

بسم الله الرحمن الرحيم

جامعة الخرطوم - كلية العلوم الرياضية  
قسم الحاسوب

بحث مقدم لنيل درجة البكالوريوس في علوم الحاسوب بعنوان:

# بَرْمَجَة وَ تَصْمِيمُ نِظَامِ تَشْغِيلٍ "نظام إقرأ"

إعداد الباحث : أحمد عصام عبد الرحيم أحمد  
إشراف : د. مصطفى بابكر صديق

١٣ يونيو ٢٠١٠

تم تنضيد هذا البحث باستخدام نظام  $\text{\LaTeX}$  تحت نظام لينوكس وويندوز.  
جميع الحقوق محفوظة © ٢٠١٠ أحمد عصام عبد الرحيم أحمد.

يسمح بنسخ، توزيع و/أو تعديل هذا المستند ضمن شروط إتفاقية ترخيص المستندات الحرة جنو الإصدار ١.٢ أو أي إصدار لاحق تم نشره من قبل مؤسسة البرمجيات الحرة، دون أية أقسام ثابتة، نصوص غلاف أمامي ونصوص غلاف خلفي. لقد تمت إضافة نسخة من إتفاقية الترخيص في القسم المعنون (إتفاقية ترخيص المستندات الحرة GNU).

# المحتويات

## I. الأساسيات Basics ٣

### ١. مقدمة عن أنظمة التشغيل ٥

- ١.١. ما هو نظام التشغيل . . . . . ٨
- ١.١.١. نظام التشغيل كجهاز تخيلي . . . . . ٨
- ٢.١.١. نظام التشغيل كمدير للموارد والعتاد . . . . . ٨
- ٢.١. تاريخ أنظمة التشغيل . . . . . ٩
- ١.٢.١. الجيل الصفري (١٦٢٤-١٩٤٥): الحواسيب الميكانيكية . . . . . ٩
- ٢.٢.١. الجيل الأول (١٩٤٥-١٩٥٥): الصمامات المفرغة و لوحات التوصيل ١٢
- ٣.٢.١. الجيل الثاني (١٩٥٥-١٩٦٥): الترانزستورات . . . . . ١٥
- ٤.٢.١. الجيل الثالث (١٩٦٥-١٩٨٠): الدوائر المتكاملة . . . . . ١٥
- ٥.٢.١. الجيل الرابع (من ١٩٨٠ حتى الان): الحواسيب الشخصية . . . . . ١٥

### ٢. معمارية حواسيب x86 ١٧

- ١.٢. معمارية النظام . . . . . ١٨
- ١.١.٢. مسار النظام System Bus . . . . . ١٨
- ٢.١.٢. متحكم الذاكرة . . . . . ٢٠
- ٣.١.٢. متحكم الإدخال والإخراج . . . . . ٢٠
- ٢.٢. المعالج . . . . . ٢١
- ١.٢.٢. دورة تنفيذ التعليمات . . . . . ٢٢
- ٢.٢.٢. أنماط عمل المعالج CPU Modes . . . . . ٢٢
- ٣.٢.٢. النمط الحقيقي Real Mode . . . . . ٢٤
- ٤.٢.٢. النمط المحمي Protected Mode . . . . . ٢٥
- ٥.٢.٢. النمط الغير حقيقي والنمط التخيلي . . . . . ٢٧
- ٦.٢.٢. معمارية معالجات x86 . . . . . ٢٧

## II. إقلاع الحاسب Booting ٣٥

٣٧	٣. إقلاع الحاسب ومحمل النظام Bootloader
٣٧	١.٣. إقلاع الحاسب
٣٨	٢.٣. محمل النظام Bootloader
٤٠	٣.٣. مخطط الذاكرة
٤٠	٤.٣. برمجة محمل النظام
٤١	١.٤.٣. عرض رسالة ترحيبية
٤٣	٢.٤.٣. معلومات قطاع الإقلاع
٥٢	٣.٤.٣. تحميل قطاع من القرص باستخدام المقاطعة 0x13 int
٥٤	٥.٣. مقدمة الى نظام FAT12
٥٥	١.٥.٣. قيود نظام FAT12
٥٦	٢.٥.٣. هيكلية نظام FAT12 على القرص
٥٩	٣.٥.٣. هيكلية القرص المرن
٥٩	٤.٥.٣. القراءة و الكتابة من نظام FAT12
٧١	٤. برمجة محمل النظام - المرحلة الثانية
٧١	١.٤. الانتقال الى النمط المحمي
٧٢	١.١.٤. جدول الواصفات العام Global Descriptor Table
٧٦	٢.١.٤. العنوان في النمط المحمي PMode Memory Addressing
٧٦	٣.١.٤. الانتقال الى النمط المحمي
٧٨	٢.٤. تفعيل البوابة A20
٧٨	١.٢.٤. متحكم لوحة المفاتيح 8042 والبوابة A20
٧٩	٢.٢.٤. طرق تفعيل البوابة A20
٨٥	٣.٤. أساسيات ال VGA
٨٦	١.٣.٤. عنوان الذاكرة في متحكمات VGA
٨٧	٢.٣.٤. طباعة حرف على الشاشة
٩٢	٣.٣.٤. طباعة السلاسل النصية strings
٩٣	٤.٣.٤. تحديث المؤشر Hardware Cursor
٩٦	٥.٣.٤. تنظيف الشاشة Clear Screen
٩٧	٤.٤. تحميل النواة

## III. النواة Kernel

١٠٥

١٠٧	٥. مقدمة حول نواة نظام التشغيل
١٠٧	١.٥ نواة نظام التشغيل
١٠٨	١.١.٥ مستويات التجريد
١٠٩	٢.٥ وظائف نواة النظام
١٠٩	١.٢.٥ إدارة الذاكرة
١١٠	٢.٢.٥ إدارة العمليات
١١٠	٣.٢.٥ نظام الملفات
١١٠	٣.٥ هيكلية وتصميم النواة
١١٠	١.٣.٥ النواة الضخمة Monolithic Kernel
١١١	٢.٣.٥ النواة المصغرة MicroKernel
١١١	٣.٣.٥ النواة الهجينة Hybrid Kernel
١١١	٤.٥ برمجة نواة النظام
١١٢	١.٤.٥ تحميل وتنفيذ نواة PE
١١٥	٢.٤.٥ تطوير بيئة التشغيل للغة سي++
١٢٠	٣.٤.٥ نقل التنفيذ الى النواة
١٢١	٥.٥ نظرة على شفرة نظام إقرأ
١٢٢	٦.٥ مكتبة السي القياسية
١٢٦	٧.٥ دالة طباعة المخرجات للنواة
١٢٧	٦. المقاطعات Interrupts
١٢٧	١.٦ المقاطعات البرمجية Software Interrupts
١٢٧	١.١.٦ المقاطعات في النمط الحقيقي
١٢٩	٢.١.٦ المقاطعات في النمط المحمي
١٣١	٣.١.٦ أخطاء المعالج
١٣٣	٤.١.٦ إنشاء جدول الواصفات العام GDT
١٣٧	٥.١.٦ إنشاء جدول المقاطعات IDT
١٣٧	٢.٦ متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller
١٣٧	١.٢.٦ المقاطعات العتادية Hardware Interrupts
١٣٨	٢.٢.٦ برمجة متحكم PIC
١٤٦	٣.٦ المؤقتة Programmable Interval Timer
١٤٦	١.٣.٦ برمجة المؤقتة PIT
١٤٩	٤.٦ توسعة طبقة HAL
١٥٠	١.٤.٦ دعم PIC
١٥٤	٢.٤.٦ دعم PIT
١٥٧	٣.٤.٦ واجهة HAL الجديدة

١٦٧	٧. إدارة الذاكرة
١٦٧	١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management
١٦٧	٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management
١٦٩	٨. مشغلات الاجهزة Device Driver
١٦٩	١.٨. برمجية مشغل لوحة المفاتيح Keyboard Driver
١٦٩	٢.٨. برمجية مشغل القرص المرن Floppy Disk Driver
١٦٩	٣.٨. برمجية متحكم DMAC
١٧١	٩. أنظمة الملفات
١٧٣	١. ترجمة وتشغيل البرامج
١٧٣	١.١. نظام ويندوز
١٧٣	٢.١. نظام لينوكس
١٧٥	ب. المراجع
١٧٩	ج. شفرة نظام إقرأ
١٨١	د. إتفاقية ترخيص المستندات الحرة GNU FDL

## الأمثلة التوضيحية

٦	Assembly Language	.١.١
٤٠	Smallest Bootloader	.٣.١
٤١	Welcom to OS World	.٣.٢
٤٤	Bios Parameter Block	.٣.٣
٤٥	BPB example	.٣.٤
٤٨	Hex value of bootloader	.٣.٥
٤٩	Complete Example	.٣.٦
٥٢	Reset Floppy Drive	.٣.٧
٥٤	Read Floppy Disk Sectors	.٣.٨
٦٠	Hello Stage2	.٣.٩
٦١	Load Root directory	.٣.١٠
٦٢	Find Stage2 Bootloader	.٣.١١
٦٣	Load FAT Table	.٣.١٢
٦٤	Convert Cluster number to LBA	.٣.١٣
٦٥	Convert LBA to CHS	.٣.١٤
٦٦	Load Cluster	.٣.١٥
٦٦	Read Sectors Routine	.٣.١٦
٦٨	Read FAT entry	.٣.١٧
٧٣	GDT	.٤.١
٧٥	Load GDT into GDTR	.٤.٢
٧٧	Switching to Protected Mode	.٤.٣
٧٩	Enable A20 by System Control Port 0x92	.٤.٤
٨٠	Enable A20 by BIOS int 0x15	.٤.٥
٨١	Wait Input/Output	.٤.٦
٨٢	Enable A20 by Send 0xdd	.٤.٧
٨٤	Enable A20 by write to output port of Keyboard Controller	.٤.٨
٨٧	Print 'A' character on screen	.٤.٩
٨٩	putch32 routine	.٤.١٠
٩٢	puts32 routine	.٤.١١
٩٤	Move Hardware Cursor	.٤.١٢
٩٦	Clear Screen	.٤.١٣
٩٧	Hello Kernel	.٤.١٤

٩٩	Loading and Executing Kernel: Full Example	٤.١٥
١١٢	Portable Executable Header	٥.١
١١٥	Getting Kernel entry	٥.٢
١١٦	Global new/delete operator	٥.٣
١١٦	Pure virtual function call handler	٥.٤
١١٦	Floating Point Support	٥.٥
١١٧	Object Initializer	٥.٦
١١٩	Delete Object	٥.٧
١٢٠	Kernel Entry routine	٥.٨
١٢٢	null.h:Definition of NULL in C and C++	٥.٩
١٢٣	size_t.h:Definition of size_t in C/C++	٥.١٠
١٢٤	stdint.h:typedef data type	٥.١١
١٢٥	cstdint:C++ typedef data type	٥.١٢
١٢٥	ctype.h:determine character type	٥.١٣
١٣٠	Example of interrupt descriptor	٦.١
١٣٠	Value to put in IDTR	٦.٢
١٣٣	include/hal.h:Hardware Abstraction Layer Interface	٦.٣
١٣٥	hal/gdt.cpp:Install GDT	٦.٤
١٤١	Initialization Control Words 1	٦.٥
١٤٢	Initialization Control Words 2	٦.٦
١٤٣	Initialization Control Words 3	٦.٧
١٤٣	Initialization Control Words 4	٦.٨
١٤٤	Send EOI	٦.٩
١٤٩	PIT programming	٦.١٠
١٥٠	hal/pic.h: PIC Interface	٦.١١
١٥٢	hal/pic.cpp: PIC Implementation	٦.١٢
١٥٤	hal/pit.h: Pit Interface	٦.١٣
١٥٥	hal/pit.cpp: PIT Implementation	٦.١٤
١٥٧	New HAL Interface	٦.١٥
١٥٨	New HAL Impelmentation	٦.١٦
١٦١	kernel/main.cpp	٦.١٧
١٦١	kernel/exception.h	٦.١٨
١٦٣	kernel/exception.cpp	٦.١٩
١٦٣	kernel/panic.cpp	٦.٢٠



## قائمة الأشكال

٥	الشكل العام لأوامر المعالج x86	١.١
٧	طبقات الحاسب	٢.١
٩	آلة باسكال	٣.١
١٠	آلة Step Reckoner في متحف بألمانيا	٤.١
١٠	محرك الفروق بعد أن قام ابن باباياج بتجميعه	٥.١
١١	المحرك التحليلي بمتحف في لندن	٦.١
١٢	حاسبة Z1 بعد إعادة انشائها في متحف بألمانيا	٧.١
١٢	حاسبة Atanasoff بعد إعادة انشائها في جامعة Iowa State	٨.١
١٣	حاسبة Harvard Mark I	٩.١
١٣	آلة إنجما الألمانية لتشفير الرسائل وفكها	١٠.١
١٤	الحاسبة colossus التي كسرت شفرة إنجما	١١.١
١٤	الحاسبة ENIAC	١٢.١
١٥	الحاسبة EDVAC	١٣.١
١٧	معمارية حواسيب x86	١.٢
١٨	المسارات في الحواسيب الشخصية x86	٢.٢
٢٠	الجسر الشمالي	٣.٢
٢٦	تداخل المقاطع في النمط الحقيقي	٤.٢
٢٧	حلقات المعالج	٥.٢
١٠٤	محمل النظام أثناء العمل	١.٤
١٠٤	بدء تنفيذ النواة	٢.٤
١٣٧	متحكم المقاطعات القابل للبرمجة 8259A	١.٦
١٣٩	مشابك متحكم PIC	٢.٦
١٤٦	المؤقتة القابلة للبرمجة 8253	٣.٦
١٤٧	مشابك المؤقتة PIT	٤.٦
١٦٤	واجهة النظام بعد توسعة طبقة HAL	٥.٦
١٦٥	دالة تخدم المقاطعات الافتراضية	٦.٦



## قائمة الجداول

٢٢	مخطط الذاكرة لحواسيب x86	١.٢
٢٣	منافذ الإدخال والإخراج لحواسيب x86	٢.٢
٢٨	الأوامر التي تتطلب صلاحية الحلقة صفر	٣.٢
٢٠٠٤	EFLAGS الأعلام مسجل	٣٢
١٣٨	مقاطعات العتاد لحواسيب x86	١.٦
١٤٠	مسجل IRR/ISR/IMR	٢.٦
١٤١	عناوين المنافذ لمتحكم PIC	٣.٦
١٤١	الأمر الأول ICW1	٤.٦
١٤٣	الأمر الثالث للمتحكم الرئيسي ICW3 for Primary PIC	٥.٦
١٤٣	الأمر الثالث للمتحكم الثانوي ICW3 for Slave PIC	٦.٦
١٤٤	الأمر الرابع ICW4	٧.٦
١٤٤	أمر التحكم الثاني OCW2	٨.٦
١٤٥	أمر OCW2	٩.٦
١٤٨	مسجلات المؤقتة 8253 PIT	١٠.٦



## ملخص البحث

الهدف الرئيسي لهذه الدراسة هو توفير بحثاً متكاملًا للطلاب والباحث في مجال برمجة أنظمة التشغيل وذلك لإفتقار المكتبة العربية لهذه النوعية من البحوث ، و تناولت هذه الدراسة العديد من أساسيات ومفاهيم نظم التشغيل (إقلاع النظام ، الإنتقال الى النمط المحمي ، إدارة الذاكرة ، أنظمة الملفات وتعريفات العتاد) وكيفية برمجتها من الصفر دون الإعتماد على أي مكونات أو مكتبات خارجية. ويحوي هذا البحث شفرة نظام تشغيل ( تم تسميته بنظام إقرأ) والذي تم برمجته ليكون عوناً ودليلاً للطلاب أثناء دراسته في هذا المجال. و رؤية الباحث في هذه الدراسة هي أن تستخدم كمنهج لتدريس الطلاب في مادة نظم التشغيل وأن تُدرس الشفرة المصدرية للنظام . ولا يُقتصر على ذلك بل يستمر التطوير في النظام ليكون أداة تعليمية (مفتوحة المصدر) للطلاب والباحثين.



القسم I.

الأساسيات Basics

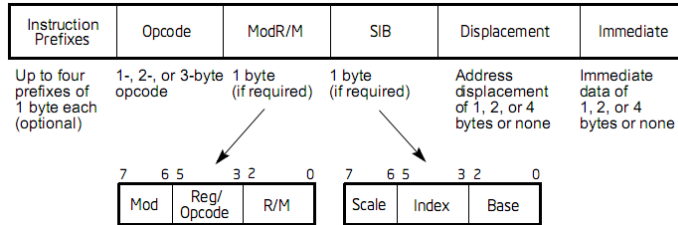




# ١. مقدمة عن أنظمة التشغيل

جهاز الحاسب هو مجموعة من الشرائح الإلكترونية والعتاديات والمتحكمات المرتبطة مع بعضها لتوفير منصة تشغيلية للبرامج والتي بدورها لن يعمل هذا الجهاز. ويمكن تقسيم البرامج بحسب طبيعة عملها ووظيفتها الى قسمين هما **برامج المستخدم** والتي صممت خصيصاً لحل مشاكل المستخدم و **برامج النظام** والتي تتحكم في عتاد وموارد الحاسب ، ويعتبر نظام التشغيل مثالا لبرامج النظام حيث يدير عتاد وموارد الحاسب بالإضافة الى ميزة مهمة وهي توفر بيئة تشغيل وهمية (Virtual Machine) لبرامج المستخدم. ويوضح التعريف السابق عدداً من المفاهيم التي يجب الوقوف عليها وتوضيحها بشكل مفصل. فجهاز الحاسب هو منصة تشغيلية حقيقية للأوامر ويأتي ذلك بسبب وجود متحكم خاص لمعالجة الأوامر وتنفيذها ، هذا المتحكم هو المعالج (Processor) حيث يعمل على تنفيذ الأوامر (من عمليات حسابية ومنطقية) وإرسال النتائج الى الأماكن المطلوبة. وتسمى مجموعة الأوامر التي ينفذها المعالج باسم **البرامج**، وبسبب تكلفة بناء المعالج فانه غالباً ما يتعرف على عدداً معيناً من الأوامر والتي تعرف بمجموعة الأوامر (Instruction Set). لذلك حتى يتم تنفيذ أوامر أي برنامج فانها يجب أن تُكتب وفقاً لمجموعة الأوامر التي يدعمها المعالج<sup>١</sup>. والشكل ١.١ يوضح نموذجاً عاماً لتعليمات وأوامر المعالج التي تتكون منها البرامج. وجزءاً منها هي اختيارية وسنركز هنا على ال OPCODE والتي تمثل أوامر المعالج.

شكل ١.١: الشكل العام لأوامر المعالج x86



وتشكل أوامر المعالج لغة برمجية من خلالها يمكن برجة الحاسب وكتابة البرامج لحل مشاكل المستخدم ، هذه اللغة تسمى بلغة الآلة (Machine Language). وتتكون هذه اللغة من الرموز 0 و 1 حيث أن أوامر المعالج ما هي الا سلسلة معينة من هذه الرموز. فمثلاً لتعيين القيمة 31744 للمسجل AX<sup>٢</sup> يجب أن يحوي البرنامج على الأمر 101110001100000000000111. وبالتالي تكون عملية كتابة برنامج

<sup>١</sup> استحدثت عن معالجات إنتل ٣٢ بت في هذا البحث نظراً لأنها الأكثر انتشاراً.  
<sup>٢</sup> المسجلات هي ذواكر بداخل المعالج.

متكامل بهذه اللغة أمراً في غاية الصعوبة وكذلك مهمة تنقيح البرنامج وتطويره في المستقبل هي معقدة أيضاً. لذلك ظهرت لغة التجميع لحل هذه المشكلة حيث أن اللغة تدعم مسميات ومختصرات للمسجلات ولأوامر المعالج ، فمثلاً الأمر السابق في لغة التجميع يكون بالصورة التالية.

#### Example ١.١ : Assembly Language

```
١ MOV AX, 0x7C00 ; Instead of 101110001100000000000111
```

والذي يجب تحويله الى لغة الآلة حتى يتمكن المعالج من تنفيذه ، هذا المحول يسمى بالمجمع والذي يقوم بتحويل أمر لغة التجميع الى ما يقابله بلغة الآلة<sup>٣</sup>. ولم تنجح لغة التجميع في توفير لغة عالية المستوى تبسط عملية برمجة البرامج بشكل أكبر وذلك بسبب أنها مختصرات للغة الآلة لذلك سرعان ما تم تطوير لغات عالية المستوى مثل لغة السي<sup>٤</sup> والسي++ بحيث تكتب البرامج فيها بشكل مبسط بعيداً عن تعقيدات الآلة وأوامرها ومسجلاتها وتدعم هذه اللغات عدداً من التراكيب وجمل التحكم العالية المستوى. ولكي ينفذ المعالج برامج هذه اللغات فإنه يجب أولاً ترجمة الشفرة المصدرية الى ما يقابلها بلغة التجميع وهذا يتم عن طريق برنامج يسمى المترجم (Compiler) وبعدها يقوم المجمع بتحويل شفرة التجميع الى برنامجاً بلغة الآلة والذي يستطيع المعالج تنفيذه.

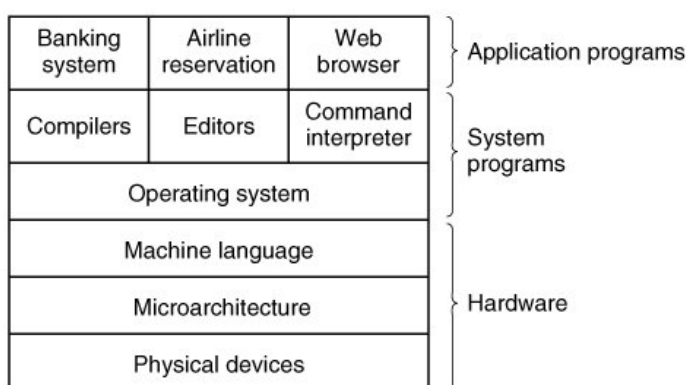
□ بخصوص تمثيل البيانات والبرامج في الحاسب فإنها تمثل بطرق مختلفة تختلف على حسب وحدة التخزين ولكنها في الآخر تستخدم المنطق الثنائي وهو وجود طاقة كهربائية أم لا ، فمثلاً تكون الذاكرة الرئيسية DRAM من ملايين المكثفات (Capacitors) والترانزستورات (Transistors) لتكوين خلايا الذاكرة (Memory Cells) ، و تتكون كل خلية (والتي تشكل بت واحد من الذاكرة) من مكثف وترانزستور بحيث يحفظ المكثف قيمة الخلية (البت) والتي هي إما وجود إلكترون (منطقياً تساوي 1) وإما عدمها (منطقياً تساوي 0) ويعمل الترانزستور على تغيير قيمة المكثف . وعلى هذا الشكل تحفظ جميع الأوامر والبرامج في الذاكرة الرئيسية ويأتي دور المعالج لتنفيذ هذه الأوامر حيث يقوم بقراءتها وفهم وظيفتها (Decode) وتنفيذها ومن ثم يقوم بحفظ النتائج. ولكي ينفذ المعالج أي برنامج فإن البرنامج يجب أن يتواجد على الذاكرة الرئيسية وليس على أحد الدوائر الثانوية (مثل القرص الصلب).

حتى الان لم نذكر وظيفة نظام التشغيل لأن بيئة التشغيل الحقيقية هي المعالج وليست نظام التشغيل أو غيره من البرامج وعلى المبرمج الإلمام بكيفية برمجة عتاد ومتحكمات الحاسب وكيفية طباعة المخرجات على الشاشة وقراءة البيانات من متحكم لوحة المفاتيح ولا يقتصر على ذلك بل على المبرمج توفير طرقاً ودوالاً لإدارة الذاكرة من حجز المقاطع وتحريرها وكذلك إدارة جميع عتاد الحاسب. كل ذلك يجعل عملية كتابة البرامج مستحيلة وهذا ما أدى الى ظهور طبقة برمجية (Layer) تدير عتاد وموارد الحاسب

<sup>٣</sup> كل أمر بلغة التجميع يقابله أمراً واحداً بلغة الآلة لذلك حقيقة لا يوجد فرقاً في أداء البرامج المكتوبة بأي منهم ولا في حجم الملف الناتج ، وإنما يظهر الفرق في سهولة تطوير البرامج بلغة التجميع ولكن على حساب أنه يجب تحويلها عن طريق المجمع.  
<sup>٤</sup> تم تطوير لغة السي بهدف برمجة نظام يونيكس Unix في معامل بيل.

وتوفر واجهة برمجية للمبرمج لكي يتعامل مع هذه الموارد. هذه الطبقة سميت **بنظام التشغيل (Operating System)**. الهدف الرئيسي لهذه الطبقة هي عزل المبرمج عن تعقيدات العتاد بحيث أن إدارة هذه العتاديات أصبحت من مهمة هذه الطبقة وفي نفس الوقت توفر واجهة برمجية (أو جهاز تخيلي) للاستفادة من هذه العتاديات. والشكل ٢.١ يوضح موضع هذه الطبقة (نظام التشغيل) في حالة قسمنا جهاز الحاسب إلى عدة طبقات [?]. وأدنى طبقة هي طبقة العتاديات (Device Level) حيث تتكون من المتحكمات

شكل ٢.١: طبقات الحاسب



ومن الشرائح المتكاملة (Integrated Circuit) والأسلاك وكل ما يتعلق بالأجهزة المادية. يلي هذه الطبقة طبقة Microarchitecture وفيها تظهر برمجيات (Microprogram) تتحكم في عمل المتحكمات لكي تؤدي وظيفتها فمثلاً برمج ال data path بداخل المعالج والذي يقوم في كل دورة للساعة (Clock Cycle) بجلب قيمتين من المسجلات إلى وحدة الحساب والمنطق (Arithmetic Logic Unit) التي تجري عليهم عملية ما ومن ثم تقوم بحفظ النتيجة في أحد المسجلات. وظيفة data path هي تنفيذ الأوامر والتعليمات وذلك بارسالها إلى وحدة الحساب والمنطق، وتشكل مجموعة الأوامر المدعومة وكذلك المسجلات المرئية للمبرمج لغة التجميع طبقة مجموعة الأوامر (Instruction Set Architecture) وتسمى هذه الطبقة طبقة الآلة (Machine Language) حيث تحوي على كل الأوامر التي يدعمها المعالج بما فيها أوامر القراءة والكتابة من مسجلات متحكمات العتاد (Device Controller). يلي هذه الطبقة طبقة نظام التشغيل والتي تفصل وتعزل العتاد عن المستخدم فبدلاً من أن يقوم المبرمج ببرمجة متحكم القرص الصلب ونظام للملفات حتى يتمكن من قراءة ملف على القرص فإن النظام يوفر واجهة مبسطة بالصورة `read(fd, buffer, size)`. وأخيراً توجد طبقة البرامج (برامج النظام والمستخدم) ولا تصنف الكثير من برامج النظام ضمن نظام التشغيل حيث أن البرامج التي تتبع لنظام التشغيل يجب أن تعمل في

مستوى النواة (Kernel Mode) وليس في المستويات الأخرى<sup>٥</sup>.

## ١.١. ما هو نظام التشغيل

من الصعب إيجاد تعريفاً واضحاً لأنظمة التشغيل فما يعتبره البعض تابعاً لنظام ما لا يعتبره الآخرون كذلك. لكن ما تم الاتفاق عليه هو أن **نظام التشغيل** يدير عتاد وموارد الحاسب ويوفر واجهة برمجية (جهاز تخيلي) من خلالها يمكن الاستفادة من هذه الموارد.

### ١.١.١. نظام التشغيل كجهاز تخيلي

مما سبق نجد أن الواجهة التي تقدمها طبقة الآلة (Machine Language Level) هي بدائية ويصعب استخدامها في كتابة البرامج ، فكما ذكرنا كمثال للقراءة من ملف على القرص يجب أن يحوي البرنامج على شفرة لنظام الملفات حتى نعرف عنوان الملف الفيزيائي على القرص، وكذلك يجب أن يحوي البرنامج على شفرة للتعامل مع متحكم القرص الصلب وهي شفرة ليست باليسيرة حيث للقراءة من القرص يجب تحديد رقم المقطع ورقم الرأس ورقم المسار وتحديد الذاكرة المؤقتة (Buffer) حتى يتم تحميل المقاطع إليها. كل هذه الأمور لو استمرت بهذا الشكل لما وصلت التطبيقات لما هي عليها الآن، لذلك كان الحل هو بإيجاد طبقة نظام التشغيل والتي توفر واجهة أو أوامر مبسطة وبمجردة من تفاصيل وتعقيدات العتاد لكي تستخدمها البرامج بدلا من الأوامر التي توفرها طبقة الآلة .

### ٢.١.١. نظام التشغيل كمدير للموارد والعتاد

بعد أن تم عزل المبرمج بواسطة طبقة نظام التشغيل فان هذه الطبقة تقدم بجانب الواجهة البرمجية إدارة لعتاد الحاسب (المعالج،الذاكرة،الأقراص الصلبة والمرنة، كرت الشبكة، وغيرها من المتحكمات) ، ومهمة إدارة العتاد تتركز في حجز العتاد وتحريره ، فمثلاً يقوم نظام التشغيل بإدارة المعالج نفسه وذلك بأن يحجز المعالج لبرنامج ما ومن ثم يحجز المعالج ويحجزه لبرنامج آخر (تعدد المهام Multitasking) وكذلك يدير النظام أهم موارد الحاسب وهي الذاكرة الرئيسية وذلك بحجز مقاطع من الذاكرة (Memory Blocks) بناءاً على طلب برامج المستخدم وكذلك عملية تحرير الذواكر وإدارة الذاكرة التخيلية ومفهوم الصفحات.

<sup>٥</sup> في الفصل الثاني بإذن الله سيتم الحديث عن مستويات الحماية في المعالجات.

## ٢.١. تاريخ أنظمة التشغيل

خلال سنوات مضت تطورت أنظمة التشغيل تطوراً ملحوظاً من أنظمة تشغيل لبرنامجاً واحداً الى أنظمة موزعة تسمح بتشغيل أكثر من برنامج على عدة حواسيب مختلفة. هذا التطور سببه الرئيسي تطور الحاسبات والمعالجات وازدياد حجم الذواكر بشكل رهيب. وفي هذا الجزء سنلقي نظرة على تطور أجيال الحواسيب وبعض أنظمة التشغيل التي استخدمت في تلك الفترات.

### ١.٢.١. الجيل الصفري (١٦٢٤-١٩٤٥): الحواسيب الميكانيكية

أول محاولة لبناء آلة حسابية كانت من قبل العالم الفرنسي بليز باسكال<sup>٦</sup> في عام ١٦٤٢ عندما كان عمره ١٩ عاماً وذلك لمساعدة والده الذي كان يعمل محصلاً للضرائب لمصلحة الحكومة الفرنسية. هذه الآلة (وتعرف بالاسم *Pascaline*) هي ميكانيكية بالكامل وتوفر فقط عملية الجمع والطرح (انظر الشكل ٣.١).

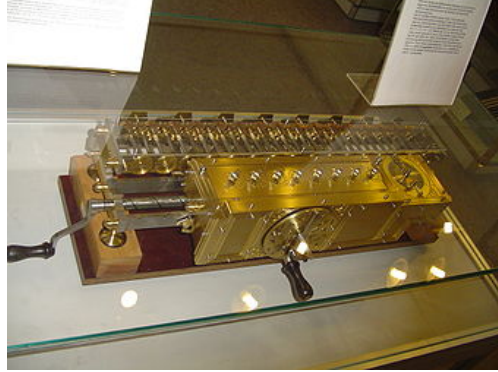
شكل ٣.١: آلة باسكال



وبعد حوالي ٣٠ عاماً قام العالم الرياضي جوتفريد ليبتر Gottfried Wilhelm Leibniz ببناء آلة حسابية ميكانيكية أخرى (تم الإنتهاء منها في عام ١٦٩٤ وسميت بالاسم *Step Reckoner*) ولكن هذه المرة أصبح من الممكن إجراء العمليات الحسابية الأربعة: الجمع والطرح والضرب والقسمة (الشكل ٤.١). ومضت حوالي ١٥٠ عاماً بدون أي شيء يذكر حتى قام البروفيسور شارلز باباج Charles Babbage بتصميم آلة محرك الفروق *Difference engine* (انظر الشكل ٥.١)، وهي آلة ميكانيكية أيضاً تشابه آلة باسكال في أنها لا توفر سوى عمليتي الجمع والطرح لكن هذه الآلة تم تصميمها لغرض حساب قيم دوال كثيرات الحدود باستخدام طرق التقريب المنتهية (*Method of Finite Differences*). وما ميز هذه الآلة هي طريقة إخراج النتائج حيث تنقش النتائج على ألواح نحاسية. وعلى الرغم من أن آلة الفروق عملت جيداً

<sup>٦</sup>والذي تم تسمية لغة البرمجة باسكال باسمه تشريفاً له.

شكل ٤.١: آلة Step Reckoner في متحف بألمانيا



شكل ٥.١: محرك الفروق بعد أن قام ابن بابايج بتجميعه



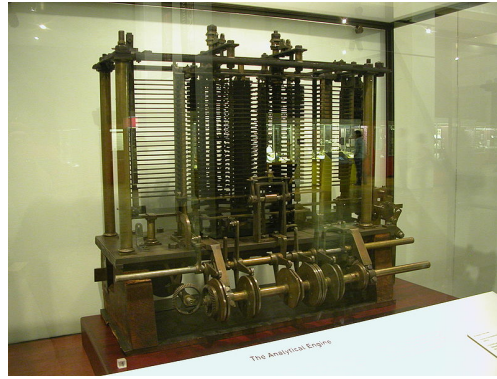
إلا أن تصميمها كان يسمح بحساب خوارزمية واحدة فقط<sup>٧</sup> وهذا ما جعل شارلز بابايج يعيد محاولته مجدداً ويستهلك جزءاً ضخماً من وقته ومن ثروة حكومته في بناء آلة أخرى عُرفت بالمحرك التحليلي Analytical Engine (انظر الشكل ٦.١). هذا المحرك (وهو أيضاً آلة ميكانيكية بالكامل) احتوى على أربع مكونات: المخزن (الذاكرة Memory)، الطاحنة (وحدة الحساب Computation Unit)، وحدة الإدخال (قارئ البطاقات المثقبة Punched Card Reader) ووحدة الإخراج (البطاقات المثقبة واللوحات المطبوعة). ويتكون المخزن من ١٠٠٠ كلمة (Word) بطول ٥٠ رقم صحيح وتستخدم لحفظ المتغيرات والنتائج، أما الطاحنة فتستقبل الوسائط من المخزن وتجري عليهم أي من العمليات الرياضية الأربعة ومن ثم تحفظ النتائج في المخزن. الميزة الأساسية للمحرك التحليلي هو قدرته على حل عدد كبير من المشاكل

<sup>٧</sup> في عام ١٩٩١ قام متحف العلوم بلندن ببناء نموذج مكتمل لمحرك الفروق.

## ٢.١. تاريخ أنظمة التشغيل

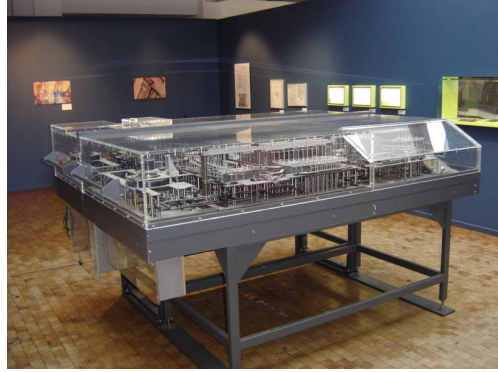
(على عكس محرك الفروق) حيث تكتب البرامج في بطاقات مثقبة ويتم قرائتها إلى المحرك بواسطة قارئاً لهذه البطاقات. وتحتوي هذه البطاقات على أوامر موجهة إلى المحرك لكي يقوم بقراءة عدد من المخزن ويجري عملية ما (جمع مثلاً) ومن ثم يحفظ النتيجة في المخزن أيضاً ، وكذلك تحتوي أوامر أخرى مثل المقارنة بين عددين والتفرع ونقل التنفيذ. ولأن المحرك قابل للبرمجة (بلغة شبيهة بلغة التجميع) فقد استعان شارلز باباج بالبرمجة آدا لوفلاس Ada Lovelace والتي صنفت كأول مبرمج في التاريخ. ولسوء الحظ لم ينجح شارلز باباج في أن يزيد من دقة المحرك ربما لأنه يحتاج إلى آلاف من التروس والعجلات. وبشكل أو بآخر يعتبر شارلز باباج الجد الأول للحواسيب الحالية حيث أن فكرة عمل المحرك التحليلي مشابهة للحواسيب الحالية.

شكل ٦.١: المحرك التحليلي بمتحف في لندن

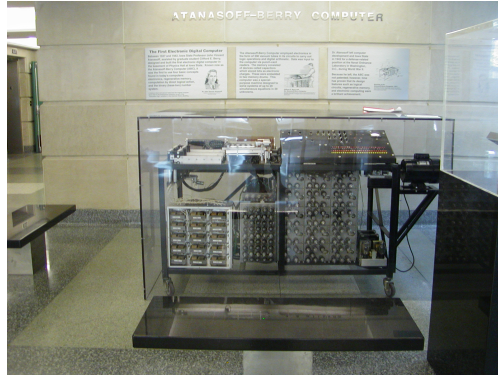


وفي أواخر ١٩٣٠ قام الطالب الألماني كونارت تسوزا Konrad Zuse ببناء آلة حسابية ولكنها تعتمد على الريلاي (Relay) وسميت بجهاز Z1 (انظر الشكل ٧.١) وتعتبر أول حاسبة تعتمد على الريلاي وعلى المنطق الثنائي في عملها. ولسوء الحظ تم تدمير الحاسبة Z1 في انفجار في برلين أثناء الحرب العالمية الثانية عام ١٩٤٣. وعلى الرغم من أن تصميم Z1 لم يؤثر في تصميم الحواسيب التي تليه بسبب تدميره هو وجميع خطط بنائه إلا أنه يعتبر أحد التصميمات التي كان لها أثرها ذاك الوقت. وبعد برهة من الزمن قام جون أتاناسوف John Vincent Atanasoff بتصميم جهاز Atanasoff (انظر الشكل ٩.١) والذي كان مدهشاً في وقته حيث يستخدم التحسب الثنائي (Binray Arithmetic) ويجوي مكثفات للذاكرة ولكن الجهاز لم يكن عملياً. وفي بدايات ١٩٤٠ قام هوارد ايكن Howard Aiken بتصميم الحاسبة ASCC - والتي أعيد تسميتها إلى Harvard Mark I - في جامعة هارفارد (انظر الشكل ٩.١). وقد أتبع هوارد نفس منهج باباج وقرر أن يعتمد على الريلاي وأن يصمم حاسبة للأغراض العامة والتي فشل بها شارلز باباج. وفي عام ١٩٤٤ تم الإنتهاء من تصميمها وتم تصميم نسخة محسنة أيضاً سميت ب Harvard Mark II. ومع هذا التصميم انتهى عصر الحاسبات الميكانيكية والريلاي وبدأ عصر جديد.

شكل ٧.١: حاسبة Z1 بعد إعادة انشائها في متحف بألمانيا



شكل ٨.١: حاسبة Atanasoff بعد إعادة انشائها في جامعة Iowa State

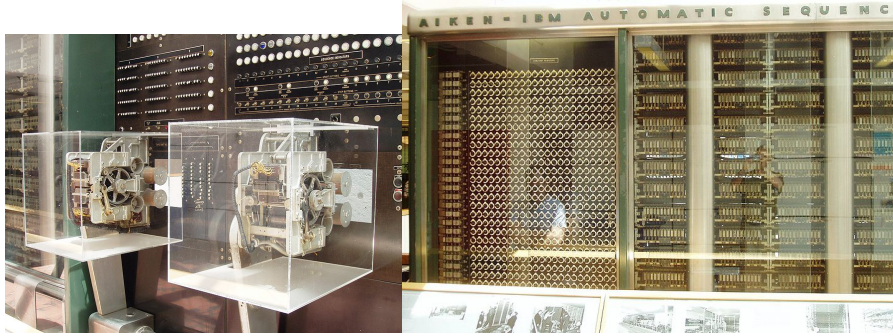


## ٢.٢.١. الجيل الأول (١٩٤٥-١٩٥٥): الصمامات المفرغة و لوحات التوصيل

في بدايات الحرب العالمية الثانية ، كان أمير البحرية في برلين يرسل رسائل الى الغواصات الألمانية عبر موجات الراديو والتي استطاع حلفاؤها البريطانيون التقاطها ، ولسوء حظهم كانت الرسائل ترسل مشفرة وذلك عن طريق شفرة خاصة تنتج من قبل جهاز يسمى بجهاز إنجما (Enigma Machine) والذي تم تصنيعه لتشفير الرسائل وفك تشفيرها . وقد تمكنت الإستخبارات البريطانية من الحصول على أحد هذه الأجهزة وذلك بعد الإتفاق مع الإستخبارات البولندية والتي كانت قد سرقت جهازاً من الألمان. وحتى يتمكن البريطانيون من فك شفرة الرسائل فان هذا يتطلب وقتاً وعمليات حسابية طويلة ، لذلك سرعان ما أسست معملًا سريًا يحوي على حاسبة الكترونية عرفت بالاسم Colossus. هذه الحاسبة تم تصميمها من قبل عدة أشخاص وشارك فيها العالم آلان تورنج وأصبحت جاهزة للعمل في عام ١٩٤٣. وبسبب



## ٢.١. تاريخ أنظمة التشغيل



(ب) الإدخال والإخراج والتحكم

(أ) الجزء الأيسر من حاسبة Mark I

شكل ٩.١: حاسبة Harvard Mark I

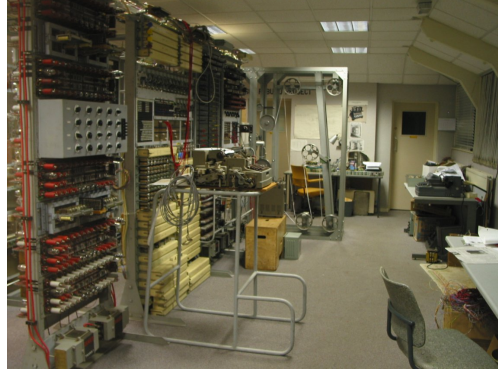
شكل ١٠.١: آلة إنجما الألمانية لتشفير الرسائل وفكها



كون المشروع سريا وتم التكتّم عليه لما يقارب ٣٠ عاما فان هذا النوع من الحواسيب لم يؤثر على تصاميم الحواسيب الحديثة ولكن يجدر بالذكر أن هذه الحاسبة تعتبر أول حاسبة إلكترونية قابلة للبرمجة تستخدم

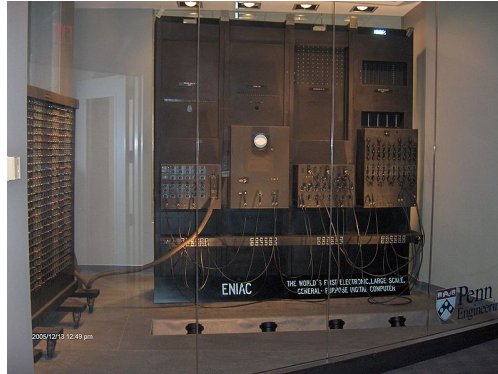
الصمامات الهوائية في حساباتها.

شكل ١١.١: الحاسبة colossus التي كسرت شفرة إنجما



وفي عام ١٩٤٣ قدم جون موكلي John Mauchley مقترحاً الى الجيش الأمريكي طالباً تمويله بالمال للبدء بتصميم حاسبة إلكترونية لحساب جداول إطلاق المدفعية بدلاً من حسابها يدوياً وذلك لتقليل الأخطاء وكسب الوقت، وقد تمت الموافقة على المشروع وبدأ جون موكلي وطالبه الخريج إيكريت ببناء حاسبة تم تسميتها بالاسم إيناك ENIAC اختصاراً للجملة Electronic Numerical Integrator And Computer. وتتكون من ١٨٠٠٠ صماماً مفرغاً (Vacuum Tubes) و ١٥٠٠ حاكمة (Relays)، وتزن الحاسبة ٣٠ طن وتستهلك ١٤٠ كيلو واط من الطاقة. وداخلها تحتوي الحاسبة على ٢٠ مسجل كل منهم يسع عدداً صحيحاً بطول ١٠ خانات. وتتم برجة إيناك عن طريق ٦٠٠٠ مفتاح Switch.

شكل ١٢.١: الحاسبة ENIAC

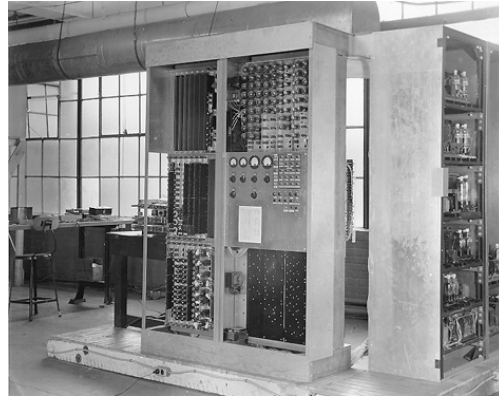


وقد تم الإنتهاء من تصميم إيناك عام ١٩٤٦ ، الوقت الذي كانت الحرب قد انتهت ولم تستخدم الحاسبة

## ٢.١. تاريخ أنظمة التشغيل

لهدفها الرئيسي. وبعد ذلك نظم جون موكلي وطالبه إيكريت مدرسة صيفية لوصف مشروعاتهم للباحثين والمهتمين. الأمر الذي أسفر عن ظهور عدد كبير من الحاسبات الضخمة. وأول حاسبة بعدها كانت EDSAC في عام ١٩٤٩ بواسطة ويلكس في جامعة كامبردج. كذلك تلتها الحاسبات JOHNIAC و ILLIAC وغيرهم. بعد ذلك بدأ جون موكلي وإيكريت بالعمل على حاسبة أخرى سميت بالاسم EDVAC اختصاراً للجملة Electronic Discrete Variable Automatic Computer والتي كانت تعمل بالأرقام الثنائية بدلاً من العشرية (كما في إيناك). بعد ذلك توقف المشروع بسبب أن جون وإيكريت قد أنشأ شركتهم الخاصة.

شكل ١٣.١: الحاسبة EDVAC



٣.٢.١. الجيل الثاني (١٩٥٥-١٩٦٥): الترانزستورات

٤.٢.١. الجيل الثالث (١٩٦٥-١٩٨٠): الدوائر المتكاملة

٥.٢.١. الجيل الرابع (من ١٩٨٠ حتى الآن): الحواسيب الشخصية



## ٢. معمارية حواسيب x86

حواسيب عائلة x86 تتبع لمعمارية العالم جون نويمان (John von Neumann architecture) والتي تنص على أن أي تصميم لجهاز حاسب يجب أن يتكون من الثلاث وحدات التالية :

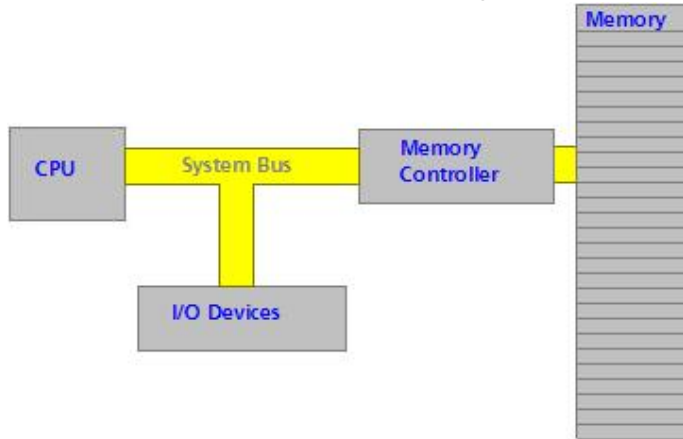
١. معالج أو وحدة معالجة مركزية (Central Processing Unit).

٢. ذاكرة (Memory).

٣. أجهزة إدخال وإخراج (I/O Devices).

الوحدة الاولى هي وحدة المعالجة والتي تقوم بتنفيذ الأوامر والعمليات الحسابية ، أما الوحدة الثانية فهي تحوي البيانات والتعليمات والأوامر التي يجب لوحدة المعالجة أن تنفذها ، وأخيراً وحدات الإدخال والإخراج وهي الاجهزة التي تستخدم في ادخال البيانات واخراجها.(انظر الشكل ١.٢ حيث يوضح مثلاً لهذه المعمارية) ويربط بين كل هذه الأجزاء هو مسار النظام (System Bus) وفيما يلي سنستعرض وظيفة كل جزء على حدة.

شكل ١.٢: معمارية حواسيب x86

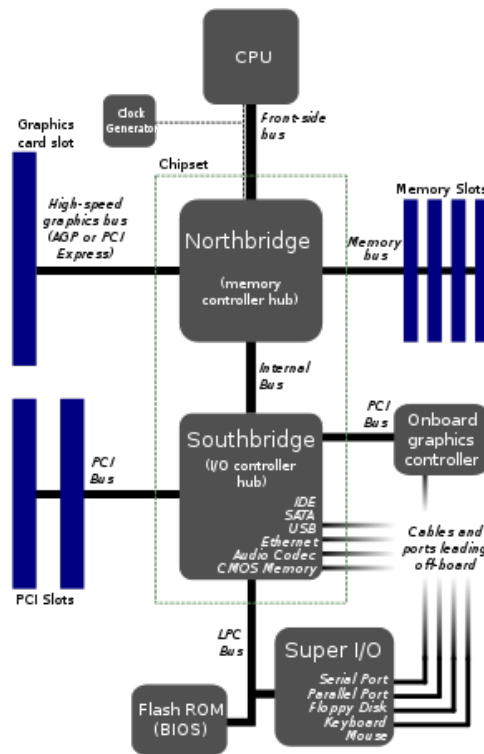


## ١.٢. معمارية النظام

## ١.١.٢. مسار النظام System Bus

يربط مسار النظام (System Bus) <sup>١</sup> وحدة المعالجة المركزية (CPU) مع متحكم الذاكرة الرئيسية . وظيفة هذه المسارات هي نقل البيانات بين أجزاء الحاسب المختلفة. والشكل ٢.٢ يوضح الصورة العامة للمسارات في أجهزة الحواسيب الشخصية (Personal Computers). ويتألف مسار النظام من ثلاث مسارات وهي مسار البيانات (Data Bus) ومسار العناوين (Address Bus) ومسار التحكم (Control Bus).

شكل ٢.٢: المسارات في الحواسيب الشخصية x86



<sup>١</sup> ويسمى أيضا Front-side Bus.

### مسار البيانات Data Bus

مسار البيانات هو عبارة عن خطوط (Lines) كل خط يمثل بت واحد. وغالباً ما يكون هناك 32 خط (أي أن مسار البيانات بطول 32-bit) ويستخدم هذا المسار في نقل البيانات (Data) من المعالج (وتحديداً من وحدة التحكم Control Unit) الى متحكم الذاكرة (الى الجسر الشمالي NorthBridge) تحديداً نظراً لأن متحكم الذاكرة يطبق على عليه). ويسبب أن حجم مسار البيانات هو حجم ثابت فان هذا يتطلب معالجة خاصة عند ارسال بيانات بطول أقل من طول مسار البيانات ، فغالباً ما يقوم المعالج باضافة أصفار في الخطوط الغير مستخدمة (Padding). أما في حالة إرسال بيانات بطول أكبر فان عملية نقلها تتم على عدة مراحل وفي كل مرحلة ترسل 32-bit من البيانات .

### مسار العناوين Address Bus

يستخدم مسار العناوين في نقل عنوان الذاكرة المراد استخدامه سواءاً للقراءة منه أو الكتابة عليه ، ويحدد حجم مسار العناوين أكبر عنوان يمكن الوصول اليه في الذاكرة وبالتالي يحدد لنا حجم الذاكرة التي يستطيع الحاسب التعامل معها . وفي الأجهزة التي تستخدم معالجات انتل 8086 كان حجم هذا المسار هو 20-bit وبالتالي فان أقصى ذاكرة يتعامل معها هذا المعالج هي 1 MB<sup>2</sup> أما في معالجات 80286/80386 فان حجم هذا مسار هو 24-bit وفي المعالجات التي تليها تم زيادة هذا الحجم الى 32-bit وبالتالي يمكن تنصيب ذاكرة بحجم 4 GB ، وفي المعالجات الحديثة تم زيادة هذا الحجم ، ولكننا سنقتصر في هذا البحث على المعالجات التي تدعم مسار عناوين بطول 32-bit بسبب انتشارها وسيطرتها لمدة من الزمن على أجهزة الحواسيب الشخصية.

### مسار التحكم Control Bus

يستخدم مسار التحكم في ارسال الأوامر مثل أمر القراءة من العنوان الموجود على مسار العناوين أو أمر الكتابة على العنوان المطلوب . ويتألف هذا المسار من عدد من الخطوط وكل خط (بت) يؤدي وظيفة محددة. أحد هذه الخطوط هو خط الكتابة WRITE والذي يعني أن العنوان الموجود على خط العناوين يجب أن تُعَيَّن له القيمة الموجودة في مسار البيانات . الخط الآخر هو خط القراءة READ والذي يدل على أن العنوان الموجود في مسار العناوين يجب أن تُقرأ قيمته الى مسار البيانات . آخر خط يهمنا هو خط الولوج ACCESS والذي يحدد ما اذا كان العنوان موجه الى متحكم الذاكرة أم الى متحكم الإدخال والإخراج وفي حالة كانت قيمة هذا الخط هي القيمة 1 فان هذا يعني أن العنوان موجه الى متحكم أجهزة الإدخال والإخراج وبالتالي سيتم القراءة من هذا العنوان أو الكتابة اليه وذلك بحسب قيمة الخطين READ and WRITE.

<sup>2</sup> ناتجة من حساب 2 مرفوع للقوة 20.

## ٢.١.٢. متحكم الذاكرة

قبل أن نذكر وظيفة هذا المتحكم يجب إعطاء نبذة عن ماهية المتحكمات (Controllars) في جهاز الحاسب. ويُعرف **المتحكم** بأنه شريحة تتحكم بعتاد ما تحوي العديد من المسجلات الداخلية وظيفتها هو استقبال الأوامر وتنفيذها على العتاد. ويمكن أن نعرفها بأنها شريحة للربط ما بين الأوامر البرمجية الى أوامر تنفذ على عتاد ما. وأي متحكم يحوي العديد من المسجلات سواءاً كانت لإرسال واستقبال البيانات أو للأوامر ، وأي مسجل يجب أن يأخذ رقم فريد يميزه عن بقية المسجلات الموجودة في هذا المتحكم أو في أي متحكم آخر وذلك حتى تتمكن من التعامل معه برمجياً ، هذا الرقم يعرف باسم المنفذ (Port) وسنطلع عليه لاحقاً. وعمل المتحكم يبدأ عندما يُرسل أمر اليه حيث يبدأ المتحكم في تنفيذ هذا الأمر ومن ثم يضع النتيجة في أحد مسجلاته ويرسل إشارة (Interrupt) الى المعالج لكي يقوم بقرائة القيمة.

نعود الى متحكم الذاكرة الرئيسية والذي يتواجد غالباً على متحكم الجسر الشمالي (NorthBridge) إنظر الشكل ٣.٢. حيث تكمن وظيفته الأساسية في استقبال الأوامر المرسلة الى الذاكرة وتنفيذها ، ويقوم هذا المتحكم بتوجيه العناوين المرسلة الى أي من شرائح الذاكرة كذلك يقوم بإعادة تنعيش (Refresh) هذه الذاكرة طيلة عمل الحاسب حتى لا تفقد الذاكرة محتوياتها.

شكل ٣.٢: الجسر الشمالي

يعتبر هذا الجسر حلقة الوصل ما بين المعالج والذاكرة الرئيسية والبايوس وذاكرة الفيديو ومتحكم الإدخال والإخراج حيث يستقبل الأوامر ويقوم بتوجيهها الى المتحكم المطلوب.



## ٣.١.٢. متحكم الإدخال والإخراج

يستخدم متحكم الإدخال والإخراج (ويسمى أيضا الجسر الجنوبي SouthBridge) في ربط متحكمات أجهزة الإدخال والإخراج مع المعالج وهذا يتضح من الشكل ٢.٢. حيث يظهر أن الجسر الشمالي يرتبط مباشرة مع المعالج بينما الجسر الجنوبي يرتبط مع الجسر الشمالي والذي بدوره يربط متحكمات عتاد الإدخال والإخراج في الحاسب. وكل جهاز يرتبط بالحاسب (مثل لوحة المفاتيح أو الفأرة أو الطابعة ...الخ) لديه متحكم بداخل الجهاز ومتحكم آخر بداخل الحاسب ، حيث يرسل المتحكم الموجود بداخل الحاسب الأوامر الى المتحكم الموجود بداخل العتاد . ولبرمجة أي جهاز فانه يجب برمجة المتحكم الموجود



في الحاسب وهذا يتم عن طريق معرفة المسجلات (Registers) الموجودة به ووظيفة كل مسجل فيه حتى تتمكن من إرسال الأوامر الصحيحة اليه. هذه المسجلات تأخذ أرقاماً معينة تسمى منافذ برمجية (Software Ports) بحيث تميز هذه الأرقام المسجلات من بعضها البعض<sup>٣</sup>.

### المنافذ Ports

يستخدم مفهوم المنافذ في علوم الحاسب للدلالة على عدة أشياء فمثلاً في مجال برمجة الشبكات تكون برامج الخادم لها رقم منفذ معين حتى تسمح لبرامج العميل بالاتصال معها، كذلك توجد المنافذ الموجودة في اللوحة الأم لوصول عتاد الحاسب بها ، أيضاً أي مسجل في متحكم على الجهاز لديه رقم منفذ وهذا ما نقصده في حديثنا عن المنافذ في هذا البحث. ويمكن الوصول لمنافذ المتحكمات والتي تعرف ب I/O ports باستخدام تعليمة المعالج in port\_address والتعليمة out port\_address حيث تستخدم الأولى لقراءة قيمة من مسجل في متحكم ووضعها في أحد مسجلات المعالج أما التعليمة الثانية تستخدم لكتابة قيمة في مسجل للمعالج إلى مسجل في المتحكم . وعند استخدام أحد هذين الأمرين فإن ذلك يعني أن العنوان موجه إلى متحكم الإدخال والإخراج وليس إلى متحكم الذاكرة حيث يقوم المعالج بتعيين قيمة الخط ACCESS الموجود في مسار التحكم (Control Bus) وبالتالي يستجيب متحكم الإدخال والإخراج ويقرأ هذا العنوان ويقوم بتوجيهه إلى المتحكم المطلوب . وهناك بعض الأجهزة تستخدم عناوين الذاكرة للوصول للمتحكم الخاص بها وهو ما يعرف ب Memory Mapped I/O حيث عند كتابة أي بيانات على هذه العناوين فإن ذلك يعني كتابة هذه البيانات على متحكمات للأجهزة وليس على الذاكرة الرئيسية. فمثلاً عند الكتابة على عنوان الذاكرة 0xa000:0x0 فإن هذا يؤدي إلى الكتابة على شاشة الحاسب نظراً لأن هذا العنوان هو موجه (Memory Mapped) مع متحكم شاشة الحاسب والجدول ١.٢ يوضح خريطة الذاكرة في حواسيب x86، ولا تحتاج الكتابة لمثل هذه العناوين استخدام الأوامر in/out بعكس الكتابة في عناوين المنافذ I/O port .

عناوين منافذ الإدخال والإخراج (Port I/O) هي عناوين تستخدمها المسجلات الموجودة على المتحكمات ويقوم البايوس بمهمة ترقيم هذه المسجلات ، والجدول ٢.٢ يعرض قائمة بعناوين المنافذ ووظيفة كل منهم.

### ٢.٢. المعالج

يعتبر المعالج هو المحرك الرئيسي لجهاز الحاسب حيث يستقبل الأوامر ويقوم بتنفيذها .

<sup>٣</sup>هناك بعض المسجلات لبعض المتحكمات تأخذ نفس الرقم ، لكن طبيعة الأمر المرسل (قراءة أو كتابة) هو الذي يحدد المسجل الذي يجب التعامل معه.

جدول ١.٢: مخطط الذاكرة لحواسيب x86

عنوان البداية	عنوان النهاية	الوصف
0x00000	0x003ff	جدول المقاطعات IVT
0x00400	0x004ff	منطقة بيانات البايوس
0x00500	0x07bff	غير مستخدمة
0x07c00	0x07dff	برنامج محمل النظام
0x07e00	0x9ffff	غير مستخدمة
0xa0000	0xfffff	ذاكرة الفيديو Video RAM
0xb0000	0xb7777	ذاكرة الفيديو أحادية اللون Monochrome VRAM
0xb8000	0xbffff	ذاكرة الفيديو الملونة Color VRAM
0xc0000	0xc7fff	ذاكرة BIOS Video ROM
0xc8000	0xfffff	منطقة BIOS Shadow Area
0xf0000	0xfffff	نظام البايوس

## ١.٢.٢. دورة تنفيذ التعليمات

لكي يُنفذ المعالج البرامج الموجودة على الذاكرة فإن هذا يتطلب بعضاً من الخطوات التي يجب أن يقوم بها ، وفي كل دقة للساعة (Clock tick) يقوم المعالج بالبدء بخطوة من هذه الخطوات ، وفيما يلي سرداً لها .

أولاً مرحلة جلب البيانات (Fetch) وفيها يتم جلب البيانات من الذاكرة الرئيسية الى المسجلات بداخل المعالج.

ثانياً مرحلة تفسير البيانات (Decode).

ثالثاً مرحلة تنفيذ البيانات (Execute).

رابعاً مرحلة حفظ النتائج (Write back).

## ٢.٢.٢. أنماط عمل المعالج CPU Modes

عندما طرحت شركة إنتل أول اصداراً من معالجات 16-bit لم يكن هناك ما يعرف بأنماط المعالج حيث كان المعالج يعمل بنمط واحد وهو ما يعرف الآن بالنمط الحقيقي (Real Mode) ، في هذا النمط يقوم المعالج بتنفيذ أي أمر موجه اليه ولا يوجد ما يُعرف بصلاحيات التنفيذ حيث يمكن لبرنامج المستخدم أي يقوم بتنفيذ أمر يتسبب في ايقاف النظام عن العمل (مثل الأمر hlt) ، كذلك توجد عددٌ من المشاكل في هذا النمط فمثلاً لا توجد حماية للذاكرة من برمجيات المستخدم ولا يوجد أي دعم لمفهوم تعدد المهام (Multitasking). لذلك سارعت إنتل بادخال عدة أنماط على بنية المعالج لتحل هذه المشاكل ، بحيث

## جدول ٢.٢: منافذ الإدخال والإخراج لحواسيب x86

الاستخدام	رقم المنفذ
Slave DMA controller	0000-000f
System	0010-001F
First Interrupt controller (8259 chip)	0020-0021
Second interrupt controller	0030-0031
Programable Interval Timer 1 (8254 chip)	0040-0043
Programable Interval Timer 2	0048-004B
System devices	0050-006F
NMI Enable / Real Time Clock	0070-0071
DMA Page registers	0080-008B
System devices	0090-009F
Slave interrupt controller	00A0-00A1
Master DMA controller	00C0-00DE
System devices	00F0-00FF
System devices	0100-0167
IDE Interface - Quaternary channel	0168-016F
IDE interface - Secondary channel	0170-0177
IDE Interface - Tertiary channel	01E8-01EF
IDE interface - Primary channel	01F0-01F7
Games Port (joystick port)	0200-0207
Usually used by sound cards, also used by NOVEL NETWARE KEY CARD	0220-022F
Plug and Play hardware	0270-0273
Parallel Port *	0278-027A
Sometimes used for LCD Display I/O	0280-028F
Alternate VGA Video Display Adaptor assignment (secondary address)	02B0-02DF
GPIB 0, data aquisition card 0 (02E1 to 02E3 only)	02E0-02E7
Serial Port - COM 4	02E8-02EF
Serial Port - COM 2	02F8-02FF
Often used as a default for Network Interface cards (was prototype card)	0300-031F
ST506 and ESDI Hard Disk Drive Interface (mostly used in PX/XT and early PC/AT)	0320-023F
MPU-401 (midi) interface, on Sound Cards	0330-0331
Sometimes used for Network Interface cards	0360-036F
Another address used by the Secondary IDE Controller (see 0170-0177)	0376-0377
Parallel Port *	0378-037A
FM (sound) synthesis port on sound cards	0388-038B
MDA, EGA and VGA Video Display Adaptor (only 03B0 to 03BB used)	03B0-03BB
Parallel Port (originally only fitted to IBM mono display adaptors) *	03BC-03BF
EGA / VGA Video Display Adaptor, (Primary address)	03C0-03DF
PCIC PCMCIA Port Controller	03E0-03E7
Serial Port - COM 3	03E8-03EF
Floppy Disk Drive Interface	03F0-03F6
Another address used by the Primary IDE Controller (see 01F0-01F7)	03F7-03f7
Serial Port - COM 1	03F8-03FF
Windows sound system (used by many sound cards)	0533-0537

يُمكن للمعالج أي يعمل في أي نمط وأن يقوم بالتحويل وقتما شاء. ويُعرّف **نمط المعالج** بأنه طريقة معينة يتبعها المعالج أثناء عمله لتنفيذ الأوامر فمثلاً يحدد النمط المستخدم ما إذا كان هناك حماية لعنوانين الذاكرة بحيث لا يمكن لبرنامج لا يمتلك صلاحيات معينة الوصول لأي منطقة في الذاكرة.

### ٣.٢.٢. النمط الحقيقي Real Mode

هذا النمط هو الذي يبدأ الجهاز الحاسب بالعمل عندما يقلع وهذا بسبب أن حواسيب x86 تم تصميمها بحيث تدعم الأجهزة القديمة وحتى تحافظ انتل على ذلك فإن هذا ما جعلها تدع المعالج يبدأ بالنمط الحقيقي عند الإقلاع توافقاً مع الحواسيب القديمة ، وبعد ذلك عندما يستلم نظام التشغيل زمام التحكم بالحاسب فإنه مخير ما بين الإستمرار بالعمل في هذا النمط وبالتالي يسمى هذا النظام **نظام تشغيل 16-bit** وبين تحويل نمط المعالج الى النمط الآخر وهو النمط المحمي (Protected Mode) وبالتالي يسمى النظام **نظام تشغيل 32-bit**. في هذا النمط يستخدم المعالج مسجلات من طول 16-bit (مثلاً المسجلات ax,bx,cx,dx,...etc) ويستخدم عنوانة **المقطع:الإزاحة (Segment:Offset)** للوصول الى الذاكرة الرئيسية - سيتم شرحها في الفقرة التالية- وأيضاً يدعم ذاكرة بحجم 1 ميجابايت ولا يقدم أي دعم لحماية الذاكرة والذاكرة التخيلية (Virtual Memory) ولا يوفر حماية للذاكرة من برمجيات المستخدم.

#### عنوانة المقطع:الإزاحة (Segment:Offset Addressing)

بعد طرح أنتل لمعالج 8086 وهو أول معالج ١٦ بت ، ظهرت مشكلة حجم الذاكرة حيث أن طول المسجلات المستخدمة في هذا المعالج (مسجلات البيانات والعناوين) هو ١٦ بت وهذا ما سمح للمسجل بأن يتعامل مع ٦٤ كيلوبايت فقط من الذاكرة على الرغم من أن مسار العناوين (Address Bus) في هذه الأجهزة كان بحجم ٢٠ بت وهو ما يسمح باستخدام ذاكرة بحجم ١ ميجا. الى هنا كان الخيار أمام شركة أنتل هو بزيادة حجم المسجلات الموجودة بداخل المعالج ولكن هذا الحل كان مكلفاً جداً آنذاك نظراً لأن هذه المسجلات هي ذواكر من النوع SRAM وهو نوع مكلفاً على الرغم من إمكانياته العالية. ما فعلته انتل هو إيجاد طريقة مختلفة لعنوانة الذاكرة فبدلاً من استخدام مسجل واحد للوصول الى عناوين الذاكرة تم استخدام مسجلين كل منهما بطول ١٦ بت ، الفكرة كانت في تقسيم الذاكرة الى مقاطع (Segments) ويُستخدم أحد المسجلات للدلالة على رقم أو عنوان المقطع (Segment Number or Address) وبالتالي هناك ٦٥٥٣٦ مقطع مختلف؛ ويُستخدم المسجل الآخر للوصول الى العناوين بداخل المقطع وهي ما تعرف بالقيم (Offsets) بداخل المقطع وبالتالي كل مقطع يحوي ٦٥٥٣٦ بايت (أي أن حجم المقطع هو ٦٤ كيلوبايت). إذا يُعرّف **المقطع Segments** بأنها منطقة من الذاكرة بحجم ٦٤ كيلوبايت ويمكن الوصول الى أي مقطع وذلك بتحميل رقم المقطع أو عنوان المقطع الى أي من مسجلات

٤ هذا ناتج من حساب  $2^{16}$ .

المقاطع الموجودة بداخل المعالج (مثل المسجلات CS, SS, DS, ES) - سيتم شرحها لاحقاً - ، ويمكن الوصول الى محتويات المقطع **الإزاحة Offset** وذلك بتحميل العنوان المطلوب الوصول اليه الى أي من مسجلات القيم (تبدأ العناوين في أي مقطع من العنوان 0x0 الى 0xffff). هذه الطريقة التي اقترحتها انتل للوصول الى عناوين الذاكرة خلقت لنا مفهوم العنوان المنطقي (Logical Address) حيث لكي نصل الى أي مكان في الذاكرة فانه يجب تحديد عنوان المقطع والعنوان بداخل هذا المقطع وذلك على الشكل Segment:Offset حيث الجزء الأول يحدد عنوان المقطع والجزء الثاني يحدد العنوان بداخل المقطع. مهمة المعالج حالياً هي تحويل العنوان المنطقي الى عنوان فيزيائي أو حقيقي لكي يقوم بارساله عبر مسار العناوين الى متحكم الذاكرة ، و طريقة التحويل تعتمد على أن الإزاحة (Offset) يتم جمعها الى عنوان المقطع (Segment) ° ولكن بعد أن يتم ضربها في العدد ١٦ وذلك بسبب أن أي مقطع يبدأ بعد ١٦ بايت من المقطع السابق له . والتحويل يتم كالآتي :

$$physical\_address = segment * 0x10 + offset$$

فمثلاً العنوان المنطقي 0x07c0:0x0000 يتم تحويله وذلك بضرب العنوان 0x07c0 بالعدد ١٦ (أو العدد 0x10 بالنظام السادس عشر) ليصبح هكذا 0x07c00، وبعد ذلك يتم جمعه الى ال Offset ليخرج العنوان الفيزيائي 0x07c00.

### مشكلة تداخل المقاطع

ذكرنا في الفقرة السابقة أن أي مقطع يبدأ مباشرة بعد ١٦ بايت من المقطع السابق له ، وهذا يعني أن المقاطع متداخلة حيث يمكن الوصول لعنوان فيزيائي معين بأكثر من طريقة مختلفة. مثلاً في مثالنا السابق استخدمنا العنوان المنطقي 0x07c0:0x0000 للوصول الى المنطقة الذاكرة 0x07c00 ، ويمكن أن نستبدل العنوان المنطقي السابق بالعنوان 0x0000:0x7c00 وبعد اجراء التحويل سنحصل على نفس العنوان الفيزيائي 0x07c00. وفي الحقيقة هناك ٤٠٩٦ طريقة مختلفة للوصول لعنوان في الذاكرة ٦ والشكل ٤.٢ يوضح لنا تداخل هذه المقاطع.

هذا التداخل **Overlapping** سمح لأي برنامج ما إمكانية الوصول الى بيانات برنامج آخر والكتابة عليها وهذا ما جعل النمط الحقيقي ضعيف من ناحية حماية أجزاء الذاكرة.

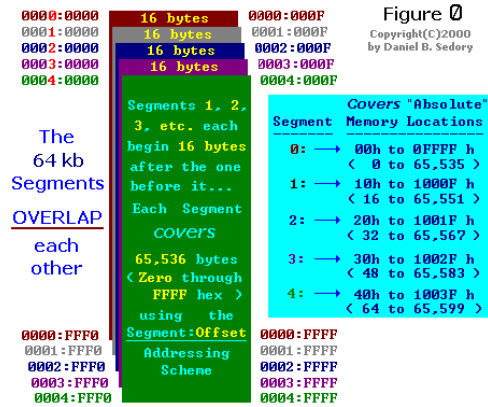
### ٤.٢.٢ . النمط المحمي Protected Mode

بعد أن تم التعرف على هذه المشاكل سارعت أنتل باصدار المعالج 80286 والذي كان أول معالج يعمل في نمطين (الحقيقي والمحمي) . هذا المعالج (والمعالجات التي تليها) حل أهم مشكلة وهي حماية مقاطع الذاكرة

° بحيث نعتبر عنوان المقطع هو عنوان بداية (Base Address) لعناوين القيم (Offset).

٦ انظر الى مقالة الكاتب Daniel B. Sedory على الرابط <http://mirror.href.com/thestarman/asm/debug/Segments.html>

شكل ٤.٢: تداخل المقاطع في النمط الحقيقي



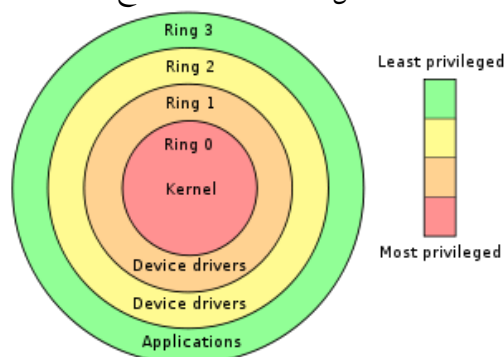
من الوصول العشوائي من قبل برامج المستخدم وذلك عن طريق وصف مقاطع الذاكرة وصلاحيات الوصول إليها في جداول تسمى جداول الوصفات (Descriptor Table). المعالج 80386 هو أول معالج ٣٢ بت يستخدم مسجلات بحجم ٣٢ بت وحجم مسار البيانات أيضا بنفس الحجم مما سمح بإمكانية التعامل مع ذاكرة بحجم ٤ جيجابايت. كذلك تم إضافة دعم للذاكرة التخيلية ومفهوم الصفحات (Paging) ودعم تعدد المهام. وفي هذا البحث سيتم الحديث عن معالجات ٣٢ بت باعتبارها أحد الأكثر انتشاراً حتى وقتنا هذا، وعلى الرغم من ظهور معالجات ٦٤ بت إلا أن الدراسة حول معالجات ٣٢ بت تعتبر هي الأساس نظراً لأن المعالجات الحديثة ما هي الا تطوير و إضافات للمفاهيم الموجودة على المعالجات السابقة.

### حلقات المعالج CPU Rings

عندما يعمل المعالج في النمط المحمي فإن هذا يضمن حماية للذاكرة من برمجيات المستخدم ، وهذا بسبب توصيف الذاكرة وصلاحيات الوصول لها في جدول يستخدمه المعالج لعنونة الذاكرة وهو جدول الواصفات. نظام الصلاحيات الذي تم ادخاله الى المعالج عند عمله في النمط المحمي يسمى **بحلقات المعالج (CPU Rings)**، هذه الحلقات تحدد مستوى الحماية المطلوب لكي يستخدمها المعالج في تقرير ما اذا كان تنفيذ أمر ما يحتاج الى صلاحية أعلى أم لا، وكذلك لكي يقرر ما اذا كان الوصول الى عنوان معين في الذاكرة مسموح باستخدام صلاحية معينة أم لا. وتوجد أربع حلقات للمعالج تبدأ من الحلقة صفر (Ring0) وتنتهي بالحلقة ٣ (Ring3). الحلقة صفر تسمى نمط النواة (Kernel Mode) بسبب أن أي برنامج يعمل في الحلقة صفر لديه الصلاحيات الكاملة على النظام بالوصول الى أي عنوان في الذاكرة وتنفيذ أي تعليمية حتى لو تسببت في إيقاف النظام عن العمل (المسؤولية تقع على البرنامج) لذلك غالباً البرامج التي تعمل في الحلقة صفر هي البرامج التي تتبع لنظام التشغيل. أما الحلقة ٣ تسمى بنمط المستخدم (User Mode)

حيث أن البرامج التي تعمل عليها لا تملك صلاحيات لتنفيذ العديد من الأوامر (مثل الامر `cli` والأمر `hlt`) ولا تملك الوصول إلى أي عنوان في الذاكرة بخلاف مساحة العنوان التخيلية (`Virtual Address Space`) الخاصة بالبرنامج نفسه وهذا ما رفع درجة حماية الذاكرة إلى أقصى حد ممكن ، والشكل ٥.٢ يوضح هذه الحلقات وصلاحياتها. وعندما يبدأ النظام بالإقلاع فإن المعالج يكون في النمط الحقيقي وهو نمط لا يحوي على حلقات حيث أنه يمكن تنفيذ كل الأوامر والوصول إلى أي عنوان في الذاكرة ، وعند التحويل إلى النمط المحمي (`PMODE`) فإن المعالج يكون في الحلقة صفر (`Kernel Mode`) ، ويتم تحويل الحلقة إلى حلقة معينة تلقائياً عند نقل التنفيذ إلى عنوان في الذاكرة موصوف في جدول الواصفات بأنه يعمل بتلك الحلقة.

شكل ٥.٢: حلقات المعالج



## ٥.٢.٢. النمط الغير حقيقي والنمط التخيلي

### ٦.٢.٢. معمارية معالجات x86

أي معالج يتعرف على مجموعة من الأوامر تسمى `Instruction Set` بعضاً منها تتطلب صلاحية معينة (الحلقة صفر) لكي يقوم المعالج بتنفيذها (انظر الجدول ٣.٢ لمعرفة هذه الأوامر) وإلا فإن هذا سيتسبب في حدوث خطأ من المعالج يسمى الخطأ العام (`General Protection Fault`) والذي ان لم تتوفر دالة تتعامل معه (`Exception Handler`) فإن هذا يؤدي إلى توقف النظام عن العمل. وتحتوي معالجات x86 العديد من المسجلات منها ما يستخدم للأغراض العامة (`General Registers`) ومنها ما يستخدم لحفظ العناوين وأرقام المقاطع (`Segments Registers`) وتوجد أيضاً مسجلات لا يمكن استخدامها إلا في برامج الحلقة صفر (أي النواة) حيث أن التغيير فيها يؤثر على عمل النظام وأخيراً هناك مجموعة من المسجلات الداخلية للمعالج والتي لا يمكن الوصول لها برمجياً. والقائمة التالية توضح هذه المسجلات :

جدول ٣.٢: الأوامر التي تتطلب صلاحية الحلقة صفر  
تنفيذ هذه الأوامر من قبل برمجيات المستخدم يؤدي الى حدوث خطأ وتوقف النظام عن العمل في حالة لم تتوفر دالة تتعامل مع هذا الخطأ.

الوصف	الأمر
تحميل جدول الواصفات العام الى المسجل GDTR	LGDT
تحميل جدول الواصفات الخاص الى المسجل LDTR	LLDT
تحميل مسجل المهام LTR	LTR
نقل بيانات الى مسجل تحكم MOV cr_x	MOV cr_x
تحميل new Machine Status WORD LMSW	LMSW
نقل بيانات الى مسجل تنقيح MOV dr_x	MOV dr_x
تصغير Task Switch Flag في مسجل التحكم الأول CLTS	CLTS
Invalidate Cache without writeback INVD	INVD
Invalidate TLB Entry INVLPG	INVLPG
Invalidate Cache with writeback WBINVD	WBINVD
إيقاف عمل المعالج HLT	HLT
قراءة مسجل MSR RDMSR	RDMSR
الكتابة الى مسجل MSR WRMSR	WRMSR
قراءة Performance Monitoring Counter RDPMSR	RDPMSR
قراءة time Stamp Counter RDTSC	RDTSC



- مسجلات عامة : RAX (EAX(AH/AL)), RBX (EBX(BH/BL)), RCX (ECX(CH/CL)), RDX (EDX(DH/DL)).
- مسجلات عناوين:
- مسجلات مقاطع: CS, SS, ES, DS, FS, GS.
- مسجلات إزاحة : RSI (ESI (SI)), RDI (EDI (DI)), RBP (EBP (BP)), RSP (ESP (SP)), RIP (EIP (IP)).
- مسجل الأعلام : RFLAGS (EFLAGS (FLAGS)).
- مسجلات التنقيح : DR0, DR1, DR2, DR3, DR4, DR5, DR6, DR7.
- مسجلات التحكم : CR0, CR1, CR2, CR3, CR4, CR8.
- مسجلات الاختبار : TR1, TR2, TR3, TR4, TR5, TR6, TR7.
- مسجلات أخرى : mm0, mm1, mm2, mm3, mm4, mm5, mm6, mm7, xmm0, xmm1, xmm2, xmm3, xmm4, xmm5, xmm6, xmm7, GDTR, LDTR, IDTR, MSR, and TR.

#### المسجلات العامة General Purpose Registers

في المعالجات ٣٢ بت يوجد ٤ أربع مسجلات عامة طول كل منها هو ٣٢ بت (٤ بايت) وتقسم أي من هذه المسجلات الى جزئين: الجزء الأعلى (High Order Word) وهو بطول ١٦ بت والجزء الأدنى (Low Order Word) وهو أيضا بطول ١٦ بت ، كذلك يُقسم الجزء الأدنى الى جزئين: الجزء الأعلى (High Order Byte) وهو بطول ٨ بت والجزء الأدنى (Low Order Byte) وهو أيضا بطول ٨ بت. على سبيل المثال مسجل EAX حيث يقسم الى جزء أعلى (لا يمكن الوصول اليه بشكل مباشر) وجزء أسفل وهو AX الذي يُقسم أيضا الى قسمين AH و AL. كل مسجل من هذه المسجلات العامة يستخدم لأي شيء لكن هناك بعض الاستخدامات الغالبة لكل منهم توضحها القائمة التالية.

- المسجل EAX: يستخدم لنقل البيانات والعمليات الحسابية.
- المسجل EBX: يستخدم في الوصول للذاكرة بشكل غير مباشر وذلك باستخدام مسجل آخر يعمل كعنوان رئيسي Base Address.
- المسجل ECX: يستخدم في عمليات التكرار والعد.
- المسجل EDX: يستخدم في تخزين البيانات.

### مسجلات المقاطع Segment Registers

مسجلات المقاطع تستخدم لتخزين أرقام وعناوين المقاطع (Segments) وتوجد ٦ مسجلات مقاطع تستخدم في النمط الحقيقي كما يلي:

- المسجل CS: يحوي عنوان بداية مقطع الشفرة للبرنامج المراد تنفيذه.
- المسجل DS: يحوي عنوان بداية مقطع البيانات للبرنامج المراد تنفيذه.
- المسجل SS: يحوي عنوان بداية مقطع المكس للبرنامج المراد تنفيذه.
- المسجل ES: يحوي عنوان بداية مقطع البيانات للبرنامج المراد تنفيذه.
- المسجل FS: يحوي عنوان مقطع بعيد.
- المسجل GS: يستخدم للأغراض العامة.

أما في النمط المحمي (PMode) فإن هذه المسجلات لا تشير إلى مقاطع البرامج والبيانات وإنما تشير إلى واصفات معينة في جدول الواصفات العام ، هذه الواصفات تحدد عنوان بداية المقطع ونوع المقطع (يحوي شفرات أم بيانات ) وتحدد صلاحية التنفيذ وصلاحية القراءة والكتابة فيها - كما سنرى ذلك في الفصل الرابع بإذن الله-.

### مسجلات الإزاحة Offset Registers

بجانب مسجلات المقاطع فإن الوصول إلى الذاكرة في النمط الحقيقي يتطلب عنوان الإزاحة بداخل المقطع ، وتوجد ٤ مسجلات إزاحة في معالجات x86 حجم كل منها هو ٣٢ بت في الأنظمة ٣٢ بت و ١٦ بت في أنظمة ١٦ بت. والقائمة التالية توضح هذه المسجلات:

- المسجل SI: يحوي عنوان الإزاحة في مقطع البيانات.
- المسجل DI: نفس الوظيفة السابقة.
- المسجل BP: يحوي عنوان الإزاحة بداخل مقطع المكس ويمكن استخدام للإشارة على أي عنوان في أي مقطع آخر.
- المسجل SP: يحوي عنوان الإزاحة بداخل مقطع المكس.

### مؤشر التعليمات Instruction Pointer

هذا المسجل (IP) يمثل إزاحة بداخل مقطع الشفرة (CS) وهو يحوي عنوان التعليمات التالية التي سيقوم المعالج بتنفيذها ، والعنوان CS:IP يمثل العنوان الفيزيائي للتعليمات التالية. هذا المسجل هو بطول ٣٢ بت (EIP) في أنظمة ٣٢ بت و ١٦ بت (IP) في أنظمة ١٦ بت، وهو مسجل لا يمكن تغيير محتواه باستخدام تعليمات المعالج MOV وإنما يتم تغيير محتواه عن القفز الى مكان آخر للتنفيذ.

### مسجل الأعلام FLAGS Register

مسجل الأعلام هو مسجل بحجم ٣٢ بت (EFLAGS) في أنظمة ٣٢ بت و بحجم ١٦ بت (FLAGS) في أنظمة ١٦ بت ، وهذا المسجل هو عبارة عن بتات (بالحجم السابق ذكره) كل بت لديه وظيفة محددة ، وينقسم بشكل عام الى بتات حالة (Status) بحيث تعكس حالة الأوامر التي يقوم المعالج بتنفيذها و بتات تحكم (Control) بحيث تتحكم في بعض الخصائص و بتات للنظام (System). والجدول ٤.٢ يوضح وظيفة كل بت في هذا المسجل.

ويحدد البتين IOPL مستوى الحماية المطلوب لتنفيذ مجموعة من الأوامر (مثل الأوامر CLI,STI,IN,OUT) حيث لن يتم تنفيذ مثل هذه التعليمات إلا في حالة كان مستوى الحماية الحالي Current Privilege Level أعلى من أو مساوياً للقيمة الموجودة في البتين IOPL<sup>٧</sup> ، وغالباً ما تكون القيمة هي صفر دلالة على أن التعليمات السابقة لا يتم تنفيذها الا لبرامج النواة (Ring0).

### مسجلات التحكم Control Registers

توجد في معالجات ٣٢ بت ستة مسجلات للتحكم في سلوك وعمل المعالج وهي CRO, CR1, CR2, CR3, CR4, CR8 ، ونظراً لخطورة التعامل معها فان هذه المسجلات لا يمكن الوصول لها إلا عند العمل في نمط النواة (Kernel Moder/Ring0) ولا يُمكن لبرمجيات المستخدم الوصول الى هذه المسجلات والتعامل معها. وفي الوقت الحالي يهملنا فقط أول مسجل تحكم وهو CRO حيث من خلاله يمكن أو نقوم بعملية تحويل نمط المعالج من النمط الحقيقي الى النمط المحمي (PMode) وكذلك يمكن أن نقوم بتفعيل خاصية الصفحات (Paging) ، والتركيبية التالية توضح محتويات كل بت في مسجل التحكم CRO وهو مسجل بحجم ٣٢ بت.

- Bit 0 (PE) : Puts the system into protected mode.
- Bit 1 (MP) : Monitor Coprocessor Flag This controls the operation of the WAIT instruction.
- Bit 2 (EM) : Emulate Flag. When set, coprocessor instructions will generate an exception

<sup>٧</sup>أعلى مستوى حماية هو الحلقة صفر (Ring0) ويليهما الحلقة ١ ثم ٢ و ٣.

جدول ٤.٢: مسجل الأعلام EFLAGS

اسم البت	رقم البت	الإستخدام
CF	0	Carry Flag - Status bit
-	1	محجوزة
PF	2	Parity Flag
-	3	محجوزة
AF	4	Adjust Flag - Status bit
-	5	محجوزة
ZF	6	Zero Flag - Status bit
SF	7	Sign Flag - Status bit
TF	9	Trap Flag - System Flag
IF	9	Interrupt Enabled Flag - System Flag
DF	10	Direction Flag - Control Flag
OF	11	Overflow Flag - Status bit
IOPL	12-13	I/O Privilege Level - Control Flag
NT	14	Nested Task Flag - Control Flag
-	15	محجوزة
RF	16	Resume Flag (386+ Only) - Control Flag
VM	17	v8086 Mode Flag (386+ Only) - Control Flag
AC	18	Alignment Check (486SX+ Only) - Control Flag
VIF	19	Virtual Interrupt Flag (Pentium+ Only) - Control Flag
VIP	20	Virtual Interrupt Pending (Pentium+ Only) - Control Flag
ID	21	Identification (Pentium+ Only) - Control Flag
-	22-31	محجوزة

- Bit 3 (TS) : Task Switched Flag This will be set when the processor switches to another task.
- Bit 4 (ET) : ExtensionType Flag. This tells us what type of coprocessor is installed.
  - 0 - 80287 is installed
  - 1 - 80387 is installed.
- Bit 5 (NE): Numeric Error
  - 0 - Enable standard error reporting
  - 1 - Enable internal x87 FPU error reporting
- Bits 6-15 : Unused
- Bit 16 (WP): Write Protect
- Bit 17: Unused
- Bit 18 (AM): Alignment Mask
  - 0 - Alignment Check Disable
  - 1 - Alignment Check Enabled (Also requires AC flag set in EFLAGS and ring 3)
- Bits 19-28: Unused
- Bit 29 (NW): Not Write-Through
- Bit 30 (CD): Cache Disable
- Bit 31 (PG) : **Enables Memory Paging.**
  - 0 - Disable
  - 1 - Enabled and use CR3 register



## القسم II.

# إقلاع الحاسب Booting





## ٣. إقلاع الحاسب ومحمل النظام Bootloader

أحد أهم الأساسيات في برمجة نظام تشغيل هي كتابة محمل 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 متمثلاً في النواة بالبدء بتنفيذ الاوامر والمهام 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 ، وفي الفصل الثالث سنعود مجدداً الى الحديث عن أنظمة الملفات.

### ١.٣. 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<sup>١</sup> الى اللوحة الام ( MotherBoard ) والتي تقوم بتوجيهها الى وحدة مزود الطاقة (Power Supply Unit). بعد ذلك يأتي دور وحدة PSU لكي تقوم بمهمة تزويد الحاسب وملحقاته بالكمية المطلوبة من الطاقة، وإرسال اشارة Power Good الى 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 ومخلفات 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 تعليمية 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 تعليمية 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 بناء على ترتيبها في اعدادات ال BIOS في برنامج Setup ، وفي حالة لم يجد ال BIOS جهازاً قابلاً للإقلاع 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

<sup>١</sup> هذه الإشارة تحوي على بت ( bit ) تدل قيمته اذا كانت 1 على أنه تم تشغيل الحاسب.

بتحميل القطاع الأول منه ( يحوي هذا القطاع على برنامج المحمل) الى الذاكرة الرئيسية وبالتحديد الى العنوان الفيزيائي 0x07c00 وسيُنقل التنفيذ الى المحمل.

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 لاحقاً- فانه لن يمكن الاستفادة من هذه المقاطعات بل سيتسبب استخدامها في حدوث استثناءات ( Exception ) توقف عمل الحاسب.

## ٢.٣. 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 إضافات ( header,symbol table,...etc ) وهو ما يعرف أيضا بـ Flat Binary .

الشرط الأول يُقيد وظيفة المحمل وقدرته على توفير خصائص متقدمة<sup>٢</sup>، 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 وانتهاءً بتغيير نمط المعالج الى النمط المحمي والقفز الى النواة للمباشرة في تنفيذها . كل ذلك يحتاج الى

<sup>٢</sup>مثل خاصية ال Safe Mode

حجم أكبر من الحجم المشروط لذلك عادة ما يلجأ مبرمجوا المحملات الى تجزئتها على مرحلتين وهو ما يسمى بـ Multi-Stage Boot Loader . الشرط الثاني للمحمل وهو أن يتواجد على أول قطاع في القرص وهو يحمل العنوان الفيزيائي التالي:

• القطاع رقم 1

• المسار رقم 0

• الرأس رقم 0

وتحقيق 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 للإقلاع من غيره ، حيث حتى يكون القطاع قابلا للإقلاع فإنه يجب أن يحمل التوقيع 0xAA55 في البايت رقم 510 و 511 . وبدون هذا التوقيع فإن البايوس (وتحديدا مقاطعة رقم 0x19) لن تعرف على هذا القطاع كقطاع قابل 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 تُحمل النواة وتنفذ 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 وجب تقسيم مهمة محمل النظام الى مرحلتين - كما سنرى ذلك - ، أما في حالة كانت النواة 16-bit فإنه 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 فإنه ينقل التنفيذ الى أول بايت فيه والذي يجب ان يكون قابلا 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 محمل النظام، فغالبا تستخدم لغة التجميع (Assembly 16-bit) لأسباب كثيرة ، منها أن الحاسب عندما يبدأ العمل فإن المعالج يكون في النمط الحقيقي تحقيقا لأغراض التوافقية ( 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 محمل النظام بلغة السي مستحيلا ! فهناك كم 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 .

### ٣.٣. مخطط الذاكرة

أثناء مرحلة الإقلاع وعندما يُنقل التنفيذ إلى محمل النظام فإن الذاكرة الرئيسية ل

### ٣.٤. برمجة محمل النظام

المثال ٣.١ يوضح أصغر محمل للنظام يمكن كتابته وتنفيذه ، باستخدام المجمع <sup>٣</sup>NASM وهو مجمع متعدد المنصات ويوفر ميزة إنتاج ملفات ثنائية object code .

#### Example ٣.١: 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-($-$$) db 0 ; append zeros.
١١
١٢ ; $ is the address of first instruction (should be 0
    x07c00) .
١٣ ; $$ 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.
```

<sup>٣</sup>راجع الملحق المعرفة كيفية استخدام المجمع لترجمة المحمل وكيفية نسخه إلى floppy disk or CD ليتم القلاع منه سواء كان على جهاز فعلي أو على جهاز تخيلي (Virtual Machine) .

### ٤.٣. . برمجية محمل النظام

وعندما يبدأ الجهاز بالعمل فان البايوس يقوم بنسخ هذا المحمل الى العنوان 0x7c00:0x0000 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 قبل البايوس.

### ٣.٤.١. عرض رسالة ترحيبية

طالما ما زلنا نعمل في النمط الحقيقي فان ذلك يمكننا من استخدام مقاطعات البايوس ، وفي المثال ٣.٢ تم عرض رسالة باستخدام مقاطعة البايوس 0x10 int الدالة 0xe .

#### Example ٣.٢: 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      ; init 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 0xaa55
٧٠    db 0xaa

```

النتيجة :

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 للإشارة إلى المكان الصحيح. و بداية نذكر أن البايوس عندما ينقل التنفيذ إلى برنامج محمل النظام الذي قمنا بكتابته فإنه 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 الأخرى مثل ds, es, ss, fs, gs. وكما نعلم أن العنوان الفيزيائي لمحمل النظام هو 0x07c00 يمكن الوصول إليه بأكثر من 4000 طريقة مختلفة، لكن سوف نقتصر على استخدام العنوان 0x07c0:0 أو العنوان 0x07c00:0x0 نظراً لأن هذه هي القيم الفعلية التي تستخدمها البايوس.

وفي حالة استخدام العنوان الأولي فإن مسجلات المقاطع يجب أن تحوي القيمة 0x07c0 (كما في المثال أعلاه) أما بقية العنوانين (سواء للمتغيرات وال label) فإنها يجب أن تبدأ من القيمة 0x0، وكما هو معروف أن المجموعات عندما تبدأ في عملية ترجمة الملف إلى ملف ثنائي فإنها تبدأ بترقيم العنوانين بدءاً من العنوان 0x0 لذلك كانت وظيفة الموجه org هي عمل إعادة تعيين (relocating) للعناوين بالقيمة التي تم كتابتها، وفي المثال أعلاه كانت القيمة هي 0x0، أما في حالة استخدام الطريقة الثانية للوصول إلى مكان محمل النظام فإن مسجلات المقاطع يجب أن تحوي القيمة 0x0 بينما المسجلات الأخرى يجب أن تبدأ قيمها من العنوان 0x7c00، وهذا لا يمكن بالوضع الطبيعي لأن المجموعات ستبدأ من العنوان 0x0 لذلك يجب استخدام الموجه org وتحديد قيمة ال relocate بالقيمة 0x7c00.

### ٢.٤.٣. معلومات قطاع الإقلاع

إضافة إلى محمل النظام فإن قطاع الإقلاع boot sector يجب أن يحوي كذلك على معلومات تساعد في وصف نظام الملفات المستخدم ووصف القرص الذي سيتم الإقلاع منه، هذه المعلومات تحوي معرف

OEM وتحتوي بيانات BIOS Parameter Block (تختصر ب BPB) ويجب أن تبدأ كل هذه البيانات من البايت رقم ٤3. 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 ٣.٣: 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.
١٠ media_descriptor     db      0xf0       ; 1.44 floppy
      disk.
١١ sectors_per_fat       dw      0x9        ; 9 sector per
      fat.
١٢ sectors_per_track     dw      0x12      ; 18 sector per
      track.
١٣ number_of_heads      dw      0x2        ; 2 heads per
      platter.
١٤ hidden_sectors       dd      0x0        ; no hidden
      sector.
١٥ total_sectors_large  dd      0x0
١٦
١٧ ; Extended BPB.
١٨
١٩ drive_number          db      0x0
٢٠ flags                db      0x0
٢١ signature            db      0x29      ; must be 0x28
      or 0x29.

```

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



### ٣.٤.٤. برمجة محمل النظام

```
٢٢ volume_id          dd          0x0          ; serial number
        written when foramt the disk.
٢٣ volume_label       db          "MOS FLOPPY " ; 11 byte.
٢٤ system_id          db          "fat12  "    ; 8 byte.
```

المثال ٣.٤.٤ يوضح شفرة المحمل بعد اضافة بيانات OEM and BPB.

#### Example ٣.٤: BPB example

```
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 ; OEM Id and BIOS Parameter Block (BPB)
١٣ ;*****
١٤
١٥ ; must begin at byte 3(4th byte), if not we should add nop
        instruction.
١٦
١٧ OEM_ID          db          "eqraOS  "    ; Name of your
        OS, Must be 8 byte! no more no less.
١٨
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 format 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 egraOS, 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.

```

### ٤.٣. برمجۃ محمل النظام

---

```
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      0xaa55
١٠٢     db      0xaa

```

والمخرج ٣.٥ 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 تُنفذ كتعليمات خاطئة ، كذلك يجب التأكد من آخر بايتين وأنها تحمل التوقيع الصحيح.

#### Example ٣.٥: 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.@f...
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	٢<.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	æÿúð....
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

### ٤.٣. برمجة محمل النظام

ويمكن الاستفادة من هذه المحررات والتعديل المباشر في قيم الهيكس للملف الثنائي<sup>٥</sup>، فمثلا يمكن حذف التوقيع واستبداله بأي رقم ومحاولة الإقلاع من القرص ! بالتأكد لا يمكن الإقلاع 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 إعادة تجميع الملف الثنائي باستخدام أي من برامج ال Disassembler وإدخال تعليمة قفز بعد استدعاء دالة طباعة السلسلة الى ما قبلها.

#### Example ٣.٦: 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.
```

<sup>٥</sup> في حالة لم تتمكن من الوصول الى ملف المصدر source code.

```

٢٤ media_descriptor    db      0xf0      ; 1.44 floppy
      disk.
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 format 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 egraOS, 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.

```

### ٤.٣. برمجۃ محمل النظام

---

```
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 times 510-($-$$) db 0 ; append zeros.
٩٧
٩٨
٩٩ ; finally the boot signature 0xaa55
١٠٠ db 0x55
١٠١ db 0xaa

```

### ٣.٤.٣. تحميل قطاع من القرص باستخدام المقاطعة int 0x13

بعد أن تم تشغيل محمل النظام لعرض رسالة ترحيبية ، فإن مهمة المحمل الفعلية هي تحميل وتنفيذ المرحلة الثانية له حيث كما ذكرنا سابقاً أن برمجية محمل النظام ستكون على مرحلتين وذلك بسبب القيود على حجم المرحلة الأولى ، وتكمن وظيفة المرحلة الأولى في البحث عن المرحلة الثانية من محمل النظام ونقل التنفيذ إليها ، وبعدها يأتي دور المرحلة الثانية في البحث عن نواة النظام ونقل التحكم إليها. وسنتناول الآن كيفية تحميل مقطع من القرص المرن إلى الذاكرة الرئيسية ونقل التحكم إليها باستخدام مقاطعة البايوس int 0x13 وفي الفصل السادس سيتم دراسة الموضوع بالتفصيل عن طريق البرمجة المباشرة لمتحكم controller القرص المرن.

#### إعادة القرص المرن

عند تكرار القراءة من القرص المرن فإنه يجب في كل مرة أن نعيد مكان القراءة والكتابة إلى 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 إذا حدث خطأ ، 0x0 إذا تمت العملية بنجاح.

مثال:

#### Example ٣.٧: Reset Floppy Drive

```

\ 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 ٣.٨: 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.

```

## ٥.٣ . 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 فإن التعامل مع القرص سيكون مستحيلاً ! حيث لن نعرف ماهي المساحات الغير مستخدمة من الأخرى ولن نستطيع ان نقوم بقراءة ملف طلبه المستخدم لعرضه على الشاشة ! وبشكل عام فإن نظام الملفات يتكون من:

- برنامج للقراءة والكتابة من القرص وسنطلق عليه اسم المحرك (Driver).
  - وجود هيكلية بيانات Data Structure معينة على القرص، يتعامل معها درايفر نظام الملفات.
- وحيث أن برمجة برنامج القراءة والكتابة تعتمد كلياً على هيكلية نظام الملفات على القرص ، فإننا سنبدأ بالحديث عنها أولاً وسوف نأخذ نظام FAT12 على قرص مرن كمثال ، نظراً لبساطة هذا النظام وخلوه من التعقيدات وفي الفصل الخامس -بإذن الله- سيتم التطرق الى أنظمة ملفات أخرى بالتفصيل.

### ١.٥.٣. 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 فإنه 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 إلا أنه يعتبر أساس جيد للأنظمة المتقدمة لذا وجب دراسته.

## ٣.٥.٢. هيكلية نظام FAT12 على القرص

عند تهيئة القرص المرن<sup>٦</sup> (Format) بنظام FAT12 فإن تركيبة القرص تكون على الشكل التالي:

وأول مقطع هو مقطع الإقلاع (Boot Sector) ويحوي شفرة محمل النظام (المرحلة الأولى) بالإضافة إلى 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 إلى مقاطعة البايوس int 0x13 التي تقوم بالقراءة من القرص كذلك في حالة ما أردنا التعامل المباشر مع متحكم القرص المرن. ونظراً لصعوبة هذه العنوان والتي تعرف ب Absolute Sector فإن أنظمة الملفات تتعامل مع نظام عنوان مختلف للوصول إلى محتويات القرص ، فبدلاً من ذكر كل من المقطع والمسار والرأس للوصول إلى مقطع ما فإن هذه العنوان تستخدم فقط رقم للمقطع . نظام العنوان الذي تستخدمه أنظمة الملفات يسمى بالعنوان المنطقية (Logical Sector Addressing) ويختصر ب LBA هو نظام بسيط يعتمد على ترقيم المقاطع بشكل متسلسل بدءاً من مقطع الإقلاع (Boot Sector) والذي يأخذ العنوان 0 ، والمقطع الثاني 1 وهكذا هلم جرا حتى نصل إلى آخر مقطع في القرص. وبما أنه يجب استخدام العنوان الحقيقية بدلاً من المنطقية لحظة القراءة من القرص (تذكر مقاطعة البايوس int 0x13 والمسجلات التي يجب إدخال قيمها) فإنه يجب إيجاد طريقة للتحويل من العنوان الحقيقية إلى المنطقية -سنناقش الموضوع لاحقاً-. ننتقل إلى المقطع التالي لمقطع الإقلاع وهو مقطع (أو عدة مقاطع) يمكن أن يحجزها المبرمج لاداء أي وظيفة يريد لها وتسمى المقاطع المحجوزة الإضافية Extra Reserved Sectors ، والمقصود بمحجوزة أي أنه لا يوجد لها وجود في دليل FAT ، ومقطع الإقلاع هو مقطع محجوز دائماً 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 إلى محتويات هذا المقطع الإضافي (إن كان له وجود) فإن العنوان الحقيقي له هو المقطع 2 المسار 0 الرأس 0 ، أما العنوان المنطقي له هو المقطع 1. وبشكل عام فإنه في الغالب لا يتم استخدام مقاطع إضافية سوى مقطع الإقلاع . المقطع الثالث هو جدول FAT ، وهو جدول يحوي سجلات بطول 12 بت عن كل كلستر (Cluster) في القرص ، بيانات هذا السجل توضح ما إذا كان الكلستر قيد الاستخدام أم لا ، وهل هو آخر كلستر للملف أم لا وإذا كان ليس باخر فإنه يوضح لنا الكلستر التالي للملف ، ويوضح الشكل التالي تركيبة هذا الجدول

إذاً هذا وظيفة هذا الجدول هي معرفة الكاسترات الخالية من غيرها كذلك الوظيفة الأخرى هي معرفة جميع الكلسترات للملف ما ويتم ذلك بالنظر إلى قيمة السجل (قيمة ال 12 بت) ، والقيم هي :

• القيمة 0x00: تدل على أن الكلستر خالي.

• القيمة 0x01: تدل على أن الكلستر محجوز.

<sup>٦</sup> سواءاً كانت التهيئة من قبل درايفر نظام الملفات الذي سنقوم ببرمجته أو كانت من قبل نظام التشغيل المستخدم أثناء عملية التطوير ، فمثلاً في ويندوز يمكن إعادة تهيئة القرص المرن بنظام FAT12 .

- القيم من 0x02 الى 0xfef : تدل على عنوان الكلستر التالي (معنى 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 0xffff : 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 المقابلة 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 ٧1 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 بالإضافة الى شرح مبسط عن هيكلية 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 بالإضافة 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 بدءاً من البايث الاول الى الاخير:

- البايثات 0-7: اسم الملف (وفي حالة كان الحجم أقل من 8 بايت يجب استخدام حرف المسافة لتعبئة المتبقي).
- البايثات 8-10: امتداد الملف (يجب استخدام المسافة أيضا لتعبئة المتبقي).
- البايث 11: خصائص الملف وهي :
  - البت 0: القراءة فقط.
  - البت 1: مخفي.
  - البت 2: ملف نظام.
  - البت 3: اسم القرص Volume Label.

<sup>٧</sup> بافتراض الوضع الغالب وهو عدم وجود مقاطع إضافية باستثناء مقطع الإقلاع

- 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.
  - 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 Length 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.

### ٣.٥.٣. هيكلية 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 تهيئة (Format) له.

### ٣.٥.٤. القراءة و الكتابة من نظام FAT12

حتى تتمكن من التعامل مع القرص المرن (قراءة وكتابة القطاعات) فانه يلزمنا برمجية درايفر لنظام FAT12 والذي سيعمل كوسيط بين المستخدم وبين القرص المرن، بمعنى أن أي طلب لقراءة ملف ما يجب أن تذهب أولاً الى نظام FAT12 حيث سيقدر 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 تتمكن 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 الخوارزميات 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 ترحيبية 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 ٣.٩: 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 أولاً 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 الحالية (Buffer) لكي يتم نقل هذا الجدول اليها. والشفرة التالية توضح كيفية عمل ذلك.

#### Example ٣.١٠ : Load Root directory

```

١      ;-----
٢      ; Compute Root Directory Size
٣      ;-----
٤
٥      xor cx,cx
٦      mov ax,32      ; every root entry size are 32 byte.
٧      mul word[root_directory] ; dx:ax = 32*224 bytes
٨      div word[bytes_per_sector]
٩      xchg ax,cx      ; cx = number of sectors to load.
١٠
١١      ;-----
١٢      ; Get start sector of root directory
١٣      ;-----
١٤

```

```

١٥     mov al,byte[total_fats]      ; there are 2 fats.
١٦     mul word[sectors_per_fat]   ; 9*2 sectors
١٧     add ax,word[reserved_sectors] ; ax = start sector of
        root directory.
١٨
١٩     mov word[data_region],ax
٢٠     add word[data_region],cx    ; data_region = start sector
        of data.
٢١
٢٢
٢٣     ;-----
٢٤     ; Load Root Dir at 0x07c0:0x0200 above bootloader.
٢٥     ;-----
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 إذا كان الملف غير موجود فنصدر رسالة خطأ ونوقف النظام عن العمل. والشفرة التالية توضح ذلك.

#### Example ٣.١١: 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 الجدول بها . والشفرة التالية توضح ذلك.

#### Example ٣.١٢: 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 Internal Fragmentation لذلك يجب اختيار حجم ملائم، وفي تنفيذ نظام FAT12 على قرص مرن اخترنا أن كل كلستر يقابل قطاع واحد فقط من القرص المرن. المشكلة التي ستواجهنا هي كيفية قراءة كلستر من القرص، فالقرص المرن لا يقرأ أي قطاع إلا بتحديد العنوان المطلق له Absolute Address ولذلك يجب تحويل رقم الكلستر إلى عنوان مطلق وتحويل عنوان LBA أيضاً إلى عنوان مطلق. التحويل من رقم Cluster إلى عنوان LBA يتم كالآتي:

#### Example ٣.١٣: 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 لاحقاً-  
وللتحويل من عنوان LBA الى عنوان Absolute Address :

#### Example ٣.١٤: Convert LBA to CHS

```

١ ; *****
٢ ; lba_to_chs: Convert LBA to CHS.
٣ ;   input:
٤ ;       ax: LBA.
٥ ; output:
٦ ;       absolute_sector
٧ ;       absolute_track
٨ ;       absolute_head
٩ ; *****
١٠ lba_to_chs:
١١
١٢     ; absolute_sector = (lba % sectors_per_track) + 1
١٣     ; absolute_track  = (lba / sectors_per_track) /
        number_of_heads
١٤     ; absolute_head   = (lba / sectors_per_track) %
        number_of_heads
١٥
١٦     xor dx,dx
١٧     div word[sectors_per_track]
١٨     inc dl
١٩     mov byte[absolute_sector],dl
٢٠
٢١     xor dx,dx
٢٢     div word[number_of_heads]
٢٣     mov byte[absolute_track],al
٢٤     mov byte[absolute_head],dl

```

```

٢٥
٢٦     ret

```

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

#### Example ٣.١٥: Load Cluster

```

١  ;-----
٢  ; Load all clusters(stage2.sys)
٣  ; At address 0x050:0x0
٤  ;-----
٥
٦      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 لاحقاً عند التحويل إلى النمط المحمي بدالة أخرى 32-bit.

#### Example ٣.١٦: 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 لآخر 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 أولاً تطبيق القانون :

$$cluster = cluster + (cluster/2)$$

وقراءة 16 بت ، وفي حالة ما اذا كان رقم الكلستر هو رقم زوجي فيجب عمل Mask لآخر 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 ٣.١٧: 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,0xffff
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,0xffff                    ; check end of file, last cluster?
٤٠          jnb 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
```

---

## ٤. برمجة محمل النظام – المرحلة الثانية

بسبب القيود على حجم محمل النظام فإن هذا قد أدى الى تقسيم المهمة الى مرحلتين حيث اقتضت 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 المحمي PMode.
- 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.

### ٤.١. 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 فإنه يجب تجهيز بعض الجداول تسمى جداول الواصفات Descriptor Table وبدون تجهيز هذه الجداول فإن 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.

## ١.١.٤. جدول الوصفات 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 كل 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 41-43: 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:
    - System descriptor: 0
    - Code or Data Descriptor: 1
  - 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: Segment type: نوع المقطع

– 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 وللبينات Code and Data Descriptor، بحيث يمكن القراءة و الكتابة من أول بايت في الذاكرة الى آخر الذاكرة 0xffffffff.

#### Example ٤.١: GDT

```

١ ;*****
٢ ; Global Descriptor Table
٣ ;*****
٤
٥ begin_of_gdt:
٦
٧ ; Null Descriptor: start at 0x0.
٨
٩     dd  0x0      ; fill 8 byte with zero.
١٠    dd  0x0
١١
١٢ ; Code Descriptor: start at 0x8.
١٣
١٤    dw  0xffff    ; limit low.
١٥    dw  0x0       ; base low.
١٦    db  0x0       ; base middle.
١٧    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 بالوصفة الخالية Null Descriptor وحجمها 8 بايت ومتحوياتها تكون صفراً في العادة ، أما الوصفة التالية لها فهي واصفة مقطع الشفرة Code Descriptor وتوضح المقطع من الذاكرة الذي سيتستخدم 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 الوصفة 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 حالياً 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.
- البابت 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 0xffff أي 1 ميجا من الذاكرة ، ولاحقاً عندما يتم تفعيل بوابة A20 ستمكن 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.
- البابت الاخير في واصفة مقطع الكود (البابت 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. إذا 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 تماماً لوصفة الكود باستثناء البت رقم 43 حيث يحدد ما اذا كان المقطع كود أم بيانات.
- وبعد إنشاء هذا الجدول (GDT) في الذاكرة ، يجب أن يحمل المسجل gdt على حجم هذا الجدول ناقصاً واحد وعلى عنوان بداية الجدول، ويتم ذلك عن طريق إنشاء مؤشرا الى جدول GDT ومن ثم استخدام الامر lgdt (وهو أمر يعمل فقط في الحلقة صفر Ring0) ، والشفرة التالية توضح ذلك.

## Example ٤.٢: 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 gdt_r
١٣ sti ; enable interrupt

```

```

١٤      popa          ; restore registers.
١٥
١٦      ret
١٧
١٨
١٩ ;*****
٢٠ ; gdt_ptr: data structure used by gdt
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

```

#### ٢.١.٤. العنوان في النمط المحمي PMode Memory Addressing

في النمط الحقيقي يستخدم المعالج عنوان 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 عنوان 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 بتحديدده 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 لأن في النمط المحمي يمكن استخدام مسجلات 32-bit فانه يمكن عنوانة 4 جيجا من الذاكرة<sup>١</sup>.

#### ٣.١.٤. 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.

<sup>١</sup> يفرض أن بوابة A20 تم تفعيلها.



#### ١.٤ . الانتقال الى النمط المحمي

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 ٤.٣: Switching to Protected Mode

```
١  ;-----
٢  ; Load gdt into gtr.
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.

```

## ٢.٤. تفعيل البوابة A20

بوابة A20 Gate هي عبارة عن OR Gate موجودة على ناقل النظام System Bus<sup>٢</sup> 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 A31-A20 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، ابتداءً 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.

## ٤.٢.١. 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 القراءة والكتابة من مسجلات المتحكم الداخلية. وبسبب ارتباط بوابة A20 مع

<sup>٢</sup>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 ويرسل الى المتحكم الموجود بداخل لوحة المفاتيح والذي بدوره 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.

## ٢.٢.٤ 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 محمولة 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 ٤.٤: Enable A20 by System Control Port 0x92

```
١ ;*****
٢ ; enable_a20_port_0x92:
٣ ; Enable A20 with System Control port 0x92
```

```

٤ ;*****
٥
٦ enable_a20_port_0x92:
٧
٨     push ax      ; save register.
٩
١٠    mov al,2      ; set bit 2 to enable A20
١١    out 0x92,al
١٢
١٣    pop ax       ; restore register.
١٤    ret

```

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

### بواسطة البايوس

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

#### Example ٤.٥: Enable A20 by BIOS int 0x15

```

١ ;*****
٢ ; enable_a20_bios:
٣ ;   Enable A20 with BIOS int 0x15 routine 0x2401
٤ ;*****
٥
٦ enable_a20_bios:
٧
٨     pusha      ; save all registers
٩
١٠    mov ax,0x2401 ; Enable A20 routine.
١١    int 0x15
١٢
١٣    popa       ; restore registers
١٤    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 الأوامر الى المتحكم ولقراءة حالة المتحكم (Status). حيث يتم إرسال الأوامر الى المتحكم عن طريق المنفذ 0x64 وإذا 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 حالياً 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 المرسلة اليه (wait input) وكيفية انتظار المتحكم الى ان يأتي بنتيجة ما (wait output).

#### Example ٤.٦: 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 أوامر إلى المتحكم فإن يجب استخدام المنفذ 0x64 وتوجد الكثير من الأوامر ، ونظرا لأن هذا الجزء غير مخصص لبرمجة متحكم لوحة المفاتيح فإننا سنناقش فقط الأوامر التي نهمنا حاليا ، وفي الفصل السادس سنعود إلى الموضوع بالتفصيل إن شاء الله.

وقائمة الأوامر حاليا:

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

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

#### Example ٤.٧: Enable A20 by Send 0xdd

```

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

```

## ٢.٤ . تفعيل البوابة A20

```
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 ٤.٨: 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 because 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 لمتحكم 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 المكس (Stack) ، وبعد ذلك تم ارسال أمر الكتابة الى منفذ الخروج لمتحكم لوحة المفاتيح (الامر 0xd1) وانتظار المتحكم حتى 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.

## ٣.٤. أساسيات ال VGA

في عام 1987 قامت IBM بتطوير مقياس لمحكمات شاشة الحاسب وهو Video Graphics Array واختصاراً VGA وجائت تسميته ب Array نظراً لانه تم تطويره كشريحة واحدة single 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<sup>٣</sup>. ال Video Buffer هو مقطع من الذاكرة segment of memory يعمل كذاكرة للشاشة Memory Mapped ، وعند بداية التشغيل فان البايوس يخصص مساحة من الذاكرة بدءاً 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 وكيفية عرض البيانات.

النمط الرسومي All Point Addressable Graphics Mode يعتمد على البكسلات ، حيث يمكن التعامل مع كل بكسل 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 فهي Screen Buffer ومجرد الكتابة عليها فان التأثير سيظهر مباشرة على الشاشة.

ومقياس VGA هو مبني على المقاييس السابقة ، ابتداءً من مقياس Monochrome Display Adapter ويسمى اختصاراً MDA والذي طوره 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 (واختصاراً 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.

#### ٤.٣.١. عنوان الذاكرة في متحكمات VGA

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

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

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

<sup>٣</sup> اشرح هذه المكونات سيكون في الفصل الخامس باذن الله ، وسيتم التركيز على بعض الاشياء بحسب الحاجة حالياً.

## Example ٤.٩: Print 'A' character on screen

```

\
٢ %define VIDEO_MEMORY 0xb8000 ; Base Address of Mapped
    Video Memory.
٢ %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.

```

## ٢.٣.٤. طباعة حرف على الشاشة

لطباعة حرف على الشاشة يجب ارسال الحرف الى عنوان الـ Video Memory وحتى تتمكن من طباعة العديد من الحروف فانه يجب انشاء متغيران (x,y) لحفظ المكان الحالي للصف والعمود ومن ثم تحويل هذا المكان الى عنوان في الـ Video Memory. 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 الـ 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:

$$videomemory = 0xb0000$$

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

ويسبب أن هناك 80 حرف في كل عمود فانه يجب ضرب قيمة y بـ 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.

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

وبارسل 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. البايث 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.

إذاً 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 ٤.١٠ : 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 ?

```

```
٢٤ ;-----
٢٥
٢٦     cmp bl,0xa      ; if character is newline ?
٢٧     je new_row     ; yes, jmp at end.
٢٨
٢٩ ;-----
٣٠ ; Calculate the memory offset
٣١ ;-----
٣٢     ; 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:
٣٣     ; offset = x_pos*2 + y_pos*COLUMNS*2
٣٤
٣٥     xor eax,eax
٣٦
٣٧     mov al,2
٣٨     mul byte[x_pos]
٣٩     push eax       ; save the first section of formula.
٤٠
٤١     xor eax,eax
٤٢     xor ecx,ecx
٤٣
٤٤     mov ax,COLUMNS*2 ; 80*2
٤٥     mov cl,byte[y_pos]
٤٦     mul ecx
٤٧
٤٨     pop ecx
٤٩     add eax,ecx
٥٠
٥١     add eax,VIDEO_MEMORY ; eax = address to print the
        character.
٥٢
٥٣ ;-----
٥٤ ; Print the chracter.
٥٥ ;-----
٥٦
٥٧     mov edi,eax
٥٨
٥٩     mov byte[edi],bl ; print the character,
٦٠     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 b1) 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.

## ٣.٣.٤. طباعة السلاسل النصية strings

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

## Example ٤.١١: 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 السلسلة النصية وطباعته الى أن نصل الى نهاية السلسلة (القيمة 0x0) ، وبعد ذلك سيتم تحديث المؤشر وذلك عن طريق متحكم CRT Controller ونظراً لأن التعامل معه بطيء قليلاً فإن تحديث المؤشر سيكون بعد طباعة السلسلة وليس بعد طباعة كل حرف .

#### ٤.٣.٤ . تحديث المؤشر Hardware Cursor

عند طباعة حرف أو سلسلة نصية فإن مؤشر الكتابة لا يتحرك من مكانه الا عند تحديده يدوياً ، 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 . ولإرسال بيانات الى هذا المتحكم ، فيجب أولاً تحديد نوع البيانات وذلك بإرسالها الى مسجل Index Register ومن ثم إرسال البيانات الى مسجل البيانات Data Register ، وفي حواسيب x86 فإن مسجل البيانات يأخذ العنوان 0x3d5 ومسجل Index Register يأخذ العنوان 0x3d4 . والجدول التالي يوضح القيم التي يمكن إرسالها 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 ٤.١٢: Move Hardware Cursor

```

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

```

٦ ;          bh: y pos.
٧ ; *****
٨
٩ bits    32
١٠
١١ move_cursor:
١٢
١٣     pusha          ; Save Registers.
١٤
١٥     ;-----
١٦     ; Calculate the offset.
١٧     ;-----
١٨     ; offset = x-pos + y-pos*COLUMNS
١٩
٢٠     xor ecx,ecx
٢١     mov cl,byte[x-pos]
٢٢
٢٣     mov eax,COLUMNS
٢٤     mul byte[y-pos]
٢٥
٢٦     add eax,ecx
٢٧     mov ebx,eax
٢٨
٢٩     ;-----
٣٠     ; Cursor Location Low.
٣١     ;-----
٣٢
٣٣     mov al,0xf
٣٤     mov dx,0x3d4
٣٥     out dx,al
٣٦
٣٧     mov al,b1
٣٨     mov dx,0x3d5
٣٩     out dx,al
٤٠
٤١     ;-----
٤٢     ; Cursor Location High.
٤٣     ;-----
٤٤
٤٥     mov al,0xe
٤٦     mov dx,0x3d4
٤٧     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

```

### ٥.٣.٤. تنظيف الشاشة Clear Screen

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

#### Example ٤.١٣: 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

```

#### ٤.٤. تحميل النواة

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

#### ٤.٤. تحميل النواة

الى هنا تنتهي مهمة المرحلة الثانية من محمل النظام Second Stage Bootloader ويتبقى فقط البحث عن النواة ونقل التحكم اليها<sup>٤</sup>. 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 ، وابتداءً من الفصل القادم سنترك لغة التجميع جانباً ونبدأ العمل بلغة السي والسي++ .

وبما أننا نعمل في النمط المحمي PMode فلا يمكننا أن نستخدم مقاطعة البايوس int 0x13 لتحميل مقاطعات 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 لاحقاً.

وحيث أن النمط المحمي يسمح باستخدام ذاكرة حتى 4 جيجا ، فان النواة سنقوم بتحميلها على العنوان 0x100000 أي عند 1 ميغا من الذاكرة . لكن علينا التذكر بأن النمط الحقيقي لا يدعم الوصول الى العنوان 0x100000 لذلك سنقوم بتحميل النواة أولاً 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 ٤.١٤: 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
```

<sup>٤</sup>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

```

#### ٤.٤. تحميل النواة

```
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 إليها ، 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 ميغا 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<sup>٥</sup>.

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<sup>٦</sup>.

#### Example ٤.١٥: 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.
```

<sup>٥</sup>راجع الفصل السابق لمعرفة التفاصيل.

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

```
١٣ %include "a20.inc"           ; Enable A20 routines.
١٤ %include "fat12.inc"        ; FAT12 driver.
١٥ %include "common.inc"      ; common declarations.
١٦
١٧ ;*****
١٨ ; data and variable
١٩ ;*****
٢٠
٢١ hello_msg    db    0xa,0xd,"Welcome to eqraOS Stage2",0xa,0xd
                ,0
٢٢ fail_message db    0xa,0xd,"KERNEL.SYS is Missing. press
                any key to reboot...",0
٢٣
٢٤
٢٥
٢٦ ; *****
٢٧ ; entry point of stage2 bootloader.
٢٨ ; *****
٢٩
٣٠ stage2:
٣١
٣٢ ;-----
٣٣ ; Set Registers.
٣٤ ;-----
٣٥
٣٦ cli
٣٧
٣٨ xor ax, ax
٣٩ mov ds, ax
٤٠ mov es, ax
٤١
٤٢ mov ax, 0x0
٤٣ mov ss, ax
٤٤ mov sp, 0xFFFF
٤٥
٤٦ sti
٤٧
٤٨ ;-----
٤٩ ; Load gdt into gdtr.
٥٠ ;-----
٥١
٥٢ call load_gdt
```



#### ٤.٤. تحميل النواة

---

```
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 ;-----

```

#### ٤.٤. تحميل النواة

---

```
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 ١.٤: محمل النظام أثناء العمل

```
Plex86/Bochs VGABios 0.6c 08 Apr 2009
This UGA/VE Bios is released under the GNU LGPL

Please visit :
. http://bochs.sourceforge.net
. http://www.nongnu.org/vgabios

Bochs VBE Display Adapter enabled

Bochs BIOS - build: 09/28/09
$Revision: 1.235 $ $Date: 2009/09/28 16:36:02 $
Options: apmbios pcibios eltorito rombios32

Press F12 for boot menu.

Booting from Floppy...
eagraOS 0.1 Copyright 2010 Ahmad Essam
.....
Welcome to eagraOS Stage2
-
```

شكل ٢.٤: بدء تنفيذ النواة

```
eagraOS v0.1 Copyright (C) 2010 Ahmad Essam
University of Khartoum - Faculty of Mathematical Sceinces.

eagraOS
```

### القسم III.

## النواة Kernel



## ٥. مقدمة حول نواة نظام التشغيل

أحد أهم المكونات في نظام التشغيل هي نواة النظام (Kernel) حيث تدير هذه النواة عتاد وموارد الحاسب وتوفر واجهة برمجية عالية تسمح لبرامج المستخدم من الاستفادة من هذه الموارد بشكل جيد. وتعتبر برمجية نواة النظام من أصعب المهمات البرمجية على الإطلاق ، حيث تؤثر هيكلته وتصميمه على كافة نظام التشغيل وهذا ما يميز بعض الانظمة ويجعلها قابلة للعمل في أجهزة معينة. وفي هذا الفصل سنلقي نظرة على النواة وبرمجتها باستخدام لغة السي و السي++ وكذلك سيتم الحديث عن طرق تصميم النواة وميزات وعيوب كل على حدة.

### ١.٥. نواة نظام التشغيل

تعرف نواة نظام التشغيل بأنها الجزء الأساسي في النظام والذي تعتمد عليه بقية مكونات نظام التشغيل. ويمكن دور نواة النظام في التعامل المباشر مع عتاد الحاسب وإدارته بحيث تكون طبقة برمجية تبعد برامج المستخدم من تفاصيل وتعقيدات العتاد ، ولا تقتصر على ذلك بل توفر واجهة برمجية مبسطة (يمكن استخدامها من لغة البرمجة المدعومة على النظام) بحيث تمكن برامج المستخدم الاستفادة من موارد الحاسب . وفي الحقيقة لا يوجد قانون ينص على إلزامية وجود نواة للنظام ، حيث يمكن لبرنامج ما (يعمل في الحلقة صفر) التعامل المباشر مع العتاد ومع كل الجداول في الحاسب والوصول الى أي بايت في الذاكرة لكن هذا ما سيجعل عملية كتابة البرامج عملية شبه مستحيلة ! حيث يجب على كل مبرمج يريد كتابة تطبيق بسيط أن يجيد برمجة العتاد وأساسيات الاقلاع حتى يعمل برنامجه ، اضافة على ذلك لا يمكن تشغيل أكثر من برنامج في نفس الوقت نظرا لعدم وجود بنية تحتية توفر مثل هذه الخصائص ، ولاننسى اعداد وتهيئة جداول النظام وكتابة وظائف التعامل مع المقاطعات والأخطاء، ودوال حجز وتحرير الذاكرة وغيرها من الخصائص الضرورية لأي برنامج. كل هذا يجعل عملية تطوير برنامج للعمل على حاسب ما بدون نواة له أمراً غير مرغوب ، خاصة إذا ذكرنا أن البرنامج يجب تحديثه مجدداً عند نقله الى منصة أخرى ذات عتاد مختلف. اذاً يمكن أن نقول أن نواة النظام هي الجزء الأهم في نظام التشغيل ككل ، حيث تدير النواة عتاد الحاسب من المعالج والذاكرة الرئيسية والأقراص الصلبة والمرنة وغيرها من الأجهزة المحيطة بالحاسب. وحتى نفهم علاقة النواة مع بقية أجزاء النظام ، فانه يمكن تقسيم الحاسب الى عدة مستويات من التجريد بحيث كل مستوى يخدم المستوى الذي يليه .

### ١.١.٥. مستويات التجريد

العديد من البرمجيات يتم بنائها على شكل مستويات ، وظيفة كل مستوى هو توفير واجهة للمستوى الذي يليه بحيث تخفي هذه الواجهة العديد من التعقيدات والتفاصيل وكذلك ربما يحمي مستوى ما بعض الخصائص من المستوى الذي يليه ، وغالبا ما يتبع نظام التشغيل لهذا النوع من البرمجيات حيث يمكن تقسيم النظام ككل الى عدة مستويات.

#### المستوى الأول: مستوى العتاد

مستوى العتاد هو أدنى مستوى يمكن أن نعرفه ويظهر على شكل متحكمات لعتاد الحاسب ، حيث يرتبط متحكم ما في اللوحة الأم مع متحكم آخر في العتاد نفسه. وظيفة المتحكم في اللوحة الأم هي التخاطب مع المتحكم الاخر في العتاد والذي بدوره يقوم بتنفيذ الأوامر المستقبلية. كيف يقوم المتحكم بتنفيذ الأوامر ؟ هذا هو دور المستوى الثاني.

#### المستوى الثاني: مستوى برامج العتاد Firmware

برامج العتاد (Firmware) هي برامج موجودة على ذاكرة بداخل المتحكم (غالبا ذاكرة EEPROM) ، وظيفة هذه البرامج هي تنفيذ الأوامر المرسلة الى المتحكم. ومن الامثلة على مثل هذه البرمجيات برنامج البايوس وأي برنامج موجود في أي متحكم مثل متحكم لوحة المفاتيح.

#### المستوى الثالث: مستوى النواة (الحلقة صفر)

النواة وهي أساس نظام التشغيل ، وظيفتها ادارة موارد الحاسب وتوفير واجهة لبقية أجزاء النظام ، وتعمل النواة في الحلقة صفر ، اي أنه يمكن تنفيذ أي أمر والوصول المباشر الى أي عنوان في الذاكرة.

#### المستوى الرابع: مستوى مشغلات الأجهزة (الحلقة ١ و ٢)

مشغلات الأجهزة هي عبارة عن برامج للنظام وظيفتها التعامل مع متحكمات العتاد (وذلك عن طريق النواة) سواء لقراءة النتائج او لارسال الأوامر ، هذه البرامج تحتاج الى أن تعمل في الحلقة ١ و ٢ حتى تتمكن من تنفيذ العديد من الأوامر ، وفي حالة تم تنفيذها على الحلقة صفر فان هذا قد يؤدي الى خطورة تعطل النظام في حالة كان هناك عطل في احد المشغلات كذلك ستكون صلاحيات المشغل عالية فقد يقوم أحد المشغلات بتغيير أحد جداول المعالج مثل جدول الواصفات العام (GDT) والذي بدوره قد يعطل النظام.



### المستوى الخامس: مستوى برامج المستخدم (الحلقة ٣)

المستوى الاخير وهو مستوى برامج المستخدم حيث لا يمكن لهذه البرامج الوصول الى النواة وانما تتعامل فقط مع واجهة برمجة التطبيقات (Application Programming Interface) والتي تعرف بدوال (API).

## ٢٠٥. وظائف نواة النظام

تختلف مكونات ووظائف نواة نظام التشغيل تبعاً لطريقة التصميم المتبعة، فهناك العديد من الطرق لتصميم الأنوية بعضهاً منها يجعل ما هو متعارف عليه بأنه يتبع لنواة النظام برنامج للمستخدم (User Program)<sup>١</sup> والبعض الاخر عكس ذلك. لذلك سنذكر حالياً المكونات الشائعة في نواة النظام وفي القسم التالي عند الحديث عن هيكلية وطرق تصميم الأنوية سنفصل أكثر في هذه المكونات ونقسمها بحسب طريقة التصميم.

### ١.٢.٥. إدارة الذاكرة

أهم وظيفة لنواة النظام هي إدارة الذاكرة حيث أن أي برنامج يجب ان يتم تحميله على الذاكرة الرئيسية قبل أن يتم تنفيذه، لذلك من مهام مدير الذاكرة هي معرفة الأماكن الشاغرة، والتعامل مع مشاكل التجزئة (Fragmentation) حيث من الممكن أن تحوي الذاكرة على الكثير من المساحات الصغيرة والتي لا تكفي لتحميل أي برنامج أو حتى حجز مساحة لبرنامج ما. أحد المشاكل التي على مدير الذاكرة التعامل معها هي معرفة مكان تحميل البرنامج، حيث يجب أن يكون البرنامج مستقلاً عن العناوين (Position Independent) لكي يتم تحميله وإلا فلن نعرف ما هو عنوان البداية (Base Address) لهذا البرنامج. فلو فرضنا ان لدينا برنامج binary ونريد تحميله الى الذاكرة فهنا لن نتمكن من معرفة ما هو العنوان الذي يجب أن يكون عليه البرنامج، لذلك عادة فإن الناتج من عملية ترجمة وربط أي برنامج هو انها تبدأ من العنوان  $0 \times 0$ ، وهكذا سنتمكن دوماً من تحميل أي برنامج في بداية الذاكرة. بهذا الشكل لن نتمكن من تنفيذ أكثر من برنامج واحد، حيث سيكون هناك برنامجاً واحداً فقط يبدأ من العنوان  $0 \times 0$ ، والحل لهذه المشاكل هو باستخدام مساحة العنوان التخيلية (Virtual Address Space) حيث يتم تخصيص مساحة تخيلية من الذاكرة لكل برنامج بحيث تبدأ العنوانية تخيلية من  $0 \times 0$  وبهذا تم حل مشكلة تحميل أكثر من برنامج وحل مشكلة relocation. ومساحة العنوان التخيلية (VAS) هي مساحة من العناوين لكل برنامج بحيث تبدأ من ال  $0 \times 0$  ومفهوم هذه المساحة هو أن كل برنامج سيتعامل مع مساحة العناوين الخاصة به وهذا ما يؤدي الى حماية الذاكرة، حيث لن يستطيع أي برنامج الوصول الى أي عنوان آخر بخلاف العناوين الموجودة في VAS. ونظراً لعدم ارتباط ال VAS مع الذاكرة الرئيسية فإنه يمكن ان يشير عنوان تخيلي الى ذاكرة اخرى بخلاف الذاكرة الرئيسية (مثلاً القرص الصلب). وهذا يحل مشكلة انتهاء المساحات الخالية في

<sup>١</sup>المقصود أنها برامج تعمل في الحلقة ٣.

الذاكرة. ويجدر بنا ذكر أن التحويل بين العناوين التخيلية الى الحقيقية يتم عن طريق العتاد بواسطة وحدة ادارة الذاكرة بداخل المعالج (Memory Management Unit). وكذلك مهمة حماية الذاكرة والتحكم في الذاكرة Cache وغيرها من الخصائص والتي سيتم الإطلاع عليها في الفصل الثامن - بمشيئة الله-.

## ٢.٢.٥. إدارة العمليات

## ٣.٢.٥. نظام الملفات

## ٣.٥. هيكلية وتصميم النواة

توجد العديد من الطرق لتصميم الأنوية وسنستعرض بعض منها في هذا البحث ، لكن قبل ذلك يجب الحديث عن طريقة مفيدة في هيكلية وتصميم الأنوية الا وهي تجريد العتاد (Hardware Abstraction) أي بمعنى فصل النواة من التعامل المباشر مع العتاد ، وانشاء طبقة برمجية (Software Layer) تسمى طبقة HAL (اختصارا لكلمة Hardware Abstraction Layer) بين النواة وبين العتاد ، وظيفة طبقة HAL هي توفير واجهة لعتاد الحاسب بحيث تمكن النواة من التعامل مع العتاد.

فصل النواة من العتاد تتيح العديد من الفوائد ، أولاً شفرة النواة ستكون أكثر مقروئية وأسهل في الصيانة والتعديل لأن النواة ستتعامل مع واجهة أخرى أكثر سهولة من تعقيدات العتاد ، الميزة الثانية والأكثر أهمية هي امكانية نقل النواة (Porting) لأجهزة ذات عتاد مختلف (مثل SPARC, MIPS,...etc) بدون التغيير في شفرة النواة ، فقط سيتم تعديل طبقة HAL من ناحية التطبيق (Implementation) بالاضافة الى إعادة كتابة مشغلات الأجهزة (Devcie Drivers) مجدداً<sup>٢</sup>.

## ١.٣.٥. النواة الضخمة Monolithic Kernel

تعتبر الأنوية المصممة بطريقة Monolithic<sup>٣</sup> أسرع وأكثر أنوية في العمل وذلك نظرا لان كل برامج النظام (System Process) تكون ضمن النواة وتعمل في الحلقة صفر ، والشكل التالي يوضح مخطط لهذه الأنوية. المشكلة الرئيسية لهذا التصميم هو انه عند حدوث أي مشكلة في أي من برامج النظام فان النظام سوف يتوقف عن العمل وذلك نظرا لانها تعمل في الحلقة صفر وكما ذكرنا سابقا أن أي خلل في هذا المستوى يؤدي الى توقف النظام عن العمل. مشكلة اخرى يمكن ذكرها وهي ان النواة غير مرنة بمعنى أنه لتغيير نظام الملفات مثلا يجب إعادة تشغيل النظام مجددا.

<sup>٢</sup>أغلب أنوية أنظمة التشغيل الحالية تستخدم طبقة HAL ، هل تساءلت يوما كيف يعمل نظام جنو/لينوكس على أجهزة سطح المكتب والأجهزة المضمنة!  
<sup>٣</sup>كلمة Mono تعني واحد ، أما كلمة Lithic فتعني حجري ، والمقصود بأن النواة تكون على شكل كتلة حجرية ليست مرنة وتطورها وصيانتها معقد.

وكأثلة على أنظمة تشغيل تعمل بهذا التصميم هي أنظمة يونكس ولينوكس ، وأنظمة ال DOS القديمة وويندوز ما قبل NT.

### ٢.٣.٥ . النواة المصغرة MicroKernel

الأنوية MicroKernel هي الأكثر ثباتا واستقرار ومرونة والأسهل في الصيانة والتعديل والتطوير وذلك نظرا لان النواة تكون أصغر ما يمكن ، حيث أن الوظائف الأساسية فقط تكون ضمن النواة وهي ادارة الذاكرة وادارة العمليات (مجدول العمليات، أساسيات IPC)، أما بقية برامج النظام مثل نظام الملفات ومشغلات الأجهزة وغيرها تتبع لبرامج المستخدم وتعمل في نمط المستخدم (الحلقة ٣) ، وهذا يعني في حالة حدوث خطأ في هذه البرامج فان النظام لن يتأثر كذلك يمكن تغيير هذه البرامج (مثلا تغيير نظام الملفات) دون الحاجة الى اعادة تشغيل الجهاز حيث أن برامج النظام تعمل كبرامج المستخدم . والشكل التالي يوضح مخطط هذه الأنوية. المشكلة الرئيسية لهذا التصميم هو بطئ عمل النظام وذلك بسبب أن برامج النظام عليها أن تتخاطب مع بعضها البعض عن طريق تمرير الرسائل (Message Passing) أو مشاركة جزء من الذاكرة (Shared Memory) وهذا ما يعرف ب Interprocess Communication. وأشهر مثال لنظام تشغيل يتبع هذا التصميم هو نظام مينكس الاصدار الثالث.

### ٣.٣.٥ . النواة الهجينة Hybrid Kernel

هذا التصميم للنواة ما هو إلا مزيج من التصميمين السابقين ، حيث تكون النواة MicroKernel لكنها تطبق ك Monolithic Kernel ، ويسمى هذا التصميم Hybrid Kernel أو Modified MicroKernel. والشكل التالي يوضح مخطط لهذا التصميم. وكأثلة على أنظمة تعمل بهذا التصميم هو أنظمة ويندوز التي تعتمد على معمارية NT ، ونظام BeOS و 9 Plane.

## ٤.٥ . برجة نواة النظام

يمكن برجة نواة نظام التشغيل بأي لغة برجة ، لكن يجب التأكد من أن اللغة تدعم استخدام لغة التجميع (Inline Assembly) حيث أن النواة كثيرا ما يجب عليها التعامل المباشر مع أوامر لغة التجميع (مثلا عند تحميل جدول الواصفات العام وجدول المقاطعات وكذلك عند غلق المقاطعات وتفعيلها وغيرها). الشيء الآخر الذي يجب وضعه في الحسبان هو أنه لا يمكن استخدام لغة برجة تعتمد على مكتبات في وقت التشغيل (ملفات dll مثلا) دون إعادة برجة هذه المكتبات (مثال ذلك لا يمكن استخدام لغات دوت نت دون إعادة برجة إطار العمل). وكذلك لا يمكن الاعتماد على دوال النظام الذي تقوم بتطوير نظامك الخاص فيه (مثلا لن تتمكن من استخدام new لحجز الذاكرة وذلك لانها تعتمد كليا على نظام التشغيل، أيضا دوال الادخال والاخراج تعتمد كليا على النظام).

لذلك غالبا تستخدم لغة السي والسي++ لبرمجة أنوية أنظمة التشغيل نظرا لما تتمتع به اللغتين من ميزات فريدة تميزها عن باقي اللغات ، وتنتشر لغة السي بشكل أكبر لاسباب كثيرة منها هو أنها لا تحتاج الى مكتبة وقت التشغيل (RunTime Library) حتى تعمل البرامج المكتوبة بها على عكس لغة سي++ والتي تحتاج الى (RunTime Library) لدعم الكثير من الخصائص مثل الاستثناءات و دوال البناء والهدم. وفي حالة استخدام لغة سي أو سي++ فانه يجب إعادة تطوير اجزاء من مكتبة السي والسي++ القياسية (Standard C/C++ Library) وهي الأجزاء التي تعتمد على نظام التشغيل مثل دوال printf و scanf و دوال حجز الذاكرة malloc/new وتحريرها free/delete. ونظرا لاننا بصدد برمجة نظام 32 بت ، فان النواة أيضا يجب أن تكون 32 بت وهذا يعني أنه يجب استخدام مترجم سي أو سي++ 32 بت . مشكلة هذه المترجمات أن المخرج منها (البرنامج) لا يأتي بالشكل الثنائي فقط (Flat Binary) ،<sup>٤</sup> وانما يضاف على الشفرة الثنائية العديد من الأشياء Headers,...etc. ولتحميل مثل هذه البرامج فانه يجب البحث عن نقطة الإنطلاق للبرنامج (main routine) ومن ثم البدء بتنفيذ الأوامر منها. وسيتم استخدام مترجم فيجوال سي++ لترجمة النواة ، وفي الملحق سيتم توضيح خطوات تهيئة المشروع وازالة أي اعتمادية على مكتبات أو ملفات وقت التشغيل. وسنعيد كتابة النواة التي قمنا ببرمجتها بلغة التجميع في الفصل السابق ولكن بلغة السي والسي++ ، وسناقش كيفية تحميل وتنفيذ هذه النواة حيث أن المخرج من مترجم فيجوال سي++ هو ملف تنفيذي (Portable Executable) ولديه صيغة محددة يجب التعامل معها حتى تتمكن من تنفيذ الدالة الرئيسية للنواة (main()) ، كذلك سنبدأ في تطوير ملفات وقت التشغيل للغة سي++ وذلك حتى يتم دعم بعض خصائص اللغة والتي تحتاج الى دعم وقت التشغيل مثل دوال البناء والهدم والدوال الظاهرية (Pure Virtual Function) ، وفي الوقت الحالي لا يوجد دعم للإستثناءات (Exceptions) في لغة السي++ .

#### ٥.٤.١. تحميل وتنفيذ نواة PE

بما أننا سنستخدم مترجم فيجوال سي++ والذي يخرج لنا ملف تنفيذي (Portable Executable) فانه يجب أن نعرف ما هي شكل هذه الصيغة حتى تتمكن عند تحميل النواة أن ننقل التنفيذ الى الدالة الرئيسية وليست الى أماكن أخرى. ويمكن استخدام مترجمات سي++ أخرى (مثل مترجم g++) لكن يجب ملاحظة أن هذا المترجم يخرج لنا ملف بصيغة ELF وهي صيغة الملفات التنفيذية على نظام جنو/لينوكس. والشكل التالي يوضح صيغة ملف PE الذي نحن بصدد التعامل معه. يوجد أربع اضافات (headers) لصيغة PE سنطلع عليها بشكل سريع وفي حالة قمنا بتطوير محمل خاص لهذه الصيغة فسيتم دراستها بالتفصيل. ويمكن أن نصف هذه الاضافات بلغة السي++ كالتالي.

#### Example ٥.١ : Portable Executable Header

١

<sup>٤</sup>كبرنامج محمل النظام الذي قمنا بتطويره في بداية هذا البحث.

```

٢ // header information format for PE files
٣
٤ typedef struct _IMAGE_DOS_HEADER { // DOS .EXE header
٥     unsigned short e_magic; // Magic number (Should be MZ
٦     unsigned short e_cblp; // Bytes on last page of file
٧     unsigned short e_cp; // Pages in file
٨     unsigned short e_crlc; // Relocations
٩     unsigned short e_cparhdr; // Size of header in
    paragraphs
١٠    unsigned short e_minalloc; // Minimum extra paragraphs
        needed
١١    unsigned short e_maxalloc; // Maximum extra paragraphs
        needed
١٢    unsigned short e_ss; // Initial (relative) SS value
١٣    unsigned short e_sp; // Initial SP value
١٤    unsigned short e_csum; // Checksum
١٥    unsigned short e_ip; // Initial IP value
١٦    unsigned short e_cs; // Initial (relative) CS value
١٧    unsigned short e_lfarlc; // File address of relocation
        table
١٨    unsigned short e_ovno; // Overlay number
١٩    unsigned short e_res[4]; // Reserved words
٢٠    unsigned short e_oemid; // OEM identifier (for
        e_oeminfo)
٢١    unsigned short e_oeminfo; // OEM information; e_oemid
        specific
٢٢    unsigned short e_res2[10]; // Reserved words
٢٣    long e_lfanew; // File address of new exe header
٢٤ } IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;
٢٥
٢٦
٢٧ // Real mode stub program
٢٨
٢٩ typedef struct _IMAGE_FILE_HEADER {
٣٠     unsigned short Machine;
٣١     unsigned short NumberOfSections;
٣٢     unsigned long TimeDateStamp;
٣٣     unsigned long PointerToSymbolTable;
٣٤     unsigned long NumberOfSymbols;
٣٥     unsigned short SizeOfOptionalHeader;
٣٦     unsigned short Characteristics;
٣٧ } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

```

```

٣٨
٣٩ typedef struct _IMAGE_OPTIONAL_HEADER {
٤٠     unsigned short    Magic;
٤١     unsigned char     MajorLinkerVersion;
٤٢     unsigned char     MinorLinkerVersion;
٤٣     unsigned long     SizeOfCode;
٤٤     unsigned long     SizeOfInitializedData;
٤٥     unsigned long     SizeOfUninitializedData;
٤٦     unsigned long     AddressOfEntryPoint;    // offset of
        kernel_entry
٤٧     unsigned long     BaseOfCode;
٤٨     unsigned long     BaseOfData;
٤٩     unsigned long     ImageBase;    // Base address of
        kernel_entry
٥٠     unsigned long     SectionAlignment;
٥١     unsigned long     FileAlignment;
٥٢     unsigned short    MajorOperatingSystemVersion;
٥٣     unsigned short    MinorOperatingSystemVersion;
٥٤     unsigned short    MajorImageVersion;
٥٥     unsigned short    MinorImageVersion;
٥٦     unsigned short    MajorSubsystemVersion;
٥٧     unsigned short    MinorSubsystemVersion;
٥٨     unsigned long     Reserved1;
٥٩     unsigned long     SizeOfImage;
٦٠     unsigned long     SizeOfHeaders;
٦١     unsigned long     CheckSum;
٦٢     unsigned short    Subsystem;
٦٣     unsigned short    DllCharacteristics;
٦٤     unsigned long     SizeOfStackReserve;
٦٥     unsigned long     SizeOfStackCommit;
٦٦     unsigned long     SizeOfHeapReserve;
٦٧     unsigned long     SizeOfHeapCommit;
٦٨     unsigned long     LoaderFlags;
٦٩     unsigned long     NumberOfRvaAndSizes;
٧٠     IMAGE_DATA_DIRECTORY DataDirectory[DIRECTORY_ENTRIES];
٧١ } IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;

```

ما نريد الحصول عليه هو عنوان الدالة الرئيسية للنواة (kernel entry()) والتي سيبدأ تنفيذ النواة منها ، هذا العنوان موجود في أحد المتغيرات في آخر إضافة (header) وهي IMAGE\_OPTIONAL\_HEADER ، وحتى نحصل على عنوان هذه الأضافة يجب أن نبدأ من أول إضافة وذلك بسبب أن الإضافة الثانية ذات حجم متغير وليست ثابتة مثل بقية الإضافات.

وبالنظر الى أول إضافة IMAGE DOS HEADER وبالتحديد الى المتغير `ImageBase` حيث يحوي عنوان الإضافة الثالثة IMAGE FILE HEADER والتي هي اضافة ثابتة الحجم ، ومنها نصل الى آخر إضافة ونقرأ المتغير `AddressOfEntryPoint` الذي يحوي عنوان `offset` للدالة الرئيسية وكذلك نقرأ المتغير `ImageBase` والذي يحوي عنوان البداية للدالة ويجب اضافته لقيمة ال `offset` ، وبعد ذلك يتم نقل التنفيذ الى الدالة بواسطة الامر `call`. والشفرة التالية توضح طريقة ذلك (ويتم تنفيذها في المرحلة الثانية من محمل النظام مباشرة بعدما يتم تحميل النواة الى الذاكرة على العنوان `KERNEL PMODE BASE`).

#### Example ٥.٢: Getting Kernel entry

```

١
٢
٣ mov ebx, [KERNEL_PMODE_BASE+60]
٤ add ebx, KERNEL_PMODE_BASE ; ebx = _IMAGE_FILE_HEADER
٥
٦ add ebx, 24 ; ebx = _IMAGE_OPTIONAL_HEADER
٧
٨ add ebx, 16 ; ebx point to AddressOfEntryPoint
٩
١٠ mov ebp, dword[ebx] ; ebp = AddressOfEntryPoint
١١
١٢ add ebx, 12 ; ebx point to ImageBase
١٣
١٤ add ebp, dword[ebx] ; ebp = kernel_entry
١٥
١٦ cli
١٧
١٨ call ebp
```

### ٢.٤.٥ . تطوير بيئة التشغيل للغة سي++

حتى تتمكن من استخدام جميع خصائص لغة سي++ فانه يجب كتابة بعض الشفرات التشغيلية (startup) والتي تمهد وتعرف العديد من الخصائص في اللغة ، وفي هذا الجزء سيتم تطوير مكتبة وقت التشغيل للغة سي++ (C++ Runtime Library) وذلك نظراً لأننا قد الغينا الإعتماد على مكتبة وقت التشغيل التي تأتي مع المترجم المستخدم في بناء النظام (النظام الخاص بنا) حيث أن هذه المكتبة تعتمد على نظام التشغيل المستخدم في عملية التطوير مما يسبب مشاكل استدعاء دوال ليست موجودة. وبدون تطوير هذه المكتبة فلن يمكن تهيئة الكائنات العامة (Global Object) وحذف الكائنات ، وكذلك لن يمكن استخدام بعض المعاملات (new, delete) و RTTI والاستثناءات (Exceptions).

## المعاملات العامة Global Operator

سيتم تعريف معامل حجز الذاكرة (new) وتحريرها (delete) في لغة السي++ ، ولكن لاننا حالياً لم نرمج مديراً للذاكرة فان التعريف سيكون حالياً. والمقطع التالي يوضح ذلك.

## Example ٥.٣: Global new/delete operator

```

١
٢ void* __cdecl ::operator new (unsigned int size){return 0;}
٣ void* __cdecl operator new[] (unsigned int size){return 0;}
٤ void __cdecl ::operator delete (void * p){}
٥ void __cdecl operator delete[] (void * p){}
```

## Pure virtual function call handler

ايضا يجب تعريف دالة للتعامل مع الدوال الظاهرية النقية (Pure virtual function) °، حيث سيقوم المترجم باستدعاء الدالة () purecall أينما وجد عملية استدعاء لدالة Pure virtual ، لذلك أن أردنا دعم الدوال Pure virtual يجب تعريف الدالة purecall ، وحالياً سيكون التعريف كالآتي.

## Example ٥.٤: Pure virtual function call handler

```

١
٢ int __cdecl ::purecall() { for (;;) return 0; }
```

## دعم الفاصلة العائمة Floating Point Support

لدعم الفاصلة العائمة (Floating Point) في سي++ فانه يجب تعيين القيمة 1 للمتغير fltused ، وكذلك يجب تعريف الدالة () fto12 sse والتي تحول من النوع float الى النوع long كالتالي.

## Example ٥.٥: Floating Point Support

```

١
٢
٣ extern "C" long __declspec (naked) _ftol2_sse() {
٤     int a;
```

° عند تعريف دالة بألمة Pure virtual داخل أي فئة فإن هذا يدل على أن الفئة مجردة (Abstract) ويجب إعادة تعريف الدالة (Override) في الفئات المشتقة من الفئة التي تحوي هذه الدالة، والا ستكون الفئة المشتقة .  
 ° هذه الدالة يقوم مترجم الفيجوال سي++ باستدعائها. وقد تختلف من مترجم لآخر.



```

٥ #ifdef i386
٦     _asm {
٧         fistp [a]
٨         mov ebx, a
٩     }
١٠ #endif
١١ }
١٢
١٣ extern "C" int _fltused = 1;

```

### تهيئة الكائنات العامة والسكنة

عندما يجد المترجم كائناً فإنه يضيف مهيئاً ديناميكياً له (Dynamic initializer) في قسم خاص من البرنامج<sup>٧</sup> وهو القسم crt. . وقبل أن يعمل البرنامج فإن وظيفة مكتبة وقت التشغيل هي استدعاء وتنفيذ كل المهيئات وذلك حتى تأخذ الكائنات قيمها الابتدائية (عبر دالة البناء Constructor). وبسبب أننا أزلنا مكتبة وقت التشغيل فإنه يجب إنشاء القسم crt. وهذا يتم عن طريق موجهات المعالج التمهيدي (Preprocessor) الموجودة في المترجم.

هذا القسم crt. يحوي مصفوفة من مؤشرات الدوال (Function Pointer)، ووظيفة مكتبة وقت التشغيل هي استدعاء كل الدوال الموجودة وذلك بالمرور على مصفوفة المؤشرات الموجودة. و يجب أن نعلم أن مصفوفة المؤشرات موجودة حقيقة داخل القسم crt:xcu. حيث أن الجزء الذي يلي العلامة dollar sign يحدد المكان بداخل القسم، وحتى تتمكن من استدعاء وتنفيذ الدوال عن طريق مصفوفة المؤشرات فإنه يجب إنشاء مؤشر إلى بداية القسم crt:xcu. وفي نهايته، مؤشر البداية سيكون في القسم crt:xca. وهو يسبق القسم crt:xcu. مباشرة، ومؤشر النهاية سيكون في القسم crt:xcz. يلي القسم crt:xcu مباشرة.

وبخصوص القسم crt. الذي سننشئه فإننا لا نملك صلاحيات قراءة وكتابة فيه، لذلك الحل في أن نقوم بدمج هذا القسم مع قسم البيانات data. . والشفرة التالية توضح ما سبق.

#### Example ٥.٦: Object\_INITIALIZER

```

١
٢ // Function pointer typedef for less typing
٣ typedef void (__cdecl *_PVFV)(void);
٤
٥ // __xc_a points to beginning of initializer table

```

<sup>٧</sup> في أي برنامج تنفيذي يوجد العديد من الأقسام، مثلاً قسم البيانات data. وقسم الشفرة code. والمكدس stack. وغيرها.

```
٦ #pragma data_seg(".CRT$XCA")
٧ _PVFV __xc_a[] = { 0 };
٨
٩ // __xc_z points to end of initializer table
١٠ #pragma data_seg(".CRT$XCZ")
١١ _PVFV __xc_z[] = { 0 };
١٢
١٣ // Select the default data segment again (.data) for the
    rest of the unit
١٤ #pragma data_seg()
١٥
١٦ // Now, move the CRT data into .data section so we can read/
    write to it
١٧ #pragma comment(linker, "/merge:.CRT=.data")
١٨
١٩
٢٠ // initialize all global initializers (ctors, statics,
    globals, etc..)
٢١ void __cdecl _initterm ( _PVFV * pfbegin, _PVFV * pfend ) {
٢٢
٢٣     //! Go through each initializer
٢٤     while ( pfbegin < pfend )
٢٥     {
٢٦         //! Execute the global initializer
٢٧         if ( *pfbegin != 0 )
٢٨             (**pfbegin) ();
٢٩
٣٠         //! Go to next initializer inside the initializer
            table
            ++pfbegin;
٣١     }
٣٢ }
٣٣
٣٤
٣٥ // execute all constructors and other dynamic initializers
٣٦ void _cdecl init_ctor() {
٣٧
٣٨     _atexit_init();
٣٩     _initterm(__xc_a, __xc_z);
٤٠ }
```

---

## حذف الكائنات

لكي يتم حذف الكائنات (Objects) يجب انشاء مصفوفة من مؤشرات دوال الهدم (deinitializer array) ، وذلك بسبب أن المترجم عندما يجد دالة هدم فإنه يضيف مؤشراً الى دالة الهدم بداخل هذه المصفوفة وذلك حتى يتم استدعائها لاحقاً (عند استدعاء الدالة (exit ( )، ويجب تعريف الدالة atexit حيث أن مترجم الفيچوال سي++ يقوم باستدعائها عندما يجد أي كائن ، وظيفة هذه الدالة هي اضافة مؤشر لدالة هدم الكائن الى مصفوفة المؤشرات ، وبخصوص مصفوفة المؤشرات فإنه يمكن حفظها في أي مكان على الذاكرة . والشفرة التالية توضح ما سبق.

## Example ٥.٧: Delete Object

```

١
٢ /* function pointer table to global deinitializer table
٣ static _PVFV * pf_atexitlist = 0;
٤
٥ // Maximum entries allowed in table. Change as needed
٦ static unsigned max_atexitlist_entries = 32;
٧
٨ // Current amount of entries in table
٩ static unsigned cur_atexitlist_entries = 0;
١٠
١١ /// initialize the de-initializer function table
١٢ void __cdecl _atexit_init(void) {
١٣
١٤     max_atexitlist_entries = 32;
١٥
١٦     // Warning: Normally, the STDC will dynamically allocate
١٧     this. Because we have no memory manager, just choose
١٨     // a base address that you will never use for now
١٩     pf_atexitlist = (_PVFV *)0x5000;
٢٠ }
٢١ /// adds a new function entry that is to be called at
٢٢ shutdown
٢٣ int __cdecl atexit(_PVFV fn) {
٢٤
٢٥     /// Insure we have enough free space
٢٦     if (cur_atexitlist_entries >= max_atexitlist_entries)
٢٧         return 1;
٢٨     else {
```

```

٢٩    ///! Add the exit routine
٣٠    *(pf_atexitlist++) = fn;
٣١    cur_atexitlist_entries++;
٣٢    }
٣٣    return 0;
٣٤ }
٣٥
٣٦ ///! shutdown the C++ runtime; calls all global de-
    initializers
٣٧ void _cdecl exit () {
٣٨
٣٩    ///! Go through the list, and execute all global exit
        routines
٤٠    while (cur_atexitlist_entries—) {
٤١
٤٢        ///! execute function
٤٣        *(--pf_atexitlist)() ;
٤٤    }
٤٥ }

```

### ٣.٤.٥. نقل التنفيذ الى النواة

بعد أن قمنا بعمل تحليل (Parsing) لصيغة ملف PE ونقل التنفيذ الى الدالة `kernel_entry()` والتي تعتبر أول دالة يتم تنفيذها في نواة النظام ، وأول ما يجب تنفيذه فيها هو تحديد قيم مسجلات المقاطع وانشاء مكسدس (Stack) وبعد ذلك يجب تهيئة الكائنات العامة ومن ثم استدعاء الدالة `main()` التي تحوي شفرة النواة ، واخيرا عندما تعود الدالة `main()` يتم حذف الكائنات وايقاف النظام (Hang). والشفرة التالية توضح ذلك

#### Example ٥.٨: Kernel Entry routine

```

١
٢ extern void _cdecl main ();    // main function.
٣ extern void _cdecl init_ctor(); // init constructor.
٤ extern void _cdecl exit ();    // exit.
٥
٦ void _cdecl kernel_entry ()
٧ {
٨
٩ #ifdef i386

```

## ٥.٥. نظرة على شفرة نظام إقرأ

---

```
١٠  _asm {
١١      cli
١٢
١٣      mov ax, 10h          // select data descriptor in GDT.
١٤      mov ds, ax
١٥      mov es, ax
١٦      mov fs, ax
١٧      mov gs, ax
١٨      mov ss, ax          // Set up base stack
١٩      mov esp, 0x90000
٢٠      mov ebp, esp        // store current stack pointer
٢١      push ebp
٢٢  }
٢٣  #endif
٢٤
٢٥      // Execute global constructors
٢٦      init_ctor();
٢٧
٢٨      // Call kernel entry point
٢٩      main();
٣٠
٣١      // Cleanup all dynamic dtors
٣٢      exit();
٣٣
٣٤  #ifdef i386
٣٥      _asm cli
٣٦  #endif
٣٧
٣٨      for(;;);
٣٩  }
```

---

وتعريف الدالة `main()` حالياً سيكون خالياً.

## ٥.٥. نظرة على شفرة نظام إقرأ

أهم الخصائص التي يجب مراعاتها أثناء برمجة نواة نظام التشغيل هي خاصية المحمولة على صعيد الأجهزة والمنصات<sup>٨</sup> وخاصية قابلية توسعة النواة (Expandability) و لذلك تم الاتفاق على أن تصميم نواة نظام

---

<sup>٨</sup>على عكس محمل النظام Bootloader والذي يعتمد على معمارية العتاد والمعالج.

تشغيل إقرأ سيتم بنائها على طبقة HAL حتى تسمح لأي مطور فيما بعد إعادة تطبيق هذه الطبقة لدعم أجهزة وعتاد آخر. وحتى نحصل على أعلى قدر من المحمولية وقابلية التوسعة في نواة النظام فإنه سيتم تقسيم الشفرات البرمجية للنواة الى وحدات مستقلة بحيث تؤدي كل وحدة وظيفة ما ، وفي نفس الوقت يجب أن تتوافر واجهة عامة (Interface) لكل وحدة بحيث يتمكن من الاستفادة من خدمات هذه الوحدة دون الحاجة لمعرفة تفاصيلها الداخلية. وفي بداية تصميم المشروع فإن عملية تصميم الواجهة تعتبر أهم بكثير من عملية برمجة محتويات الوحدة أو ما يسمى بالتنفيذ (Impelmentation) نظراً لأن التنفيذ قد لا يؤثر على هيكلية المشروع ومعماريته مثلما تؤثر الواجهة .

- eqraOS:

- boot: first-stage and second-stage bootloader.
- core:
  - \* kernel: Kernel program PE executable file type.
  - \* hal: Hardware abstraction layer.
  - \* lib: Standard library runtime and standard C/C++ library.
  - \* include: Standard include headers.
  - \* debug: Debug version of eqraOS.
  - \* release: Final release of eqraOS.

## ٦.٥. مكتبة السي القياسية

نظراً لأنه قد تم إلغاء الاعتماد على مكتبة السي والسي++ القياسية أثناء تطوير نواة نظام التشغيل فإنه يجب انشاء هذه المكتبة حتى يتمكن من استخدام لغة سي وسي++ ، وبسبب أن عملية إعادة برمجة هذه المكتبات يتطلب وقتاً وجهداً فنانا سنركز على بعض الملفات المستخدمة بكثرة ونترك البقية للتطوير لاحقاً.

### تعريف NULL

في لغة سي++ يتم تعريف NULL على أنها القيمة 0 بينما في لغة السي تعرف ب 0 (void\*).

Example ٥.٩: null.h: Definition of NULL in C and C++

```

١
٢ #ifndef NULL_H
٣ #define NULL_H
٤
```

```

٥ #if define (_MSC_VER) && (_MSC_VER >= 1020)
٦ #pragma once
٧ #endif
٨
٩ #ifndef NULL
١٠ #undef NULL
١١ #endif
١٢
١٣ #ifdef __cplusplus
١٤ extern "C"
١٥ {
١٦ #endif
١٧
١٨ /* C++ NULL definition */
١٩ #define NULL 0
٢٠
٢١ #ifdef __cplusplus
٢٢ }
٢٣ #else
٢٤
٢٥ /* C NULL definition */
٢٦ #define NULL (void*)0
٢٧
٢٨ #endif
٢٩
٣٠ #endif //NULL_H

```

وعند ترجمة النواة بـ مترجم سي++ فإن القيمة `__cplusplus` تكون معرّفة لديه ، أما في حالة ترجمة النواة بـ مترجم سي فإن المترجم لا يُعرّف تلك القيمة.

### تعريف `size_t`

يتم تعريف `size_t` على أنها عدد صحيح 32-bit بدون إشارة (unsigned).

Example ٥.١٠ : size\_t.h: Definition of size\_t in C/C++

```

١
٢ #ifndef SIZE_T_H
٣ #define SIZE_T_H
٤

```

١٢٣

```

٥ #ifndef __cplusplus
٦ extern "C"
٧ {
٨ #endif
٩
١٠ /* Standard definition of size_t */
١١ typedef unsigned size_t;
١٢
١٣ #ifndef __cplusplus
١٤ }
١٥ #endif
١٦
١٧
١٨ #endif //SIZE_T_H

```

### إعادة تعريف أنواع البيانات

أنواع البيانات (Data Types) تختلف حجمها بحسب المترجم والنظام الذي تم ترجمة البرنامج عليه ، ويفضل أن يتم إعادة تعريفها (typedef) لتوضيح الحجم والنوع في آن واحد .

Example ٥.١١: stdint.h:typedef data type

```

١
٢ #ifndef STDINT_H
٣ #define STDINT_H`
٤
٥ #define __need_wint_t
٦ #define __need_wchar_t
٧
٨ /* Exact-width integer type */
٩ typedef char int8_t;
١٠ typedef unsigned char uint8_t;
١١ typedef short int16_t;
١٢ typedef unsigned short uint16_t;
١٣ typedef int int32_t;
١٤ typedef unsigned int uint32_t;
١٥ typedef long long int64_t;
١٦ typedef unsigned long long uint64_t;
١٧
١٨

```



```

١٩ // to be continue..
٢٠
٢١ #endif //STDINT_H

```

ولدعم ملفات الرأس للغة سي++ فان الملف السابق سيتم تضمينه في ملف cstdint وهي التسمية التي تتبعها السي++ في ملفات الرأس<sup>٩</sup>.

#### Example ٥.١٢: cstdint:C++ typedef data type

```

١
٢ #ifndef CSTDINT_H
٣ #define CSTDINT_H
٤
٥ #include <stdint.h>
٦
٧ #endif //CSTDINT_H

```

### نوع الحرف

ملف ctype.h يحوي العديد من الماكرو (Macros) والتي تحدد نوع الحرف (عدد، حرف، حرف صغير، مسافة، حرف تحكم،... الخ).

#### Example ٥.١٣: ctype.h:determine character type

```

١
٢ #ifndef CTYPE_H
٣ #define CTYPE_H
٤
٥ #ifdef _MSC_VER
٦ #pragma warning (disable:4244)
٧ #endif
٨
٩ #ifdef __cplusplus
١٠ extern "C"
١١ {
١٢ #endif
١٣
١٤ extern char _ctype[];

```

<sup>٩</sup>ملفات الرأس للغة سي++ تتبع نفس هذا الأسلوب لذلك لن يتم ذكرها مجدداً وسنكتفي بذكر ملفات الرأس للغة سي.

```

١٥
١٦ /* constants */
١٧
١٨ #define CT_UP      0x01 // upper case
١٩ #define CT_LOW     0x02 // lower case
٢٠ #define CT_DIG     0x04 // digit
٢١ #define CT_CTL     0x08 // control
٢٢ #define CT_PUN     0x10 // punctuation
٢٣ #define CT_WHT     0x20 // white space (space,cr,lf,tab).
٢٤ #define CT_HEX     0x40 // hex digit
٢٥ #define CT_SP      0x80 // sapce.
٢٦
٢٧ /* macros */
٢٨
٢٩ #define isalnum(c)   ( (_ctype+1)[(unsigned)(c)] & (CT_UP|
    CT_LOW|CT_DIG) )
٣٠ #define isalpha(c)  (( _ctype + 1)[(unsigned)(c)] & (CT_UP
    | CT_LOW))
٣١ #define iscntrl(c)   (( _ctype + 1)[(unsigned)(c)] & (
    CT_CTL))
٣٢
٣٣
٣٤ // to be continue..
٣٥
٣٦ #ifdef __cplusplus
٣٧ }
٣٨ #endif
٣٩
٤٠ #endif // CTYPE_H

```

---

دعم الدوال بعدد غير محدود من الوسائط

## ٧.٥. دالة طباعة المخرجات للنواة

## ٦. المقاطعات Interrupts

المقاطعات هي طريقة لإيقاف المعالج بشكل مؤقت من تنفيذ عملية ما (Current Process) والبدء بتنفيذ أوامر أخرى . وكمثال على ذلك هو عند الضغط على أي حرف في لوحة المفاتيح فان هذا يولد مقاطعة (Interrupt) تأتي كإشارة الى المعالج بأن يوقف ما يعمل عليه حالياً ويحفظ كل القيم التي يحتاجها لكي يستطيع مواصلة ما تم قطعه ، وفي حالة وجود دالة للتعامل مع هذه المقاطعة (مقاطعة لوحة المفاتيح) وتسمى دالة معالجة المقاطعة (Interrupt Handler) أو دالة خدمة المقاطعة (Interrupt Service Routine) فان التنفيذ ينتقل اليها تلقائياً ، و يتم فيها معالجة هذه المقاطعة (مثلاً يتم قراءة الحرف الذي تم ادخاله من متحكم لوحة المفاتيح ومن ثم ارساله الى متغير في الذاكرة) وعندما تنتهي دالة معالجة المقاطعة من عملها فان المعالج يعود ليكمل تنفيذ العملية التي كان يعمل عليها. والمقاطعات إما تكون مقاطعات عتادية (Hardware Interrupt) وتصدر من عتاد الحاسب أو تكون برمجية (Software Interrupt) وتصدر من خلال البرامج عن طريق تعليمة `int n`. كذلك هناك مقاطعات يصدرها المعالج نفسه عند حدوث خطأ ما (مثلاً عن القسمة على العدد صفر أو عند حدوث Page Fault) وتسمى هذه المقاطعات بأخطاء المعالج أو استثناءات المعالج (Exceptions) ويجب معالجة هذه الأخطاء (Error Handler) لأنها توقف عمل النظام في حالة لم تتوفر دالة لمعالجتها.

### ١.٦. المقاطعات البرمجية Software Interrupts

المقاطعات البرمجية هي مقاطعات يتم اطلاقها من داخل البرنامج (عن طريق الأمر `int n`) لنقل التنفيذ الى دالة أخرى تعالج هذه المقاطعة (Interrupt handler)، وغالباً ما تستخدم هذه المقاطعات في برامج المستخدم (Ring3 user mode) للاستفادة من خدمات النظام (مثلاً للقراءة والكتابة في أجهزة الإدخال والإخراج حيث لا توجد طريقة أخرى لذلك في نمط المستخدم).

#### ١.١.٦. المقاطعات في النمط الحقيقي

في النمط الحقيقي عندما يتم تنفيذ أمر المقاطعة (وهو ما يسمى بطلب تنفيذ المقاطعة (Interrupt Request) وتختصر بـ IRQ) فان المعالج يأخذ رقم المقاطعة المطلوب تنفيذها ويذهب بها الى جدول المقاطعات (Interrupt Vector Table) ، هذا الجدول يبدأ من العنوان الحقيقي `0x0` وينتهي عند العنوان `0x3ff`

ويحتوي كل سجل فيه على عنوان دالة معالجة المقاطعة (IR) والتي يجب تنفيذها لتخدم المقاطعة المطلوبة. حجم العنوان هو أربع بايت وتكون كالتالي:

- Byte 0: Low offset address of IR.
- Byte 1: High offset address of IR.
- Byte 2: Low Segment address of IR.
- Byte 3: High Segment Address of IR.

ويتكون الجدول من 256 مقاطعة (وبحسبة بسيطة يكون حجم الجدول هو 1024 بايت وهي ناتجة من ضرب عدد المقاطعات في حجم كل سجل)، بعض منها محجوز والبعض الآخر يستخدمه المعالج والبقية متروكة لمبرمج نظام التشغيل لدعم المزيد من المقاطعات. وبسبب أن الجدول يتكون فقط من عناوين لدوال معالجة المقاطعات فإن هذا يمكننا من وضع الدالة في أي مكان على الذاكرة ومن ثم وضع عناوينها داخل هذا السجل (يتم هذا عن طريق مقاطعات البايوس)، والجدول التالي يوضح IVT والمقاطعات الموجودة فيه.

Base Address	Interrupt Number	Description
0x000	0	Divide by 0
0x004	1	Single step (Debugger)
0x008	2	Non Maskable Interrupt (NMI) Pin
0x00C	3	Breakpoint (Debugger)
0x010	4	Overflow
0x014	5	Bounds check
0x018	6	Undefined Operation Code
0x01C	7	No coprocessor
0x020	8	Double Fault
0x024	9	Coprocessor Segment Overrun
0x028	10	Invalid Task State Segment (TSS)
0x02C	11	Segment Not Present
0x030	12	Stack Segment Overrun
0x034	13	General Protection Fault (GPF)
0x038	14	Page Fault
0x03C	15	Unassigned
0x040	16	Coprocessor error
0x044	17	Alignment Check (486+ Only)
0x048	18	Machine Check (Pentium/586+ Only)
0x05C	19-31	Reserved exceptions
0x068 - 0x3FF	32-255	Interrupts free for software use

## ٢.١.٦. المقاطعات في النمط المحمي

في النمط المحمي يستخدم المعالج جدولاً خاصاً يسمى بجدول واصفات المقاطعات (Interrupt Descriptor Table) ويختصر ب IDT ، هذا الجدول يشابه جدول IVT حيث يتكون من 256 واصفة كل واصفة مخصصة لمقاطعة ما (إذاً الجدول يحوي 256 مقاطعة) ، حجم كل واصفة هو 8 بايت تحوي عنوان دالة معالجة المقاطعة (IR) و نوع الناخب (selector type: code or data) في جدول GDT الذي تعمل عليه دالة معالجة المقاطعة ، بالإضافة الى مستوى الحماية المطلوب والعديد من الخصائص توضحها التركيبة التالية.

- Bits 0-15:
  - Interrupt / Trap Gate: Offset address Bits 0-15 of IR
  - Task Gate: Not used.
- Bits 16-31:
  - Interrupt / Trap Gate: Segment Selector (Useually 0x10)
  - Task Gate: TSS Selector
- Bits 31-35: Not used
- Bits 36-38:
  - Interrupt / Trap Gate: Reserved. Must be 0.
  - Task Gate: Not used.
- Bits 39-41:
  - Interrupt Gate: Of the format 0D110, where D determines size
    - \* 01110 - 32 bit descriptor
    - \* 00110 - 16 bit descriptor
  - Task Gate: Must be 00101
  - Trap Gate: Of the format 0D111, where D determines size
    - \* 01111 - 32 bit descriptor
    - \* 00111 - 16 bit descriptor
- Bits 42-44: Descriptor Privilege Level (DPL)
  - 00: Ring 0
  - 01: Ring 1
  - 10: Ring 2
  - 11: Ring 3

- Bit 45: Segment is present (1: Present, 0:Not present)
- Bits 46-62:
  - Interrupt / Trap Gate: Bits 16-31 of IR address
  - Task Gate: Not used

والمثال التالي يوضح انشاء واصفة واحدة بلغة التجميع حتى يسهل تتبع القيم ، وسيتم كتابة مثال كامل لاحقا بلغة السي.

#### Example ٦.١ : Example of interrupt descriptor

```

١
٢ idt_descriptor:
٣     baseLow      dw    0x0
٤     selector     dw    0x8
٥     reserved     db    0x0
٦     flags        db    0x8e          ; 010001110
٧     baseHi       dw    0x0

```

المتغير الأول baseLow هو أول 16 بت من عنوان دالة معالجة المقاطعة IR ويكمل الجزء الآخر من العنوان المتغير baseHi وفي هذا المثال العنوان هو 0x0. بمعنى أن دالة تخدم المقاطعة ستكون في العنوان 0x0. وبما أن دالة معالجة (تخدم) المقاطعة تحوي شفرة برمجية للتنفيذ وليست بيانات (Data) فإن قيمة المتغير selector يجب أن تكون 0x8 للإشارة إلى ناخب الشفرة (Code Selector) في جدول الواصفات العام (GDT). أما المتغير flags فإن قيمته هي 010001110b دلالة على أن الواصفة هي 32-bit وأن مستوى الحماية هو الحلقة صفر (Ring0). وبعد أن يتم انشاء أغلب الواصفات بشكل متسلسل (في أي مكان على الذاكرة) ، يجب أن ننشئ جدول IDT وهذا يتم عن طريق حفظ عنوان أول واصفة في متغير وليكن idt\_start وعنوان نهاية الواصفات في المتغير idt\_end ومن ثم انشاء مؤشر يسمى idt\_ptr والذي يجب أن يكون في صورة معينة بحيث يحفظ عنوان بداية الجدول ونهايته :

#### Example ٦.٢ : Value to put in IDTR

```

١ idt_ptr:
٢     limit dw idt_end - idt_start ; bits 0-15 is size of idt
٣     base dd idt_start           ; base of idt

```

هذا المؤشر يجب أن يتم تحميله إلى المسجل IDTR (وهو مسجل داخل المعالج) عن طريق تنفيذ الأمر lidt بالشكل التالي lidt [idt\_ptr].

بعد تنفيذ هذا الأمر فإن جدول المقاطعات سيتم استبداله بالجدول الجديد والذي نجد عنوانه بداخل المسجل idtr ، وهذا الأمر لا يُنفَّذ إلا إذا كانت قيمة العلم (CPL flag) هي صفر.

وعند حدوث أي مقاطعة فإن المعالج ينهي الأمر الذي يعمل عليه و يأخذ رقم المقاطعة ويذهب به إلى جدول IDT (عنوان هذا الجدول يتواجد بداخل المسجل IDTR) ، وبعد ذلك يقوم بحساب مكان الوصفة بالمعادلة  $int\_num * 8$  وذلك بسبب أن حجم كل واصفة في جدول IDT هو 8 بايت. وقبل أن ينقل التنفيذ إلى دالة معالجة المقاطعة فإنه يجب أن يقوم بعملية حفظ للمكان الذي توقف فيه حتى يستطيع أن يتابع عمله عندما تعود دالة معالجة المقاطعة . ويتم حفظ الأعلام EFLAGS ومسجل مقطع الشفرة CS ومسجل عنوان التعليمة التالية IP في المكس (Stack) الحالي ، وفي حالة حدوث خطأ ما فإنه يتم دفع شفرة الخطأ (Error Code) إلى المكس أيضاً. وشفرة الخطأ هي بطول 32-bit وتتبع التركيبة التالية.

- Bit 0: External event
  - 0: Internal or software event triggered the error.
  - 1: External or hardware event triggered the error.
- Bit 1: Description location
  - 0: Index portion of error code refers to descriptor in GDT or current LDT.
  - 1: Index portion of error code refers to gate descriptor in IDT.
- Bit 2: GDT/LDT. Only use if the descriptor location is 0.
  - 0: This indicates the index portion of the error code refers to a descriptor in the current GDT.
  - 1: This indicates the index portion of the error code refers to a segment or gate descriptor in the LDT.
- Bits 3-15: Segment selector index. This is an index into the IDT, GDT, or current LDT to the segment or gate selector bring referenced by the error code.
- Bits 16-31: Reserved.

وعندما تنتهي دالة معالجة المقاطعة من عملها فإنه يجب أن تنفذ الأمر `iretd` أو `iret` حتى يتم ارجاع القيم التي تم دفعها إلى المكس (قيم الأعلام FLAGS). وبالتالي يُكْمَل المعالج عمله.

### ٣.١.٦. أخطاء المعالج

خلال تنفيذ المعالج للأوامر فإنه ربما يحدث خطأ ما مما يجعل المعالج يقوم بتوليد استثناء يعرف باستثناء المعالج ، ويوجد له عدة أنواع:

- الخطأ Fault: عندما تعمل دالة معالجة هذا النوع من الاستثناء فربما يتم اصلاح هذا الخطأ ، وعنوان العودة الذي يتم دفعه الى المكس هو عنوان الأمر الذي تسبب في هذا الخطأ.
- الخطأ Trap: عنوان العودة هو عنوان التعليمه التي تلي الأمر الذي تسبب في الخطأ.
- الخطأ Abort: لا يوجد عنوان للعودة ، ولن يكمل البرنامج عمله بعد انتهاء دالة معالجة الخطأ.

والجدول التالي يوضح أخطاء المعالج والمقاطعات التي يقوم بتوليدها.

Interrupt Number	Class	Description
0	Fault	Divide by 0
1	Trap/Fault	Single step
2	Unclassed	Non Maskable Interrupt (NMI) Pin
3	Trap	Breakpoint
4	Trap	Overflow
5	Fault	Bounds check
6	Fault	Invalid OPCode
7	Fault	Device not available
8	Abort	Double Fault
9	Abort	Coprocessor Segment Overrun
10	Fault	Invalid Task State Segment
11	Fault	Segment Not Present
12	Fault	Stack Fault Exception
13	Fault	General Protection Fault
14	Fault	Page Fault
15	-	Unassigned
16	Fault	x87 FPU Error
17	Fault	Alignment Check
18	Abort	Machine Check
19	Fault	SIMD FPU Exception
20-31	-	Reserved
32-255	-	Available for software use

ويجدر بنا الوقوف على ملاحظة كئنا قد ذكرناها في الفصول السابقة وهي إلغاء المقاطعات (بواسطة الأمر cli) عند الانتقال الى النمط المحمي حتى لا يتسبب في حدوث خطأ General Protection Fault وبالتالي توقف النظام عن العمل وسبب ذلك هو أن عدم تنفيذ الأمر cli يعني أن المقاطعات العادية مفعلة وبالتالي أي عتاد يمكنه أن يرسل مقاطعة الى المعالج لكي ينقل التنفيذ الى دالة تخديمها . وعند بداية الانتقال الى النمط المحمي فان جدول المقاطعات IDT لم يتم انشاءه وأي محاولة لاستخدامه سيؤدي الى هذا الخطأ. أحد المتحكمات التي ترسل مقاطعات الى المعالج بشكل ثابت هو متحكم Prprogrammable Interval Timer وتختصر بـ PIT وهي تمثل ساعة النظام System Timer بحيث ترسل مقاطعة بشكل دائم الى المعالج والذي بدوره ينقل التنفيذ الى دالة تخدم هذه المقاطعة . وبسبب أن جدول المقاطعات غير متواجد



في بداية المرحلة الثانية من محمل النظام وكذلك لا توجد دالة لتخديم هذه المقاطعة فان هذا يؤدي الى توقف النظام ، لذلك يجب ايقاف المقاطعات العتادية لحين انشاء جدول المقاطعات وكتابة دوال معالجة المقاطعات. كذلك توجد مشكلة أخرى لبعض المقاطعات العتادية حيث انها تستخدم نفس أرقام المقاطعات التي يستخدمها المعالج للإستثناءات وحلها هو بإعادة برمجة الشريحة المسؤولة عن استقبال الاشارات من العتاد وتحويلها الى مقاطعات وارسالها الى المعالج ، هذه الشريحة تسمى **Programmable Interrupt Controller** وتختصر ب **PIC** ويجب إعادة برمجتها وتغيير ارقام المقاطعات للأجهزة التي تستخدم أرقاماً متشابهة.

وفيما يلي سيتم إنشاء جدول المقاطعات (IDT) باستخدام لغة السي وتوفير ال 256 دالة لمعالجة المقاطعات وحاليا سيقصر عمل الدوال على طباعة رسالة ، وقبل ذلك سنقوم بإنشاء جدول الواصفات العام (GDT) مجدداً (أي سيتم الغاء الجدول الذي قمنا بإنشائه في مرحلة الاقلاع) وبعد ذلك سنبدأ في برمجة متحكم PIC واعادة ترقيم مقاطعات الأجهزة وكذلك برمجة ساعة النظام لارسال مقاطعة بوقت محدد.

#### ٤.١.٦. إنشاء جدول الواصفات العام GDT

الهدف الرئيسي في نواة نظام التشغيل هي المحمولية على صعيد المنصات ، وهذا ما أدى الى اعتماد فكرة طبقة HAL والتي يقبع تحتها كل ما يتعلق بعتاد الحاسب وادارته وكل ما يجعل النظام معتمداً على معمارية معينة أيضاً نجده تحت طبقة HAL ، و جدول الواصفات العام - كما ذكرنا في الفصول السابقة- يحدد ويقسم لنا الذاكرة الرئيسية كأجزاء قابلة للتنفيذ وأجزاء تحوي بيانات وغيرها ، ونظراً لأن إنشاء هذا الجدول يعتمد على معمارية المعالج والأوامر المدعومة فيه فانه يجب ان يقع تحت طبقة HAL<sup>٢</sup> وهذا يعني أن نقل النظام الى معمارية حاسوب آخر يتطلب فقط إعادة برمجة طبقة HAL .

بداية سنبدأ بتصميم الواجهة العامة لطبقة HAL ويجب أن نراعي أن تكون الواجهة مفصولة تماماً عن التطبيق حتى يتمكن أي مطور من إعادة تطبيقها لاحقاً على معمارية حاسوب آخر.

Example ٦.٣: include/hal.h:Hardware Abstraction Layer Interface

```
١
٢ #ifndef HAL_H
٣ #define HAL_H
٤
٥ #ifndef i386
٦ #error "HAL is not implemented in this platform"
٧ #endif
٨
٩ #include <stdint.h>
١٠
١١ #ifdef _MSC_VER
```

<sup>٢</sup>من منظور آخر هذه الجداول (GDT,LDT and IDT) هي جداول للمعالج لذلك يجب أن تكون في طبقة HAL.

```

١٢ #define interrupt __declspec(naked)
١٣ #else
١٤ #define interrupt
١٥ #endif
١٦
١٧ #define far
١٨ #define near
١٩
٢٠
٢١ /* Interface */
٢٢
٢٣ extern int _cdecl hal_init();
٢٤ extern int _cdecl hal_close();
٢٥ extern void _cdecl gen_interrupt(int);
٢٦
٢٧
٢٨ #endif // HAL_H

```

وحالياً واجهة طبقة HAL مكونة من ثلاث دوال تم الإعلان عنها بأنها extern وهذا يعني أن أي تطبيق (Implementation) لهذه الواجهة يجب أن يُعرّف هذه الدوال. الدالة الاولى هي hal\_init() والتي تقوم بتهيئة العتاد وجدول المعالج بينما الدالة الثانية hal\_close() تقوم بعملية الحذف والتحرير وأخيراً الدالة gen\_interrupt() والتي تم وضعها لغرض تجربة إرسال مقاطعة برمجية والتأكد من أن دالة معالجة المقاطعة تعمل كما يرام.

نعود بالحديث الى جدول الواصفات العام (GDT) <sup>٣</sup> حيث سيتم انشائه بلغة السي وهذا ما سيسمح لنا باستخدام تراكيب عالية للتعبير عن الجدول و المؤشر مما يعطي وضوح ومقروئية أكثر في الشفرة. وسوف نحتاج الى تعريف ثلاث دوال <sup>٤</sup>:

- الدالة i386\_gdt\_init: تقوم بتهيئة واصفة خالية وواصفة للشفرة وللبيانات وكذلك انشاء مؤشر الجدول.
- الدالة i386\_gdt\_set\_desc: دالة تهيئة الواصفة حيث تستقبل القيم وتعينها الى الواصفة المطلوبة.
- الدالة gdt\_install: تقوم بتحميل المؤشر الذي يحوي حجم الجدول وعنوان بدايته الى المسجل GDTR.

والشفرة التالية توضح كيفية انشاء الجدول <sup>٥</sup>.

<sup>٣</sup>راجع ١.١.٤.

<sup>٤</sup>لغرض التنظيم والتقسيم لا أكثر ولا أقل.

<sup>٥</sup>راجع شفرة النظام لقراءة ملف الرأس hal/gdt.h.

Example ٦.٤: hal/gdt.cpp:Install GDT

```

١
٢ #include <string.h>
٣ #include "gdt.h"
٤
٥ static struct gdt_desc _gdt[MAX_GDT_DESC];
٦ static struct gdtr _gdtr;
٧
٨
٩ static void gdt_install();
١٠
١١
١٢ static void gdt_install() {
١٣ #ifdef _MSC_VER
١٤     _asm lgdt [_gdtr];
١٥ #endif
١٦ }
١٧
١٨ extern void i386_gdt_set_desc(uint32_t index, uint64_t base,
    uint64_t limit, uint8_t access, uint8_t grand) {
١٩
٢٠     if ( index > MAX_GDT_DESC )
٢١         return;
٢٢
٢٣     // clear the desc.
٢٤     memset((void*)&_gdt[index], 0, sizeof(struct gdt_desc));
٢٥
٢٦     // set limit and base.
٢٧     _gdt[index].low_base = uint16_t(base & 0xffff);
٢٨     _gdt[index].mid_base = uint8_t((base >> 16) & 0xff);
٢٩     _gdt[index].high_base = uint8_t((base >> 24) & 0xff);
٣٠     _gdt[index].limit = uint16_t(limit & 0xffff);
٣١
٣٢     // set flags and grandularity bytes
٣٣     _gdt[index].flags = access;
٣٤     _gdt[index].grand = uint8_t((limit >> 16) & 0x0f);
٣٥     _gdt[index].grand = _gdt[index].grand | grand & 0xf0;
٣٦ }
٣٧
٣٨ extern gdt_desc* i386_get_gdt_desc(uint32_t index) {
٣٩     if ( index >= MAX_GDT_DESC )

```

```
٤٠     return 0;
٤١     else
٤٢         return &_gdt[index];
٤٣ }
٤٤
٤٥ extern int i386_gdt_init() {
٤٦
٤٧     // init _gdt
٤٨     _gdt.limit = sizeof(struct gdt_desc) * MAX_GDT_DESC - 1;
٤٩     _gdt.base = (uint32_t)&_gdt[0];
٥٠
٥١     // set null desc.
٥٢     i386_gdt_set_desc(0,0,0,0,0);
٥٣
٥٤     // set code desc.
٥٥     i386_gdt_set_desc(1,0,0xffffffff,
٥٦         I386_GDT_CODE_DESC | I386_GDT_DATA_DESC | I386_GDT_READWRITE
٥٧         | I386_GDT_MEMORY,    // 10011010
٥٨         I386_GDT_LIMIT_HI | I386_GDT_32BIT | I386_GDT_4K
٥٩         // 11001111
٦٠
٦١ );
٦٢
٦٣     // set data desc.
٦٤     i386_gdt_set_desc(2,0,0xffffffff,
٦٥         I386_GDT_DATA_DESC | I386_GDT_READWRITE | I386_GDT_MEMORY,
٦٦         // 10010010
٦٧         I386_GDT_LIMIT_HI | I386_GDT_32BIT | I386_GDT_4K    //
٦٨         11001111
٦٩
٧٠ );
٧١
٧٢     // install gdt
٧٣     gdt_install();
٧٤
٧٥     return 0;
٧٦ }
```

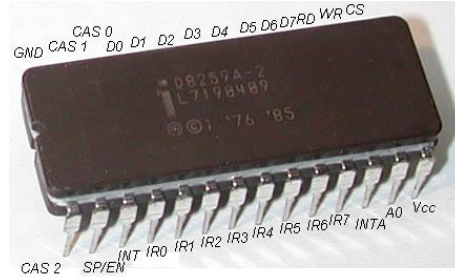
---

### ٥.١.٦ إنشاء جدول المقاطعات IDT

## ٢.٦. متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller

السبب الرئيسي في تعطيل المقاطعات العتادية عند الانتقال الى النمط المحمي (PMode) هو بسبب عدم توفر دوال لمعالجة المقاطعات في تلك اللحظة ، وحتى لو قمنا بتوفير ال ٢٥٦ دالة لمعالجة المقاطعات فان هنالك مشكلة استخدام نفس رقم المقاطعة لأكثر من غرض ، فمثلا مؤقتة النظام PIT التي ترسل مقاطعات بشكل دائم تستخدم المقاطعة رقم ٨ والتي هي أيضا أحد استثناءات المعالج ، لذلك في كلتا الحالات سيتم استدعاء دالة تخدم واحدة وهو شيء مرفوض تماماً. لذلك الحل الوحيد هو بإعادة برمجة المتحكم المسؤول عن استقبال الإشارات من متحكمات العتاد وتعيين أرقام مختلفة بخلاف تلك الأرقام التي يستخدمها المعالج للأخطاء والاستثناءات ، هذا المتحكم (انظر الشكل ١.٦) وظيفته هي استقبال إشارات من متحكمات العتاد ومن ثم يقوم بتحويلها الى أرقام مقاطعات تُرسل بعد ذلك الى المعالج الذي يقوم بنقل التنفيذ إليها ، ويعرف هذا المتحكم بمتحكم PIC اختصاراً ل Programmable Interrupt Controller ويعرف أيضا بالإسم 8259A ، وفي هذا البحث سنستخدم المسمى متحكم PIC.

شكل ١.٦.: متحكم المقاطعات القابل للبرمجة 8259A



### ١.٢.٦ المقاطعات العتادية Hardware Interrupts

قبل أن نبدأ في الدخول في تفاصيل متحكم PIC يجب إعطاء نبذة عن المقاطعات العتادية حيث ذكرنا أنها مقاطعات تختلف عن المقاطعات البرمجية من ناحية أن مصدرها يكون من العتاد وليس من برنامج ما ، وهذا ما أدى الى ظهور لقب مسير للأحداث (Interrupt Driven) على أجهزة الحاسب. حيث قديماً لم يكن هناك طريقة للتعامل مع العتاد إلا باستخدام حلقة برمجية (loop) على مسجل ما في متحكم العتاد حتى تتغير قيمته دلالة على أن هناك قيمة أو نتيجة قد جاءت من العتاد ، هذه الطريقة في التخاطب مع

جدول ١.٦: مقاطعات العتاد لحواسيب x86

رقم المشبك (الدبوس)	رقم المقاطعة	الوصف
IRQ0	0x08	المؤقتة Timer
IRQ1	0x09	لوحة المفاتيح
IRQ2	0x0a	يُربط مع متحكم PIC ثانوي
IRQ3	0x0b	المنفذ التسلسلي ٢
IRQ4	0x0c	المنفذ التسلسلي ١
IRQ5	0x0d	منفذ التوازي ٢
IRQ6	0x0e	متحكم القرص المرن
IRQ7	0x0f	منفذ التوازي ١
IRQ8/IRQ0	0x70	ساعة ال CMOS
IRQ9/IRQ1	0x71	CGA vertical retrace
IRQ10/IRQ2	0x72	محجوزة
IRQ11/IRQ3	0x73	محجوزة
IRQ12/IRQ4	0x74	محجوزة
IRQ13/IRQ5	0x75	وحدة FPU
IRQ14/IRQ6	0x76	متحكم القرص الصلب
IRQ15/IRQ7	0x77	محجوزة

العتاد تسمى Polling<sup>٦</sup> وهي تضيق وقت المعالج في انتظار قيمة لا يُعرف هل ستظهر أم لا وقد تم إلغاؤها في التخاطب مع العتاد حيث الان أصبح أي متحكم عتاد يدعم إرسال الإشارات (وبالتالي المقاطعات) الى المعالج والذي قد يعمل على عملية أخرى ، وهكذا تم الاستفادة من وقت المعالج وأصبح التخاطب هو غير متزامن (Asynchronous) بدلاً من متزامن (Synchronous). وعندما يبدأ الحاسب في الإقلاع فان نظام البايوس يقوم بتقييم عتاد الحاسب وإعطاء رقم مقاطعة لكل متحكم وبسبب تكرار هذه الأرقام فانه يجب تغييرها لأرقام أخرى وهذا يتم بسهولة في النمط الحقيقي وذلك باستخدام مقاطعات البايوس أما في النمط المحمي فيجب أن نقوم بالتخاطب المباشر مع المتحكم الذي لديه أرقام المقاطعات ومن ثم تغييرها . والجدول ١.٦ يوضح أرقام المقاطعات لمتحكمات الحاسب.

## ٢.٢.٦. برمجة متحكم PIC

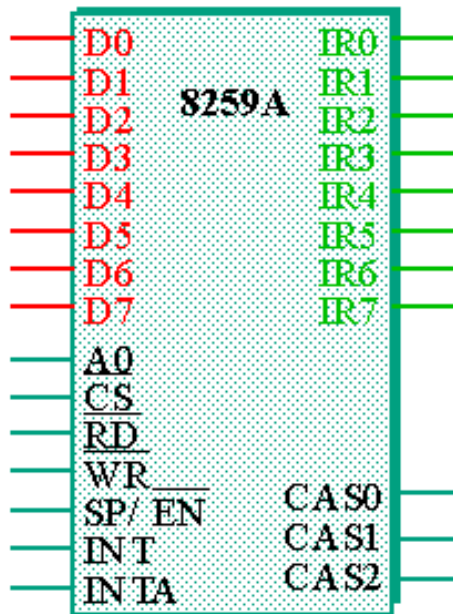
متحكم PIC يستقبل إشارات (Signals) من متحكمات العتاد والتي تكون موصولة به ومن ثم يقوم بتحويلها الى أرقام مقاطعات لكي يقوم المعالج بنقل التنفيذ الى دالة تخدمها ، ويراعي متحكم PIC أولية متحكمات العتاد ، فمثلا لو تم إرسال إشارتين في نفس الوقت الى متحكم PIC فان المتحكم سوف

<sup>٦</sup>وتسمى أيضا ب Busy Waiting.

يراعي الأولوية ويقوم بإرسال رقم مقاطعة العناد ذو الأولوية أولاً وبعد أن تنتهي دالة تقديم المقاطعة يقوم المتحكم بإرسال الرقم الآخر . ونظراً لتعقيدات بناء المتحكم فإنه يتعامل فقط مع ٨ أجهزة مختلفة (أي ٨ مقاطعات IRQ) وهذا ما أدى مصنعي الحاسب إلى توفير متحكم PIC آخر يعرف بالمتحكم الثانوي (Secondary/Slave PIC) . المتحكم الرئيسي (Primary PIC) يوجد داخل المعالج ويرتبط مع المتحكم الثانوي والذي يتواجد في الجسر الجنوبي (SouthBridge) .

### مشابك المتحكم PIC's Pins

تعتبر مشابك المتحكم هي طريقة إرسال البيانات من المتحكم إلى المعالج (أو إلى متحكم رئيسي) ، ونظراً لأن كل مشبك لديه وظيفة محددة فإنه يجب دراسة هذه المشابك ولكن لن نفصل كثيراً حيث أن الموضوع متشعب ويخص دراسي المنطق الرقمي (Digital Logic) . ويوضح الشكل ٢٠٦ هذه المشابك.



شكل ٢٠٦: مشابك متحكم PIC

حيث أن المشابك D0-D7 هي لإرسال البيانات إلى متحكم PIC أما المشابك CAS0, CAS1, CAS2 تستخدم للتخاطب بين متحكمات PIC الرئيسية والثانوية ، والمشبك INT يرتبط مع مشبك للمعالج وهو INTR كذلك المشبك INTA يرتبط مع مشبك المعالج INTA وهذه المشابك لها العديد من الفوائد حيث عندما يقوم المعالج بتنفيذ أي مقاطعة فإنه يقوم بتعطيل قيم العلمين IF and TF وهذا ما يجعل مشبك المعالج INTR يغلق مباشرة وبالتالي لا يمكن لمتحكم PIC إرسال أي مقاطعة عبر مشبكه INT حيث أن الجهة المقابلة لها تم غلقها وبالتالي لا يمكن لمقاطعة أن تقطع مقاطعة أخرى وإنما يتم حجزها في مسجل داخل PIC إلى أن ينتهي المعالج من تنفيذ المقاطعة والعودة بإشارة (تسمى إشارة نهاية المقاطعة End Of Interrupt) تدل على أن المقاطعة قد انتهت. أخيراً ما يهمنا في هذه المشابك هي مشابك IR0...IR7 وهي مشابك ترتبط مع متحكمات العناد المراد استقبال الإشارات منه عند حدوث شيء معين (الضغط على حرف في لوحة المفاتيح مثلاً) ويمكن لهذه المشابك أن ترتبط مع متحكمات PIC أخرى ولا يوجد شرط ينص على وجوب توفر متحكمين PIC وإنما يمكن ربط كل مشبك من هذه المشابك الثمانية مع متحكم PIC وهكذا سيتواجد ٨ متحكمات تدعم حتى ٢٥٦ مقاطعة

جدول ٢.٦: مسجل IRR/ISR/IMR

IRQ Number (Slave controller)	IRQ Number (Primary controller)	Bit Number
IRQ8	IRQ0	0
IRQ9	IRQ1	1
IRQ10	IRQ2	2
IRQ11	IRQ3	3
IRQ12	IRQ4	4
IRQ13	IRQ5	5
IRQ14	IRQ6	6
IRQ15	IRQ7	7

عتادية مختلفة. ويجب ملاحظة أن متحكم العتاد الذي يرتبط بأول مشبك IRO لديه الأولوية الأولى في التنفيذ وهكذا على التوالي.

### مسجلات متحكم PIC

يحتوي متحكم PIC على عدة مسجلات داخلية وهي:

- مسجل الأوامر (Command Register): ويستخدم لإرسال الأوامر إلى المتحكم ، وهناك عدد من الأوامر مثل أمر القراءة من مسجل ما أو أمر إرسال إشارة EOI.
- مسجل الحالة (Status Register): وهو مسجل للقراءة فقط حيث تظهر عليه حالة المتحكم.
- مسجل طلبات المقاطعات (Interrupt Request Register): يحفظ هذا المسجل الأجهزة التي طلبت تنفيذ مقاطعتها وهي بانتظار وصول إشعار (Acknowledges) من المعالج ، والجدول ٢.٦ يوضح بتات هذا المسجل.
- وفي حالة كانت قيمة أي بت هي ١ فهذا يعني أن متحكم العتاد بانتظار الإشعار من المعالج.
- مسجل الخدمة (In Service Register (ISR): يدل على المسجل على أن طلب المقاطعة قد نجح وأن الإشعار قد وصل لكن لم تنتهي دالة تخدم المقاطعة من عملها.
- مسجل (Interrupt Mask Register (IMR): يحدد هذا المسجل ما هي المقاطعات التي يجب تجاهلها وعدم إرسال إشعار لها وذلك حتى يتم التركيز على المقاطعات الأهم.

والجدول ٣.٦ يوضح عناوين منافذ المسجلات في حواسيب x86.



## ٢.٦. متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller

جدول ٣.٦: عناوين المنافذ لمتحكم PIC

الوصف	رقم المنفذ
Primary PIC Command and Status Register	0x20
Primary PIC Interrupt Mask Register and Data Register	0x21
Secondary (Slave) PIC Command and Status Register	0xA0
Secondary (Slave) PIC Interrupt Mask Register and Data Register	0xA1

جدول ٤.٦: الأمر الأول ICW1

الوصف	القيمة	رقم البت
إرسال الأمر ICW4	IC4	0
هل يوجد متحكم PIC واحد	SNGL	1
تأخذ القيمة صفر في حواسيب x86	ADI	2
نمط عمل المقاطعة	LTIM	3
بت التهيئة	1	4
تأخذ القيمة صفر في حواسيب x86	0	5
تأخذ القيمة صفر في حواسيب x86	0	6
تأخذ القيمة صفر في حواسيب x86	0	7

### برمجة متحكم PIC

لبرمجة متحكم PIC وإعادة ترقيم المقاطعات فإن ذلك يتطلب إرسال بعض الأوامر إلى المتحكم بحيث تأخذ هذه الأوامر نمط معين تُحدّد بها عمل المتحكم. وتوجد أربع أوامر تهيئة يجب إرسالها لتهيئة المتحكم تعرف بـ Initialization Control Words وتختصر بأوامر تهيئة ICW ، وكذلك توجد ثلاث أوامر تحكم في عمل متحكم PIC تعرف بـ Operation Control Words وتختصر بـ OCW . وفي حالة توفر أكثر من متحكم PIC على النظام فيجب أن تُرسل أوامر التهيئة إلى المتحكم الآخر كذلك. الأمر الأول **ICW1** وهو أمر التهيئة الرئيسي والذي يجب إرساله أولاً إلى المتحكم الرئيسي والثانوي ويأخذ ٧ بتات ويوضح الجدول ٤.٦ هذه البتات ووظيفة كل بت.

حيث أن البت الأول يحدد ما إذا كان يجب إرسال أمر التحكم ICW4 أم لا وفي حالة كان قيمة البت هي ١ فإنه يجب إرسال الأمر ICW4 أما البت الثاني فغالباً يأخذ القيمة صفر دلالة على أن هناك أكثر من متحكم PIC في النظام ، والبت الثالث غير مستخدم أما الرابع فيحدد نمط عمل المقاطعة هل هي Level Triggered Mode أم Edge Triggered Mode ، أما البت الخامس فيجب أن يأخذ القيمة ١ دلالة على أننا سنقوم بتهيئة متحكم PIC وبقية البتات غير مستخدمة في حواسيب x86. والشفرة ٦.٥ توضح إرسال الأمر الأول إلى متحكم PIC الرئيسي والثانوي.

## Example ٦.٥: Initialization Control Words 1

```

١ ; Setup to initialize the primary PIC. Send ICW 1
٢ mov al, 0x11 ; 00010001
٣ out 0x20, al
٤
٥ ; Send ICW 1 to second PIC command register
٦ out 0xA0, al

```

**الأمر الثاني ICW2** يستخدم لإعادة تغيير عناوين جدول IVT الرئيسية للطلبات المقاطعات IRQ وبالتالي عن طريق هذا الأمر يمكن أن نغير أرقام المقاطعات لل IRQ الى أرقام أخرى . ويجب أن يرسل هذا الأمر مباشرة بعد الأمر الأول كذلك يجب أن يتم اختيار أرقاماً غير مستخدمة من قبل المعالج حتى لا تقع في نفس المشكلة السابقة ( وهي أكثر من IRQ يستخدم نفس رقم المقاطعة وبالتالي لديهم دالة تخديم واحدة). والمثال ٦.٦ يوضح كيفية تغيير أرقام IRQ لمتحكم PIC الرئيسي والثانوي بحيث يتم استخدام أرقام المقاطعات ٣٢-٣٩ للمتحكم الأول والأرقام من ٤٠-٤٧ للمتحكم الثانوي وهي أرقاماً خالية لا يستخدمها المعالج وتقع مباشرة بعد آخر مقاطعة للمعالج الذي يستخدم ٣٢ مقاطعة بدءاً من الصفر وانتهاءً بالمقاطعة ٣١.

## Example ٦.٦: Initialization Control Words 2

```

١ ; send ICW 2 to primary PIC
٢ mov al, 0x20
٣ out 0x21, al
٤ ; Primary PIC handled IRQ 0..7. IRQ 0 is now mapped to
   interrupt number 0x20
٥
٦
٧ ; send ICW 2 to secondary PIC
٨ mov al, 0x28
٩ out 0xA1, al
١٠ ; Secondary PIC handles IRQ's 8..15. IRQ 8 is now mapped
    to use interrupt 0x28

```

**الأمر الثالث ICW3** يستخدم في حالة كان هناك أكثر من متحكم PIC حيث يجب أن نحدد رقم طلب المقاطعة IRQ التي يستخدمها المتحكم الثانوي للتخاطب مع المتحكم الرئيسي. وفي حواسيب x86 غالباً ما يستخدم IRQ2 لذا يجب إرسال هذا الأمر الى المتحكم، لكن كل متحكم يتوقع الأمر بصيغة معينة يوضحها الجدولان ٥.٦ و ٦.٦ . ويجب إرسال الأمر بحسب الصيغة التي يقبلها مسجل البيانات للمتحكم ، فمتحكم PIC الرئيسي يستقبل رقم IRQ على شكل ٧ بت بحيث يتم تفعيل رقم البت المقابل لرقم IRQ وفي مثالنا يرتبط المتحكم الرئيسي

## ٢.٦. متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller

جدول ٥.٥.٦: الأمر الثالث للمتحكم الرئيسي ICW3 for Primary PIC

رقم البت	القيمة	الوصف
0-7	S0-S7	رقم IRQ التي يتصل بها المتحكم الثانوي

جدول ٦.٦: الأمر الثالث للمتحكم الثانوي ICW3 for Slave PIC

رقم البت	القيمة	الوصف
0-2	ID0	رقم IRQ التي يتصل بها مع المتحكم الرئيسي
3-7	3-7	محمولة

مع الثانوي عبر IRQ2 لذلك يجب تفعيل قيمة البت ٢ (أي يجب إرسال القيمة 0000100b وهي تعادل 0x4) بينما المتحكم الثانوي يقبل رقم IRQ عن طريق إرسال قيمته على الشكل الثنائي وهي ٢ (وتعادل بالترميز الثنائي 010) وبقيّة البتات محمولة (انظر جدول ٦.٦)، والمثال ٦.٧ يوضح كيفية إرسال الأمر الثالث إلى المتحكمين.

### Example ٦.٧: Initialization Control Words 3

```
١ ; Send ICW 3 to primary PIC
٢ mov al, 0x4 ; 0x04 => 0100, second bit (IR line 2)
٣ out 0x21, al ; write to data register of primary PIC
٤
٥ ; Send ICW 3 to secondary PIC
٦ mov al, 0x2 ; 010=> IR line 2
٧ out 0xA1, al ; write to data register of secondary PIC
```

الأمر الرابع ICW4 هو آخر أمر تحكم يجب إرساله إلى المتحكمين ويأخذ التركيبة التي يوضحها جدول ٧.٦. وفي الغالب لا يوجد حوجة لتفعيل كل هذه الخصائص، فقط أول بت يجب تفعيله حيث يستخدم مع حواسيب x86. والمثال ٦.٨ يوضح كيفية إرسال الأمر الرابع إلى المتحكم الرئيسي والثانوي.

### Example ٦.٨: Initialization Control Words 4

```
١ mov al, 1 ; bit 0 enables 80x86 mode
٢
٣ ; send ICW 4 to both primary and secondary PICs
٤ out 0x21, al
٥ out 0xA1, al
```

جدول ٧.٦: الأمر الرابع ICW4

رقم البت	القيمة	الوصف
0	uPM	يجب تفعيل هذا البت في حواسيب x86
1	AEOI	جعل المتحكم يقوم بإرسال إشارة EOI
2	M/S	If set (1), selects buffer master. Cleared if buffer slave.
3	BUF	If set, controller operates in buffered mode
4	SFNM	تأخذ القيمة صفر في حواسيب x86
5-7	0	تأخذ القيمة صفر في حواسيب x86

جدول ٨.٦: أمر التحكم الثاني OCW2

رقم البت	القيمة	الوصف
0-2	L0/L1/L2	Interrupt level upon which the controller must react
3-4	0	محجوزة
5	EOI	End of Interrupt (EOI)
6	SL	Selection
7	R	Rotation option

وبعد إرسال هذه الأوامر الأربع تكتمل عملية تهيئة متحكم PIC الرئيسي والثانوي ، وفي حالة حدوث أي مقاطعة من متحكم لعتاد ما ، فإن أرقام المقاطعات التي سترسل إلى المعالج هي الأرقام التي قمنا بتعيينها في الأمر الثاني (وتبدأ من ٣٢ إلى ٤٧) وهي تختلف بالطبع عن الأرقام التي يستخدمها المعالج. وبخصوص أوامر التحكم الثلاث OCW فلن نحتاج إليها جميعاً وسيتم الحديث عن الأمر الثاني OCW2 نظراً لأنه يجب أن يُرسل دائماً بعد أن تنتهي دالة تخدم المقاطعة من عملها وذلك حتى يتم السماح لبقية المقاطعات أن تأخذ دوراً لمعالجتها. والجدول ٨.٦ يوضح البتات التي يجب إرسالها إلى مسجل التحكم . ويهملنا البتات ٥-٧ حيث أن قيمهم تحدد بعض الخصائص التي يوضحها الجدول ٩.٦. والمثال ٦.٩ يوضح كيفية إرسال إشارة نهاية عمل دالة تخدم المقاطعة (EOI) حيث يجب ضبط البتات لإختيار Non specific EOI command.

## Example ٦.٩: Send EOI

```

١ ; send EOI to primary PIC
٢
٣ mov al, 0x20 ; set bit 4 of OCW 2
٤ out 0x20, al ; write to primary PIC command register

```

جدول ٩.٦: أمر OCW2

Description	EOI Bit	SL Bit	R Bit
Rotate in Automatic EOI mode (CLEAR)	0	0	0
Non specific EOI command	1	0	0
No operation	0	1	0
Specific EOI command	1	1	0
Rotate in Automatic EOI mode (SET)	0	0	1
Rotate on non specific EOI	1	0	1
Set priority command	0	1	1
Rotate on specific EOI	1	1	1

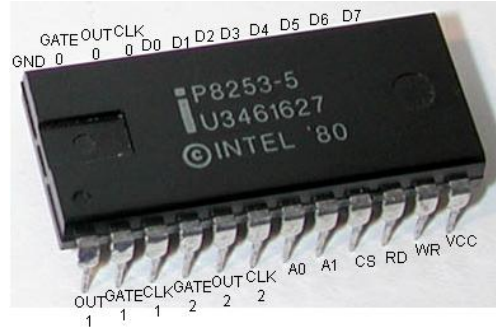
### كيف تعمل مقاطعات العتاد

عندما يحتاج متحكم أي عتاد لفت انتباه المعالج إلى شيء ما فأول خطوة يقوم بها هي إرسال إشارة إلى متحكم PIC (وعلى سبيل المثال سنفرض أن هذا المتحكم هو متحكم المؤقتة PIT والتي ترتبط بالمشبك IRQ) هذه الإشارة ترسل عبر مشبك IRQ، حينها يقوم متحكم PIC بتسجيل طلب المتحكم IRQ في مسجل يسمى مسجل طلبات المقاطعات (Interrupt Request Register) ويعرف اختصاراً بمسجل IRR. هذا المسجل بطول ٨ بت كل بت فيه يمثل رقم IRQ ويتم تفعيل أي بت عند طلب مقاطعة من المتحكم، وفي مثالنا سيتم تفعيل البت 0 بسبب أن المؤقتة ترتبط مع IRQ. بعد ذلك يقوم متحكم PIC بفحص مسجل Interrupt Mask Register ليتأكد من أنه لا توجد هناك مقاطعة ذات أولوية أعلى حيث في هذه الحالة على المقاطعة الجديدة أن تنتظر حتى يتم تخدم كل المقاطعات ذات الأولوية. وبعد ذلك يُرسل PIC إشارة إلى المعالج من خلال مشبك INTA لأخبار المعالج بأن هناك مقاطعة يجب تنفيذها. وهنا يأتي دور المعالج حيث يقوم بالإنتهاء من تنفيذ الأمر الحالي الذي يعمل عليه ومن ثم يقوم بفحص قيمة العلم IF حيث في حالة كانت غير مفعلة فإن المعالج سوف يتجاهل طلب تنفيذ المقاطعة، أما إذا وجد المعالج قيمة العلم مفعلة فإنه يقوم بإرسال إشعار (Acknowledges) عبر مشبك INTR إلى متحكم PIC الذي بدوره يستقبلها من مشبك INTA ويضع رقم المقاطعة ورقم IRQ في المشابك D0-D7، وأخيراً يفعل قيمة البت ٠ في مسجل In Service Register دلالة على أن مقاطعة المؤقتة جاري تنفيذها. وعندما يحصل المعالج على رقم المقاطعة فإنه يقوم بوقف العملية التي يعمل عليها ويحفظ قيم مسجل الأعلام ومسجل CS and EIP وإذا كان المعالج يعمل في النمط الحقيقي فإنه يأخذ رقم المقاطعة ويذهب بها كدليل إلى جدول المقاطعات IVT حيث يجد عنوان دالة تخدم المقاطعة ومن ثم ينقل التنفيذ إليها، أما إذا كان المعالج يعمل في النمط المحمي فإنه يأخذ رقم المقاطعة ويذهب بها إلى جدول واصفات المقاطعات حيث يجد دالة تخدم المقاطعة. وعندما تنتهي دالة تخدم المقاطعة من عملها فإنها يجب أن ترسل إشارة EOI حتى يتم تفعيل المقاطعات مجدداً.

### ٣.٦. المؤقتة Programmable Interval Timer

المؤقتة هي شريحة Dual Inline Package (DIP) تحوي ثلاث عدادات (Counters or Channels) تعمل كمؤقتات لإدارة ثلاث أشياء (انظر الشكل ٣.٦). العداد الأول ويُعرف بمؤقت النظام (System Timer) وظيفته إرسال طلب مقاطعة (IRQ0) إلى متحكم PIC وذلك لتنفيذ مقاطعة ما كل فترة محددة ، هذه الفترة يتم تحديدها عند برمجة هذه المؤقتة ويُستفاد من هذه المؤقتة في عملية تزامن العمليات وتوفير بنية تحتية لمفهوم تعدد العمليات والمسالك (Multitask and Multithread) حيث أن الفترة التي تقوم بها مؤقتة النظام لإصدار طلب المقاطعة سيكون هو الوقت المحدد لأي عملية (Process) موجودة في طابور العمليات (Process Queue) وبعد ذلك تُرسل العملية إلى آخر الصف في حالة لم تنتهي من عملها بعد ويبدأ المعالج في تنفيذ العملية التالية تحت نفس الفترة المحددة. أما العداد الثاني فيُستخدم في عملية تنعيش الذاكرة الرئيسية (RAM refreshing) حتى تحافظ على محتوياتها من الفقدان أثناء عمل الحاسب ويجدر بنا ذكر أن هذه المهمة قد أُحيلت إلى متحكم الذاكرة (Memory Controller) وأصبحت هذه المؤقتة لا تستخدم في العادة. أما العداد الأخير فيستخدم في عملية إرسال الصوت إلى سماعات الحاسب<sup>٧</sup> (PC Speaker).

شكل ٣.٦: المؤقتة القابلة للبرمجة 8253

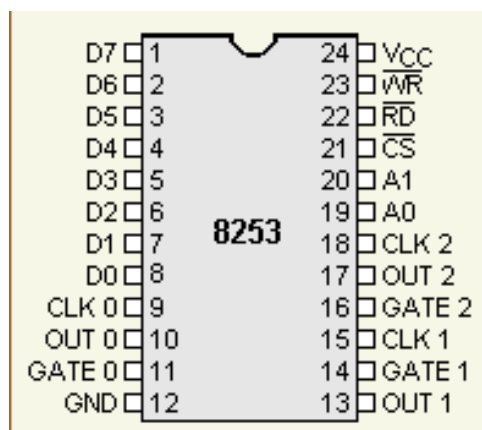


#### ١.٣.٦. برمجة المؤقتة PIT

مؤخراً تم نقل المؤقتة من اللوحة الأم (MotherBoard) كشرريحة DIP مستقلة إلى الجسر الشمالي (SouthBridge). وسوف نركز على برمجة العداد الأول وهو مؤقت النظام حيث أنه يوفر الدعم العتادي اللازم للنظام حتى يدعم تعدد العمليات والمسالك.

<sup>٧</sup> لا يُقصد بهذه كرت الصوت وإنما يوجد في كل حاسب سماعات داخلية تستخدم في إصدار الصوت والنعيمات وأحد استخداماتها لإصدار رسائل الخطأ بعد عملية فحص الحاسب (POST) في مرحلة الإقلاع.

### مشابك المؤقتة PIT's Pins



شكل ٤.٦: مشابك المؤقتة PIT

تُرسل الأوامر والبيانات الى المؤقتة وذلك عبر مسار البيانات (Data Bus) حيث يرتبط هذا المسار مع مشابك البيانات في المؤقتة وهي ٨ مشابك D0...D7 وتمثل ٨ بتات. وعند إرسال بيانات الى المؤقتة (عملية كتابة) فان مشبك الكتابة WR يأخذ قيمة منخفضة دلالة على أن هناك عملية إرسال بيانات الى المؤقتة وكذلك في حالة قراءة بيانات من المؤقتة فإن مشبك القراءة RD يأخذ قيمة منخفضة دلالة على أن هناك عملية قراءة من المؤقتة. ويتحكم في مشبك القراءة والكتابة مشبك CS حيث تحدد قيمته تعطيل أو تفعيل عمل الشبكين السابقين ، ويرتبط مشبك CS مع مسار العناوين (Address Bus) بينما يرتبط مشبك القراءة والكتابة مع مسار التحكم

(Control Bus). وتحدد قيمة المشبكين A0,A1

واللذان يرتبطان مع مسار العناوين- المسجلات المطلوب الوصول اليها داخل المؤقتة. أما المشابك (CLK, OUT, and GATE) فهي لكل عداد بداخل المؤقتة أي بمعنى أنه توجد ثلاث مشابك من كل واحدة منهم ، ويعتبر المشبكين (CLK (Clock Input) and GATE) مشابك إدخال للعداد بينما المشبك (OUT) مشبك إخراج حيث يستخدم لربط العداد مع العناد فمثلا مشبك الإخراج في العداد الأول (مؤقتة النظام) يرتبط مع متحكم PIC حيث من خلاله تستطيع مؤقتة النظام إرسال طلب المقاطعة (IRQ0) الى متحكم PIC والذي يقوم بتحويل الطلب الى المعالج لكي ينفذ دالة التخدم.

### مسجلات المؤقتة PIT

توجد ٤ مسجلات بداخل المؤقتة PIT ، ثلاث منها تستخدم للعدادات (الأول والثاني والثالث) حيث من خلالها يمكن قراءة قيمة العداد أو الكتابة فيه ، وطول مسجل العداد هو ١٦ بت . ويسبب أن مشابك البيانات التي تربط المؤقتة ومسار البيانات هي من الطول ٨ بت فانه لن تتمكن من إرسال البيانات بهذا الشكل . لذلك يجب إستخدام مسجل اخر وهو مسجل التحكم (Control Word) بحيث قبل إرسال بيانات أو قراءة بيانات من أي عداد فانه يجب إرسال الأمر المطلوب الى مسجل التحكم وبعد ذلك يتم إرسال البيانات أو قرائتها. والجدول ١٠.٦ يوضح هذا المسجلات وعنوان منافذ الإدخال والإخراج المستخدمة للتعامل معها ، ويجب ملاحظة قيم خط القراءة والكتابة وخط العنوان (A0,A1) حيث تؤثر قيمهم في تحديد نوع العملية المطلوبة (قراءة أم كتابة ورقم العداد). وتوضح التركيبة التالية ماهية البتات المستخدمة في مسجل التحكم (وهو مسجل بطول ٨ بت) حيث يجب إرسال قيم معينة حتى تتمكن من

جدول ١٠.٦: مسجلات المؤقتة 8253 PIT

اسم المسجل	رقم المنفذ	خط RD	خط WR	خط A0	خط A1	الوظيفة
Counter 0	0x40	1	0	0	0	كتابة الى المسجل 0 قراءة المسجل 0
Counter 1	0x41	1	0	0	1	كتابة الى المسجل 1 قراءة المسجل 1
Counter 2	0x42	1	0	1	0	كتابة الى المسجل 2 قراءة المسجل 2
Control Word	0x43	1	0	1	1	كتابة Control Word لا توجد عملية

القراءة أو الكتابة في عداد ما.

- Bit 0: (BCP) Binary Counter
  - 0: Binary
  - 1: Binary Coded Decimal (BCD)
- Bit 1-3: (M0, M1, M2) Operating Mode. See above sections for a description of each.
  - 000: Mode 0: Interrupt or Terminal Count
  - 001: Mode 1: Programmable one-shot
  - 010: Mode 2: Rate Generator
  - 011: Mode 3: Square Wave Generator
  - 100: Mode 4: Software Triggered Strobe
  - 101: Mode 5: Hardware Triggered Strobe
  - 110: Undefined; Don't use
  - 111: Undefined; Don't use
- Bits 4-5: (RL0, RL1) Read/Load Mode. We are going to read or send data to a counter register
  - 00: Counter value is latched into an internal control register at the time of the I/O write operation.
  - 01: Read or Load Least Significant Byte (LSB) only
  - 10: Read or Load Most Significant Byte (MSB) only
  - 11: Read or Load LSB first then MSB



- Bits 6-7: (SC0-SC1) Select Counter. See above sections for a description of each.
  - 00: Counter 0
  - 01: Counter 1
  - 10: Counter 2
  - 11: Illegal value

والمثال ٦.١٠ يوضح كيفية برمجة عداد مؤقت النظام لإرسال طلب مقاطعة كل 100Hz (كل ١٠ milliseconds) ، وهذا يتم عن طريق إرسال أمر التحكم أولاً ومن ثم إرسال الوقت المطلوب الى العداد المطلوب.

#### Example ٦.١٠: PIT programming

```
١          ; COUNT = input hz / frequency
٢
٣  mov dx, 1193180 / 100 ; 100hz, or 10 milliseconds
٤
٥  ; FIRST send the command word to the PIT. Sets binary
   counting,
٦  ; Mode 3, Read or Load LSB first then MSB, Channel 0
٧
٨  mov al, 110110b
٩  out 0x43, al
١٠
١١ ; Now we can write to channel 0. Because we set the "Load
   LSB first then MSB" bit, that is
١٢ ; the way we send it
١٣
١٤ mov ax, dx
١٥ out 0x40, al ;LSB
١٦ xchg ah, al
١٧ out 0x40, al ;MSB
```

---

## ٤.٦. توسعة طبقة HAL

طبقة HAL تبعد نواة النظام من التعامل المباشر مع العتاد وتعمل كواجهة أو طبقة ما بين النواة والعتاد ، وفيها نجد تعريفات العتاد. وسيتم إضافة أوامر برمجة متحكم PIC التي تقوم بإعادة تعيين أرقام المقاطعات

بداخل هذه الطبقة وكذلك سيتم إضافة شفرة برمجة المؤقتة و التي تحدد الوقت اللازم للمؤقتة لكي تقوم بإرسال طلب المقاطعة (IRQ0) .

### ٦.٤.١. دعم PIC

في القسم ٢.٢.٦ تم عرض متحكم PIC وكيفية برمجته بالتفصيل ، وفي هذا القسم سيتم تطبيق ما تم عرضه على نواة نظام إقرأ. ويوجد ملفين لمتحكم PIC الأول هو ملف الرأس (hal/pic.h) الذي يحوي الإعلان عن الدوال وكذلك الثوابت والثاني هو ملف التطبيق (hal/pic.cpp) الذي يحوي على تعريف تلك الدوال. والمثال ٦.١١ يعرض ملف الرأس الذي يغلف العديد من الأرقام والعناوين في صورة ثوابت (باستخدام الماكرو) بحيث تزيد من مقروئية ووضوح الشفرة<sup>٨</sup>.

Example ٦.١١: hal/pic.h: PIC Interface

```

١ // PIC 1 Devices IRQ
٢ #define I386_PIC_IRQ_TIMER          0
٣ #define I386_PIC_IRQ_KEYBOARD      1
٤ #define I386_PIC_IRQ_SERIAL2       3
٥ #define I386_PIC_IRQ_SERIAL1       4
٦ #define I386_PIC_IRQ_PARALLEL2     5
٧ #define I386_PIC_IRQ_DESKETTE      6
٨ #define I386_PIC_IRQ_PARALLEL1     7
٩
١٠ // PIC 2 Devices IRQ
١١ #define I386_PIC_IRQ_CMOSTIMER     0
١٢ #define I386_PIC_IRQ_CGARETRACE   1
١٣ #define I386_PIC_IRQ_AUXILIARY    4
١٤ #define I386_PIC_IRQ_FPU          5
١٥ #define I386_PIC_IRQ_HDC          6
١٦
١٧ // Operation Command Word 2 (OCW2)
١٨ #define I386_PIC_OCW2_MASK_L1     1
١٩ #define I386_PIC_OCW2_MASK_L2     2
٢٠ #define I386_PIC_OCW2_MASK_L3     4
٢١ #define I386_PIC_OCW2_MASK_EOI    0x20
٢٢ #define I386_PIC_OCW2_MASK_SL     0x40
٢٣ #define I386_PIC_OCW2_MASK_ROTATE 0x80
٢٤
٢٥

```

<sup>٨</sup>إرجع الى القسم ٢.٢.٦ لمعرفة وظيفة هذه القيم الثابتة.

```

٢٦ // Operation Command Word 3 (OCW3)
٢٧ #define I386_PIC_OCW3_MASK_RIS      1
٢٨ #define I386_PIC_OCW3_MASK_RIR      2
٢٩ #define I386_PIC_OCW3_MASK_MODE      4
٣٠ #define I386_PIC_OCW3_MASK_SMM      0x20
٣١ #define I386_PIC_OCW3_MASK_ESMM     0x40
٣٢ #define I386_PIC_OCW3_MASK_D7      0x80
٣٣
٣٤
٣٥ // PIC 1 port address
٣٦ #define I386_PIC1_COMMAND_REG        0x20
٣٧ #define I386_PIC1_STATUS_REG         0x20
٣٨ #define I386_PIC1_IMR_REG            0x21
٣٩ #define I386_PIC1_DATA_REG           0x21
٤٠
٤١
٤٢ // PIC 2 port address
٤٣ #define I386_PIC2_COMMAND_REG        0xa0
٤٤ #define I386_PIC2_STATUS_REG         0xa0
٤٥ #define I386_PIC2_IMR_REG            0xa1
٤٦ #define I386_PIC2_DATA_REG           0xa1
٤٧
٤٨ // Initializing Command Word 1 (ICW1) Mask
٤٩ #define I386_PIC_ICW1_MASK_IC4       0x1
٥٠ #define I386_PIC_ICW1_MASK_SNGL      0x2
٥١ #define I386_PIC_ICW1_MASK_ADI       0x4
٥٢ #define I386_PIC_ICW1_MASK_LTIM      0x8
٥٣ #define I386_PIC_ICW1_MASK_INIT      0x10
٥٤
٥٥
٥٦ // Initializing Command Word 4 (ICW4) Mask
٥٧ #define I386_PIC_ICW4_MASK_UPM       0x1
٥٨ #define I386_PIC_ICW4_MASK_AEOI     0x2
٥٩ #define I386_PIC_ICW4_MASK_MS       0x4
٦٠ #define I386_PIC_ICW4_MASK_BUF       0x8
٦١ #define I386_PIC_ICW4_MASK_SFNM      0x10
٦٢
٦٣
٦٤ // Initializing command 1 control bits
٦٥ #define I386_PIC_ICW1_IC4_EXPECT     1
٦٦ #define I386_PIC_ICW1_IC4_NO         0
٦٧ #define I386_PIC_ICW1_SNGL_YES       2

```

```

٦٨ #define I386_PIC_ICW1_SNGL_NO          0
٦٩ #define I386_PIC_ICW1_ADI_CALLINTERVAL4  4
٧٠ #define I386_PIC_ICW1_ADI_CALLINTERVAL8  0
٧١ #define I386_PIC_ICW1_LTIM_LEVELTRIGGERED 8
٧٢ #define I386_PIC_ICW1_LTIM_EDGETRIGGERED  0
٧٣ #define I386_PIC_ICW1_INIT_YES          0x10
٧٤ #define I386_PIC_ICW1_INIT_NO           0
٧٥
٧٦ // Initializing command 4 control bits
٧٧ #define I386_PIC_ICW4_UPM_86MODE         1
٧٨ #define I386_PIC_ICW4_UPM_MCSMODE       0
٧٩ #define I386_PIC_ICW4_AEOI_AUTOEOI      2
٨٠ #define I386_PIC_ICW4_AEOI_NOAUTOEOI    0
٨١ #define I386_PIC_ICW4_MS_BUFFERMASTER  4
٨٢ #define I386_PIC_ICW4_MS_BUFFERSLAVE    0
٨٣ #define I386_PIC_ICW4_BUF_MODEYES       8
٨٤ #define I386_PIC_ICW4_BUF_MODENO        0
٨٥ #define I386_PIC_ICW4_SFNM_NESTEDMODE    0x10
٨٦ #define I386_PIC_ICW4_SFNM_NOTNESTED    0
٨٧
٨٨
٨٩ extern uint8_t i386_pic_read_data(uint8_t pic_num);
٩٠ extern void i386_pic_send_data(uint8_t data, uint8_t pic_num)
    ;
٩١ extern void i386_pic_send_command(uint8_t cmd, uint8_t
    pic_num);
٩٢ extern void i386_pic_init(uint8_t base0, uint8_t base1);

```

وتحتوي الواجهة ٤ دوال منها دالتان للقراءة والكتابة من مسجل البيانات ودالة لإرسال الأوامر الى مسجل التحكم والدالة الأخيرة هي لتهيئة المتحكم وهي الدالة التي يجب استدعاؤها. والمثال ٦.١٢ يوضح تعريف هذه الدوال.

#### Example ٦.١٢: hal/pic.cpp: PIC Implementation

```

١ uint8_t i386_pic_read_data(uint8_t pic_num) {
٢     if (pic_num > 1)
٣         return 0;
٤
٥     uint8_t reg = (pic_num == 1)?I386_PIC2_DATA_REG:
        I386_PIC1_DATA_REG;
٦     return inportb(reg);
٧ }

```

```

٨
٩ void i386_pic_send_data(uint8_t data,uint8_t pic_num) {
١٠     if (pic_num > 1)
١١         return;
١٢
١٣     uint8_t reg = (pic_num == 1)?I386_PIC2_DATA_REG:
        I386_PIC1_DATA_REG;
١٤     outportb(reg,data);
١٥ }
١٦
١٧ void i386_pic_send_command(uint8_t cmd,uint8_t pic_num) {
١٨
١٩     if (pic_num > 1)
٢٠         return;
٢١
٢٢     uint8_t reg = (pic_num == 1)?I386_PIC2_COMMAND_REG:
        I386_PIC1_COMMAND_REG;
٢٣     outportb(reg,cmd);
٢٤ }
٢٥
٢٦
٢٧
٢٨ void i386_pic_init(uint8_t base0,uint8_t base1) {
٢٩
٣٠     uint8_t icw = 0;
٣١
٣٢     disable_irq();          /* disable hardware interrupt (cli) */
٣٣
٣٤     /* init PIC, send ICW1 */
٣٥     icw = (icw & ~I386_PIC_ICW1_MASK_INIT) |
        I386_PIC_ICW1_INIT_YES;
٣٦     icw = (icw & ~I386_PIC_ICW1_MASK_IC4) |
        I386_PIC_ICW1_IC4_EXPECT;
٣٧     /* icw = 0x11 */
٣٨
٣٩     i386_pic_send_command(icw,0);
٤٠     i386_pic_send_command(icw,1);
٤١
٤٢     /* ICW2 : remapping irq */
٤٣     i386_pic_send_data(base0,0);
٤٤     i386_pic_send_data(base1,1);
٤٥

```

```

٤٦  /* ICW3 : irq for master/slave pic*/
٤٧  i386_pic_send_data(0x4,0);
٤٨  i386_pic_send_data(0x2,1);
٤٩
٥٠  /* ICW4: enable i386 mode. */
٥١  icw = (icw & ~I386_PIC_ICW4_MASK_UPM) |
        I386_PIC_ICW4_UPM_86MODE ; /* icw = 1 */
٥٢  i386_pic_send_data(icw,0);
٥٣  i386_pic_send_data(icw,1);
٥٤
٥٥  }

```

## ٦.٤.٢. دعم PIT

## Example ٦.١٣: hal/pit.h: Pit Interface

```

١  #define I386_PIT_COUNTER0_REG      0x40
٢  #define I386_PIT_COUNTER1_REG      0x41
٣  #define I386_PIT_COUNTER2_REG      0x42
٤  #define I386_PIT_COMMAND_REG       0x43
٥
٦  #define I386_PIT_OCW_MASK_BINCOUNT 0x1
٧  #define I386_PIT_OCW_MASK_MODE      0xe
٨  #define I386_PIT_OCW_MASK_RL       0x30
٩  #define I386_PIT_OCW_MASK_COUNTER  0xc0
١٠
١١
١٢ #define I386_PIT_OCW_BINCOUNT_BINARY 0x0
١٣ #define I386_PIT_OCW_BINCOUNT_BCD   0x1
١٤
١٥ #define I386_PIT_OCW_MODE_TERMINALCOUNT 0x0
١٦ #define I386_PIT_OCW_MODE_ONESHOT      0x2
١٧ #define I386_PIT_OCW_MODE_RATEGEN      0x4
١٨ #define I386_PIT_OCW_MODE_SQUAREWAVEGEN 0x6
١٩ #define I386_PIT_OCW_MODE_SOFTWARETRIG 0x8
٢٠ #define I386_PIT_OCW_MODE_HARDWARETRIG 0xa
٢١
٢٢ #define I386_PIT_OCW_RL_LATCH          0x0
٢٣ #define I386_PIT_OCW_RL_LSBONLY       0x10

```

```

٢٤ #define I386_PIT_OCW_RL_MSBONLY      0x20
٢٥ #define I386_PIT_OCW_RL_DATA        0x30
٢٦
٢٧ #define I386_PIT_OCW_COUNTER_0      0x0
٢٨ #define I386_PIT_OCW_COUNTER_1      0x40
٢٩ #define I386_PIT_OCW_COUNTER_2      0x80
٣٠
٣١ extern void i386_pit_send_command(uint8_t cmd);
٣٢ extern void i386_pit_send_data(uint16_t data,uint8_t counter
    );
٣٣ extern uint8_t i386_pit_read_data(uint16_t counter);
٣٤ extern uint32_t i386_pit_set_tick_count(uint32_t i);
٣٥ extern uint32_t i386_pit_get_tick_count();
٣٦ extern void i386_pit_start_counter(uint32_t freq,uint8_t
    counter,uint8_t mode);
٣٧ extern void _cdecl i386_pit_init();
٣٨ extern bool _cdecl i386_pit_is_initialized();

```

#### Example ٦.١٤: hal/pit.cpp: PIT Implementation

```

١ static volatile uint32_t _pit_ticks = 0;
٢ static bool _pit_is_init = false;
٣
٤ void _cdecl i386_pit_irq();
٥
٦ void i386_pit_send_command(uint8_t cmd) {
٧     outportb(I386_PIT_COMMAND_REG,cmd);
٨ }
٩
١٠ void i386_pit_send_data(uint16_t data,uint8_t counter) {
١١     uint8_t port;
١٢
١٣     if (counter == I386_PIT_OCW_COUNTER_0)
١٤         port = I386_PIT_COUNTER0_REG;
١٥     else if ( counter == I386_PIT_OCW_COUNTER_1)
١٦         port = I386_PIT_COUNTER1_REG;
١٧     else
١٨         port = I386_PIT_COUNTER2_REG;
١٩
٢٠     outportb(port,uint8_t (data));
٢١ }

```

```

٢٢
٢٣ uint8_t i386_pit_read_data(uint16_t counter) {
٢٤     uint8_t port;
٢٥
٢٦     if (counter == I386_PIT_OCW_COUNTER_0)
٢٧         port = I386_PIT_COUNTER0_REG;
٢٨     else if ( counter == I386_PIT_OCW_COUNTER_1)
٢٩         port = I386_PIT_COUNTER1_REG;
٣٠     else
٣١         port = I386_PIT_COUNTER2_REG;
٣٢
٣٣     return inportb(port);
٣٤ }
٣٥
٣٦ uint32_t i386_pit_set_tick_count(uint32_t i) {
٣٧     uint32_t prev = _pit_ticks;
٣٨     _pit_ticks = i;
٣٩     return prev;
٤٠ }
٤١
٤٢ uint32_t i386_pit_get_tick_count() {
٤٣     return _pit_ticks;
٤٤ }
٤٥
٤٦ void i386_pit_start_counter(uint32_t freq,uint8_t counter,
٤٧                             uint8_t mode) {
٤٨     if (freq == 0)
٤٩         return;
٥٠
٥١     uint16_t divisor = uint16_t(1193181/uint16_t(freq));
٥٢
٥٣     /* send operation command */
٥٤     uint8_t ocw = 0;
٥٥
٥٦     ocw = (ocw & ~I386_PIT_OCW_MASK_MODE) | mode;
٥٧     ocw = (ocw & ~I386_PIT_OCW_MASK_RL) | I386_PIT_OCW_RL_DATA
٥٨         ;
٥٩     ocw = (ocw & ~I386_PIT_OCW_MASK_COUNTER) | counter;
٦٠
٦١     i386_pit_send_command(ocw);
٦٢
٦٣     /* set frequency rate */

```



```

٦٢ i386_pit_send_data(divisor & 0xff,0);
٦٣ i386_pit_send_data((divisor >> 8) & 0xff,0);
٦٤
٦٥ /* reset ticks count */
٦٦ _pit_ticks = 0;
٦٧ }
٦٨
٦٩ void _cdecl i386_pit_init() {
٧٠     set_vector(32,i386_pit_irq);
٧١     _pit_is_init = true;
٧٢ }
٧٣
٧٤ bool _cdecl i386_pit_is_initialized() {
٧٥     return _pit_is_init;
٧٦ }
٧٧
٧٨ void _cdecl i386_pit_irq() {
٧٩
٨٠     _asm {
٨١         add esp,12
٨٢         pushad
٨٣     }
٨٤
٨٥     _pit_ticks++;
٨٦
٨٧     int_done(0);
٨٨
٨٩     _asm {
٩٠         popad
٩١         iretd
٩٢     }
٩٣ }

```

### ٣.٤.٦. واجهة HAL الجديدة

المثال ٦.١٥ يوضح الواجهة العامة لطبقة HAL

#### Example ٦.١٥: New HAL Interface

```

١ extern int _cdecl hal_init();
٢ extern int _cdecl hal_close();
٣ extern void _cdecl gen_interrupt(int);

```

```

٤ extern void _cdecl int_done(unsigned int n);
٥ extern void _cdecl sound(unsigned int f);
٦ extern unsigned char _cdecl inportb(unsigned short port_num)
  ;
٧ extern void _cdecl outportb(unsigned short port_num, unsigned
  char value);
٨ extern void _cdecl enable_irq();
٩ extern void _cdecl disable_irq();
١٠ extern void _cdecl set_vector(unsigned int int_num, void (
  _cdecl far & vect) ());
١١ extern void (_cdecl far * _cdecl get_vector(unsigned int
  int_num)) ();
١٢ extern const char* _cdecl get_cpu_vendor();
١٣ extern int _cdecl get_tick_count();

```

#### Example ٦.١٦: New HAL Impelmentation

```

١ int _cdecl hal_init() {
٢     i386_cpu_init();
٣     i386_pic_init(0x20, 0x28);
٤     i386_pit_init();
٥     i386_pit_start_counter(100, I386_PIT_OCW_COUNTER_0,
        I386_PIT_OCW_MODE_SQUAREWAVEGEN);
٦
٧     /* enable irq */
٨     enable_irq();
٩
١٠    return 0;
١١ }
١٢
١٣ int _cdecl hal_close() {
١٤     i386_cpu_close();
١٥     return 0;
١٦ }
١٧
١٨ void _cdecl gen_interrupt(int n) {
١٩     #ifdef _MSC_VER
٢٠     _asm {
٢١         mov al, byte ptr [n]
٢٢         mov byte ptr [address+1], al
٢٣         jmp address

```

```

٢٤
٢٥     address:
٢٦         int 0    // will execute int n.
٢٧     }
٢٨ #endif
٢٩ }
٣٠
٣١
٣٢ void _cdecl int_done(unsigned int n) {
٣٣     if (n > 16)
٣٤         return;
٣٥
٣٦     if (n > 7)
٣٧         /* send EOI to pic2 */
٣٨         i386_pic_send_command(I386_PIC_OCW2_MASK_EOI,1);
٣٩
٤٠         /* also send to the primary pic */
٤١         i386_pic_send_command(I386_PIC_OCW2_MASK_EOI,0);
٤٢     }
٤٣
٤٤ void _cdecl sound(unsigned int f) {
٤٥     outportb(0x61,3 | unsigned char(f << 2));
٤٦ }
٤٧
٤٨ unsigned char _cdecl inportb(unsigned short port_num) {
٤٩ #ifdef _MSC_VER
٥٠     _asm {
٥١         mov dx,word ptr [port_num]
٥٢         in al,dx
٥٣         mov byte ptr [port_num],al
٥٤     }
٥٥ #endif
٥٦
٥٧     return unsigned char(port_num);
٥٨ }
٥٩
٦٠
٦١ void _cdecl outportb(unsigned short port_num,unsigned char
    value) {
٦٢ #ifdef _MSC_VER
٦٣     _asm {
٦٤         mov al,byte ptr[value]

```

```

٦٥     mov dx,word ptr[port_num]
٦٦     out dx,al
٦٧ }
٦٨ #endif
٦٩ }
٧٠
٧١ void _cdecl enable_irq() {
٧٢ #ifdef _MSC_VER
٧٣     _asm sti
٧٤ #endif
٧٥ }
٧٦
٧٧ void _cdecl disable_irq() {
٧٨ #ifdef _MSC_VER
٧٩     _asm cli
٨٠ #endif
٨١ }
٨٢
٨٣ void _cdecl set_vector(unsigned int int_num,void (_cdecl far
    & vect)()) {
٨٤     i386_idt_install_ir(int_num,I386_IDT_32BIT|
        I386_IDT_PRESENT /*10001110*/,0x8 /*code desc*/,vect);
٨٥ }
٨٦
٨٧ void (_cdecl far * _cdecl get_vector(unsigned int int_num)
    ) {
٨٨     idt_desc* desc = i386_get_idt_ir(int_num);
٨٩
٩٠     if (desc == 0)
٩١         return 0;
٩٢
٩٣     uint32_t address = desc->base_low | (desc->base_high <<
        16);
٩٤
٩٥     I386_IRQ_HANDLER irq = (I386_IRQ_HANDLER) address;
٩٦     return irq;
٩٧ }
٩٨
٩٩ const char* _cdecl get_cpu_vendor() {
١٠٠     return i386_cpu_vendor();
١٠١ }
١٠٢

```

```
١٢ int _cdecl get_tick_count() {
١٤     return i386_pit_get_tick_count();
١٥ }
```

---

#### Example ٦.١٧: kernel/main.cpp

```
١ int _cdecl main()
٢ {
٣     hal_init();
٤     enable_irq();
٥
٦     set_vector(0, (void (_cdecl &)(void)) divide_by_zero_fault);
٧     set_vector(1, (void (_cdecl &)(void)) single_step_trap);
٨     set_vector(2, (void (_cdecl &)(void)) nmi_trap);
٩     set_vector(3, (void (_cdecl &)(void)) breakpoint_trap);
١٠    set_vector(4, (void (_cdecl &)(void)) overflow_trap);
١١    set_vector(5, (void (_cdecl &)(void)) bounds_check_fault);
١٢    set_vector(6, (void (_cdecl &)(void)) invalid_opcode_fault);
١٣    set_vector(7, (void (_cdecl &)(void)) no_device_fault);
١٤    set_vector(8, (void (_cdecl &)(void)) double_fault_abort);
١٥    set_vector(10, (void (_cdecl &)(void)) invalid_tss_fault);
١٦    set_vector(11, (void (_cdecl &)(void)) no_segment_fault);
١٧    set_vector(12, (void (_cdecl &)(void)) stack_fault);
١٨    set_vector(13, (void (_cdecl &)(void))
        general_protection_fault);
١٩    set_vector(14, (void (_cdecl &)(void)) page_fault);
٢٠    set_vector(16, (void (_cdecl &)(void)) fpu_fault);
٢١    set_vector(17, (void (_cdecl &)(void)) alignment_check_fault
        );
٢٢    set_vector(18, (void (_cdecl &)(void)) machine_check_abort);
٢٣    set_vector(19, (void (_cdecl &)(void)) simd_fpu_fault);
٢٤
٢٥    ...
```

---

#### Example ٦.١٨: kernel/exception.h

```
١ /* Exception Handler */
٢
٣ /* Divide by zero */
٤ extern void _cdecl divide_by_zero_fault(uint32_t cs, uint32_t
    eip, uint32_t eflags);
```

```

٥
٦ /* Single step */
٧ extern void _cdecl single_step_trap(uint32_t cs,uint32_t eip
    ,uint32_t eflags);
٨
٩ /* No Maskable interrupt trap */
١٠ extern void _cdecl nmi_trap(uint32_t cs,uint32_t eip,
    uint32_t eflags);
١١
١٢ /* Breakpoint hit */
١٣ extern void _cdecl breakpoint_trap(uint32_t cs,uint32_t eip,
    uint32_t eflags);
١٤
١٥ /* Overflow trap */
١٦ extern void _cdecl overflow_trap(uint32_t cs,uint32_t eip,
    uint32_t eflags);
١٧
١٨ /* Bounds check */
١٩ extern void _cdecl bounds_check_fault(uint32_t cs,uint32_t
    eip,uint32_t eflags);
٢٠
٢١ /* invalid opcode instruction */
٢٢ extern void _cdecl invalid_opcode_fault(uint32_t cs,uint32_t
    eip,uint32_t eflags);
٢٣
٢٤ /* Device not available */
٢٥ extern void _cdecl no_device_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags);
٢٦
٢٧ /* Double Fault */
٢٨ extern void _cdecl double_fault_abort(uint32_t cs,uint32_t
    err,uint32_t eip,uint32_t eflags);
٢٩
٣٠ /* Invalid TSS */
٣١ extern void _cdecl invalid_tss_fault(uint32_t cs,uint32_t
    err,uint32_t eip,uint32_t eflags);
٣٢
٣٣ /* Segment not present */
٣٤ extern void _cdecl no_segment_fault(uint32_t cs,uint32_t err
    ,uint32_t eip,uint32_t eflags);
٣٥
٣٦ /* Stack fault */

```

```
٣٧ extern void _cdecl stack_fault(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٣٨
٣٩ /* General Protection Fault */
٤٠ extern void _cdecl general_protection_fault(uint32_t cs,
    uint32_t err,uint32_t eip,uint32_t eflags);
٤١
٤٢ /* Page Fault */
٤٣ extern void _cdecl page_fault(uint32_t cs,uint32_t err,
    uint32_t eip,uint32_t eflags);
٤٤
٤٥ /* FPU error */
٤٦ extern void _cdecl fpu_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags);
٤٧
٤٨ /* Alignment Check */
٤٩ extern void _cdecl alignment_check_fault(uint32_t cs,
    uint32_t err,uint32_t eip,uint32_t eflags);
٥٠
٥١ /* Machine Check */
٥٢ extern void _cdecl machine_check_abort(uint32_t cs,uint32_t
    eip,uint32_t eflags);
٥٣
٥٤ /* FPU Single Instruction Multiple Data (SIMD) error */
٥٥ extern void _cdecl simd_fpu_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags);
```

---

**Example ٦.١٩: kernel/exception.cpp**

```
١ /* Divide by zero */
٢ void _cdecl divide_by_zero_fault(uint32_t cs,uint32_t eip,
    uint32_t eflags) {
٣     kernel_panic("Divide by 0");
٤     for (;;)
٥ }
```

---

**Example ٦.٢٠: kernel/panic.cpp**

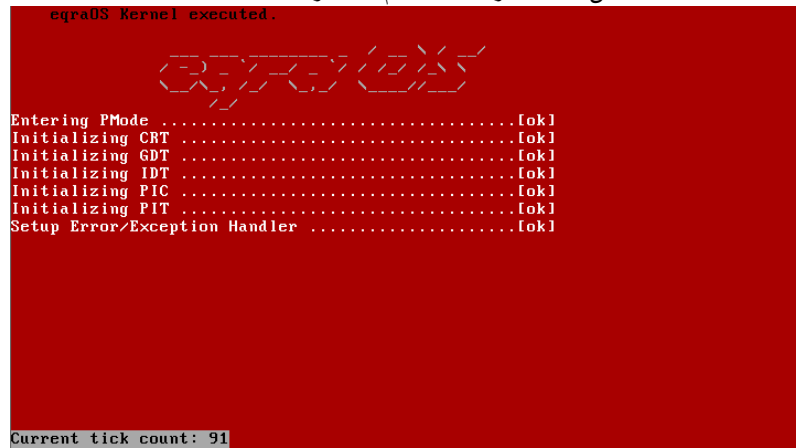
```
١ void _cdecl kernel_panic(const char* msg,...) {
٢
```

```

٣  disable_irq();
٤
٥  va_list args;
٦  va_start(args, msg);
٧  /* missing */
٨  va_end(args);
٩
١٠ char* panic = "\nSorry, egraOS has encountered a problem
    and has been shutdown.\n\n";
١١
١٢ kclear(0x1f);
١٣ kgoto_xy(0,0);
١٤ kset_color(0x1f);
١٥ kputs(panic);
١٦ kprintf("*** STOP: %s", msg);
١٧
١٨ /* hang */
١٩ for (;;) ;
٢٠ }

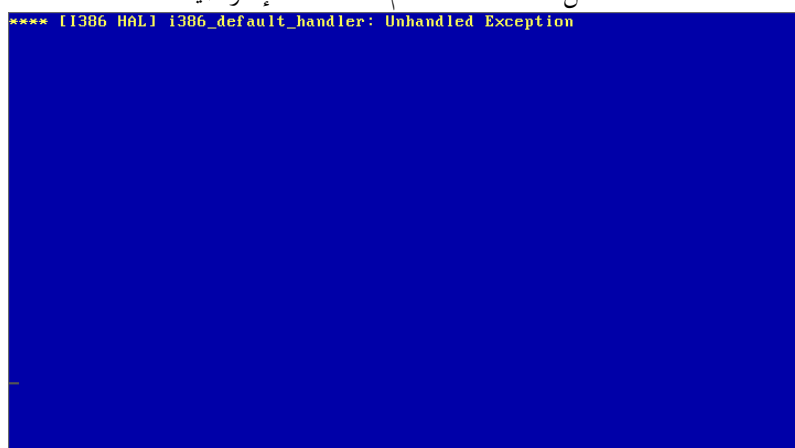
```

شكل ٥.٦: واجهة النظام بعد توسعة طبقة HAL





شكل ٦.٦.: دالة تخدم المقاطعات الافتراضية





## ٧. إدارة الذاكرة

١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management

٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management



## ٨. مشغلات الاجهزة Device Driver

٨.١. برمجة مشغل لوحة المفاتيح Keyboard Driver

٨.٢. برمجة مشغل القرص المرن Floppy Disk Driver

انظر الى شفرة النظام.

٨.٣. برمجة متحكم DMAC

انظر الى شفرة النظام.



## ٩ . أنظمة الملفات

انظر الى شجرة النظام.





## ١. ترجمة وتشغيل البرامج

لتطوير نظام التشغيل يجب استخدام مجموعة من الأدوات واللغات التي تساعد وتيسر عملية التطوير وفي هذا الفصل سيتم عرض هذه الأدوات وكيفية استخدامها.  
اعداد مترجم فيجوال سي++ لبرمجة النواة.

### ١.١. نظام ويندوز

### ٢.١. نظام لينوكس



## ب. المراجع



# Bibliography

- [?] William Stallings, *Operating System: Internals and Design Principles*. Prentice Hall, 5th Edition, 2004.
- [?] Andrew S. Tanenbaum ,Albert S Woodhull, *Operating Systems Design and Implementation*. Prentice Hall, 3rd Edition, 2006.
- [?] Michael Tischer, Bruno Jennrich, *PC Intern: The Encyclopedia of System Programming*. Abacus Software, 6th Edition, 1996.
- [?] Hans-Peter Messmer, *The Indispensable PC Hardware Book*. Addison-Wesley Professional, 4th Edition, 2001.
- [?] Andrew S. Tanenbaum, *Structured Computer Organization*. Prentice Hall, 4th Edition, 1998.
- [?] Ytha Yu,Charles Marut, *Asssembly Language Programming and Organization IBM PC*. McGraw-Hill/Irwin, 1st Edition, 1992.
- [?] Intel® Manuals, *Intel® 64 and IA-32 Architectures Software Developer's Manuals*. <http://www.intel.com/products/processor/manuals/>
- [?] OSDev: <http://wiki.osdev.org>
- [?] brokenthorn: <http://brokenthorn.com>
- [??] Computer Sciense Student's Community in Sudan: <http://sudancs.com>



## ج. شفرة نظام إقرأ

كود النظام





**د. إتفاقية ترخيص المستندات الحرة GNU FDL**