجلات العناوين تحوي العنوان داخل مقطع ما ، ويتم ضرب عنوان المقطع بالعدد 0x10 وجمع ال offset اليه للحصول على العنوان النهائي والذي سيمر بداخل مسار العنوان Address Bus.
أما النمط المحمي PMode فانه يستخدم عنونة Descriptor:Offset وذلك بأن تكون مسجلات المقاطع تحوي عنوان أحد الواصفات التي قمنا ببنائها (مثلا مسجل CS يحوي العنوان 0x8 ومسجل البيانات DS يحوي العنوان 0x10) ، وال offset سيتم جمعها الى عنوان بداية المقطع Base Address والذي قمنا بتحديده في جدول الواصفات كذلك سيتم التأكد من أن هذا العنوان لا يتجاوز حجم المقطع Segment Limit أيضا سيتم التأكد من مستوى الصلاحية وأنه يمكن الوصول للعنوان المطلوب. ونظراً لان في النمط المحمي يمكن استخدام مسجلات 32-bit فانه يمكن عنونة 4 جيجا من الذاكرة[١].

٣.١.١ الانتقال الى النمط المحمي

بعد إنشاء جدول GDT وتحميل مسجل GDTR يمكن الانتقال الى النمط المحمي عن طريق تفعيل البت الاول في مسجل التحكم cr0، وكما هو معروف أن هذا النمط لا يستخدم مقاطعات البايوس لذا يجب تعطيل عمل المقاطعات قبل الانتقال حتى لا تحدث أي مشاكل.

[١] بفرض أن بوابة A20 تم تفعيلها.

وبعد الانتقال الى النمط المحمي فان يجب تعيين الواصفة التي يجب استخدامها لمسجلات المقاطع ، وبالنسبة لمسجل CS فانه يمكن تعديل قيمته وذلك عن طريق تنفيذ far jump ،والكود التالي يوضح طريقة الانتقال الى النمط المحمي وتعديل قيم مسجلات المقاطع.

Example ١.٣: Switching to Protected Mode

```
١    ;————————————————————
٢    ; Load gdt into gdtr.
٣    ;————————————————————
٤
٥      call load_gdt
٦
٧    ;————————————————————
٨    ; Go to PMode.
٩    ;————————————————————
١٠     ; just set bit 0 from cr0 (Control Register 0).
١١
١٢     cli          ; important.
١٣     mov eax,cr0
١٤     or eax,0x1
١٥     mov cr0,eax      ; entering pmode.
١٦
١٧
١٨   ;————————————————————
١٩   ; Fix CS value
٢٠   ;————————————————————
٢١     ; select the code descriptor
٢٢     jmp 0x8:stage3
٢٣
٢٤
٢٥ ;*************************
٢٦ ; entry point of stage3
٢٧ ;*************************
٢٨
٢٩ bits 32      ; code now 32—bit
٣٠
٣١ stage3:
٣٢
٣٣   ;————————————————;
٣٤   ; Set Registers.
٣٥   ;————————————————;
٣٦
```

```
٣٧      mov ax,0x10     ; address of data descriptor.
٣٨      mov ds,ax
٣٩      mov ss,ax
٤٠      mov es,ax
٤١      mov esp,0x90000   ; stack begin from 0x90000.
٤٢
٤٣
٤٤    ;—————————————————;
٤٥    ; Hlat the system.
٤٦    ;—————————————————;
٤٧      cli     ; clear interrupt.
٤٨      hlt     ; halt the system.
```

٢.١ تفعيل البوابة A20

بوابة A20 Gate هي عبارة عن OR Gate موجودة على ناقل النظام System Bus [٢]والهدف منها هو التحكم في عدد خطوط العناوين Address Line، حيث كانت الاجهزة قديما (ذات المعالجات التي تسبق معالج 80286) تحوي على 20 بت (خط) للعناوين (20 address line) ، وعندما صدرت اجهزة IBM PC والتي احتوت على معالج 80286 تم زيادة خط العناوين الى 32 خط وهكذا أصبح من الممكن عنونة 4 جيجا من الذاكرة ، وحتى يتم الحفاظ على التوافقية مع الاجهزة السابقة فانه يمكن التحكم في بوابة A20 من فتح الخطوط A20–A31 واغلاقها.
هذه البوابة مرتبطة مع متحكم 8042 وهو متحكم لوحة المفاتيح (Keyboard Controller) ، وعند تفعيل البت رقم 1 في منفذ خروج البيانات (output data port) التابع لمتحكم لوحة المفاتيح فان هذا يفتح بوابة A20 وبهذا نستطيع الوصول الى 4 جيجا من الذاكرة ، ابتداءاً من العنوان 0x0–0xffffffff
وعند اقلاع الحاسب فان البايوس يقوم بتفعيل هذه البوابة لأغراض حساب حجم الذاكرة واختبارها ومن ثم يقوم بغلقها مجدداً للحفاظ على التوافقية مع الاجهزة القديمة.
وتوجد العديد من الطرق لتفعيل هذه البوابة ، العديد منها يعمل على أجهزة معينة لذلك سيتم ذكر العديد من الطرق واستخدام أكثر الطرق محمولية على صعيد الاجهزة المختلفة.

١.٢.١ متحكم لوحة المفاتيح 8042 والبوابة A20

عند الانتقال الى النمط المحمي (PMode) فانه لن يمكن استخدام مقاطعات البايوس ويجب التعامل المباشر مع متحكم أي عتاد والقراءة والكتابة من مسجلات المتحكم الداخلية . وبسبب ارتباط بوابة A20 مع

[٢]توجد البوابة تحديداً على خط العناوين رقم 20

متحكم لوحة المفاتيح فانه لا بد من التعامل مع هذا المتحكم لتفعيل البوابة ، وهذا يتم عن طريق استخدام أوامر المعالج in والامر out.
وبخصوص متحكم لوحة المفاتيح (متحكم 8042) فغالبا ما تأتي على شكل شريحة Integrated Circuit أو تكون مضمنة داخل اللوحة الأم (Motherboard) وتكون في ال South Bridge.ويرتبط هذا المتحكم مع متحكم آخر بداخل لوحة المفاتيح ، وعند الضغط على زر ما فانه يتم توليد Make Code ويُرسل الى المتحكم الموجود بداخل لوحة المفاتيح والذي بدروه يقوم بارساله الى متحكم 8042 عن طريق منفذ الحاسب (Hardware Port) .وهنا يأتي دور متحكم 8042 حيث يقوم بتحويل Make code الى Scan Code ويحفظها في مسجلاته الداخلية Buffer هذا المسجل يحمل الرقم 0x60 في أجهزة IBM and Compatible PC، وهذا يعني أنه في حالة قراءة هذا المسجل (عن طريق الأمر in) فانه يمكن قراءة القيمة المدخلة.
وفي الفصل السادس سيتم مناقشة متحكم لوحة المفاتيح بالتفصيل ، وسنكتفي هنا فقط بتوضيح الأجزاء المتعلقة بتفعيل بوابة A20.

٢.٢.١ طرق تفعيل البوابة A20

بواسطة System Control Port 0x92

في بعض الاجهزة يمكن استخدام أحد منافذ الادخال والاخراج وهو I/O part 0x92 لتفعيل بوابة A20 ، وعلى الرغم من سهولة هذه الطريقة الا أنها تعتبر أقل محمولية وبعض الاجهزة لا تدعمها ، وفيما يلي توضيح للبتات على هذا المنفذ:

- البت 0: تفعيل هذا البت يؤدي الى عمل reset للنظام والعودة الى النمط الحقيقي.
- البت 1: القيمة 0 لتعطيل بوابة A20 ، والقيمة 1 لتفعيلها.
- البت 2: لا تستخدم.
- البت 3: power on password bytes
- البتات 4-5: لا تستخدم.
- البتات 6-7: HDD activity LED : القيمة 0: off، القيمة 1: on.

والمثال التالي يوضح طريقة تفعيل البوابة .

Example ١.٤: Enable A20 by System Control Port 0x92

```
;*************************************************
; enable_a20_port_0x92:
;   Enable A20 with System Control port 0x92
```

```
;************************************************
enable_a20_port_0x92:
    push ax      ; save register.
    mov al,2     ; set bit 2 to enable A20
    out 0x92,al
    pop ax       ; restore register.
    ret
```

ويجب ملاحظة أن هذه الطريقة لا تعمل في كل الاجهزة وربما يكون هناك ارقام مختلفة للمنافذ ، ويعتمد في الآخر على مصنعي اللوحات الام ويجب قراءة كتيباتها لمعرفة العناوين.

بواسطة البايوس

يمكن استخدام مقاطعة البايوس int 0x15 الدالة 0x2401 لتفعيل بوابة A20 ، والدالة 0x2400 لتعطيلها.مع التذكير بأن يجب أن يكون المعالج في النمط الحقيقي حتى نتمكن من استدعاء هذه المقاطعة، والكود التالي يوضح طريقة التفعيل باستخدام البايوس.

Example ١.٥: Enable A20 by BIOS int 0x15

```
;************************************************
; enable_a20_bios:
;   Enable A20 with BIOS int 0x15 routine 0x2401
;************************************************
enable_a20_bios:
    pusha      ; save all registers
    mov ax,0x2401 ; Enable A20 routine.
    int 0x15
    popa       ; restore registers
    ret
```

بواسطة متحكم لوحة المفاتيح

يوجد منفذين لمتحكم لوحة المفاتيح: المنفذ 0x60 وهو يمثل ال buffer (في حالة القراءة منه يسمى Output Buffer وفي حالة الكتابة يسمى Input Buffer، والمنفذ 0x64 وهو لإرسال الاوامر الى المتحكم ولقراءة حالة المتحكم (Status). حيث يتم ارسال الأوامر الى المتحكم عن طريق المنفذ 0x64 واذا كان هناك وسائط لهذا الأمر فترسل الى ال buffer (المنفذ 0x60) وكذلك تقرأ النتائج من المنفذ 0x60.
وحيث ان تنفيذ أوامر البرنامج (عن طريق المعالج) أسرع بكثير من تنفيذ الأوامر المرسلة الى متحكم لوحة المفاتيح (وبشكل عام الى أي متحكم لعتاد ما) فانه يجب ان نوفر طرقاً لانتظار المتحكم قبل العودة الى البرنامج لاستكمال التنفيذ .
ويمكن عن طريق قراءة حالة المتحكم (عن طريق قراءة المنفذ 0x64) أن نعرف ما اذا تم تنفيذ الاوامر المرسلة ام لا ، وكذلك هل هناك نتيجة لكي يتم قرائتها في البرنامج ام لا.
وما يهمنا من البتات عند قراءة حالة المتحكم حاليا هو أول بتين فقط ، ووظيفتهما هي:

- البت 0: حالة ال Output Buffer:
  - القيمة 0: ال Output Buffer خالي (لا توجد نتيجة ، لا تقرأ الان).
  - القيمة 1: ال Output Buffer ممتلئ (توجد نتيجة ، قم بالقراءة الان).
- البت 1: حالة ال Input Buffer:
  - القيمة 0: ال Input Buffer خالي (لا توجد أوامر غير منفذة ، يمكن الكتابة الان).
  - القيمة 1: ال Input Buffer ممتلئ (توجد أوامر غير منفذة ، لا تكتب الان).

والشفرة التالية توضح كيفية انتظار المتحكم حتى ينفذ الاوامر المرسله اليه (wait input) وكيفية انتظار المتحكم الى ان يأتي بنتيجة ما (wait output).

Example ١.٦: Wait Input/Output

```
١
٢ ;****************************************
٣ ; wait_output: wait output buffer to be full.
٤ ;****************************************
٥
٦ wait_output:
٧
٨     in al,0x64     ; read status
٩     test al,0x1    ; is output buffer is empty?
١٠    je wait_output  ; yes, hang.
١١
١٢    ret       ; no,there is a result.
```

```
١٣
١٤
١٥ ;****************************************
١٦ ; wait_input: wait input buffer to be empty.
١٧         command executed already.
١٨ ;****************************************
١٩
٢٠ wait_input:
٢١
٢٢      in al,0x64     ; read status
٢٣      test al,0x2    ; is input buffer is full?
٢٤      jne wait_input  ; yes, hang.
٢٥
٢٦      ret        ; no,command executed.
```

ولإرسال اوامر الي المتحكم فان يجب استخدام المنفذ 0x64 وتوجد الكثير من الأوامر ، ونظرا لان هذا الجزء غير مخصص لبرمجة متحكم لوحة المفاتيح فاننا سنناقش فقط الاوامر التي تهمنا حاليا ، وفي الفصل السادس سنعود الى الموضوع بالتفصيل ان شاء الله.
وقائمة الاوامر حاليا:

- الأمر 0xad: تعطيل لوحة المفاتيح.
- الأمر 0xae: تفعيل لوحة المفاتيح.
- الأمر 0xd0: القراءة من Output Port.
- الأمر 0xd1: الكتابة الى Output Port.
- الأمر 0xdd: تفعيل بوابة A20.
- الأمر 0xdf: تعطيل بوابة A20.

وعن طريق الأمر 0xdd فانه يمكن تفعيل البوابة A20 بسهولة كما في الشفرة التالية ، لكن أيضا هذه الطريقة لا تعمل على كل الاجهزة حيث هناك بعض المتحكمات لا تدعم هذا الأمر.

Example ١.٧: Enable A20 by Send 0xdd

```
١
٢ ;*************************************************
٣ ; enable_a20_keyboard_controller:
٤ ;   Enable A20 with command 0xdd
٥ ;*************************************************
```

```
٦
٧ enable_a20_keyboard_controller:
٨
٩     ;cli
١٠    push ax      ; save register.
١١
١٢    mov al,0xdd   ; Enable A20 Keyboard Controller Command.
١٣    out 0x64,al
١٤
١٥    pop ax       ; restore register.
١٦    ret
```

وتوجد طريقة أخرى أكثر محمولية وهي عن طريق منفذ الخروج Output Port في متحكم لوحة المفاتيح ويمكن قراءة هذا المنفذ والكتابة اليه عن طريق ارسال الاوامر 0xd0 و 0xd1 على التوالي.
وعند قراءة هذا المنفذ (بارسال الامر d0 الى متحكم لوحة المفاتيح) فان القيم تعني:

- البت 0: System Reset:
  - القيمة 0: Reset Computer.
  - القيمة 1: Normal Operation.
- البت 1: بوابة A20:
  - القيمة 0: تعطيل.
  - القيمة 1: تفعيل.
- البتات 2-3: غير معرف.
- البت 4: Input Buffer Full.
- البت 5: Output Buffer Empty.
- البت 6: Keyboard Clock:
  - القيمة 0: High-Z.
  - القيمة 1: Pull Clock Low.
- البت 7: Keyboard Data:
  - القيمة 0: High-Z.
  - القيمة 1: Pull Data Low.

وعند تفعيل البت رقم 1 فان هذا يفعل بوابة A20 ويجب استخدام الامر or حتى يتم الحفاظ على بقية البتات .وبعد ذلك يجب كتابة القيم الى نفس المنفذ باستخدام الامر 0xd1 .
والشفرة التالية توضح كيفية تفعيل بوابة A20 عن طريق منفذ الخروج Output Port لمتحكم لوحة المفاتيح.

Example ١.٨: Enable A20 by write to output port of Keyboard Controller

```
١
٢ ;*************************************************
٣ ; enable_a20_keyboard_controller_output_port:
٤ ;   Enable A20 with write to keyboard output port.
٥ ;*************************************************
٦
٧ enable_a20_keyboard_controller_output_port:
٨
٩     cli
١٠    pusha     ; save all registers
١١
١٢    call wait_input   ; wait last operation to be finished.
١٣
١٤    ;--------------------
١٥    ; Disable Keyboard
١٦    ;--------------------
١٧    mov al,0xad   ; disable keyboard command.
١٨    out 0x64,al
١٩    call wait_input
٢٠
٢١    ;-------------------------------------
٢٢    ; send read output port command
٢٣    ;-------------------------------------
٢٤    mov al,0xd0   ; read output port command
٢٥    out 0x64,al
٢٦    call wait_output    ; wait output to come.
٢٧    ; we don't need to wait_input bescause when output came
          we know that operation are executed.
٢٨
٢٩    ;--------------------------
٣٠    ; read input buffer
٣١    ;--------------------------
٣٢    in al,0x60
٣٣    push eax    ; save data.
٣٤    call wait_input
٣٥
٣٦    ;-------------------------------------
٣٧    ; send write output port command.
٣٨    ;-------------------------------------
٣٩    mov al,0xd1   ; write output port command.
```

```
٤٠      out 0x64,al
٤١      call wait_input
٤٢
٤٣      ;—————————————
٤٤      ; enable a20.
٤٥      ;—————————————
٤٦      pop eax
٤٧      or al,2      ; set bit 2.
٤٨      out 0x60,al
٤٩      call wait_input
٥٠
٥١      ;———————————————————
٥٢      ; Enable Keyboard.
٥٣      ;———————————————————
٥٤      mov al,0xae    ; Enable Keyboard command.
٥٥      out 0x64,al
٥٦      call wait_input
٥٧
٥٨
٥٩      popa       ; restore registers
٦٠      sti
٦١
٦٢      ret
```

حيث في البداية تم تعطيل لوحة المفاتيح (عن طريق ارسال الامر 0xad) واستدعاء الدالة wait input للتأكد من أن الامر قد تم تنفيذه ومن ثم تم ارسال أمر قراءة منفذ الخروج لمتحم لوحة المفاتيح (الامر 0xda) وانتظار المتحكم حتى ينتهي من تنفيذ الامر ، وقد تم استخدام الدالة wait output لانتظار قيمة منفذ الخروج ، وبعدها تم قراءة هذه القيمة وحفظها في المكدس (Stack) ،وبعد ذلك تم ارسال أمر الكتابة الى منفذ الخروج لمتحكم لوحة المفاتيح (الامر 0xd1) وانتظار المتحكم حى ينتهي من تنفيذ الامر ومن قمنا بارسال قيمة المنفذ الخروج الجديدة بعد أن تم تفعيل البت رقم 1 وهو البت الذي يفعل بوابة A20 ، وفي الاخير تم تفعيل لوحة المفاتيح مجددا.

٣.١ أساسيات ال VGA

في عام 1987 قامت IBM بتطوير مقياس لمتحكمات شاشة الحاسب وهو Video Graphics Array واختصاراً VGA وجائت تسميته ب Array نظرا لانه تم تطويره كشريحة واحدة signle chip حيث استبدلت العديد من الشرائح والتي كانت تستخدم في مقاييس اخرى مثل MDA و CGA و EGA

، ويتكون ال VGA من Video Buffer, Video DAC , CRT Controller , Sequencer unit , Graphics Controller و Attribute Controller[٣].
ال Video Buffer هو مقطع من الذاكرة segment of memory يعمل كذاكرة للشاشة Memory Mapped ، وعند بداية التشغيل فان البايوس يخصص مساحة من الذاكرة بدءا من العنوان 0xa0000 كذاكرة للشاشة وفي حالة تم الكتابة الى هذه الذاكرة فان هذا سوف يغير في الشاشة ، هذا الربط يسمى Memory Mapping، أما ال Graphics Controller فهو الذي يقوم بتحديث محتويات الشاشة بناءاً على البيانات الموجودة في ال Video buffer.
وتدعم ال VGA نمطين للعرض الاول هو النمط النصي Text Mode والاخر هو النمط الرسومي APA Graphics Mode ويحدد النمط طريقة التعامل مع ال Video buffer وكيفة عرض البيانات.
النمط الرسومي All Point Addressable Graphics Mode يعتمد على البكسلات ، حيث يمكن التعامل مع كل بسكل موجود على حدة . والبكسل هو أصغر وحدة في الشاشة وتعادل نقطة على الشاشة. أما النمط النصي Text Mode فيعتمد على الحروف Characters ، ولتطبيق هذا النمط فان متحكم الشاشة Video Controller يستخدم ذاكرتين two buffers الاولى وهي خريطة الحروف Character Map وهي تعرف البكسلات لكل حرف ويمكن تغيير هذه الخريطة لدعم أنظمة محارف أخرى، أما الذاكرة الثانية فهيScreen Buffer وبمجرد الكتابة عليها فان التأثير سيظهر مباشرة على الشاشة.
ومقياس VGA هو مبني على المقاييس السابقة ، ابتداءا من مقياس Monochrome Display Adapter ويسمى اختصارا MDA والذي طورته IBM في عام 1981 ، و MDA لا تدعم النمط الرسومي والنمط النصي بها (يسمى Mode 7) يدعم 80 عمود و 24 صف (80*25). وفي نفس العام قامت IBM بتطوير مقياس Color Graphics Adapter (واختصارا CGA) الذي كان أول متحكم يدعم الالوان حيث يمكن عرض 16 لون مختلف.وبعد ذلك تم تطوير Enhanced Graphics Adapter.
ويجدر بنا التذكير بان متحكمات VGA متوافقة مع المقاييس السابقة Backward Compatible فعندما يبدأ الحاسب في العمل فان النمط سيكون النمط النصي Mode 7 (الذي ظهر في MDA) ، وهذا يعني اننا سنتعامل مع 80 عمود و 25 صف.

١.٣.١ عنونة الذاكرة في متحكمات VGA

عندما يبدأ الحاسب بالعمل فان البايوس يخصص العناوين من 0xa0000 الى 0xbffff لذاكرة الفيديو Video memroy (موجودة على متحكم VGA) ، هذه العناوين مقسمة كالاتي:

- من 0xb0000 الى 0xb7777: للنمط النصي أحادي اللون Monochrome Text Mode.
- من 0xb8000 الى 0xbffff: Color Text Mode.

وعند الكتابة في هذه العناوين فان هذا سوف يؤثر في الشاشة واظهار القيم التي تم كتابتها ، والمثال التالي يوضح كيفية كتابة حرف A بلون أبيض وخلفية سوداء.

[٣]شرح هذه المكونات سيكون في الفصل الخامس باذن الله ، وسيتم التركيز على بعض الاشياء بحسب الحاجة حاليا.

Example ١.٩: Print 'A' character on screen

```
١
٢ %define VIDEO_MEMORY  0xb8000    ; Base Address of Mapped
      Video Memory.
٣ %define CHAR_ATTRIBUTE  0x7      ; White chracter on black
      background.
٤
٥ mov edi,VIDEO_MEMORY
٦
٧ mov [edi],'A'            ; print A
٨ mov [edi+1],CHAR_ATTRIBUTE      ; in white foreground black
      background.
```

٢.٣.١ طباعة حرف على الشاشة

لطباعة حرف على الشاشة يجب ارسال الحرف الى عنوان ال Video Memory وحتى نتمكن من طباعة العديد من الحروف فانه يجب انشاء متغيران (x,y) لحفظ المكان الحالي للصف والعمود ومن ثم تحويل هذا المكان الى عنوان في ال Video Memoey. وفي البداية ستكون قيم (x,y) هي (0 ,0) أي ان الحرف سيكون في الجزء الاعلي من اليسار في الشاشة ويجب ارسال هذا الحرف الى عنوان بداية ال Video Memory وهو 0xb8000 (Color text Mode). ولطباعة حرف آخر فان قيم (x,y) له هي (0 ,1) ويجب ارسال الحرف الى العنوان 0xb8001 ، وسنستخدم العلاقة التالية للتحويل بين قيم (x,y) الى عناوين لذاكرة العرض Video Memory:

$$videomemory = 0xb0000$$
$$videomemory+ = x + y * 80$$

وبسبب أن هناك 80 حرف في كل عمود فانه يجب ضرب قيمة y ب 80 . والمثال التالي يوضح كيفية طباعة حرف عند (4,4) .

$$address = x + y * 80$$

$$address = 4 + 4 * 80 = 324$$

; now add the base address of video memory.

$$address = 324 + 0xb8000 = 0xb8144$$

وبارسال الحرف الى العنوان 0xb8144 فان الحرف سوف يظهر على الشاشة في الصف الخامس والعمود الخامس (الترقيم يبدأ من صفر وأول صف وعمود رقمها صفر).

وكما ذكرنا ان النمط النصي Mode 7 هو الذي يبدأ الحاسب به ، في هذا النمط يتعامل متحكم العرض مع بايتين من الذاكرة لكل حرف يراد طباعته ، بمعنى اذا ما أردنا طباعة الحرف A فانه يجب ارسال الحرف الى العنوان 0xb8000 وخصائص الحرف الى العنوان التالي له 0xb8001 وهذا يعني انه يجب تعديل قانون التحويل السابق واعتبار أن كل حرف يأخذ بايتين من الذاكرة وليس بايت واحد.
البايت الثاني للحرف يحدد لون الحرف وكثافة اللون (غامق وفاتح) والجدول التالي يوضح البتات فيه:

- البتات 0-2: لون الحرف:
  - البت 0: أحمر.
  - البت 1: أخضر.
  - البت 2: أزرق.
- البت 3: كثافة لون الحرف (0 غامق ، 1 فاتح).
- البت 4-6: لون خلفية الحرف:
  - البت 0: أحمر.
  - البت 1: أخضر.
  - البت 2: أزرق.
- البت 7: كثافة لون خلفية الحرف (0 غامق ، 1 فاتح).

وهكذا توجد 4 بت لتحديد اللون ، والجدول التالي يوضح هذه الألوان:

- 0: Black.
- 1: Blue.
- 2: Green.
- 3: Cyan.
- 4: Red.
- 5: Magneta.
- 6: Brown.
- 7: Light gray.
- 8: Dark Gray.
- 9: Light Blue.
- 10: Light Green.

- 11: Light Cyan.
- 12: Light Red.
- 13: Light Magneta.
- 14: Light Brown.
- 15: White.

اذاً لطباعة حرف على النمط Mode 7 فانه يجب ارسال الحرف وخصائصه الى ذاكرة العرض ، كما يجب مراعاة بعض الامور من تحديث المؤشر Cursor (هو خط underline يظهر ويختفي للدلالة على الموقع الحالي) و الانتقال الى الصف التالي في حالة الوصول الى اخر حرف في العمود أو في حالة كان الحرف المراد طباعته هو حرف الانتقال الى سطر جديد 0xa . والمثال التالي يوضح الدالة putch32 والتي تستخدم لطباعة حرف على الشاشة في النمط المحمي PMode.

Example ١.١٠: putch32 routine

```
١
٢ ; ******************************************
٣ ; putch32: print character in protected mode.
٤ ;   input:
٥ ;       bl: character to print.
٦ ; ******************************************
٧
٨ bits  32
٩
١٠ %define VIDEO_MEMORY  0xb8000   ; Base Address of Mapped
      Video Memory.
١١ %define COLUMNS     80      ; text mode (mode 7) has 80
      columns,
١٢ %define ROWS      25      ; and 25 rows.
١٣ %define CHAR_ATTRIBUTE  31      ; white on blue.
١٤
١٥ x_pos   db     0     ; current x position.
١٦ y_pos   db     0     ; current y position.
١٧
١٨ putch32:
١٩
٢٠     pusha      ; Save Registers.
٢١
٢٢   ;----------------------------
٢٣   ; Check if bl is new line ?
```

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

  cmp bl,0xa    ; if character is newline ?
  je new_row    ; yes, jmp at end.

;-----------------------------
; Calculate the memory offset
;-----------------------------
  ; because in text mode every character take 2 bytes: one
      for the character and one for the attribute, we must
      calculate the memory offset with the follwing
     formula:
  ; offset = x_pos*2 + y_pos*COLUMNS*2

  xor eax,eax

  mov al,2
  mul byte[x_pos]
  push eax      ; save the first section of formula.

  xor eax,eax
  xor ecx,ecx

  mov ax,COLUMNS*2    ; 80*2
  mov cl,byte[y_pos]
  mul ecx

  pop ecx
  add eax,ecx

  add eax,VIDEO_MEMORY  ; eax = address to print the
     character.

;-----------------------
; Print the chracter.
;-----------------------

  mov edi,eax

  mov byte[edi],bl          ; print the character,
  mov byte[edi+1],CHAR_ATTRIBUTE    ; with respect to the
     attribute.
```

```
٦١
٦٢    ;————————————————————
٦٣    ; Update the postions.
٦٤    ;————————————————————
٦٥
٦٦      inc byte[x_pos]
٦٧      cmp byte[x_pos],COLUMNS
٦٨      je new_row
٦٩
٧٠      jmp putch32_end
٧١
٧٢
٧٣    new_row:
٧٤
٧٥      mov byte[x_pos],0       ; clear the x_pos.
٧٦      inc byte[y_pos]         ; increment the y_pos.
٧٧
٧٨    putch32_end:
٧٩
٨٠      popa      ; Restore Registers.
٨١
٨٢      ret
```

وتبدأ هذه الدالة بفحص الحرف المراد طباعته (موجود في المسجل bl) مع حرف الانتقال الى السطر الجديد 0xa وفي حالة التساوي يتم نقل التنفيذ الى آخر جسم الدالة والذي يقوم بتصفير قيمة x وزيادة قيمة y دلالة على الانتقال الى السطر الجديد. أما في حالة كان الحرف هو أي حرف آخر فانه يجب حساب العنوان الذي يجب ارسال الحرف اليه حتى يمكن طباعته ، وكما ذكرنا أن النمط النصي Mode 7 يستخدم بايتين لكل حرف لذا سيتم استخدام العلاقة التالية للتحويل ما بين (x,y) الى العنوان المطلوب.

$$videomemory = 0xb0000$$
$$videomemory+ = x * 2 + y * 80 * 2$$

وكما يظهر في الكود السابق فقد تم حساب هذا العنوان وحفظه في المسجل eax وبعد ذلك تم طباعة الحرف المطلوب بالخصائص التي تم تحديدها مسبقا كثابت. وآخر خطوة في الدالة هي زيادة قيم (x,y) للدالة الى المكان التالي ، وهذا يتم بزيادة x فقط وفي حالة تساوت القيمة مع قيمة آخر عمود في الصف فانه يتم زيادة قيمة y وتصفير x دلالة على الانتقال الى الصف التالي.

٣.٣.١ طباعة السلاسل النصية strings

لطباعة سلسلة نصية سنستخدم دالة طباعة الحرف وسنقوم بأخذ حرف حرف من السلسة وارسالها الى دالة طباعة الحرف حتى تنتهي السلسلة ، والشفرة التالية توضح الدالة puts32 لطباعة سلسلة نصية.

Example ١.١١: puts32 routine

```
١
٢
٣ ; ****************************************
٤ ; puts32: print string in protected mode.
٥ ;   input:
٦ ;        ebx: point to the string
٧ ; ****************************************
٨
٩ bits  32
١٠
١١ puts32:
١٢
١٣     pusha      ; Save Registers.
١٤
١٥     mov edi,ebx
١٦
١٧   @loop:
١٨     mov bl,byte[edi]  ; read character.
١٩
٢٠     cmp bl,0x0       ; end of string ?
٢١     je  puts32_end     ; yes, jmp to end.
٢٢
٢٣     call putch32     ; print the character.
٢٤
٢٥     inc edi        ; point to the next character.
٢٦
٢٧     jmp @loop
٢٨
٢٩   puts32_end:
٣٠
٣١   ;-----------------------------------
٣٢   ; Update the Hardware Cursor.
٣٣   ;-----------------------------------
٣٤     ; After print the string update the hardware cursor.
٣٥
```

```
٣٦      mov bl,byte[x_pos]
٣٧      mov bh,byte[y_pos]
٣٨
٣٩      call move_cursor
٤٠
٤١      popa        ; Restore Registers.
٤٢
٤٣      ret
```

في هذه الدالة سيتم قراءة حرف حرف من السلسة النصية وطباعته الى أن نصل الى نهاية السلسلة (القيمة 0x0) ، وبعد ذلك سيتم تحديث المؤشر وذلك عن طريق متحكم CRT Controller ونظراً لان التعامل معه بطئ قليلا فان تحديث المؤشر سيكون بعد طباعة السلسلة وليس بعد طباعة كل حرف .

٤.٣.١ تحديث المؤشر Hardware Cursor

عند طباعة حرف او سلسلة نصية فان مؤشر الكتابة لا يتحرك من مكانه الا عند تحديده يدويا ، وهذا يتم عن طريق التعامل مع متحكم CRT Controller . هذا المتحكم يحوي العديد من المسجلات ولكننا سوف نركز على مسجل البيانات Data Register ومسجل نوع البيانات Index Register.
ولارسال بيانات الى هذا المتحكم ، فيجب اولا تحديد نوع البيانات وذلك بارسالها الى مسجل Index Register ومن ثم ارسال البيانات الى مسجل البيانات Data Register ، وفي حواسيب x86 فان مسجل البيانات يأخذ العنوان 0x3d5 ومسجل Index Register يأخذ العنوان 0x3d4.
والجدول التالي يوضح القيم التي يمكن ارسالها الى مسجل نوع البيانات Index Register.

- 0x0: Horizontal Total.
- 0x1: Horizontal Display Enable End.
- 0x2: Start Horizontal Blanking.
- 0x3: End Horizontal Blanking.
- 0x4: Start Horizontal Retrace Pulse.
- 0x5: End Horizontal Retrace.
- 0x6: Vertical Total.
- 0x7: Overflow.
- 0x8: Preset Row Scan.
- 0x9: Maximum Scan Line.

- 0xa: Cursor Start.
- 0xb: Cursor End.
- 0xc: Start Address High.
- 0xd: Start Address Low.
- 0xe: Cursor Location High.
- 0xf : Cursor Location Low.
- 0x10: Vertical Retrace Start.
- 0x11: Vertical Retrace End.
- 0x12: Vertical Display Enable End.
- 0x13: Offset.
- 0x14: Underline Location.
- 0x15: Start Vertical Blanking.
- 0x16: End Vertical Blanking.
- 0x17: CRT Mode Control.
- 0x18: Line Compare.

وعند ارسال أي من القيم السابقة الى مسجل Index Reigster فان هذا سيحدد نوع البيانات التي سترسل الى مسجل البيانات Data Register. ومن الجدول السابق سنجد أن القيمة `0xf` ستحدد قيمة x للمؤشر ، والقيمة `0xe` ستحدد قيمة y للمؤشر.وبعد ذلك يجب ارسال قيم x,y الى مسجل البيانات على التوالي مع ملاحظة أن متحكم CRT يتعامل مع بايت واحد لكل حرف وهذا يعني أننا سنستخدم القانون التالي للتحويل من قيم (x,y) الى عناوين.

$$videomemory = x + y * 80$$

والشفرة التالية توضح عمل الدالة move cursor والتي تعمل على تحريك المؤشر.

Example ١.١٢: Move Hardware Cursor

```
١
٢ ; ****************************************
٣ ; move_cursor: Move the Hardware Cursor.
٤ ;   input:
٥ ;       bl: x pos.
```

```
;       bh: y pos.
; ****************************************

bits  32

move_cursor:

    pusha     ; Save Registers.

  ;-------------------------
  ; Calculate the offset.
  ;-------------------------
    ; offset = x_pos + y_pos*COLUMNS

    xor ecx,ecx
    mov cl,byte[x_pos]

    mov eax,COLUMNS
    mul byte[y_pos]

    add eax,ecx
    mov ebx,eax

  ;-------------------------
  ; Cursor Location Low.
  ;-------------------------

    mov al,0xf
    mov dx,0x3d4
    out dx,al

    mov al,bl
    mov dx,0x3d5
    out dx,al

  ;-------------------------
  ; Cursor Location High.
  ;-------------------------

    mov al,0xe
    mov dx,0x3d4
    out dx,al
```

```
٤٨
٤٩      mov al,bh
٥٠      mov dx,0x3d5
٥١      out dx,al
٥٢
٥٣
٥٤      popa        ; Restore Registers.
٥٥
٥٦      ret
```

١.٣.٥ تنظيف الشاشة Clear Screen

تنظيف الشاشة هي عملية ارسال حرف المسافة بعدد الحروف الموجودة (80*25 في نمط Mode 7) و تصفير قيم (x,y) . والشفرة التالية توضح كيفية تنظيف الشاشة وتحديد اللون الازرق كخلفية لكل حرف.

Example ١.١٣: Clear Screen

```
١
٢ ; ********************************************
٣ ; clear_screen: Clear Screen in protected mode.
٤ ; ********************************************
٥
٦ bits  32
٧
٨ clear_screen:
٩
١٠      pusha      ; Save Registers.
١١      cld
١٢
١٣      mov edi,VIDEO_MEMORY     ; base address of video memory.
١٤      mov cx,2000    ; 25*80
١٥      mov ah,CHAR_ATTRIBUTE    ; 31 = white character on blue
            background.
١٦      mov al,' '
١٧
١٨      rep stosw
١٩
٢٠      mov byte[x_pos],0
٢١      mov byte[y_pos],0
```

```
٢٢
٢٣      popa        ; Restore Registers.
٢٤
٢٥      ret
```

٤.١ تحميل النواة

الى هنا تنتهي مهمة المرحلة الثانية من محمل النظام Second Stage Bootloader ويتبقى فقط البحث عن النواة ونقل التحكم اليها[٤]. وفي هذا الجزء سيتم كتابة نواة تجريبية بهدف التأكد من عملية نقل التحكم الى النواة وكذلك بهدف إعادة كتابة شفرة محمل النظام بشكل أفضل.
وسيتم استخدام لغة التجميع لكتابة هذه النواة التجريبية حيث أن الملف الناتج سيكون Pure Binary ولا يحتاج الى محمل خاص ، وابتداءاً من الفصل القادم سنترك لغة التجميع جانبا ونبدأ العمل بلغة السي والسي++ .
وبما أننا نعمل في النمط المحمي PMode فلا يمكننا أن نستخدم مقاطعة البايوس int 0x13 لتحميل قطاعات النواة الى الذاكرة ، ويجب أن نقوم بكتابة درايفر لمحرك القرص المرن أو نقوم بتحميل النواة الى الذاكرة قبل الانتقال الى النمط المحمي وهذا ما سنفعله الان ، وسنترك جزئية برمجة محرك القرص المرن لاحقا.
وحيث أن النمط المحمي يسمح باستخدام ذاكرة حتى 4 جيجا ، فان النواة سنقوم بتحميلها على العنوان 0x100000 أي عند 1 ميجا من الذاكرة .لكن علينا التذكر بأن النمط الحقيقي لا يدعم الوصول الى العنوان 0x100000 لذلك سنقوم بتحميل النواة أولا في أي عنوان خالي وليكن 0x3000 وعند الانتقال الى النمط المحمي سنقوم بنسخها الى العنوان 0x100000 ونقل التنفيذ والتحكم اليها.
والشفرة التالية توضح نواة ترحيبية.

Example ١.١٤: Hello Kernel

```
١
٢ org   0x100000       ; kernel will load at 1 MB.
٣
٤ bits  32           ; PMode.
٥
٦ jmp kernel_entry
٧
٨ %include "stdio.inc"
٩
١٠
```

[٤] الفصل التالي سيتناول موضوع النواة وكيفية برمجتها بالتفصيل.

```
١١
١٢ kernel_message  db    0xa,0xa,0xa,"                eqraOS
       v0.1 Copyright (C) 2010 Ahmad Essam"
١٣         db    0xa,0xa,    "          University of Khartoum
              — Faculty of Mathematical Sceinces.",0
١٤
١٥
١٦ logo_message  db    0xa,0xa,0xa,"                  ___ ___
       -------- _  / __ \ / __/"
١٧         db    0xa,         "               / —_) _ `/ __/ _
              `/ / /_/ /_\ \  "
١٨         db    0xa,         "               \__/\_, /_/  \_,_/
                \____//___/  "
١٩         db    0xa,         "                     /_/
                                        ",0
٢٠ ;*********************
٢١ ; Entry point.
٢٢ ;*********************
٢٣
٢٤ kernel_entry:
٢٥
٢٦   ;—————————————
٢٧   ; Set Registers
٢٨   ;—————————————
٢٩
٣٠     mov ax,0x10     ; data selector.
٣١     mov ds,ax
٣٢     mov es,ax
٣٣     mov ss,ax
٣٤     mov esp,0x90000   ; set stack.
٣٥
٣٦   ;————————————————————————————
٣٧   ; Clear Screen and print message.
٣٨   ;————————————————————————————
٣٩
٤٠     call clear_screen
٤١
٤٢     mov ebx,kernel_message
٤٣     call puts32
٤٤
٤٥     mov ebx,logo_message
٤٦     call puts32
```

```
٤٧
٤٨    ;————————————————
٤٩    ; Halt the system.
٥٠    ;————————————————
٥١
٥٢      cli
٥٣      hlt
```

والمرحلة الثانية من محمل النظام ستكون هي المسؤولة عن البحث عن النواة وتحميلها ونقل التنفيذ اليها ، وسيتم تحميلها الى الذاكرة قبل الانتقال الى النمط المحمي وذلك حتى نتكمن من استخدام مقاطعة البايوس `int 0x13` وعند الانتقال الى النمط المحمي سيتم نسخ النواة الى عنوان 1 ميجا ونقل التحكم الى النواة.

ولتحميل النواة الى الذاكرة يجب أولا تحميل Root Directory الى الذاكرة والبحث عن ملف النواة وفي حالة كان الملف موجودا سيتم قراءة عنوان أول كلستر له ، هذا العنوان سيعمل ك index في جدول FAT (والذي يجب تحميله الى الذاكرة ايضا) وسيتم قراءة القيمة المقابلة لهذا ال index والتي ستخبرنا هل ما اذا كان هذا الكلستر هو آخر كلستر للملف أم لا[٥].

والشفرة التالية توضح ملف المرحلة الثانية من المحمل stage2.asm ، وتم تقسيم الكود بشكل أكثر تنظيما حيث تم نقل أي دالة تتعلق بالقرص المرن الى الملف floppy.inc (ملف inc. هو ملف للتضمين في ملف آخر) ، والدوال المتعلقة بنظام الملفات موجودة على الملف fat12.inc ودوال الاخراج موجودة في stdio.inc ودوال تفعيل بوابة A20 موجودة على الملف a20.inc ودالة تعيين جدول الواصفات العام وكذلك تفاصيل الجدول موجودة في الملف gdt.inc ، اخيرا تم انشاء ملف common.inc لحفظ بعض الثوابت المستخدمة دائما[٦].

Example ١.١٥: Loading and Executing Kernel: Full Example

```
١
٢
٣ bits 16   ; 16—bit real mode.
٤ org 0x500
٥
٦ start:  jmp stage2
٧
٨ ;*************************
٩ ; include files:
١٠ ;*************************
١١ %include "stdio.inc"    ; standard I/O routines.
١٢ %include "gdt.inc"      ; GDT load routine.
```

[٥] راجع الفصل السابق لمعرفة التفاصيل.

[٦] جميع شفرات الملفات مرفقة مع البحث في مجلد example/ch3/boot/ وشفرة المحمل النهائية ستكون ملحقة في نهاية البحث.

```
%include "a20.inc"      ; Enable A20 routines.
%include "fat12.inc"    ; FAT12 driver.
%include "common.inc"   ; common declarations.

;*********************
; data and variable
;*********************

hello_msg   db    0xa,0xd,"Welcome to eqraOS Stage2",0xa,0xd
    ,0
fail_message  db    0xa,0xd,"KERNEL.SYS is Missing. press
    any key to reboot...",0


; ************************************
; entry point of stage2 bootloader.
; ************************************

stage2:

   ;—————————————
   ; Set Registers.
   ;—————————————

     cli

     xor ax, ax
     mov ds, ax
     mov es, ax

     mov ax, 0x0
     mov ss, ax
     mov sp, 0xFFFF

     sti

   ;———————————————————
   ; Load gdt into gdtr.
   ;———————————————————

     call load_gdt
```

```
٥٣
٥٤    ;————————————————————
٥٥    ; Enable A20.
٥٦    ;————————————————————
٥٧      call enable_a20_keyboard_controller_output_port
٥٨
٥٩    ;————————————————————
٦٠    ; Display Message.
٦١    ;————————————————————
٦٢      mov si,hello_msg
٦٣      call puts16
٦٤
٦٥    ;————————————————————
٦٦    ; Load Root Directory
٦٧    ;————————————————————
٦٨      call load_root
٦٩
٧٠    ;————————————————————
٧١    ; Load Kernel
٧٢    ;————————————————————
٧٣      xor ebx,ebx
٧٤      mov bp,KERNEL_RMODE_BASE     ; bx:bp buffer to load
          kernel
٧٥
٧٦      mov si,kernel_name
٧٧      call load_file
٧٨
٧٩      mov dword[kernel_size],ecx
٨٠      cmp ax,0
٨١      je enter_stage3
٨٢
٨٣      mov si,fail_message
٨٤      call puts16
٨٥
٨٦      mov ah,0
٨٧      int 0x16      ; wait any key.
٨٨      int 0x19      ; warm boot.
٨٩      cli          ; cannot go here!
٩٠      hlt
٩١
٩٢
٩٣    ;————————————————————
```

```
٩٤    ; Go to PMode.
٩٥    ;-------------------
٩٦
٩٧    enter_stage3:
٩٨
٩٩      ; just set bit 0 from cr0 (Control Register 0).
١٠٠
١٠١     cli          ; important.
١٠٢     mov eax,cr0
١٠٣     or eax,0x1
١٠٤     mov cr0,eax      ; entering pmode.
١٠٥
١٠٦
١٠٧   ;-----------------------
١٠٨   ; Fix CS value
١٠٩   ;-----------------------
١١٠     ; select the code descriptor
١١١     jmp CODE_DESCRIPTOR:stage3
١١٢
١١٣
١١٤ ;*************************
١١٥ ; entry point of stage3
١١٦ ;*************************
١١٧
١١٨ bits 32      ; code now 32-bit
١١٩
١٢٠ stage3:
١٢١
١٢٢   ;-------------------;
١٢٣   ; Set Registers.
١٢٤   ;-------------------;
١٢٥
١٢٦     mov ax,DATA_DESCRIPTOR       ; address of data
          descriptor.
١٢٧     mov ds,ax
١٢٨     mov ss,ax
١٢٩     mov es,ax
١٣٠     mov esp,0x90000   ; stack begin from 0x90000.
١٣١
١٣٢   ;----------------------------------------
١٣٣   ; Clear Screen and print message.
١٣٤   ;----------------------------------------
```

```
١٣٥
١٣٦       call clear_screen
١٣٧
١٣٨       mov ebx,stage2_message
١٣٩       call puts32
١٤٠
١٤١       mov ebx,logo_message
١٤٢       call puts32
١٤٣
١٤٤
١٤٥
١٤٦     ;—————————————————————
١٤٧     ; Copy Kernel at 1 MB.
١٤٨     ;—————————————————————
١٤٩       mov eax,dword[kernel_size]
١٥٠       movzx ebx,word[bytes_per_sector]
١٥١       mul ebx
١٥٢       mov ebx,4
١٥٣       div ebx
١٥٤
١٥٥       cld
١٥٦
١٥٧       mov esi,KERNEL_RMODE_BASE
١٥٨       mov edi,KERNEL_PMODE_BASE
١٥٩       mov ecx,eax
١٦٠       rep movsd
١٦١
١٦٢     ;—————————————————————
١٦٣     ; Execute the kernel.
١٦٤     ;—————————————————————
١٦٥       jmp CODE_DESCRIPTOR:KERNEL_PMODE_BASE
١٦٦
١٦٧     ;—————————————————————;
١٦٨     ; Hlat the system.
١٦٩     ;—————————————————————;
١٧٠       cli      ; clear interrupt.
١٧١       hlt      ; halt the system.
```

النتيجة:

شكل ١.١: محمل النظام أثناء العمل

شكل ٢.١: بدء تنفيذ النواة