بسم الله الرحمن الرحيم

جامعة الخرطوم – كلية العلوم الرياضية
قسم الحاسوب

بحث مقدم لنيل درجة البكالوريوس في علوم الحاسوب بعنوان:

بَرْمَجَة وَ تَصْمِيمْ نظَام تَشْغِيلْ

"نظام إقرأ"

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

٣ يوليو ٢٠١٠

جامعة الخرطوم.
كلية العلوم الرياضية – قسم الحاسوب.
تم كتابة هذا البحث باستخدام نظام LaTeX.

المحتويات

I. الأساسيات Basics **١**

١. مقدمة عن أنظمة التشغيل **٣**
١.١. ما هو نظام التشغيل ٦
١.١.١. نظام التشغيل كجهاز تخيلي ٦
٢.١.١. نظام التشغيل كمدير للموارد والعتاد ٦
٢.١. تاريخ أنظمة التشغيل ٦
١.٢.١. الجيل الصفري (١٦٢٤-١٩٤٥): الحواسيب الميكانيكية ٧
٢.٢.١. الجيل الأول (١٩٤٥-١٩٥٥): الصمامات المفرغة و لوحات التوصيل ١٠

٢. معمارية حواسيب x86 **١٥**
١.٢. معمارية النظام ١٦
١.١.٢. مسار النظام System Bus ١٦
٢.١.٢. متحكم الذاكرة ١٨
٣.١.٢. متحكم الإدخال والإخراج ١٨
٢.٢. المعالج ١٩
١.٢.٢. دورة تنفيذ التعليمات ٢٠
٢.٢.٢. أنماط عمل المعالج CPU Modes ٢٠
٣.٢.٢. النمط الحقيقي Real Mode ٢٢
٤.٢.٢. النمط المحمي Protected Mode ٢٣
٥.٢.٢. معمارية معالجات x86 . . . . . ٢٥

II. إقلاع الحاسب Booting **٣١**

٣. إقلاع الحاسب ومحمل النظام Bootloader **٣٣**
١.٣. إقلاع الحاسب ٣٣
٢.٣. محمل النظام Bootloader ٣٤

٣.٣. مخطط الذاكرة ٣٥
٤.٣. برمجة محمل النظام ٣٦
١.٤.٣. عرض رسالة ترحيبية ٣٧
٢.٤.٣. معلومات قطاع الاقلاع ٤٠
٣.٤.٣. تحميل قطاع من القرص باستخدام المقاطعة int 0x13 . . . . . ٤٨
٥.٣. مقدمة الى نظام FAT12 . . . . . ٥٠
١.٥.٣. قيود نظام FAT12 . . . . . ٥١
٢.٥.٣. هيكلة نظام FAT12 على القرص ٥١
٣.٥.٣. هيكلة القرص المرن ٥٤
٤.٥.٣. القراءة و الكتابة من نظام FAT12 . . . . . ٥٥

٤. برمجة محمل النظام – المرحلة الثانية ٦٧
١.٤. الانتقال الى النمط المحمي ٦٧
١.١.٤. جدول الواصفات العام Global Descriptor Table ٦٨
٢.١.٤. العنونة في النمط المحمي PMode Memory Addressing ٧٢
٣.١.٤. الانتقال الى النمط المحمي ٧٢
٢.٤. تفعيل البوابة A20 . . . . . ٧٤
١.٢.٤. متحكم لوحة المفاتيح 8042 والبوابة A20 . . . . . ٧٤
٢.٢.٤. طرق تفعيل البوابة A20 . . . . . ٧٥
٣.٤. أساسيات ال VGA ٨١
١.٣.٤. عنونة الذاكرة في متحكمات VGA ٨٢
٢.٣.٤. طباعة حرف على الشاشة ٨٣
٣.٣.٤. طباعة السلاسل النصية strings ٨٧
٤.٣.٤. تحديث المؤشر Hardware Cursor ٨٩
٥.٣.٤. تنظيف الشاشة Clear Screen ٩٢
٤.٤. تحميل النواة ٩٣

III. النواة Kernel ١٠١

٥. مقدمة حول نواة نظام التشغيل ١٠٣
١.٥. نواة نظام التشغيل ١٠٣
١.١.٥. مستويات التجريد ١٠٤
٢.٥. وظائف نواة النظام ١٠٥
١.٢.٥. إدارة الذاكرة ١٠٥
٣.٥. هيكلة وتصميم النواة ١٠٦
١.٣.٥. النواة الضخمة Monolithic Kernel ١٠٦
٢.٣.٥. النواة المصغرة MicroKernel ١٠٧

٣.٣.٥. النواة الهجينة Hybrid Kernel ١٠٧
٤.٥. برمجة نواة النظام ١٠٧
١.٤.٥. تحميل وتنفيذ نواة PE ١٠٨
٢.٤.٥. تطوير بيئة التشغيل للغة سي++ ١١١
٣.٤.٥. نقل التنفيذ الى النواة ١١٦
٥.٥. نظرة على شفرة نظام إقرأ ١١٧
٦.٥. مكتبة السي القياسية ١١٨

٦. المقاطعات Interrupts **١٢٣**
١.٦. المقاطعات البرمجية Software Interrupts ١٢٣
١.١.٦. المقاطعات في النمط الحقيقي ١٢٣
٢.١.٦. المقاطعات في النمط المحمي ١٢٥
٣.١.٦. أخطاء المعالج ١٢٧
٤.١.٦. إنشاء جدول الواصفات العام GDT ١٢٩
٢.٦. متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller ١٣٣
١.٢.٦. المقاطعات العتادية Hardware Interrupts ١٣٣
٢.٢.٦. برمجة متحكم PIC ١٣٤
٣.٦. المؤقتة Programmable Interval Timer ١٤١
١.٣.٦. برمجة المؤقتة PIT ١٤٢
٤.٦. توسعة طبقة HAL ١٤٥
١.٤.٦. دعم PIC ١٤٥
٢.٤.٦. دعم PIT ١٤٩
٣.٤.٦. واجهة HAL الجديدة ١٥٢

٧. إدارة الذاكرة **١٦١**
١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management ١٦١
١.١.٧. حساب حجم الذاكرة ١٦٢
٢.١.٧. خريطة الذاكرة Memory Map ١٦٤
٣.١.٧. مواصفات الإقلاع المتعدد ١٦٧
٤.١.٧. مدير الذاكرة الفيزيائية ١٧١
٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management ١٧٧

٨. مشغلات الاجهزة Device Driver **١٨٣**
١.٨. برمجة مشغل لوحة المفاتيح Keyboard Driver ١٨٣

ا. ترجمة وتشغيل البرامج **١٩٥**
ا.١. نظام ويندوز ١٩٥
ا.٢. نظام لينوكس ١٩٥

ب. شفرة نظام إقرأ **١٩٧**

الأمثلة التوضيحية

١.١. Assembly Language ٤

٣.١. Smallest Bootloader ٣٧
٣.٢. Welcom to OS World ٣٧
٣.٣. Bios Parameter Block ٤٠
٣.٤. BPB example ٤١
٣.٥. Hex value of bootloader ٤٤
٣.٦. Complete Example ٤٥
٣.٧. Reset Floppy Drive ٤٨
٣.٨. Read Floppy Disk Sectors ٤٩
٣.٩. Hello Stage2 . . . . . ٥٥
٣.١٠. Load Root directory ٥٧
٣.١١. Find Stage2 Bootloader ٥٨
٣.١٢. Load FAT Table ٥٩
٣.١٣. Convert Cluster number to LBA ٦٠
٣.١٤. Convert LBA to CHS ٦٠
٣.١٥. Load Cluster ٦١
٣.١٦. Read Sectors Routine ٦٢
٣.١٧. Read FAT entry ٦٤

٤.١. GDT ٦٩
٤.٢. Load GDT into GDTR ٧١
٤.٣. Switching to Protected Mode ٧٣
٤.٤. Enable A20 by System Control Port 0x92 . . . . . ٧٥
٤.٥. Enable A20 by BIOS int 0x15 . . . . . ٧٦
٤.٦. Wait Input/Output ٧٧
٤.٧. Enable A20 by Send 0xdd ٧٨
٤.٨. Enable A20 by write to output port of Keyboard Controller ٨٠
٤.٩. Print 'A' character on screen ٨٣
٤.١٠. putch32 routine ٨٥
٤.١١. puts32 routine ٨٧
٤.١٢. Move Hardware Cursor ٩٠
٤.١٣. Clear Screen ٩٢
٤.١٤. Hello Kernel ٩٣

٤.١٥. Loading and Executing Kernel: Full Example ٩٥

.٥.١ Portable Executable Header ١٠٨
.٥.٢ Getting Kernel entry ١١٠
.٥.٣ Global new/delete operator ١١١
.٥.٤ Pure virtual function call handler ١١٢
.٥.٥ Floating Point Support ١١٢
.٥.٦ Object Initializer ١١٣
.٥.٧ Delete Object ١١٤
.٥.٨ Kernel Entry routine ١١٦
.٥.٩ null.h:Definition of NULL in C and C++ ١١٨
.٥.١٠ size_t.h:Definition of size_t in C/C++ ١١٩
.٥.١١ stdint.h:typedef data type ١٢٠
.٥.١٢ cstdint:C++ typedef data type ١٢٠
.٥.١٣ ctype.h:determine character type ١٢١

.٦.١ Example of interrupt descriptor ١٢٦
.٦.٢ Value to put in IDTR ١٢٦
.٦.٣ include/hal.h:Hardware Abstraction Layer Interface ١٢٩
.٦.٤ hal/gdt.cpp:Install GDT ١٣٠
.٦.٥ Initialization Control Words 1 . . . . . ١٣٧
.٦.٦ Initialization Control Words 2 . . . . . ١٣٨
.٦.٧ Initialization Control Words 3 . . . . . ١٣٨
.٦.٨ Initialization Control Words 4 . . . . . ١٣٩
.٦.٩ Send EOI ١٤٠
.٦.١٠ PIT programming ١٤٤
.٦.١١ hal/pic.h: PIC Interface ١٤٥
.٦.١٢ hal/pic.cpp: PIC Implementation ١٤٧
.٦.١٣ hal/pit.h: PIt Interface ١٤٩
.٦.١٤ hal/pit.cpp: PIT Implementation ١٥٠
.٦.١٥ New HAL Interface ١٥٢
.٦.١٦ New HAL Impelmentation ١٥٣
.٦.١٧ kernel/main.cpp ١٥٦
.٦.١٨ kernel/exception.h ١٥٦
.٦.١٩ kernel/exception.cpp ١٥٨
.٦.٢٠ kernel/panic.cpp ١٥٨

.٧.١ Using int 0x12 to get size of memory ١٦٢
.٧.٢ Using int 0x15 Function 0xe801 to get size of memory ١٦٣
.٧.٣ Memory Map Entry Structure ١٦٥
.٧.٤ Get Memory Map ١٦٥

٧.٥. Multiboot Inforamtion Structure ١٦٨
٧.٦. snippet from stage2 bootloader ١٦٩
٧.٧. Kernel Entry ١٧٠
٧.٨. Physical Memory Manager Interface ١٧١
٧.٩. Physical Memory Manager Implemetation ١٧١
٧.١٠. Virtual Memory Manager Interface ١٧٧
٧.١١. Virtual Memory Manager Implemetation ١٧٨

٨.١. Keybaord Driver Interface ١٨٣
٨.٢. Keybaord Driver Implemetation ١٨٧

قائمة الأشكال

١.١. الشكل العام لأوامر المعالج x86 . . . . . ٣
٢.١. طبقات الحاسب ٥
٣.١. آلة باسكال ٧
٤.١. آلة Step Reckoner في متحف بألمانيا ٧
٥.١. محرك الفروق بعد أن قام ابن بابباج بتجميعه ٨
٦.١. المحرك التحليلي بمتحف في لندن ٩
٧.١. حاسبة Z1 بعد إعادة انشائها في متحف بألمانيا ٩
٨.١. حاسبة Atanasoff بعد إعادة انشائها في جامعة Iowa State ١٠
٩.١. حاسبة Harvard Mark I ١٠
١٠.١. آلة إنجما الألمانية لتشفير الرسائل وفكها ١١
١١.١. الحاسبة colossus التي كسرت شفرة إنجما ١٢
١٢.١. الحاسبة ENIAC ١٢
١٣.١. الحاسبة EDVAC ١٣

١.٢. معمارية حواسيب x86 . . . . . ١٥
٢.٢. المسارات في الحواسيب الشخصية x86 . . . . . ١٦
٣.٢. الجسر الشمالي ١٨
٤.٢. تداخل المقاطع في النمط الحقيقي ٢٤
٥.٢. حلقات المعالج ٢٥

١.٣. مخطط محتويات الذاكرة الرئيسية ٣٦
٢.٣. هيكلة نظام FAT12 على القرص ٥١

١.٤. محمل النظام أثناء العمل ١٠٠
٢.٤. بدء تنفيذ النواة ١٠٠

١.٦. متحكم المقاطعات القابل للبرمجة 8259A ١٣٣
٢.٦. مشابك متحكم PIC ١٣٥
٣.٦. المؤقتة القابلة للبرمجة 8253 . . . . . ١٤٢
٤.٦. مشابك المؤقتة PIT ١٤٢
٥.٦. واجهة النظام بعد توسعة طبقة HAL ١٥٩
٦.٦. دالة تخديم المقاطعات الإفتراضية ١٦٠

قائمة الجداول

٢.١. مخطط الذاكرة لحواسيب x86 . . . . . ٢٠
٢.٢. منافذ الإدخال والإخراج لحواسيب x86 . . . . . ٢١
٢.٣. الأوامر التي تتطلب صلاحية الحلقة صفر ٢٦
٢.٤ الأعلام مسجل EFLAGS ٢٩
٦.١ Interrupt Vector Table ١٢٤
٦.٢. x86 Processor Exceptions Table ١٢٨
٦.٣. مقاطعات العتاد لحواسيب x86 . . . . . ١٣٤
٦.٤. مسجل IRR/ISR/IMR ١٣٦
٦.٥. عناوين المنافذ لمتحكم PIC ١٣٦
٦.٦. الأمر الأول ICW1 . . . . . ١٣٧
٦.٧ الأمر الرئيسي للمتحكم الثالث ICW3 for Primary PIC ١٣٨
٦.٨ الأمر الثالث الثانوي للمتحكم ICW3 for Slave PIC ١٣٨
٦.٩ الأمر الرابع ICW4 . . . . . ١٣٩
٦.١٠. أمر التحكم الثاني OCW2 . . . . . ١٤٠
٦.١١. أمر OCW2 . . . . . ١٤٠
٦.١٢. مسجلات المؤقتة 8253 PIT ١٤٣

ملخص البحث

هدف هذا البحث هو برمجة نظام تشغيل (نظام إقرأ) للإستخدامات التعليمية والأكاديمية بحيث يُمكِّن الطالب من تطبيق ما تعلمه من نظريات خلال فترة الدراسة على نظام تشغيل حقيقي مما يكسبه خبرة ويؤهله للمشاركة في برمجة أنظمة تشغيل ضخمة مثل جنو/لينوكس. كذلك يهدف الى توفير بحثا باللغة العربية يشرح الأسس العلمية لكيفية برمجة نظام تشغيل من الصفر دون الإعتماد على أي مكونات خارجية في الوقت الذي تندر توفر مثل هذه البحوث المفيدة للطالب وخاصة في هذا المجال الذي يعتبر من أهم المجالات في علوم الحاسوب.

وتناولت هذه الدراسة العديد من أساسيات ومفاهيم نظم التشغيل بدءا من مرحلة إقلاع الحاسب والإنتقال الى النمط المحمي وانتهاءا ببرمجة مديرا للذاكرة وتعريفات لبعض العتاد.

و رؤية الباحث في هذه الدراسة هي أن تستخدم كمنهج لتدريس الطلاب في مادة نظم التشغيل وأن تُدرَّس الشفرة المصدرية للنظام . ولا يُقتصر على ذلك بل يَستمر التطوير في النظام ليكون أداة تعليمية (مفتوحة المصدر) للطلاب والباحثين.

مقدمة البحث

تلعب أنظمة التشغيل دوراً مهما في شتى مجالات الحياة حيث أصبحت أحد أهم الركائز الأساسية لتشغيل وادارة أي جهاز أو عتاد يعتمد على الشرائح الالتكرونية المتكاملة ، فبدءا من جهاز الحاسب الشخصي و الجوالات و الأجهزة الكفية والمضمنة (Embedded Device) و أجهزة الألعاب والصرافات الآلية وحتى أجهزة الفضاء والدورات (Orbiter) كلها تعمل بأنظمة التشغيل. ونظراً لذلك فان مجال برمجة أنظمة التشغيل يعتبر من أهم المجالات في علوم الحاسب التي يجب أن تأخذ نصيبها من البحث العلمي والتطبيق البرمجي.
وعلى الرغم من أهمية هذا المجال الا انه يندر وجود بحوثا فيه ويعود ذلك لعدة أسباب: **الأول** هو تنوع المجالات التي يجب دراستها قبل الخوض في برمجة نظام تشغيل حيث لا بد للطالب أو الباحث الإلمام بلغة السي والسي++ ولغة التجميع (Assembly) بالإضافة الى المعرفة التامة بمعمارية الحاسب من معالج وذاكرة ووحدات إدخال وإخراج. أما السبب **الثاني** فهو عدم توفر مراجع وكتبا باللغة العربية تشرح الأسس العلمية لبرمجة أنظمة التشغيل. والسبب **الثالث** هو توفر كمية كبيرة من أنظمة التشغيل في الوقت الحالي تجعل الطالب يعتقد بعدم الحوجة للبحث في هذا المجال وهذا مفهوم خاطئ حيث أن مبرمج نظام التشغيل ليس بالضرورة أن يبرمج نظاما من الصفر وانما يمكن أن يقوم بالتعديل والتطوير في أحد الأنظمة المفتوحة المصدر كذلك ربما يعمل في برمجة برامج النظام التي تتطلب الماما تاما بفاهيم أنظمة التشغيل مثل برمجة برامج اصلاح القاطاعات التالفة (Bad Sectors) واسترجاع الملفات المفقودة وغيرها من برامج النظام.
وقد وضع الكاتب نصب عينيه في هذا البحث التطرق للأمور البرمجية بتفاصيلها والتركيز على كيفية كتابة الشفرة لكل جزئية في نظام التشغيل . و لم يتم ذكر كل الجوانب النظرية في الموضوع وهذا بسبب أن الأمور النظرية في الغالب تأخذ بالطالب بعيدا وتحجب رؤيته عن حقيقة عمل نظام التشغيل . وقد تم برمجة النظام من الصفر دون الإعتماد على أي مكونات أو شفرات جاهزة (مفتوحة المصدر) ولا يمكن اعتبار هذا إعادة اختراع للعجلة ! بل هو أساس يمكن الاعتماد عليه وتعليم الطلاب عليه وهكذا يتطور المشروع ويتقدم الى الأمام وفي نفس الوقت تزداد خبرة الطالب العملية في المجال.
و يجدر بنا ايضاح أن هدف النظام (نظام إقرأ) هو للاستخدامات الأكاديمية والتعليمية وليس للمستخدم الأخير ، حيث أن الهدف هو تعليم الطالب على هذه الأداة واعداده للعمل على أنظمة ضخمة مثل جنو/لينوكس.

أحمد عصام عبد الرحيم.
٢٩/٦/٢٠١٠

القسم I.

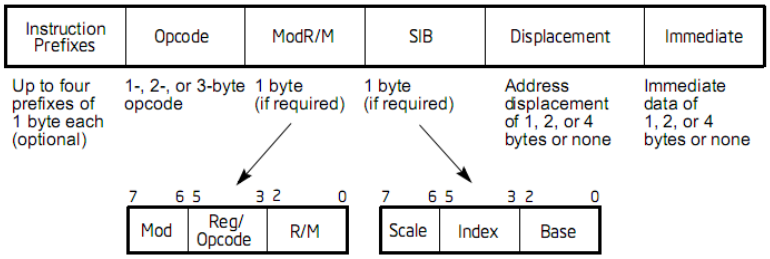
الأساسيات Basics

١. مقدمة عن أنظمة التشغيل

جهاز الحاسب هو مجموعة من الشرائح الإلكترونية والعتاديات والمتحكمات المرتبطة مع بعضها لتوفير منصة تشغيلية للبرامج و التي بدونها 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 وتوضحيها 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 معينا من الأوامر والتي تعرف بمجموعة الأوامر (Instruction Set). لذلك حتى يتم تنفيذ أوامر أي برنامج فانها يجب أن تُكتب وفقا لمجموعة الأوامر التي يدعمها المعالج[١]. والشكل ١.١ يوضح نموذجاً عاما لتعليمات وأوامر المعالج التي تتكون منها البرامج. وجزءا 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 برمجة 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. فمثلا لتعيين القيمة 31744 للمسجل AX[٢] يجب أن يحوي البرنامج على الأمر 10111000011000000000000111. وبالتالي تكون عملية كتابة برنامج متكامل بهذه اللغة أمراً

[١] سنتحدث عن معالجات انتل ٣٢ بت في هذا البحث نظراً لأنها الأكثر انتشاراً.
[٢] المسجلات هي ذواكر بداخل المعالج.

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

Example ١.١: Assembly Language

```
١   MOV AX,0x7C00 ; Instead of 10111000110000000000000111
```

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[٣]. و لم 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[٤] والسي++ بحيث تكتب البرامج فيها بشكل مبسط بعيداً عن تعقيدات الآلة وأوامرها ومسجلاتها وتدعم هذه اللغات عددا من التراكيب وجمل التحكم العالية المستوى. ولكي ينفذ المعالج برامج هذه اللغات فانه يجب أولا 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.

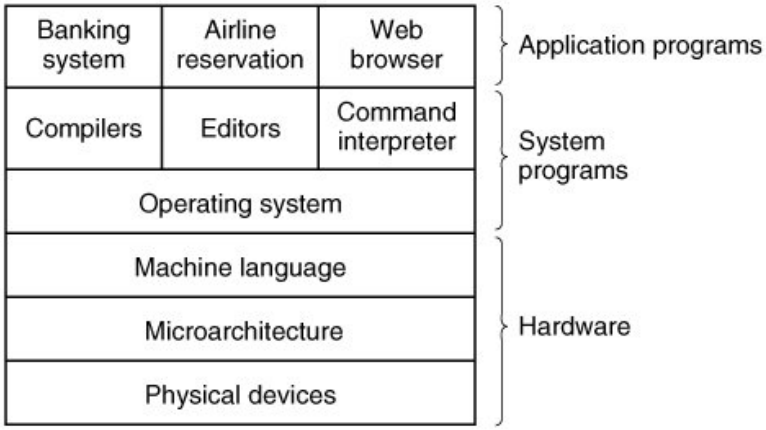
□بخصوص تمثيل البيانات والبرامج في الحاسب فإنها تمثل بطرق مختلفة تختلف على حسب وحدة التخزين و لكنها في الآخر تستخدم المنطق الثنائي وهو وجود طاقة كهربائية أم لا ، فمثلا تتكون الذاكرة الرئيسية DRAM من ملايين المكثفات (Capacitors) والترانزستورات (Transistors) لتكوين خلايا الذاكرة (Memory Cells) ، و تتكون كل خلية (والتي تشكل بت واحد من الذاكرة) من مكثف وترانزستور بحيث يحفظ المكثف قيمة الخلية (البت) والتي هي إما وجود إلكترون (منطقيا تساوي 1) وإما عدمها (منطقيا 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 بقرائتها وفهم وظيفتها (Decode) وتنفيذها ومن ثم يقوم بحفظ النتائج. ولكي ينفذ المعالج أي برنامج فان البرنامج يجب أن يتواجد على الذاكرة الرئيسية وليس على أحد الذواكر 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 طرقا 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 وتوفر واجهة برمجية للمبرمج

[٣]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.
[٤]تم تطوير لغة السي بهدف برمجة نظام يونيكس Unix في معامل بيل.

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) للإستفادة من هذه العتاديات. والشكل ٢.١ يوضح موضع هذه الطبقة (نظام التشغيل) في حالة قسمنا جهاز الحاسب الى عدة طبقات [**؟**]. وأدنى طبقة هي

شكل ٢.١.: طبقات الحاسب



طبقة العتاديات (Device Level) حيث تتكون من المتحكمات ومن الشرائح المتكاملة (Integrated Circuit) والأسلاك وكل ما يتعلق بالأجهزة المادية.يلي هذه الطبقة طبقة Microarchitecure وفيها تظهر برمجات (Mircoprogram) 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 الى 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 الى وحدة الحساب والمنطق ، وتشكل مجموعة الأوامر المدعومة وكذلك المسجلات المرئية لمبرمج 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) .

ويلي 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 فان 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[٥].

[٥] في الفصل الثاني بإذن الله سيتم الحديث عن مستويات الحماية في المعالجات.

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 الإتفاق عليه هو أن **نظام التشغيل** يدير عتاد وموارد الحاسب ويوفر واجهة برمجية (جهاز تخيلي) من خلالها يمكن الإستفادة 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 اليها.كل هذه الأمور لو استمرت بهذا الشكل لما وصلت التطبيقات لما هي عليها الان، لذلك كان الحل هو بإيجاد طبقة نظام التشغيل والتي توفر واجهة أو أوامر مبسطة ومجردة من تفاصيل وتعقيدات العتاد لكي تستخدمها البرامج بدلا من الأوامر التي توفرها طبقة الآلة .

١.١.٢. 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 يحرر 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.

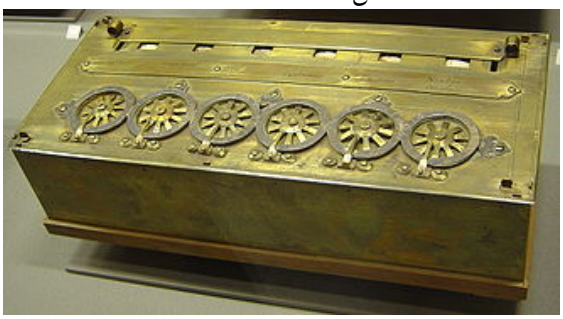
١.٢. تاريخ أنظمة التشغيل

خلال سنوات مضت تطورت أنظمة التشغيل تطوراً ملحوظاً من أنظمة تشغيل لبرنامجاً واحدا 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

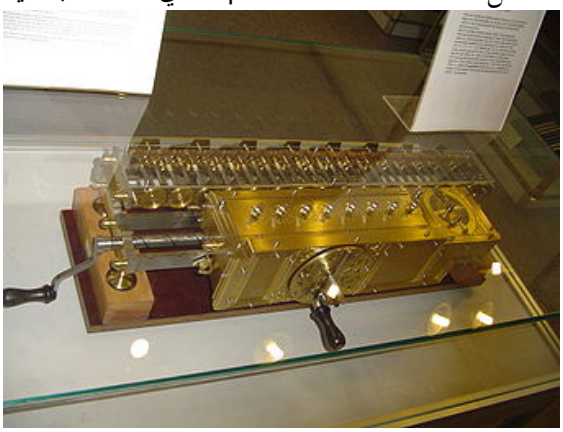
أول محاولة لبناء آلة حسابية كانت من قبل العالم الفرنسي بليز باسكال[٦] في عام ١٦٤٢ عندما كان عمره ١٩ عاماً وذلك لمساعدة والده الذي كان يعمل محصلاً للضرائب لمصلحة الحكومة الفرنسية.هذه الآلة (وتعرف بالاسم Pascaline) هي ميكانيكية بالكامل وتوفر فقط عملية الجمع والطرح (انظر الشكل ٣.١).

شكل ٣.١.: آلة باسكال



وبعد حوالي ٣٠ عاماً قام العالم الرياضي جوتفريد ليبنز 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 و الضرب والقسمة (الشكل ٤.١). ومضت

شكل ٤.١.: آلة Step Reckoner في متحف بألمانيا



حوالي ١٥٠ عاماً بدون أي شيء يذكر حتى قام البروفيسور شارلز بابباج Charles Babbage بتصميم آلة محرك الفروق Difference engine (انظر الشكل ٥.١) ، وهي آلة ميكانيكية أيضا تشابه آلة باسكال في أنها

[٦] والذي تم تسمية لغة البرمجة باسكال باسمه تشريفاً له.

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

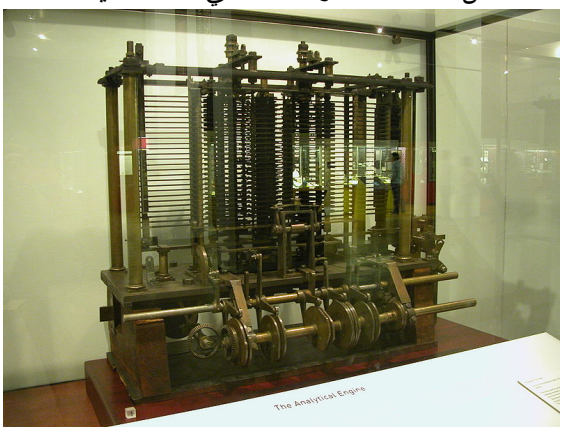
شكل ١.٥.: محرك الفروق بعد أن قام ابن بابباج بتجميعه

النتائج حيث تنقش النتائج على ألواح نحاسية. وعلى الرغم من أن آلة الفروق عملت جيداً إلا أن تصميمها كان يسمح بحساب خوارزمية واحدة فقط[٧] وهذا ما جعل شارلز بابباج يعيد محاولته مجدداً ويستهلك جزءاً ضخما من وقته ومن ثروة حكومته في بناء آلة أخرى عُرفت بالمحرك التحليلي Analytical Engine (انظر الشكل ٦.١). هذا المحرك (وهو أيضا 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 (على عكس محرك الفروق) حيث تكتب البرامج في بطاقات مثقبة ويتم قرائتها الى المحرك بواسطة قارئاً لهذه البطاقات. وتحوي هذه البطاقات على أوامر موجه الى المحرك لكي يقوم بقرائة عددين من المخزن ويجري عملية ما (جمع مثلا) ومن ثم يحفظ النتيجة في المخزن أيضا ، وكذلك تحوي أوامر أخرى مثل المقارنة بين عددين والتفرع ونقل التنفيذ. ولأن المحرك قابل للبرمجة (بلغة شبيهة بلغة التجميع) فقد استعان شارلز بابباج بالمبرمجة آدا لوفلاس Ada Lovelace والتي صنفت كأول مبرمج في التاريخ. ولسوء الحظ لم ينجح شارلز بابباج في أن يزيد من دقة المحرك ربما لأنه يحتاج الى آلافاً من التروس والعجلات. وبشكل أو بآخر يعتبر شارلز باباج الجد الأول للحواسيب الحالية حيث أن فكرة عمل المحرك التحليلي مشابهة للحواسيب الحالية.

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

[٧] في عام ١٩٩١ قام متحف العلوم بلندن ببناء نموذج مكتمل لمحرك الفروق.

شكل ١.٦.: المحرك التحليلي بمتحف في لندن

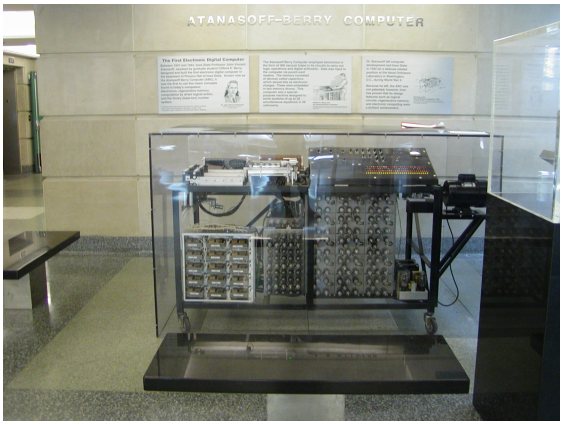


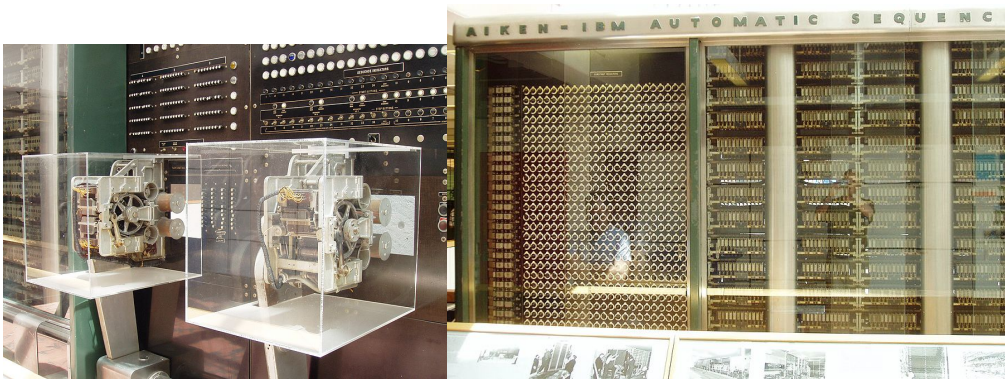
الثنائي في عملها. ولسوء الحظ تم تدمير الحاسبة Z1 في انفجار في برلين أثناء الحرب العالمية الثانية عام ١٩٤٣. وعلى الرغم من أن تصميم Z1 لم يؤثر في تصاميم الحواسيب التي تليه بسبب تدميره هو وجميع خطط بنائه إلا أنه يعتبر أحد التصاميم التي كان لها أثرها ذاك الوقت. وبعد برهة من الزمن قام جون أتاناسوف John

شكل ١.٧.: حاسبة Z1 بعد إعادة انشائها في متحف بألمانيا

Vincent Atanasoff بتصميم جهاز Atanasoff(انظر الشكل **??**) والذي كان مدهشا في وقته حيث يستخدم التحسيب الثنائي (Binray Arithmetic) ويحوي مكثفات للذاكرة ولكن الجهاز لم يكن عمليا. وفي بدايات ١٩٤٠ قام هوارد ايكين Howard Aiken بتصميم الحاسبة ASCC – والتي أعيد تسميتها الى Harvard Mark I – في جامعة هافارد (انظر الشكل ٩.١).وقد أتبع هوارد نفس منهج بابباج وقرر أن يعتمد على الريلاي وأن يصمم حاسبة للأغراض العامة والتي فشل بها شارلز بابباج.وفي عام ١٩٤٤ تم الإنتهاء من تصميمها وتم تصميم نسخة محسنة أيضا سميت ب Harvard Mark II.ومع هذا التصميم انتهى عصر الحاسبات الميكانيكية

شكل ١.٨.: حاسبة Atanasoff بعد إعادة انشائها في جامعة Iowa State





(ا) الجزء الأيسر من حاسبة Mark I (ب) الإدخال والإخراج والتحكم

شكل ١.٩.: حاسبة Harvard Mark I

والريلاي وبدأ عصر جديد.

١.٢.٢. 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 البريطانيين 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 البريطانيين من فك شفرة الرسائل فان هذا يتطلب وقتاً وعمليات حسابية طويلة ، لذلك سرعان ما أسست معملا

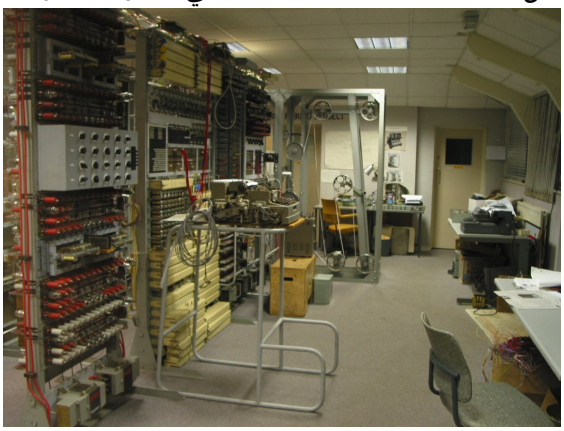
شكل ١٠.١.: آلة إنجما الألمانية لتشفير الرسائل وفكها

سريا 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 كون المشروع سريا وتم التكتم 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.

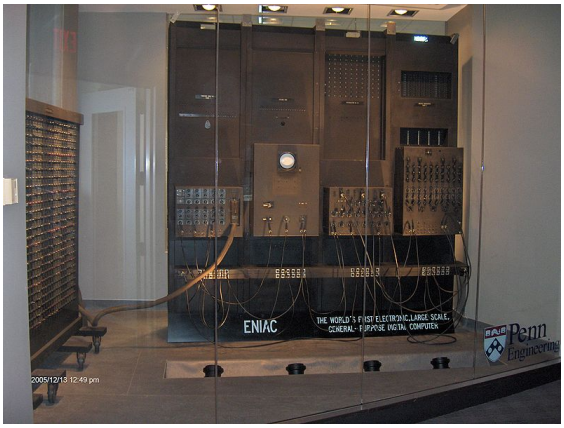
وفي عام ١٩٤٣ قدم جون موكلي John Mauchley مقترحاً الى الجيش الأمريكي طالباً تمويله بالمال للبدء بتصميم حاسبة إلكترونية لحساب جداول إطلاق المدفعيات بدلاً من حسابها يدويا وذلك لتقليل الأخطاء وكسب الوقت، وقد تمت الموافقة على المشروع وبدأ جون موكلي وطالبه الخريج إيكريت ببناء حاسبة تم تسميتها بالاسم إيناك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

وقد تم الإنتهاء من تصميم إيناك عام ١٩٤٦ ، الوقت الذي كانت الحرب قد انتهت و لم تستخدم الحاسبة لهدفها الرئيسي. وبعد ذلك نظم جون موكلي وطالبه إيكريت مدرسة صيفية لوصف مشروعهم للباحثين والمهتمين. الأمر الذي أسفر عن ظهور عدد كبير من الحاسبات الضخمة . وأول حاسبة بعدها كانت

شكل ١.١١.: الحاسبة colossus التي كسرت شفرة إنجما

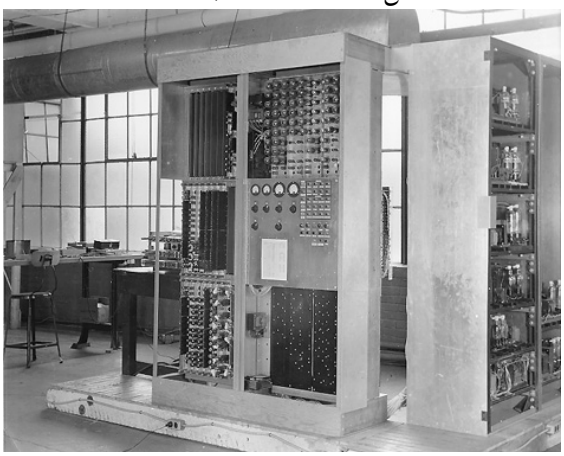


شكل ١.١٢.: الحاسبة ENIAC



EDSAC في عام ١٩٤٩ بواسطة ويلكس في جامعة كامبردج. كذلك تلته الحاسبات JOHNIAC و ILLIAC وغيرهم.بعد ذلك بدأ جون موكلي وإيكريت بالعمل على حاسبة أخرى سميت بالاسم EDVAC اختصارا للجملة Electronic Discrete Variable Automatic Computer والتي كانت تعمل بالأرقام الثنائية بدلا من العشرية (كما في إيناك).بعد ذلك توقف المشروع بسبب أن جون وإيكريت قد أنشئا شركتهم الخاصة.

شكل ١.١٣.: الحاسبة EDVAC



٢. معمارية حواسيب x86

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

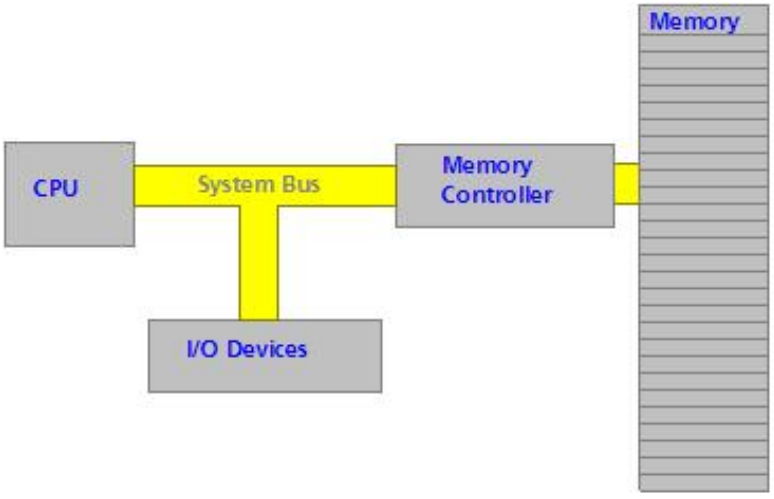
١. معالج أو وحدة معالجة مركزية (Central Processing Unit).

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 على وحدة المعالجة أن تنفذها ، **وأخيراً** وحدات الإدخال والإخراج وهي الاجهزة التي تستخدم في ادخال البيانات واخراجها.(انظر الشكل ١.٢ حيث يوضح مثالاً 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

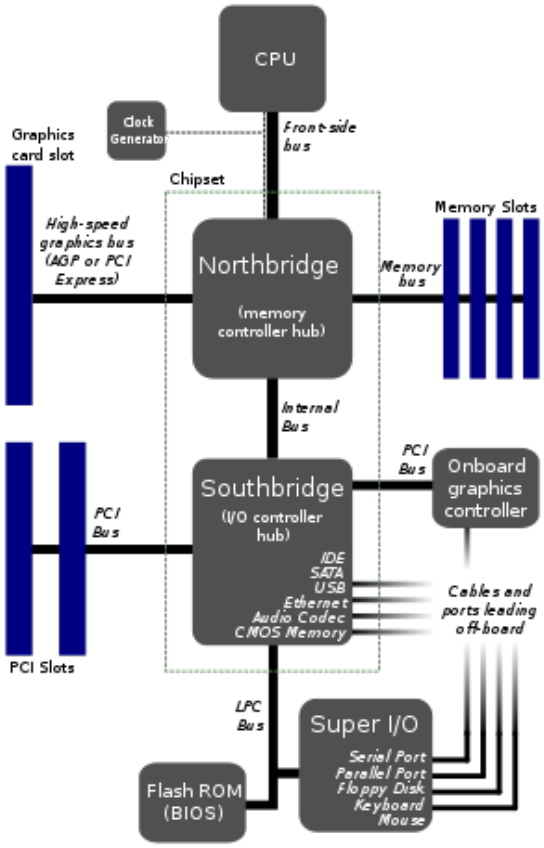


١.٢. معمارية النظام

١.١.٢. مسار النظام System Bus

يربط مسار النظام[١] (System Bus) وحدة المعالجة المركزية (CPU) مع متحكم الذاكرة الرئيسية . و 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: مسار البيانات (Data Bus) ومسار العناوين (Address Bus) ومسار التحكم (Control Bus).

شكل ٢.٢.: المسارات في الحواسيب الشخصية x86



[١] ويسمى أيضا Front-side Bus.

مسار البيانات Data Bus

مسار البيانات هو عبارة عن خطوط (Lines) كل خط يمثل بت واحد. وغالبا ما يكون هناك ٣٢ خط (أي أن مسار البيانات بطول ٣٢ بت) ويستخدم هذا المسار في نقل البيانات (Data) من المعالج (وتحديداً من وحدة التحكم Control Unit) الى متحكم الذاكرة والذي يتواجد بداخل الجسر الشمالي NorthBridge. وبسبب أن حجم مسار البيانات هو حجم ثابت فان هذا يتطلب معالجة خاصة عند ارسال بيانات بطول أقل من طول مسار البيانات ، فغالبا 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 ٣٢ بت 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 الوصل اليه في الذاكرة وبالتالي يحدد حجم الذاكرة التي يستطيع الحاسب التعامل معها . وفي الأجهزة التي تستخدم معالجات انتل 8086 كان حجم هذا المسار هو ٢٠ بت وبالتالي فان أقصى ذاكرة يتعامل معها هذا المعالج هي ١ ميجا[٢] أما في معالجات 80286/80386 فان حجم هذا مسار هو ٢٤ بت وفي المعالجات التي تليها تم زيادة هذا الحجم الى ٣٢ بت وبالتالي يمكن تنصيب ذاكرة بحجم ٤ جيجا ، وفي المعالجات الحديثة تم زيادة هذا الحجم ، ولكننا سنقتصر في هذا البحث على المعالجات التي تدعم مسار عناوين بطول ٣٢ بت 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 تُعيَّن 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 موجهٌ الى متحكم الذاكرة أم الى متحكم الإدخال والإخراج وفي حالة كانت قيمة هذا الخط هي القيمة ١ فان هذا يعني أن العنوان موجهُ الى متحكم أجهزة الإدخال والإخراج وبالتالي سيتم القراءة من هذا العنوان أو الكتابة اليه وذلك بحسب قيمة الخطين READ and WRITE.

[٢] ناتجة من حساب ٢ مرفوع للقوة ٢٠.

٢.١.٢. متحكم الذاكرة

قبل ذكر وظيفة هذا المتحكم يجب إعطاء نبذة عن ماهية المتحكمات (Controllers) في جهاز الحاسب. ويُعرَّف **المتحكم** بأنه شريحة تتحكم بعتاد ما تحوي العديد من المسجلات الداخلية وظيفتها هي استقبال الأوامر وتنفيذها على العتاد. ويمكن تعريفها كذلك بأنها شريحة للربط ما بين الأوامر البرمجية الى أوامر تنفذ على عتاد ما. وأي متحكم يحوي غالبا العديد من المسجلات سواءا 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 يمكن التعامل معه برمجياً ، هذا الرقم يعرف باسم المنفذ (Port) وسيتم الحديث عنه 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.
وعودة بالحديث عن متحكم الذاكرة الرئيسية والذي يتواجد غالبا 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 بداخله 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 نتمكن من إرسال الأوامر الصحيحة اليه.

هذه المسجلات تأخذ أرقاما معينة تسمى منافذ برمجية (Software Ports) بحيث تميز هذه الأرقام المسجلات من بعضها البعض[٣].

المنافذ Ports

يستخدم مفهوم المنافذ في علوم الحاسب للدلالة على عدة أشياء فمثلا في مجال برمجة الشبكات تكون برامج الخادم لها رقم منفذ معين حتى تسمح لبرامج العميل بالاتصال معها، كذلك ربما يقصد بالمفهوم المنافذ الموجودة في اللوحة الأم لوصل عتاد الحاسب بها ، أيضا قد يدل المفهوم على المسجلات الموجودة في المتحكمات على الجهاز وهذا ما سنقصده في حديثنا عن المنافذ في هذا البحث. و يمكن 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 الى مسجل في المتحكم . وعند استخدام أحد هذين الأمرين فان ذلك يعني أن العنوان موجه الى متحكم الإدخال والإخراج وليس الى متحكم الذاكرة حيث يقوم المعالج بتعين قيمة الخط `ACCESS` الموجود في مسار التحكم (Control Bus) وبالتالي يستجيب متحكم الإدخال والإخراج ويقرأ هذا العنوان ويقوم بتوجيهه الى 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 فان ذلك يعني كتابة هذه البيانات على متحكمات للأجهزة وليس على الذاكرة الرئيسية.
فمثلاً عند الكتابة على عنوان الذاكرة `0x0:0xa000` فان هذا يؤدي الى الكتابة على شاشة الحاسب نظراً لان 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 `port I/O` .
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 بتفيذها .

[٣]هناك بعض المسجلات لبعض المتحكمات تأخذ نفس الرقم ، لكن طبيعة الأمر المُرسل (قراءة أو كتابة) هو الذي يحدد المسجل الذي يجب التعامل معه.

جدول ٢.١.: 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	0xaffff	ذاكرة الفيديو Video RAM
0xb0000	0xb7777	ذاكرة الفيديو أحادية اللون Monochrome VRAM
0xb8000	0xbffff	ذاكرة الفيديو الملونة Color VRAM
0xc0000	0xc7fff	ذاكرة Video ROM BIOS
0xc8000	0xeffff	منطقة BIOS Shadow Area
0xf0000	0xfffff	نظام البايوس

٢.١.٢. دورة تنفيذ التعليمات

لكي يُنفذ المعالج البرامج الموجودة على الذاكرة فان هذا يتطلب بعضا من الخطوات التي يجب أن يقوم بها ، وفي كل دقة للساعة (Clock tick) يقوم المعالج بالبدء بخطوة من هذه الخطوات ، وفيما يلي سردا 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 أنتل أول اصدارة من معالجات ١٦ بت لم يكن هناك ما يعرف بأنماط المعالج حيث كان المعالج يعمل بنمط واحد وهو ما يعرف الان بالنمط الحقيقي (Real Mode) ، في هذا النمط يقوم المعالج بتنفيذ أي أمر موجه اليه ولا يوجد ما يُعرف بصلاحيات التنفيذ حيث يمكن لبرنامج للمستخدم أي يقوم بتنفيذ أمر يتسبب في ايقاف النظام عن العمل (مثل الأمر `hlt`) ، كذلك توجد عددٌ من المشاكل في هذا النمط فمثلا لا توجد حماية للذاكرة من برمجيات المستخدم ولا يوجد أي دعم لمفهوم تعدد المهام (Multitasking). لذلك سارعت أنتل بادخال عدة أنماط على بنية المعالج لتحل هذه المشاكل ، بحيث يُمكن للمعالج أي يعمل في أي نمط وأن يقوم بالتحويل وقتما شاء. ويُعرَّف **نمط المعالج** بأنه طريقة معينة يتبعها

جدول ٢.٢.: منافذ الإدخال والإخراج لحواسيب x86

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

المعالج أثناء عمله لتنفيذ الأوامر فمثلا يحدد النمط المستخدم ما إذا كان هناك حماية لعنواين 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 فان هذا ما جعلها تدع المعالج يبدأ بالنمط الحقيقي عند الإقلاع توافقاً مع الحواسيب القديمة ، وبعد ذلك عندما يستلم نظام التشغيل زمام التحكم بالحاسب فانه مخيرٌ ما بين الإستمرار بالعمل في هذا النمط وبالتالي يسمى هذا النظام **نظام تشغيل ١٦ بت** وبين تحويل نمط المعالج الى النمط الاخر وهو النمط المحمي (Protected Mode) وبالتالي يسمى النظام **نظام تشغيل ٣٢ بت**. و في هذا النمط يستخدم المعالج مسجلات من طول ١٦ بت (مثلاً المسجلات ax,bx,cx,dx,...etc) ويستخدم عنونة **المقطع:الإزاحة** (Segment:Offset) للوصول الى الذاكرة الرئيسية – سيتم شرحها في الفقرة التالية– وأيضا يدعم ذاكرة بحجم 1 ميجابايت ولا يقدم أي دعم لحماية الذاكرة والذاكرة التخيلية (Virtual Memory) ولا يوفر حماية للذاكرة من برمجيات المستخدم.

عنونة 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 جدا آنذاك نظراً لإن هذه المسجلات هي ذواكر من النوع SRAM وهو نوع مكلفاً على الرغم من إمكانياته العالية. ما فعلته انتل هو إيجاد طريقة مختلفة لعنونة الذاكرة فبدلاً من استخدام مسجل واحد للوصل 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[٤] ويُستخدم المسجل الآخر للوصل الى العناوين بداخل المقطع وهي ما تعرف بالقيم (Offsets) بداخل المقطع وبالتالي كل مقطع يحوي ٦٥٥٣٦ بايت (أي أن حجم المقطع هو ٦٤ كيلوبايت). إذاً يُعرَّف **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 المقاطع الموجودة بداخل المعالج (مثل المسجلات CS,SS,DS,ES) – سيتم شرحها لاحقا – ، ويمكن الوصول الى محتويات المقطع **الإزاحة Offset** وذلك

[٤] هذا ناتج من حساب 2^{16}.

بتحميل العنوان المطلوب الوصل 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 حاليا 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) [٥] ولكن بعد أن يتم ضربها في العدد ١٦ وذلك بسبب أن أي مقطع يبدأ بعد ١٦ بايت من المقطع السابق له . والتحويل يتم كالأتي :

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

فمثلا 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 الذاكرية 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 للوصل 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.

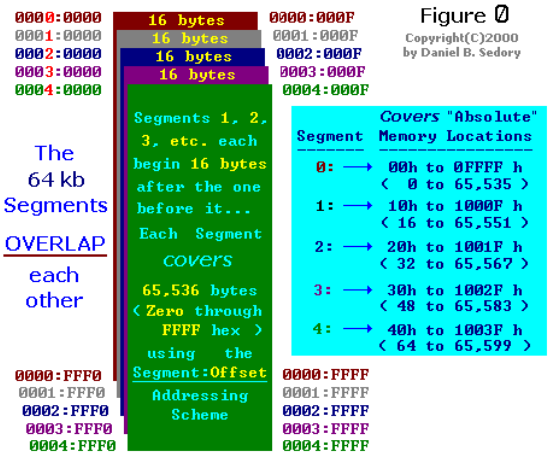
٤.٢.٢. النمط المحمي Protected Mode

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 من الوصول العشوائي من قبل برامج المستخدم وذلك عن طريق وصف مقاطع الذاكرة وصلاحيات الوصول اليها في جداول تسمى جداول الوصفات (Descriptor Table). المعالج 80386 هو أول معالج ٣٢ بت ويستخدم مسجلات بحجم ٣٢ بت وحجم مسار البيانات أيضا بنفس الحجم مما سمح بإمكانية التعامل مع ذاكرة بحجم ٤ جيجابايت .كذلك تم اضافة دعم للذاكرة التخيلية ومفهوم الصفحات (Paging) ودعم تعدد

[٥] 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 ٢.٤.: تداخل المقاطع في النمط الحقيقي



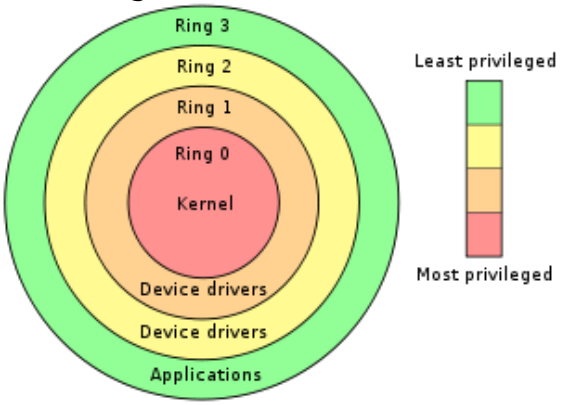
المهام. وفي هذا البحث سيتم الحديث عن معالجات ٣٢ بت باعتبارها أحد الأكثر انتشاراً حتى وقتنا هذا ،و على الرغم من ظهور معالجات ٦٤ بت إلا ان الدراسة حول معالجات ٣٢ بت تعتبر هي الأساس نظراً لان المعالجات الحديثة ما هي الا تطوير واضافات للمفاهيم الموجودة على المعالجات السابقة.

حلقات المعالج CPU Rings

عندما يعمل المعالج في النمط المحمي فان 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 مسموحٌ 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 ايقاف النظام عن العمل (المسؤولية تقع على البرنامج) لذلك غالبا 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 الى أي عنوان في الذاكرة بخلاف مساحة العنونة التخيلية (Virtual Address Space) الخاصة بالبرنامج نفسه وهذا ما رفع درجة حماية الذاكرة الى أقصى حد ممكن ، والشكل ٥.٢ يوضح هذه الحلقات وصلاحياتها. وعندما يبدأ النظام بالإقلاع فان المعالج يكون في النمط الحقيقي وهو نمط لا يحوي على حلقات حيث أنه يمكن تنفيذ كل الأوامر والوصول الى أي عنوان في الذاكرة ، وعند التحويل الى النمط المحمي (PMode) فان المعالج يكون

في الحلقة صفر (Kernel Mode) ، ويتم تحويل الحلقة الى حلقة معينة تلقائيا عند نقل التنفيذ الى عنوان في الذاكرة موصوف في جدول الواصفات بأنه يعمل بتلك الحلقة.

شكل ٢.٥.: حلقات المعالج



٢.٢.٥. 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 فان هذا سيتسبب في حدوث خطأ من المعالج يسمى الخطأ العام (General Protection Fault) والذي ان لم تتوفر دالة تتعامل معه (Exception Handler) فان هذا يؤدي الى توقف النظام عن العمل.
وتحوي معالجات x86 العديد من المسجلات منها ما يستخدم للأغراض العامة (General Registers) ومنها ما يستخدم لحفظ العناوين وأرقام المقاطع (Segments Registers) وتوجد أيضا مسجلات لا يمكن استخدامها إلا في برامج الحلقة صفر (أي النواة) حيث أن التغيير فيها يؤثر على عمل النظام وأخيرا هناك مجموعة من المسجلات الداخلية للمعالج والتي لا يمكن الوصول لها برمجياً. والقائمة التالية توضح هذه المسجلات :

- مسجلات عامة : RAX (EAX(AX/AH/AL)), RBX (EBX(BX/BH/BL)), RCX (ECX(CX/CH/CL)), RDX (EDX(DX/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.

جدول ٢.٣.: الأوامر التي تتطلب صلاحية الحلقة صفر

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

- مسجلات التحكم: CR0, CR1, CR2, CR3, CR4, CR8.
- مسجلات الإختبار: 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 الإستخدامات الغالبة لكلٌ 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 المكدس 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 الى مقاطع البرامج والبيانات وإنما تشير الى واصفات معينة في جدول الواصفات العام ، هذه الواصفات تحدد عنوان بداية المقطع ونوع المقطع (يحوي شفرات أم بيانات) وتحدد صلاحية التنفيذ وصلاحية والقراءة والكتابة فيها – كما سنرى ذلك في الفصل الرابع بإذن الله–.

مسجلات الإزاحة Offset Registers

بجانب مسجلات المقاطع فإن الوصول الى 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 المكدس ويمكن استخدام للأشارة على أي عنوان في أي مقطع آخر.

- المسجل SP: يحوي عنوان الإزاحة بداخل مقطع المكدس.

مؤشر التعليمة Instruction Pointer

هذا المسجل (IP) يمثل إزاحة بداخل مقطع الشفرة (CS) وهو يحوي عنوان التعليمة التالية التي سيقوم المعالج بتنفيذها ، والعنوان CS:IP يمثل العنوان الفيزيائي للتعليمة 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 تعليمة 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 وظيفه محدده ، 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 Priviledge Level أعلى من أو مساوياً للقيمة الموجودة في البتين IOPL[٧] ، 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 CR0, 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 يهمنا فقط أول مسجل تحكم وهو CR0 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) ، والتركيبة التالية توضح محتويات كل بت في مسجل التحكم CR0 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

[٧]أعلى مستوى حماية هو الحلقة صفر (Ring0) ويليها الحلقة ١ ثم ٢ و٣.

جدول ٤.٢.: مسجل الأعلام EFLAGS

رقم البت	اسم البت	الإستخدام
0	CF	Carry Flag - Status bit
1	-	محجوزة
2	PF	Parity Flag
3	-	محجوزة
4	AF	Adjust Flag - Status bit
5	-	محجوزة
6	ZF	Zero Flag - Status bit
7	SF	Sign Flag - Status bit
9	TF	Trap Flag - System Flag
9	IF	Interrupt Enabled Flag - System Flag
10	DF	Direction Flag - Control Flag
11	OF	Overflow Flag - Status bit
12-13	IOPL	I/O Priviledge Level - Control Flag
14	NT	Nested Task Flag - Control Flag
15	-	محجوزة
16	RF	Resume Flag (386+ Only) - Control Flag
17	VM	v8086 Mode Flag (386+ Only) - Control Flag
18	AC	Alignment Check (486SX+ Only) - Control Flag
19	VIF	Virtual Interrupt Flag (Pentium+ Only) - Control Flag
20	VIP	Virtual Interrupt Pending (Pentium+ Only) - Control Flag
21	ID	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 coprocesor 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 محملٌ له ، هذا المحمل يعمل على نسخ نواة النظام من أحد الأقراص الى الذاكرة الرئيسية ثم ينقل التنفيذ الى النواة ، وهكذا تنتهي دورة عمل المحمل ويبدأ نظام التشغيل متمثلا في النواة بالبدء بتنفيذ الاوامر والمهمات وتلبية إحتياجات المستخدم. في هذا الفصل سندرس كيفية برمجة المحمل وماهيته وسيتم الاقلاع من قرص مرن بنظام FAT12 ، فالغرض هذه المرحلة هو دراسة أساسيات المحمل وتحميل وتنفيذ نواة مبسطة .

١.٣. 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[١] الى اللوحة الام (MotherBoard) والتي تقوم بتوجيهها الى وحدة موزد الطاقة (Power Supply Unit). بعد ذلك يأتي دور وحدة PSU لكي تقوم بمهمة تزويد الحاسب وملحقاته بالكمية المطلوبة من الطاقة، وإرسال اشارة Power Good إلى اللوحة الام وبالتحديد الى نظام ال BIOS . تدل هذه الاشارة على أنه تم تزويد الطاقة الكافية ، وفورا سيبدأ برنامج الفحص الذاتي (Power on Self Test) الذي يعرف اختصاراً ب POST بفحص أجهزة ومحلقات الحاسب (مثل الذاكرة ولوحة المفاتيح والماوس والناقل التسلسلي ...الخ) والتأكد من أنها سليمة. بعدها يقوم ال POST بنقل التحكم الى نظام ال BIOS حيث سيقوم ال POST بتحميل ال BIOS الى نهاية الذاكرة 0xFFFF0 و سيقوم أيضا بوضع تعليمة قفز (jump) في أول عنوان في الذاكرة الى نهاية الذاكرة ، كذلك من مهام ال POST هي تصفير المسجلين CS:IP وهذا يعني أن أول تعليمية سينفذها المعالج هي تعليمة القفز الى نهاية الذاكرة وبالتحديد الى ال BIOS . يستلم ال BIOS التحكم ويبدأ في انشاء جدول المقاطعات (Interrupt Vector Table) وتوفير العديد من المقاطعات ،ويقوم بالمزيد من عمليات الفحص والاختبار للحاسب ، وبعد ذلك يبدأ في مهمة البحث عن نظام تشغيل في الاجهزة الموجودة بناءا على ترتيبها في اعدادات ال BIOS في برنامج Setup ،وفي حالة لم يجد ال BIOS جهازا قابلا للاقلاع في كل القائمة فانه يصدر رسالة خطأ بعدم توفر نظام تشغيل ويوقف الحاسب عن العمل (Halt) ، وفي حالة توفر جهازاً قابلاً للإقلاع سيقوم ال BIOS بتحميل القطاع الأول منه (يحوي هذا القطاع على برنامج المحمل) الى الذاكرة الرئيسية وبالتحديد الى العنوان الفيزيائي 0x07c00 وسيُنقَل التنفيذ الى المحمل.

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

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

٣.٢. محمل النظام Bootloader

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

١. حجم البرنامج يجب أن يكون 512 بايت بالضبط.

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

٣. أن يحوي شفرة تحميل النواة ونقل التنفيذ اليها.

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

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

- القطاع رقم 1

[٢] مثل خاصية ال Safe Mode

- المسار رقم 0
- الرأس رقم 0

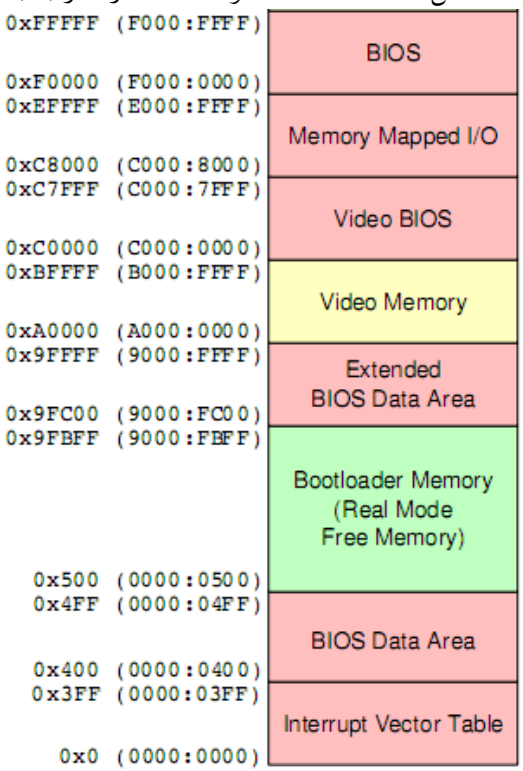
وتحقيقُ هذا الشرط ليس بالأمر المعقد خصوصا مع توفر العديد من الادوات التي تساعد على نسخ مقطع من قرص ما الى مقطع في قرص آخر ، أما الشق الثاني من الشرط فهو متعلق بتمييز القطاع الاول كقطاع قابل للاقلاع من غيره ، حيث حتى يكون القطاع قابلا للاقلاع فانه يجب أن يحمل التوقيع 0xAA55 في البايت رقم 510 و 511 . وبدون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

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

٣.٣. مخطط الذاكرة

أثناء مرحلة الإقلاع وعندما يُنقل التنفيذ الى محمل النظام فان الذاكرة الرئيسية تأخذ الشكل ١.٣ .وأول ١٠٢٤ بايت تستخدم من قبل جدول المقاطعات الذي يحوي عنوان دالة التنفيذ لكل مقاطعة للبايوس ،

شكل ٣.١.: مخطط محتويات الذاكرة الرئيسية



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

٣.٤. برمجة محمل النظام

المثال ٣.١ يوضح أصغر محمل للنظام يمكن كتابته وتنفيذه ، باستخدام المجمع NASM[٣] وهو مجمع متعدد المنصات ويوفر ميزة انتاج ملفات ثنائية object code .

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

Example 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—($—$$)  db    0   ; append zeros.
١١
١٢     ; $ is the address of first instruction (should be 0x07c00).
١٣     ; $$ is the address of current line.
١٤     ;  $—$$ means how many byte between start and current.
١٥
١٦     ; if cli and hlt take 4 byte then time directive will fill
١٧     ; 510—4 = 506 zero's.
١٨
١٩     ; finally the boot signature 0xaa55
٢٠     db    0x55  ; first byte of a boot signature.
٢١     db    0xaa  ; second byte of a  boot signature.
```

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

١.٤.٣. 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 int 0x10 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      ; intit registers
٥٠      ;————————————————
٥١
٥٢      ; because bootloader are loaded at 0x07c00 we can refrence this
           location with many different combination of segment:offset
           addressing.
٥٣
٥٤      ; So we will use either 0x0000:0x7c000 or 0x:07c0:0x0000 , and in
             this example we use 0x07c0 for segment and 0x0 for offset.
٥٥
٥٦      mov ax,0x07c0
٥٧      mov ds,ax
٥٨      mov es,ax
٥٩
٦٠      mov si,hello_msg
٦١      call puts16
٦٢
٦٣      cli     ; clear interrupt.
٦٤      hlt     ; halt the system.
٦٥
٦٦      times 510—($—$$)  db    0   ; append zeros.
٦٧
٦٨      ; finally the boot signature 0xaa55
٦٩      db    0x55
٧٠      db    0xaa
```

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

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

٣.٤.٢. معلومات قطاع الاقلاع

إضافة الى محمل النظام فان قطاع الإقلاع boot sector يجب أن يحوي كذلك على معلومات تساعد في وصف نظام الملفات المستخدم ووصف القرص الذي سيتم الاقلاع منه ، هذه المعلومات تحوي معرف OEM وتحوي بيانات BIOS Parameter Block (تختصر ب BPB) ويجب أن تبدأ كل هذه البيانات من البايت رقم 3[٤٣]. 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.
```

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

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

; Extended BPB.

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

المثال ٣.٤ يوضح شفرة المحمل بعد اضافة بيانات OEM and BPB.

Example ٣.٤: BPB example

```

;Hello Bootloader

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

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


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

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

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

```

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

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

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

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

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

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

Example 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.
٢٤ media_descriptor     db      0xf0         ; 1.44 floppy disk.
```

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

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

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

٣.٤.٣. تحميل قطاع من القرص باستخدام المقاطعة int 0x13

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

إعادة القرص المرن

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

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

النتيجة:

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

مثال:

Example ٣.٧: Reset Floppy Drive

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

قراءة المقاطع sectors

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

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

النتيجة:

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

مثال:

Example 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.
```

٣.٥. مقدمة الى نظام FAT12

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

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

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

١.٥.٣. قيود نظام FAT12

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

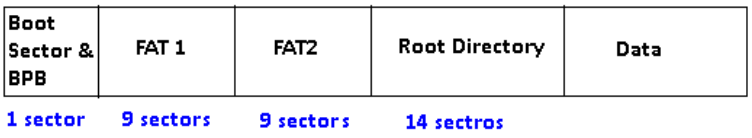
- عدم دعمه للمجلدات الهرمية ويدعم فقط مجلد واحد يسمى الجذر Root Directory.
- طول العنقود (Cluster) هو 12 بت ، بمعنى أن عدد الكلسترات هي 2^{12}.
- أسماء الملفات لا تزيد عن 12 بت.
- يستوعب كحد أقصى 4077 ملف فقط.
- حجم القرص يحفظ في 16 بت ، ولذا فانه لا يدعم الاقراص التي حجمها يزيد عن 32 MB.
- يستخدم العلامة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

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

٢.٥.٣. هيكلة نظام FAT12 على القرص

عند تهئية القرص المرن[٦] (Format) بنظام FAT12 فان تركيبة القرص تكون على الشكل ٢.٣ :

شكل ٢.٣.: هيكلة نظام FAT12 على القرص



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

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

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

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

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

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

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

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

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

٣.٥.٣. هيكلة القرص المرن

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

٤.٥.٣. القراءة و الكتابة من نظام FAT12

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

١. تحميل جدول Root Directory من القرص الى الذاكرة ومن ثم البحث عن ملف المرحلة الثانية وأخذ رقم أول كلستر له.
٢. تحميل جدول FAT من القرص الى الذاكرة ومن ثم تحميل جميع الكلسترات للملف.
٣. نقل التنفيذ الى أول بايت في المرحلة الثانية من محمل النظام.

إنشاء المرحلة الثانية من محمل النظام

بداية سنقوم بإنشاء المرحلة الثانية من محمل النظام ونسخها الى القرص المرن ، ونظراً لان تطوير نظامنا الخاص يجب ان يتم تحت نظام آخر فان هذا النظام الآخر غالبا ما يحوي درايفر لنظام ملفات FAT12 حيث يتكفل بعملية كتابة البيانات الى جدول Root Directory بالاضافة الى البحث عن كلسترات خالية في جدول FAT دون أي تدخل من قبل مطور النظام الجديد، لذلك في هذه المرحلة من التطوير سنتجاهل جزئية الكتابة في نظام FAT12 ونترك المهمة لنظام التشغيل الذي نعتمد عليه في عملية تطوير النظام الجديد ، وبهذا سيكون الدرايفر الذي سننشئه في هذا الفصل ما هو الا جزء من الدرايفر الكامل الذي سيتم تكلمته لاحقا 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 أولا والبحث عن ملف المرحلة الثانية (ذو الاسم الخارجي stage2.sys) وعند البحث يجب البحث بالاسم الداخلي الذي يستخدمه نظام الملفات لذلك يجب أن نبحث عن الملف "stage2 sys" ، ونأتي برقم الكلستر الأول للملف.

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

Example ٣.١٠: Load Root directory

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

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

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

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

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


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

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

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

Example ٣.١١: Find Stage2 Bootloader

```
;———————————————————
; Find stage2.sys
;———————————————————

  mov di,0x0200       ; di point to first entry in root dir.
  mov cx,word[root_directory] ; loop 224 time.

find_stage2:

  mov si,kernel_loader_name
  push cx
  push di
  mov cx,11          ; file name are 11 char long.

  rep cmpsb
  pop di
  je find_successfully

  mov di,32        ; point to next entry.
  pop cx

  loop find_stage2

  ; no found ?
  jmp find_fail

find_successfully:
;———————————————————
; Get first Cluster.
;———————————————————

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

تحميل جدول FAT الى الذاكرة

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

Example ٣.١٢: Load FAT Table

```
;-------------------------
; Compute FAT size
;-------------------------

  xor cx,cx
  xor ax,ax
  xor dx,dx

  mov al,byte[total_fats]     ; there are 2 fats.
  mul word[sectors_per_fat]   ; 9*2 sectors
  xchg ax,cx

;-----------------------------
; Get start sector of FAT
;-----------------------------

  add ax,word[reserved_sectors]

;---------------------------------------------------------------
; Load FAT at 0x07c0:0x0200
; Overwrite Root dir with FAT, no need to Root Dir now.
;---------------------------------------------------------------

  mov bx,0x0200
  call read_sectors
```

تحميل كلسترات الملف

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

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

Example 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 لاحقا عند التحويل الى النمط المحمي بدالة اخرى 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 لاخر 4 بت (لازالة ما تم قرائته من السجل الثاني). وفي حالة كان رقم الكلستر هو 1 فيجب قراءة السجل الثاني من جدول FAT والذي يبدأ من البت 12–23 وبسبب أنه لا يمكن قراءة 12 بت سنقوم بقراءة 16 بت أي من البت 8–23 وازالة أول 4 بت.

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

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

Example 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,0x0fff
٢٩      jmp next_cluster
٣٠
٣١    odd_cluster:
٣٢
٣٣      shr dx,4
٣٤
٣٥
٣٦    next_cluster:
٣٧      mov word[cluster_number],dx   ; next cluster to load.
٣٨
٣٩      cmp dx,0x0ff0         ; check end of file, last cluster?
٤٠      jb load_cluster         ; no, load the next cluster.
٤١
٤٢
٤٣      ; yes jmp to end
٤٤      jmp end_of_first_stage
٤٥
٤٦    find_fail:
٤٧
٤٨      mov si,fail_msg
٤٩      call puts16
٥٠
٥١      mov ah,0x0
٥٢      int 0x16    ; wait keypress.
٥٣      int 0x19    ; warm boot.
٥٤
٥٥
٥٦    end_of_first_stage:
٥٧
٥٨      ; jump to stage2 and begin execute.
٥٩      push 0x050      ; segment number.
٦٠      push 0x0      ; offset number.
٦١
٦٢      retf      ; cs:ip = 0x050:0x0
```

```
٦٣
٦٤      times 510-($-$$)  db     0    ; append zeros.
٦٥
٦٦      ; finally the boot signature 0xaa55
٦٧      db     0x55
٦٨      db     0xaa
```

٤. برمجة محمل النظام – المرحلة الثانية

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

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

١.٤. الانتقال الى النمط المحمي

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

٤.١.٤. جدول الواصفات العام Global Descriptor Table

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

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

بيانات أي واصفة Descriptor تأخذ الجدول التالي:

- البتات 0–15: تحوي أول بايتين (من بت 0 –15) من حجم المقطع.
- البتات 16–39: تحوي أول ثلاث بايتات من عنوان بداية المقطع Base Address.
- البت 40: بت الوصول Access Bit (يستخدم مع الذاكرة الظاهرية Virtual Memory).
- البتات 43-41: نوع الواصفة Descriptor Type:
  - البت 41: القراءة والكتابة:
    - Data Descriptor: القيمة 0 للقراءة فقط والقيمة 1 للقراءة والكتابة.
    - Code Descriptor: القيمة 0 للتنفيذ فقط execute والقيمة 1 للقراءة والتنفيذ.
  - البت 42: Expansion direction (Data segments), conforming (Code Segments).
  - البت 43: قابلية التنفيذ:
    - 0: اذا كان المقطع عبارة عن بيانات.
    - 1: اذا كان المقطع عبارة عن شفرة.
- البت 44: Descriptor Bit:
  - 0: System descriptor.
  - 1: Code or Data Descriptor.
- البتات 45–46: مستوى الحماية Privilege Level
  - 0: Highest (Ring 0).
  - 3: Lowest (Ring 3).

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

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

Example ٤.١: GDT

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

begin_of_gdt:

; Null Descriptor: start at 0x0.

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

; Code Descriptor: start at 0x8.

   dw  0xffff    ; limit low.
   dw  0x0     ; base low.
   db  0x0     ; base middle.
   db  10011010b ; access byte.
   db  11001111b ; granularity byte.
   db  0x0     ; base high.
```

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

هذا الجدول يبدأ بالواصفة الخالية Null Descriptor وحجمها 8 بايت ومتحوياتها تكون صفراً في العادة ، أما الواصفة التالية لها فهي واصفة مقطع الشفرة Code Descriptor وتوضح المقطع من الذاكرة الذي سيتسخدم كشفرة وما هي بدايته وحجمه وصلاحيات استخدامه حيث يمكن أن نسمح فقط للبرامج التي تعمل على مستوى النواة Kernel Mode بالدخول الى هذا المقطع.وفيما يلي شرح لمحتويات هذه الواصفة ويمكنك المطابقة مع الجدول الذي يوضح الشكل العام لكل واصفة.

تبدأ واصفة الكود Code Descriptor من العنوان 0x8 وهذا العنوان مهم جدا حيث سيكون هذا العنوان هو قيمة المسجل CS ، والبتات من 0–15 تحدد حجم المقطع Segment Limit والقيمة هي 0xffff تدل على أن أكبر حجم يمكن التعامل معه هو 0xffff.

البتات من 16–39 تمثل البتات 0–23 من عنوان بداية المقطع Base Address والقيمة التي تم اختيارها هي 0x0 وبالتالي نعرف أن عنوان بداية مقطع الكود هو 0x0 وعنوان النهاية 0xffff .

البايت رقم 6 ويسمى Access Byte يحدد العديد من الخصائص وفيما يلي توضيح لمعنى كل بت موجودة فيه:

- البت 0: Access Bit ويستخدم مع الذاكرة الظاهرية لذلك اخترنا القيمة 0.
- البت 1: بت القراءة والكتابة ، وتم اختيار القيمة 1 لذا يمكن قراءة وتنفيذ أي بايت موجودة في مقطع الكود من 0x0–0xffff.
- البت 2: expansion direction لا يهم حاليا لذا القيمة هي 0.
- البت 3: تم اختيار القيمة 1 دلالة على أن هذا مقطع شفرة Code Segment.
- البت 4: تم اختيار القيمة 1 دلالة على أن هذا مقطع للشفرة او للبيانات وليس للنظام.
- البتات 5–6: مستوى الحماية وتم اختيار القيمة 0 دلالة على أن هذا المقطع يستخدم فقط في الحلقة صفر Ring0 أو ما يسمى Kernel Mode.
- البت 7: تستخدم مع الذاكرة الظاهرية لذا تم اهمالها.

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

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

البايت الاخير في واصفة مقطع الكود (البايت رقم 8) يمثل البتات من 24-32 من عنوان بداية مقطع الكود والقيمة هي 0x0 وبالتالي عنوان بداية مقطع الكود الكلي هو 0x0 أي من أول بايت في الذاكرة.
إذاً واصفة مقطع الكود Code Descriptor حددت عنوان بداية مقطع الكود ونهايته وكذلك صلاحية التنفيذ وحددت بأن المقطع هو مقطع كود Code Segment.
الواصفة التالية هي واصفة مقطع البيانات Data Descriptor وتبدأ من العنوان رقم 0x10 وهي مشابهة تماما لواصفة الكود باستثناء البت رقم 43 حيث يحدد ما اذا كان المقطع كود أم بيانات.
وبعد إنشاء هذا الجدول (GDT) في الذاكرة ، يجب أن يَحمل المسجل gdtr على حجم هذا الجدول ناقصا واحد وعلى عنوان بداية الجدول، ويتم ذلك عن 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 gdtr
١٣     sti          ; enable interrupt
١٤     popa         ; restore registers.
١٥
```

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

٤.١.٢. العنونة في النمط المحمي PMode Memory Addressing

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

٤.١.٣. 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.
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.

[١] بفرض أن بوابة A20 تم تفعيلها.

Example ٤.٣: Switching to Protected Mode

```
١   ;————————————————————
٢   ; Load gdt into gdtr.
٣   ;————————————————————
٤
٥     call load_gdt
٦
٧   ;————————————————
٨   ; Go to PMode.
٩   ;————————————————
١٠    ; just set bit 0 from cr0 (Control Register 0).
١١
١٢    cli          ; important.
١٣    mov eax,cr0
١٤    or eax,0x1
١٥    mov cr0,eax      ; entering pmode.
١٦
١٧
١٨  ;————————————————————
١٩  ; Fix CS value
٢٠  ;————————————————————
٢١    ; select the code descriptor
٢٢    jmp 0x8:stage3
٢٣
٢٤
٢٥ ;*************************
٢٦ ; entry point of stage3
٢٧ ;*************************
٢٨
٢٩ bits 32      ; code now 32—bit
٣٠
٣١ stage3:
٣٢
٣٣  ;————————————————;
٣٤  ; Set Registers.
٣٥  ;————————————————;
٣٦
٣٧    mov ax,0x10      ; address of data descriptor.
٣٨    mov ds,ax
٣٩    mov ss,ax
```

```
٤٠      mov es,ax
٤١      mov esp,0x90000   ; stack begin from 0x90000.
٤٢
٤٣
٤٤    ;————————————————;
٤٥    ; Hlat the system.
٤٦    ;————————————————;
٤٧      cli     ; clear interrupt.
٤٨      hlt     ; halt the system.
```

٤.٢. تفعيل البوابة A20

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

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 والقراءة والكتابة من مسجلات المتحكم الداخلية . وبسبب ارتباط بوابة A20 مع متحكم لوحة المفاتيح فانه لا بد من التعامل مع هذا المتحكم لتفعيل البوابة ، وهذا يتم عن طريق استخدام أوامر المعالج in والامر out.

[٢] توجد البوابة تحديداً على خط العناوين رقم 20

وبخصوص متحكم لوحة المفاتيح (متحكم 8042) فغالبا ما تأتي على شكل شريحة Integrated Circuit أو تكون مضمنة داخل اللوحة الأم (Motherboard) وتكون في ال South Bridge.ويرتبط هذا المتحكم مع متحكم آخر بداخل لوحة المفاتيح ، وعند الضغط على زر ما فانه يتم توليد Make Code ويُرسل الى المتحكم الموجود بداخل لوحة المفاتيح والذي بدروه يقوم بارساله الى متحكم 8042 عن طريق منفذ الحاسب (Hardware Port) .وهنا يأتي دور متحكم 8042 حيث يقوم بتحويل Make code الى Scan Code ويحفظها في مسجلاته الداخلية Buffer هذا المسجل يحمل الرقم 0x60 في أجهزة IBM and Compatible PC، وهذا يعني أنه في حالة قراءة هذا المسجل (عن طريق الأمر in) فانه يمكن قراءة القيمة المدخلة.
وفي الفصل الثامن 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 محمولية وبعض الاجهزة لا تدعمها ، وفيما يلي توضيح للبتات على هذا المنفذ:

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

والمثال التالي يوضح طريقة تفعيل البوابة .

Example 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 الاجهزة وربما يكون هناك ارقام مختلفة للمنافذ ، ويعتمد في الآخر على مصنعي اللوحات الام ويجب قراءة كتيباتها لمعرفة العناوين.

بواسطة البايوس

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

Example 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 الاوامر الى المتحكم ولقراءة حالة

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

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

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

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

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

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

Example ٤.٧: Enable A20 by Send 0xdd

```
١
٢ ;*************************************************
٣ ; enable_a20_keyboard_controller:
٤ ;   Enable A20 with command 0xdd
٥ ;*************************************************
٦
٧ enable_a20_keyboard_controller:
٨
٩      ;cli
```

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

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

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

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

Example 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 bescause when output came we know
          that operation are executed.
٢٨
٢٩    ;-------------------------
٣٠    ; read input buffer
٣١    ;-------------------------
٣٢    in al,0x60
٣٣    push eax    ; save data.
٣٤    call wait_input
٣٥
٣٦    ;-----------------------------------
٣٧    ; send write output port command.
٣٨    ;-----------------------------------
```

```
٣٩      mov al,0xd1   ; write output port command.
٤٠      out 0x64,al
٤١      call wait_input
٤٢
٤٣      ;─────────────
٤٤      ; enable a20.
٤٥      ;─────────────
٤٦      pop eax
٤٧      or al,2     ; set bit 2.
٤٨      out 0x60,al
٤٩      call wait_input
٥٠
٥١      ;──────────────────
٥٢      ; Enable Keyboard.
٥٣      ;──────────────────
٥٤      mov al,0xae   ; Enable Keyboard command.
٥٥      out 0x64,al
٥٦      call wait_input
٥٧
٥٨
٥٩      popa      ; restore registers
٦٠      sti
٦١
٦٢      ret
```

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

٣.٤. أساسيات ال VGA

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

Video Buffer, Video DAC , CRT Controller , Sequencer unit , Graphics Controller و Attribute Controller[٣].

ال Video Buffer هو مقطع من الذاكرة segment of memory يعمل كذاكرة للشاشة Memory Mapped ، وعند بداية التشغيل فان البايوس يخصص مساحة من الذاكرة بدءا من العنوان 0xa0000 كذاكرة للشاشة وفي حالة تم الكتابة الى هذه الذاكرة فان هذا سوف يغير في الشاشة ، هذا الربط يسمى Memory Mapping، أما ال Graphics Controller فهو الذي يقوم بتحديث محتويات الشاشة بناءاً على البيانات الموجودة في ال Video buffer.

وتدعم ال VGA نمطين للعرض الاول هو النمط النصي Text Mode والاخر هو النمط الرسومي APA Graphics Mode ويحدد النمط طريقة التعامل مع ال Video buffer وكيفة عرض البيانات.

النمط الرسومي All Point Addressable Graphics Mode يعتمد على البكسلات ، حيث يمكن التعامل مع كل بسكل موجود على حدة . والبكسل هو أصغر وحدة في الشاشة وتعادل نقطة على الشاشة. أما النمط النصي Text Mode فيعتمد على الحروف Characters ، ولتطبيق هذا النمط فان متحكم الشاشة Video Controller يستخدم ذاكرتين two buffers الاولى وهي خريطة الحروف Character Map وهي تعرف البكسلات لكل حرف ويمكن تغيير هذه الخريطة لدعم أنظمة محارف أخرى، أما الذاكرة الثانية فهيScreen Buffer وبمجرد الكتابة عليها فان التأثير سيظهر مباشرة على الشاشة.

ومقياس VGA هو مبني على المقاييس السابقة ، ابتداءا من مقياس Monochrome Display Adapter ويسمى اختصارا MDA والذي طورته IBM في عام 1981 ، و MDA لا تدعم النمط الرسومي والنمط النصي بها (يسمى Mode 7) يدعم 80 عمود و 24 صف (80*25). وفي نفس العام قامت IBM بتطوير مقياس Color Graphics Adapter (واختصارا CGA) الذي كان أول متحكم يدعم الالوان حيث يمكن عرض 16 لون مختلف.وبعد ذلك تم تطوير Enhanced Graphics Adapter.

ويجدر بنا التذكير بان متحكمات VGA متوافقة مع المقاييس السابقة Backward Compatible فعندما يبدأ الحاسب في العمل فان النمط سيكون النمط النصي Mode 7 (الذي ظهر في MDA) ، وهذا يعني اننا سنتعامل مع 80 عمود و 25 صف.

١.٣.٤. عنونة الذاكرة في متحكمات VGA

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

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

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

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

Example ٤.٩: Print 'A' character on screen

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

٢.٣.٤. طباعة حرف على الشاشة

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

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

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

$$address = x + y * 80$$

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

; now add the base address of video memory.

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

وبارسال الحرف الى العنوان 0xb8144 فان الحرف سوف يظهر على الشاشة في الصف الخامس والعمود الخامس (الترقيم يبدأ من صفر وأول صف وعمود رقمها صفر).
وكما ذكرنا ان النمط النصي Mode 7 هو الذي يبدأ الحاسب به ، في هذا النمط يتعامل متحكم العرض

مع بايتين من الذاكرة لكل حرف يراد طباعته ، بمعنى اذا ما أردنا طباعة الحرف A فانه يجب ارسال الحرف الى العنوان 0xb8000 وخصائص الحرف الى العنوان التالي له 0xb8001 وهذا يعني انه يجب تعديل قانون التحويل السابق واعتبار أن كل حرف يأخذ بايتين من الذاكرة وليس بايت واحد.
البايت الثاني للحرف يحدد لون الحرف وكثافة اللون (غامق وفاتح) والجدول التالي يوضح البتات فيه:

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

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

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

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

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

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

$videomemory = 0xb0000$

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

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

٣.٣.٤. طباعة السلاسل النصية strings

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

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

٤.٣.٤. تحديث المؤشر Hardware Cursor

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

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

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

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

$$videomemory = x + y * 80$$

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

Example 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
```

```
move_cursor:

    pusha      ; Save Registers.

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

    xor ecx,ecx
    mov cl,byte[x_pos]

    mov eax,COLUMNS
    mul byte[y_pos]

    add eax,ecx
    mov ebx,eax

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

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

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

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

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

    mov al,bh
    mov dx,0x3d5
    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 (80*25 في نمط Mode 7) 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
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[٤]. وفي هذا الجزء سيتم كتابة نواة تجريبية بهدف التأكد من عملية نقل التحكم الى النواة وكذلك بهدف إعادة كتابة شفرة محمل النظام بشكل أفضل.
وسيتم استخدام لغة التجميع لكتابة هذه النواة التجريبية حيث أن الملف الناتج سيكون Pure Binary ولا يحتاج الى محمل خاص ، وابتداءاً من الفصل القادم سنترك لغة التجميع جانبا ونبدأ العمل بلغة السي والسي++ .
وبما أننا نعمل في النمط المحمي PMode فلا يمكننا أن نستخدم مقاطعة البايوس int 0x13 لتحميل قطاعات النواة الى الذاكرة ، ويجب أن نقوم بكتابة درايفر لمحرك القرص المرن أو نقوم بتحميل النواة الى الذاكرة قبل الانتقال الى النمط المحمي وهذا ما سنفعله الان ، وسنترك جزئية برمجة محرك القرص المرن لاحقا.
وحيث أن النمط المحمي يسمح باستخدام ذاكرة حتى 4 جيجا ، فان النواة سنقوم بتحميلها على العنوان 0x100000 أي عند 1 ميجا من الذاكرة .لكن علينا التذكر بأن النمط الحقيقي لا يدعم الوصول الى العنوان 0x100000 لذلك سنقوم بتحميل النواة أولا في أي عنوان خالي وليكن 0x3000 وعند الانتقال الى النمط المحمي سنقوم بنسخها الى العنوان 0x100000 ونقل التنفيذ والتحكم اليها.
والشفرة التالية توضح نواة ترحيبية.

Example 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
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
```

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

```
١٦ logo_message  db    0xa,0xa,0xa,"                  ---  ---  --------  -
       / __ \ / __/"
١٧          db    0xa,        "               / —_) _ `/ __/ _ `/ / /_/ /
                 _\ \  "
١٨          db    0xa,        "               \__/\_, /_/  \_,_/  \____//
                 ___/  "
١٩          db    0xa,        "                     /_/
                                        ",0
٢٠ ;*********************
٢١ ; Entry point.
٢٢ ;*********************
٢٣
٢٤ kernel_entry:
٢٥
٢٦   ;————————————————
٢٧   ; Set Registers
٢٨   ;————————————————
٢٩
٣٠     mov ax,0x10     ; data selector.
٣١     mov ds,ax
٣٢     mov es,ax
٣٣     mov ss,ax
٣٤     mov esp,0x90000   ; set stack.
٣٥
٣٦   ;————————————————————————————————
٣٧   ; Clear Screen and print message.
٣٨   ;————————————————————————————————
٣٩
٤٠     call clear_screen
٤١
٤٢     mov ebx,kernel_message
٤٣     call puts32
٤٤
٤٥     mov ebx,logo_message
٤٦     call puts32
٤٧
٤٨   ;————————————————————
٤٩   ; Halt the system.
٥٠   ;————————————————————
٥١
٥٢     cli
```

```
٥٣      hlt
```

والمرحلة الثانية من محمل النظام ستكون هي المسؤولة عن البحث عن النواة وتحميلها ونقل التنفيذ اليها ، وسيتم تحميلها الى الذاكرة قبل الانتقال الى النمط المحمي وذلك حتى نتكمن من استخدام مقاطعة البايوس `int 0x13` وعند الانتقال الى النمط المحمي سيتم نسخ النواة الى عنوان 1 ميجا ونقل التحكم الى النواة.
ولتحميل النواة الى الذاكرة يجب أولا تحميل Root Directory الى الذاكرة والبحث عن ملف النواة وفي حالة كان الملف موجودا سيتم قراءة عنوان أول كلستر له ، هذا العنوان سيعمل ك index في جدول FAT (والذي يجب تحميله الى الذاكرة ايضا) وسيتم قراءة القيمة المقابلة لهذا ال index والتي ستخبرنا هل ما اذا كان هذا الكلستر هو آخر كلستر للملف أم لا[٥].
والشفرة التالية توضح ملف المرحلة الثانية من المحمل stage2.asm ، وتم تقسيم الكود بشكل أكثر تنظيما حيث تم نقل أي دالة تتعلق بالقرص المرن الى الملف floppy.inc (ملف inc. هو ملف للتضمين في ملف آخر) ، والدوال المتعلقة بنظام الملفات موجودة على الملف fat12.inc ودوال الاخراج موجودة في stdio.inc ودوال تفعيل بوابة A20 موجودة على الملف a20.inc ودالة تعيين جدول الواصفات العام وكذلك تفاصيل الجدول موجودة في الملف gdt.inc ، اخيرا تم انشاء ملف common.inc لحفظ بعض الثوابت المستخدمة دائما[٦].

Example 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.
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
```

[٥] راجع الفصل السابق لمعرفة التفاصيل.
[٦] جميع شفرات الملفات مرفقة مع البحث في مجلد example/ch3/boot/ وشفرة المحمل النهائية ستكون ملحقة في نهاية البحث.

```
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
٥٣
٥٤    ;———————————————————
٥٥    ; Enable A20.
٥٦    ;———————————————————
٥٧      call enable_a20_keyboard_controller_output_port
٥٨
٥٩    ;—————————————————
٦٠    ; Display Message.
```

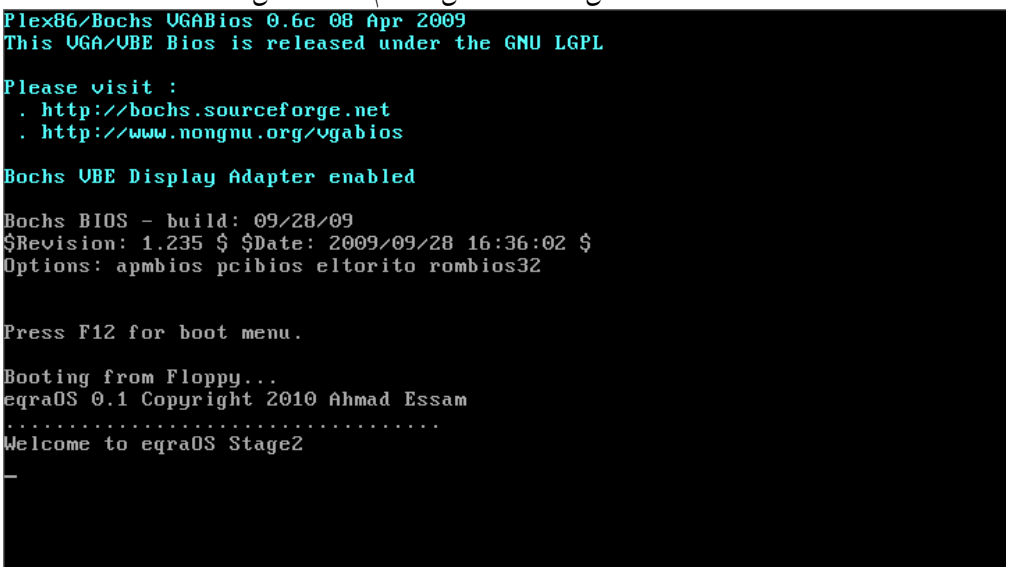
```
٦١    ;--------------------
٦٢      mov si,hello_msg
٦٣      call puts16
٦٤
٦٥    ;-------------------------
٦٦    ; Load Root Directory
٦٧    ;-------------------------
٦٨      call load_root
٦٩
٧٠    ;-------------------------
٧١    ; Load Kernel
٧٢    ;-------------------------
٧٣      xor ebx,ebx
٧٤      mov bp,KERNEL_RMODE_BASE     ; bx:bp buffer to load kernel
٧٥
٧٦      mov si,kernel_name
٧٧      call load_file
٧٨
٧٩      mov dword[kernel_size],ecx
٨٠      cmp ax,0
٨١      je enter_stage3
٨٢
٨٣      mov si,fail_message
٨٤      call puts16
٨٥
٨٦      mov ah,0
٨٧      int 0x16      ; wait any key.
٨٨      int 0x19      ; warm boot.
٨٩      cli          ; cannot go here!
٩٠      hlt
٩١
٩٢
٩٣    ;--------------------
٩٤    ; Go to PMode.
٩٥    ;--------------------
٩٦
٩٧    enter_stage3:
٩٨
٩٩      ; just set bit 0 from cr0 (Control Register 0).
١٠٠
١٠١     cli          ; important.
```

```
١٠٢     mov eax,cr0
١٠٣     or eax,0x1
١٠٤     mov cr0,eax      ; entering pmode.
١٠٥
١٠٦
١٠٧   ;———————————————
١٠٨   ; Fix CS value
١٠٩   ;———————————————
١١٠     ; select the code descriptor
١١١     jmp CODE_DESCRIPTOR:stage3
١١٢
١١٣
١١٤ ;*************************
١١٥ ; entry point of stage3
١١٦ ;*************************
١١٧
١١٨ bits 32      ; code now 32—bit
١١٩
١٢٠ stage3:
١٢١
١٢٢   ;———————————————;
١٢٣   ; Set Registers.
١٢٤   ;———————————————;
١٢٥
١٢٦     mov ax,DATA_DESCRIPTOR      ; address of data descriptor.
١٢٧     mov ds,ax
١٢٨     mov ss,ax
١٢٩     mov es,ax
١٣٠     mov esp,0x90000   ; stack begin from 0x90000.
١٣١
١٣٢   ;———————————————————————————
١٣٣   ; Clear Screen and print message.
١٣٤   ;———————————————————————————
١٣٥
١٣٦     call clear_screen
١٣٧
١٣٨     mov ebx,stage2_message
١٣٩     call puts32
١٤٠
١٤١     mov ebx,logo_message
١٤٢     call puts32
```

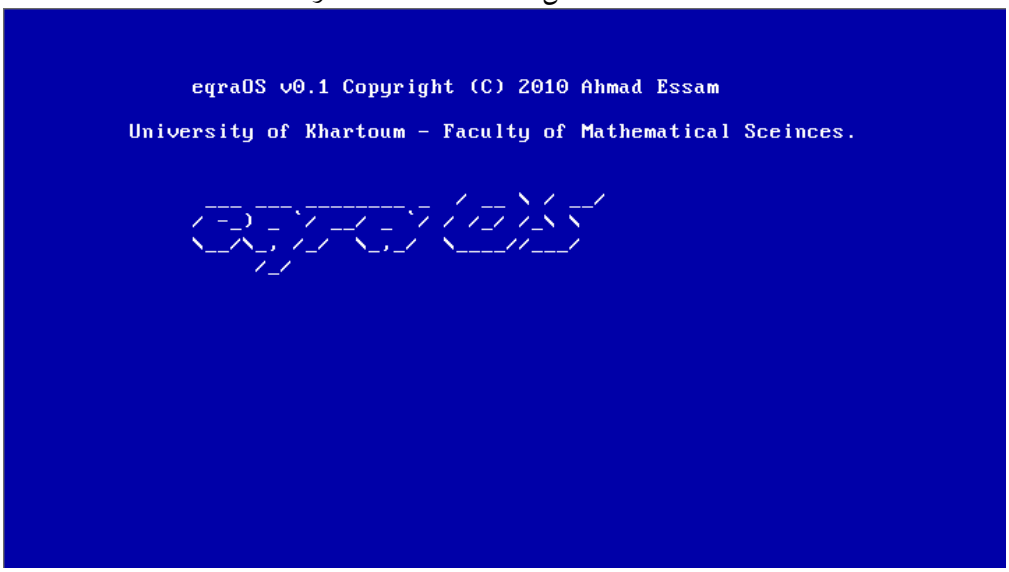
```
١٤٣
١٤٤
١٤٥
١٤٦   ;—————————————————————
١٤٧   ; Copy Kernel at 1 MB.
١٤٨   ;—————————————————————
١٤٩     mov eax,dword[kernel_size]
١٥٠     movzx ebx,word[bytes_per_sector]
١٥١     mul ebx
١٥٢     mov ebx,4
١٥٣     div ebx
١٥٤
١٥٥     cld
١٥٦
١٥٧     mov esi,KERNEL_RMODE_BASE
١٥٨     mov edi,KERNEL_PMODE_BASE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

النتيجة:

شكل ٤.١.: محمل النظام أثناء العمل



شكل ٤.٢.: بدء تنفيذ النواة



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 برمجة نواة النظام من أصعب المهمات البرمجية على الاطلاق ، 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 كلٌ على حدة.

١.٥. نواة نظام التشغيل

تعرَّف نواة نظام التشغيل بأنها الجزء الأساسي في النظام والذي تعتمد عليه بقية مكونات نظام التشغيل. ويكمن دور نواة النظام في التعامل المباشر مع عتاد الحاسب وإدارته بحيث تكوِّن 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 مرغوبا ، 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 الاهم 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 المستقبلة. 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) سواءا 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 Progeamming Interface) والتي تعرف بدوال (API).

٢.٥. وظائف نواة النظام

تختلف مكونات ووظائف نواة نظام التشغيل تبعاً لطريقة التصميم المتبعة ،فهناك العديد من الطرق لتصميم الانوية بعضاً منها يجعل ما هو متعارف عليه بأنه يتبع لنواة النظام ببرنامج للمستخدم (User Program)[١] والبعض الاخر عكس ذلك . لذلك سنذكر حالياً المكونات الشائعة في نواة النظام وفي القسم التالي عند الحديث عن هيكلة وطرق تصميم الأنوبة 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 تحمليه 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 مستقلا عن العناوين (Position Independent) لكي يتم تحمليه 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 فان الناتج من عملية ترجمة وربط أي برنامج هو انها تبدأ من العنوان 0x0، وهكذا سنتمكن دوما من تحميل أي برنامج في بداية الذاكرة. بهذا الشكل لن نتمكن من تنفيذ أكثر من برنامج واحد ، حيث سيكون هناك برنامجا واحدا فقط يبدأ من العنوان 0x0 ، والحل لهذه المشاكل هو باستخدام مساحة العنونة التخيلية (Virtual Address Space) حيث يتم تخصيص مساحة تخيلية من الذاكرة لكل برنامج بحيث تبدأ العنونة تخيليا من 0x0 وبهذا تم حل مشكلة تحميل أكثر من برنامج وحل مشكلة relocation. ومساحة العنوان التخيلية (VAS) هي مساحة من العناوين لكل برنامج بحيث تيدأ من ال 0x0 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 فانه يمكن ان يشير عنوان تخيلي الى ذاكرة اخرى بخلاف الذاكرة الرئيسية (مثلا القرص الصلب). وهذا يحل مشكلة انتهاء المساحات الخالية في الذاكرة. ويجدر بنا ذكر أن التحويل بين العناوين التخيليه الى الحقيقية يتم عن طريق العتاد بواسطة وحدة ادارة الذاكرة بداخل المعالج (Memory

[١] المقصود أنها برامج تعمل في الحلقة ٣.

Management Unit).وكذلك مهمة حماية الذاكرة والتحكم في الذاكرة Cache وغيرها من الخصائص والتي سيتم الإطلاع عليها في الفصل السابع – بمشيئة الله–.

٣.٥. هيكلة وتصميم النواة

توجد العديد من الطرق لتصميم الأنوية وسنستعرض بعض منها في هذا البحث ، لكن قبل ذلك يجب الحديث عن طريقة مفيدة في هيكلة 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 تمكِّن النواة من التعامل مع العتاد.

فصل النواة من العتاد تُتيح 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[٢].

١.٣.٥. النواة الضخمة Monolithic Kernel

تعتبر الأنوية المصممة بطريقة Monolitic[٣] أسرع وأكفأ أنوية في العمل وذلك نظرا لان كل برامج النظام (System Process) تكون ضمن النواة وتعمل في الحلقة صفر .

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 اعادة تشغيل النظام مجددا.

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

[٢] أغلب أنوية أنظمة التشغيل الحالية تستخدم طبقة HAL، هل تسائلت يوما كيف يعمل نظام جنو/لينوكس على أجهزة سطح المكتب والأجهزة المضمنة!

[٣] كلمة Mono تعني واحد ، أما كلمة Lithic فتعني حجري ، والمقصود بأن النواة تكون على شكل كتلة حجرية ليست مرنة وتطويرها وصيانتها معقد.

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 . 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. وكأمثلة على أنظمة تعمل بهذا التصميم هو أنظمة ويندوز التي تعتمد على معمارية NT ، ونظام BeOS و Plane 9.

٤.٥. برمجة نواة النظام

يمكن برمجة نواة نظام التشغيل بأي لغة برمجة ، لكن يجب التأكد من أن اللغة تدعم استخدام لغة التجميع (Inline Assemlby) حيث أن النواة كثيرا ما يجب عليها التعامل المباشر مع أوامر لغة التجميع (مثلا عند تحميل جدول الواصفات العام وجدول المقاطعات وكذلك عند غلق المقطاعات وتفعيلها وغيرها).
الشيء الاخر الذي يجب وضعه في الحسبان هو أنه لا يمكن استخدام لغة برمجة تعتمد على مكتبات في وقت التشغيل (ملفات dll مثلا) دون إعادة برمجة هذه المكتبات(مثال ذلك لا يمكن استخدام لغات دوت نت دون إعادة برمجة إطار العمل). وكذلك لا يمكن الإعتماد 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 الاستثنائات 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 الذواكر 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) [٤]، 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++ ، وسنناقش كيفية تحميل وتنفيذ هذه النواة حيث أن المخرج من مترجم الفيجوال سي++ هو ملف تنفيذي (Portable Executable) ولديه صيغة محددة يجب التعامل معها حتى نتمكن 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 للإستثنائات (Exceptions) في لغة السي++ .

١.٤.٥. 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 نتمكن 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. و يمكن أن نصف هذه الاضافات بلغة السي++ كالتالي.

Example ٥.١: Portable Executable Header

```
١
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
```

[٤] كبرنامج محمل النظام الذي قمنا بتطويره في بداية هذا البحث.

```
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 الاضافة الثانية ذات حجم متغير وليست ثابته مثل بقية الاضافات.
وبالنظر الى أول إضافة IMAGE DOS HEADER وبالتحديد الى المتغير e lfanew حيث يحوي عنوان الإضافة الثالثة IMAGE FILE HEADER والتي هي اضافة ثابته 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]  ; epb = AddressOfEntryPoint
١١
١٢  add ebx,12      ; ebx point to ImageBase
١٣
١٤  add ebp,dword[ebx]  ; epb = kernel_entry
١٥
١٦  cli
١٧
١٨  call ebp
```

٢.٤.٥. تطوير بيئة التشغيل للغة سي++

حتى نتمكن 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) و حذف الكائنات ، وكذلك لن يمكن استخدام بعض المعاملات (new,delete) و RTTI والاستثنائات (Exceptions).

المعاملات العامة Global Operator

سيتم تعريف معامل حجز الذاكرة (new) وتحريرها (delete) في لغة السي++ ، ولكن لاننا حاليا لم نبرمج مديراً للذاكرة فان التعريف سيكون خاليا.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)[٥]، حيث سيقوم المترجم باستدعاء الدالة purecall()[٦] أينما وجد عملية استدعاء لدالة Pure virtual ، لذلك أن أردنا دعم الدوال Pure virtual يجب تعريف الدالة purecall ، وحاليا سيكون التعريف كالاتي.

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 ftol2_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;
٥ #ifdef i386
٦   _asm {
٧     fistp [a]
٨     mov ebx, a
٩   }
١٠ #endif
١١ }
١٢
١٣ extern "C" int _fltused = 1;
```

[٥] عند تعريف دالة بأنها 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 .

[٦] هذه الدالة يقوم مترجم الفيجوال سي++ باستدعائها.وقد تختلف من مترجم لآخر.

تهيئة الكائنات العامة والساكنة

عندما يجد المترجم كائنا فانه يضيف مهيئاً ديناميكيا له (Dynamic initializer) في قسم خاص من البرنامج[٧] وهو القسم crt.. وقبل أن يعمل البرنامج فان 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 فانه يجب انشاء 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 نتمكن من استدعاء وتنفيذ الدوال عن طريق مصفوفة المؤشرات فانه يجب انشاء مؤشر الى 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. ويلي القسم crt:xcu. مباشرة .

وبخصوص القسم crt. الذي سننشئه فاننا 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
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
```

[٧] في أي برنامج تنفيذي يوجد العديد من الأقسام، مثلا قسم البيانات data. وقسم الشفرة code. والمكدس stack. وغيرها.

```
// Now, move the CRT data into .data section so we can read/write to it
#pragma comment(linker, "/merge:.CRT=.data")


// initialize all global initializers (ctors, statics, globals, etc..)
void __cdecl _initterm ( _PVFV * pfbegin, _PVFV * pfend ) {

  //! Go through each initializer
    while ( pfbegin < pfend )
    {
    //! Execute the global initializer
      if ( *pfbegin != 0 )
            (**pfbegin) ();

      //! Go to next initializer inside the initializer table
        ++pfbegin;
    }
}

// execute all constructors and other dynamic initializers
void _cdecl init_ctor() {

   _atexit_init();
   _initterm(__xc_a, __xc_z);
}
```

حذف الكائنات

لكي يتم حذف الكائنات (Objects) يجب انشاء مصفوفة من مؤشرات دوال الهدم (deinitializer array) ، وذلك بسبب أن المترجم عندما يجد دالة هدم فانه يضيف مؤشراً الى دالة الهدم بداخل هذه المصفوفة وذلك حتى يتم استدعائها لاحقا (عند استدعاء الدالة exit())، ويجب تعريف الدالة atexit حيث أن مترجم الفيجوال سي++ يقوم باستدعائها عندما يجد أي كائن ، وظيفة هذه الدالة هي اضافة مؤشر لدالة هدم الكائن الى مصفوفة المؤشرات ،وبخصوص مصفوفة المؤشرات فانه 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 this.
        Because we have no memory manager, just choose
١٧   // a base address that you will never use for now
١٨   pf_atexitlist = (_PVFV *)0x5000;
١٩ }
٢٠
٢١ //! adds a new function entry that is to be called at shutdown
٢٢ int __cdecl atexit(_PVFV fn) {
٢٣
٢٤   //! Insure we have enough free space
٢٥   if (cur_atexitlist_entries>=max_atexitlist_entries)
٢٦     return 1;
٢٧   else {
٢٨
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 مكدس (Stack) وبعد ذلك يجب تهئية 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
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 خاليا.

٥.٥. نظرة على شفرة نظام إقرأ

أهم الخصائص التي يجب مراعتها أثناء برمجة نواة نظام التشغيل هي خاصية المحمولية على صعيد الأجهزة والمنصات[٨] وخاصية قابلية توسعة النواة (Expandibility) و لذلك تم الإتفاق على أن تصميم نواة نظام 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 فانه 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 نتمكن من الاستفادة من خدمات هذه الوحدة دون الحاجة لمعرفة تفاصيلها الداخلية. وفي بداية تصميم المشروع فان عملية تصميم الواجهة تعتبر أهم بكثير من عملية برمجة محتويات الوحدة أو ما يسمى بالتنفيذ (Impelmentation) نظراً لان التنفيذ قد لا يؤثر على هيكلة المشروع ومعماريته مثلما تؤثر الواجهة .

- eqraOS:
  - boot: first-stage and second-stage bootloader.
  - core:
    - kernel:Kernel program PE executable file type.

[٨] على عكس محمل النظام Bootloader والذي يعتمد على معمارية العتاد والمعالج.

* 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 فانه يجب انشاء هذه المكتبة حتى نتمكن من استخدام لغة سي وسي++ ، وبسبب أن عملية إعادة برمجة هذه المكتبات يتطلب وقتاً وجهدا فاننا سنركز على بعض الملفات المستخدمة بكثرة ونترك البقية للتطوير لاحقا.

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 #pargma once
٧ #endif
٨
٩ #ifdef NULL
١٠ #undev 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 بمترجم سي++ فان القيمة __cplusplus تكون معرَّفة لديه ، أما في حالة ترجمة النواة بمترجم سي فان المترجم لا يُعرِّف 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
٥ #ifdef __cplusplus
٦ extern "C"
٧ {
٨ #endif
٩
١٠ /* Stdandard definition of size_t */
١١ typedef unsigned size_t;
١٢
١٣ #ifdef __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 اعادة 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[٩].

Example ٥.١٢: cstdint:C++ typedef data type

```
١
٢ #ifndef CSTDINT_H
```

[٩]ملفات الرأس للغة سي++ تتبع نفس هذا الأسلوب لذلك لن يتم ذكرها مجددا وسنكتفي بذكر ملفات الرأس للغة سي.

```
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[];
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
```

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 حاليا 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 Rountine) فان التنفيذ يتنقل اليها تلقائيا ، و يتم فيها معالجة هذه المقاطعة (مثلا 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 لِيُكْمِل 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 (مثلا عن القسمة على العدد صفر أو عند حدوث Page Fault) وتسمى هذه المقاطعات بأخطاء المعالج أو استثنائات 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`) لِنقل التنفيذ الى دالة أخرى تعالج هذه المقاطعة (Interrupt handler)، وغالبا ما تستخدم هذه المقاطعات في برامج المستخدم (Ring3 user mode) للاستفادة من خدمات النظام (مثلا للقراءة والكتابة في أجهزة الإدخال والإخراج حيث لا توجد طريقة اخرى 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 بـــIRQ) 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` ويحوي كل سجل فيه على عنوان دالة معالجة المقاطعة (IR) والتي يجب تنفيذها لتخديم المقاطعة المطلوبة. حجم العنوان هو أربع بايت وتكون كالتالي:

جدول ١.٦.: Interrupt Vector Table

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

- 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 نانجة من ضرب عدد المقاطعات في حجم كل سجل)، بعض منها محجوز والبعض الاخر يستخدمه المعالج والبقية متروكة لمبرمج نظام التشغيل لدعم المزيد من المقطاعات. وبسبب أن الجدول يتكون فقط من عناوين لدوال معالجة المقاطعات فان هذا يمكننا من وضع الدالة في أي مكان على الذاكرة ومن ثم وضع عنوانها داخل هذا السجل (يتم هذا عن طريق مقاطعات البايوس)، والجدول ١.٦ يوضح IVT والمقاطعات الموجودة فيه.

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 (اذاً 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 ، بالاضافة 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 determins size
    - 01110 - 32 bit descriptor
    - 00110 - 16 bit descriptor
  - Task Gate: Must be 00101
  - Trap Gate: Of the format 0D111, where D determins size
    - 01111 - 32 bit descriptor
    - 00111 - 16 bit descriptor
- Bits 42-44: Descriptor Privedlge 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 الاخر من العنوان المتغير `baseHi` وفي هذا المثال العنوان هو `0x0`.بمعنى أن دالة تخديم المقاطعة ستكون في العنوان `0x0`. وبما أن دالة معالجة (تخديم) المقاطعة تحوي شفرة برمجية للتنفيذ وليست بيانات (Data) فان قيمة المتغير `selector` يجب أن تكون `0x8` للإشارة الى ناخب الشفرة (Code Selector) في جدول الواصفات العام (GDT). أما المتغير `flags` فان 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 مؤشراً 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 الى المسجل `IDTR` (وهو مسجل داخل المعالج) عن طريق تنفيذ الامر `lidt`[١] بالشكل التالي `lidt [idt_ptr]`.

[١] بعد تنفيذ هذا الأمر فان جدول المقاطعات سيتم استبداله بالجدول الجديد والذي نجد عنوانه بداخل المسجل `idtr` ، وهذا الأمر لا يُنفَّذ إلاَّ اذا كانت قيمة العلم (CPL flag) هي صفر.

وعند حدوث أي مقاطعة فان المعالج ينهي الأمر الذي يعمل عليه و يأخذ رقم المقاطعة ويذهب به الى جدول IDT (عنوان هذا الجدول يتواجد بداخل المسجل IDTR) ، وبعد ذلك يقوم بحساب مكان الواصفة بالمعادلة `int_num * 8` وذلك بسبب أن حجم كل واصفة في جدول IDT هو 8 بايت. وقبل أن ينقل التنفيذ الى دالة معالجة المقاطعة فانه 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 المكدس (Stack) الحالي ، وفي حالة حدوث خطأ ما فانه يتم دفع شفرة الخطأ (Error Code) الى المكدس أيضا. 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 refrenced by the error code.
- Bits 16-31: Reserved.

وعندما تنتهي دالة معالجة المقاطعة من عملها فانه يجب أن تنفذ الأمر `iret` أو `iretd` حتى يتم ارجاع القيم التي تم دفعها الى المكدس (قيم الأعلام FLAGS). وبالتالي يُكْمِل المعالج عمله.

٣.١.٦. أخطاء المعالج

خلال تنفيذ المعالج للأوامر فانه 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 المكدس هو عنوان الأمر الذي تسبب في هذا الخطأ.
- الخطأ Trap: عنوان العودة هو عنوان التعليمة التي تلي الأمر الذي تسبب في الخطأ.

جدول ٢.٦.: x86 Processor Exceptions Table

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

- الخطأ Abort: لا يوجد عنوان للعودة ، ولن يكمل البرنامج عمله بعد انتهاء دالة معالجة الخطأ.

والجدول ٢.٦ يوضح أخطاء المعالج والمقاطعات التي يقوم بتوليدها.
ويجدر بنا الوقوف على ملاحظة كنّا 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 العتادية 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 انشائه وأي محاولة لاستخدامه سيؤدي الى هذا الخطأ.
أحد المتحكمات التي ترسل مقاطعات الى المعالج بشكل ثابت هو متحكم Prpgrammable Interval Timer وتختصر بمتحكم PIT وهي تمثل ساعة النظام System Timer بحيث ترسل مقاطعة بشكل دائم الى المعالج والذي بدوره ينقل التنفيذ الى دالة تخديم هذه المقاطعة . وبسبب أن جدول المقطاعات 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 المقطاعات 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 المقطاعات وحاليا سيقتصر عمل الدوال على طباعة رسالة ، وقبل ذلك سنقوم بانشاء جدول الواصفات العام (GDT) مجددا (أي سيتم الغاء الجدول الذي قمنا بانشائه 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 أيضا نجده تحت طبقة HAL ، وجدول 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[٢] 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 تماما 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
١٢ #define interrupt __declspec(naked)
١٣ #else
```

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

```
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 يُعرِّف هذه الدوال. الدالة الاولى هي hal_init() والتي تقوم بتهيئة العتاد وجداول المعالج بينما الدالة الثانية hal_close() تقوم بعملية الحذف والتحرير وأخيرا 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) [٣] 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[٤]:

- 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[٥].

Example ٦.٤: hal/gdt.cpp:Install GDT

[٣] راجع ١.١.٤.
[٤] لغرض التنظيم والتقسيم لا أكثر ولا أقل.
[٥] راجع شفرة النظام لقراءة ملف الرأس hal/gdt.h.

```
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 _gdtr
٤٨   _gdtr.limit = sizeof(struct gdt_desc) * MAX_GDT_DESC - 1;
٤٩   _gdtr.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|
         I386_GDT_MEMORY,   // 10011010
٥٧     I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K                //
         11001111
٥٨
٥٩   );
٦٠
٦١   // set data desc.
٦٢   i386_gdt_set_desc(2,0,0xffffffff,
٦٣     I386_GDT_DATA_DESC|I386_GDT_READWRITE|I386_GDT_MEMORY,  //
         10010010
٦٤     I386_GDT_LIMIT_HI|I386_GDT_32BIT|I386_GDT_4K      // 11001111
٦٥   );
٦٦
٦٧   // install gdtr
٦٨   gdt_install();
٦٩
٧٠   return 0;
٧١ }
```

٢.٦. متحكم المقاطعات القابل للبرمجة Programmable Interrupt Controller

السبب الرئيسي في تعطيل المقاطعات العتادية عند الإنتقال 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 تخديم 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 اليها ، 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 قديما 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 العتاد تسمى Polling[٦] وهي تضيع وقت المعالج في انتظار قيمة لا يُعرف هل ستظهر أم لا وقد تم إلغائها 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 الإستفادة من وقت المعالج وأصبح التخاطب هو غير متزامن (Asynchronous) بدلاً من متزامن (Synchronous). وعندما يبدأ الحاسب في الإقلاع فان نظام البايوس يقوم بترقيم عتاد

[٦] وتسمى أيضا ب Busy Waiting.

جدول ٣.٦.: 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

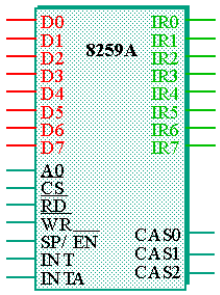
الحاسب وإعطاء رقم مقاطعة لكل متحكم وبسبب تكرار هذه الأرقام فانه يجب تغييرها لأرقام أخرى وهذا يتم بسهولة في النمط الحقيقي وذلك باستخدام مقاطعات البايوس أما في النمط المحمي فيجب أن نقوم بالتخاطب المباشر مع المتحكم الذي لديه أرقام المقاطعات ومن ثم تغييرها . والجدول ٣.٦ يوضح أرقام المقاطعات لمتحكمات الحاسب.

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 تخديمها ، ويراعي متحكم PIC أولية متحكمات العتاد ، فمثلا لو تم إرسال إشارتين في نفس الوقت الى متحكم PIC فان المتحكم سوف يراعي الأولية ويقوم بارسال رقم مقاطعة العتاد ذو الأولية أولا وبعد أن تنتهي دالة تخديم المقاطعة يقوم المتحكم بارسال الرقم الآخر . ونظراً لتعقيدات بناء المتحكم فانه يتعامل فقط مع ٨ أجهزة مختلفة (أي ٨ مقاطعات IRQ) وهذا ما أدى مصنعى الحاسب الى 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 ارسال البيانات من المتحكم الى المعالج (أو الى متحكم رئيسي) ، ونظراً لان كل مشبك لديه وظيفة محددة فانه يجب دراسة هذه المشابك ولكن لن نفصِّل كثيراً حيث أن الموضوع متشعب ويخص دراسي المنطق الرقمي (Digital Logic). ويوضح الشكل ٢.٦ هذه المشابك.



شكل ٢.٦.: مشابك متحكم PIC

حيث أن المشابك D0-D7 هي لإرسال البيانات الى 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 فانه 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 حجرها في مسجل داخل PIC الى أن ينتهي المعالج من تنفيذ المقاطعة والعودة بإشارة (تسمى إشارة نهاية المقاطعة End Of Interrupt) تدل على أن المقاطعة قد انتهت. أخيرا ما يهمنا في هذه المشابك هي مشابك IR0...IR7 وهي مشابك ترتبط مع متحكمات العتاد 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 عتادية مختلفة. ويجب ملاحظة أن متحكم العتاد الذي يرتبط بأول مشبك IR0 لديه الأولية الأولى في التنفيذ وهكذا على التوالي.

مسجلات متحكم PIC

يحوي متحكم PIC على عدة مسجلات داخلية وهي:

- مسجل الأوامر (Command Reigster): ويستخدم لإرسال الأوامر الى المتحكم ، وهناك عدد من الأوامر مثل أمر القراءة من مسجل ما أو أمر ارسال اشارة 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 (Acnowledges) من المعالج ، والجدول ٤.٦ يوضح بتات هذا المسجل.

جدول ٦.٤.: 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

جدول ٦.٥.: 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

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 تخديم المقاطعة من عملها.
- مسجل ((Interrupt Mask Register (IMR): يحدد هذا المسجل ما هي المقاطعات التي يجب تجاهلها وعدم ارسال إشعار لها وذلك حتى يتم التركيز على المقاطعات الأهم.

والجدول ٦.٥ يوضح عناوين منافذ المسجلات في حواسيب x86.

برمجة متحكم PIC

لبرمجة متحكم PIC وإعادة ترقيم المقاطعات فإن ذلك يتطلب إرسال بعض الأوامر الى المتحكم بحيث تأخذ هذه الأوامر نمط معين تُحَدَّد بها عمل المتحكم. وتوجد أربع أوامر تهيئة يجب إرسالها لتهيئة المتحكم تعرف ب Initialization Control Words وتختصر بأوامر تهيئة ICW ، وكذلك توجد ثلاث أوامر تحكم في عمل متحكم PIC تعرف ب Operation Control Words وتختصر ب OCW . وفي حالة توفر أكثر من متحكم PIC على النظام فيجب أن تُرسل أوامر التهيئة الى المتحكم الآخر كذلك. **الأمر الأول ICW1** وهو أمر التهيئة

جدول ٦.٦.: الأمر الأول ICW1

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

الرئيسي والذي يجب إرساله أولا الى المتحكم الرئيسي والثانوي ويأخذ ٧ بتات ويوضح الجدول ٦.٦ هذه البتات ووظيفة كل بت.
حيث أن البت الأول يحدد ما اذا 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 الى 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 عنواين 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 أرقاما غير مستخدمة من قبل المعالج حتى لا نقع 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 وانتهاءاً بالمقاطعة ٣١.

جدول ٦.٧.: الأمر الثالث للمتحكم الرئيسي ICW3 for Primary PIC

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

جدول ٦.٨.: 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	محجوزة

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 ٦.٧ و ٦.٨ .

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 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) وبقية البتات محجوزة (انظر جدول ٦.٨) ، والمثال ٦.٧ يوضح كيفية إرسال الأمر الثالث الى المتحكمين.

Example ٦.٧: Initialization Control Words 3

جدول ٦.٩.: 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

```
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 الى المتحكمين ويأخذ التركيبة التي يوضحها جدول ٦.٩.
وفي الغالب لا يوجد حوجة لتفعيل كل هذه الخصائص ، فقط أول بت يجب تفعيله حيث يستخدم مع حواسيب x86 . والمثال ٦.٨ يوضح كيفة إرسال الأمر الرابع الى المتحكم PIC 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
```

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 الى المعالج هي الأرقام التي قمنا بتعيينها في الأمر الثاني (وتبدأ من ٣٢ الى ٤٧) وهي تختلف بالطبع عن الأرقام التي يستخدمها المعالج. وبخصوص أوامر التحكم الثلاث OCW فلن نحتاج اليها جميعاً وسيتم الحديث عن الأمر الثاني OCW2 نظراً لأنه يجب أن يُرسل دائماً بعد أن تنتهي دالة تخديم المقاطعة من عملها وذلك حتى يتم السماح لبقية المقاطعات أن تأخذ دوراً لمعالجتها. والجدول ١٠.٦ يوضح البتات التي يجب إرسالها الى مسجل التحكم . ويهمنا البتات ٥-٧ حيث أن قيمهم تحدد بعض الخصائص التي يوضحها الجدول ١١.٦. والمثال ٦.٩ يوضح كيفية إرسال إشارة نهاية

جدول ٦.١٠.: 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

جدول ٦.١١.: 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

عمل دالة تخديم 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
```

كيف تعمل مقاطعات العتاد

عندما يحتاج متحكم أي عتاد لفت انتباه المعالج الى شيء ما فأول خطوة يقوم بها هي إرسال إشارة الى متحكم PIC (وعلى سبيل المثال سنفرض أن هذا المتحكم هو متحكم المؤقتة PIT والتي ترتبط بالمشبك IR0) هذه الإشارة ترسل عبر مشبك IR0 ، 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 IR0. بعد ذلك يقوم متحكم PIC بفحص مسجل Interrupt

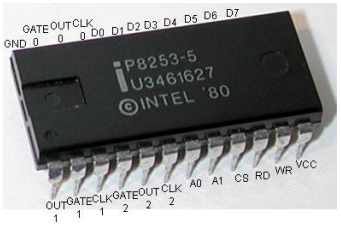
Mask Register ليتأكد من أنه لا توجد هناك مقاطعة ذات أولية أعلى حيث في هذه الحالة على المقاطعة الجديدة أن تنتظر حتى يتم تخديم كل المقاطعات ذات الأولوية. وبعد ذلك يُرسل PIC إشارة الى المعالج من خلال مشبك INTA لأخبار المعالج بأن هناك مقاطعة يجب تنفيذها. وهنا يأتي دور المعالج حيث يقوم بالإنتهاء من تنفيذ الأمر الحالي الذي يعمل عليه ومن ثم يقوم بفحص قيمة العلم IF حيث في حالة كانت غير مفعلة فان المعالج سوف يتجاهل طلب تنفيذ المقاطعة، أما إذا وجد المعالج قيمة العلم مفعلة فانه يقوم بارسال إشعار (Acnowledges) عبر مشبك INTR الى متحكم PIC الذي بدوره يستقبلها من مشبك INTA ويضع رقم المقاطعة ورقم IRQ في المشابك D0-D7 ، وأخيرا يفعل قيمة البت ٠ في مسجل In Service Register دلالة على أن مقاطعة المؤقتة جاري تنفيذها. وعندما يحصل المعالج على رقم المقاطعة فانه 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 الى جدول المقطاعات IVT حيث يجد عنوان دالة تخديم المقاطعة ومن ثم ينقل التنفيذ اليها ، أما اذا كان المعالج يعمل في النمط المحمي فانه يأخذ رقم المقاطعة ويذهب بها الى جدول واصفات المقاطعات حيث يجد دالة تخديم المقاطعة. وعندما تنتهي دالة تخديم المقاطعة من عملها فانها 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 ارسال طلب مقاطعة (IRQ0) الى 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 لاصدار طلب المقاطعة سيكون هو الوقت المحدد لأي عملية (Process) موجودة في طابور العمليات (Process Queue) وبعد ذلك تُرسل العملية الى 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 الى متحكم الذاكرة (Memory Controller) وأصبحت هذه المؤقتة لا تستخدم في العادة. أما العداد الأخير فيستخدم في عملية **إرسال الصوت الى سماعات الحاسب**[٧] (PC Speaker).

[٧] لا يُقصد بهذه كرت الصوت وإنما يوجد في كل حاسب سماعات داخلية تستخدم في إصدار الصوت والنغمات وأحد استخداماتها لإصدار رسائل الخطأ بعد عملية فحص الحاسب (POST) في مرحلة الإقلاع.

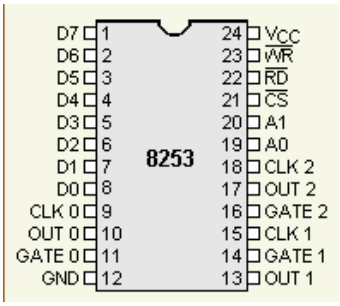
شكل ٦.٣.: المؤقتة القابلة للبرمجة 8253



٦.٣.١. برمجة المؤقتة PIT

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

مشابك المؤقتة PIT's Pins



شكل ٦.٤.: مشابك المؤقتة PIT

تُرسَل الأوامر والبيانات الى المؤقتة وذلك عبر مسار البيانات (Data Bus) حيث يرتبط هذا المسار مع مشابك البيانات في المؤقتة وهي ٨ مشابك D0...D7 وتُمثل 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). وتُحدد قيمة المشبكين A0,A1 –واللذان يرتبطان مع مسار العنواين– 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 العتاد 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 التخديم.

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 . وبسبب أن مشابك البيانات التي تربط المؤقتة ومسار البيانات هي من الطول ٨ بت فانه لن نتمكن من إرسال البيانات بهذ 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 ١٢.٦ 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

جدول ١٢.٦.: 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
		0	1	0	0	قراءة المسجل 0
Counter 1	0x41	1	0	0	1	كتابة الى المسجل 1
		0	1	0	1	قراءة المسجل 1
Counter 2	0x42	1	0	1	0	كتابة الى المسجل 2
		0	1	1	0	قراءة المسجل 2
Control Word	0x43	1	0	1	1	كتابة Control Word
		0	1	1	1	لا توجد عملية

(وهو مسجل بطول ٨ بت) حيث يجب إرسال قيم معينة حتى نتمكن من 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 بارسال طلب المقاطعة (IRQ0) .

١.٤.٦. 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[٨].

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_AUXILIRY   4
١٤ #define I386_PIC_IRQ_FPU      5
١٥ #define I386_PIC_IRQ_HDC      6
١٦
١٧ // Operation Commamd 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
```

[٨]إرجع الى القسم ٢.٢.٦ لمعرفة وظيفة هذه القيم الثابتة.

```
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
٢٦ // Operation Commamd 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 Commamd 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 Commamd 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);
```

وتحوي 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 استدعائها. 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,uint8_t
       mode) {
٤٧   if (freq == 0)
٤٨     return;
٤٩
٥٠   uint16_t divisor = uint16_t(1193181/uint16_t(freq));
٥١
٥٢   /*  send operation command  */
٥٣   uint8_t ocw = 0;
٥٤
٥٥   ocw = (ocw & ~I386_PIT_OCW_MASK_MODE) | mode;
٥٦   ocw = (ocw & ~I386_PIT_OCW_MASK_RL) | I386_PIT_OCW_RL_DATA;
٥٧   ocw = (ocw & ~I386_PIT_OCW_MASK_COUNTER) | counter;
٥٨
٥٩   i386_pit_send_command(ocw);
```

```
٦٠
٦١   /*  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 }
```

٦.٤.٣. 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 toi 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
١٠٣ int _cdecl get_tick_count() {
١٠٤   return i386_pit_get_tick_count();
١٠٥ }
```

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 /*  Execption 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

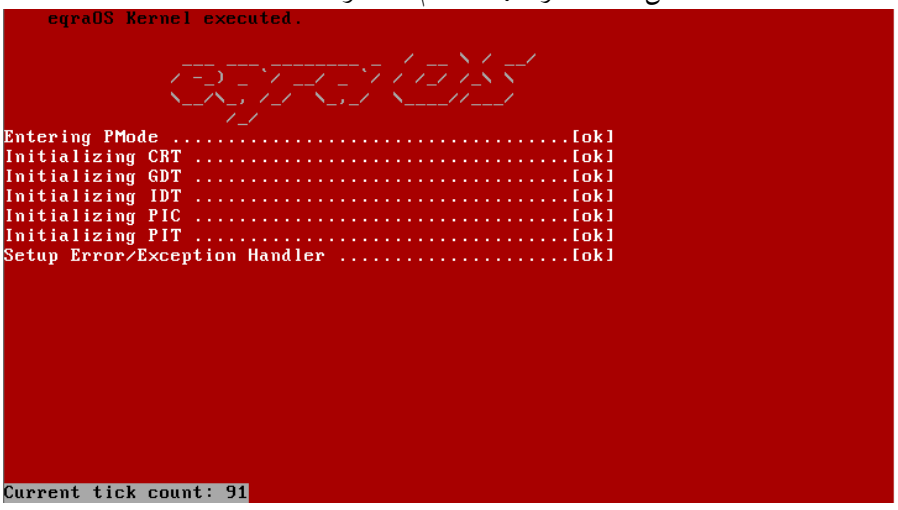
```
١ /* Divide by zero */
٢ void _cdecl divide_by_zero_fault(uint32_t cs,uint32_t eip,uint32_t
       eflags) {
٣   kernel_panic("Divide by 0");
٤   for (;;);
٥ }
```

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, eqraOS 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 ٦.٥.: واجهة النظام بعد توسعة طبقة HAL



شكل ٦.٦.: دالة تخديم المقاطعات الإفتراضية

٧. إدارة الذاكرة

تعتبر ذاكرة الحاسب الرئيسية (RAM) من أهم الموارد التي يجب على نظام التشغيل إدارتها حيث يمثل المخزن الذي يقرأ المعالج منه الأوامر ويقوم بتنفيذها ، ولا يمكن للمعالج الوصول المباشر الى أحد الذواكر الثانوية مباشرة وإنما يتم تحميل البيانات والبرامج الى الذاكرة الرئيسية حتى تصبح متاحة للمعالج. وهنا يأتي دور مدير الذاكرة (Memory Manager) حيث يراقب هذا البرنامج أجزاء الذاكرة ويتعرف على ما هو مستخدم وما هو غير مستخدم كما يوفر دوالا لحجز مقاطع الذاكرة وتحريرها ، كذلك من الممكن أن يقوم بعملية إعادة تجزئة لمقاطع الذاكرة وذلك لتوفير مساحة واستغلال أفضل.وفي هذا الفصل سيتم دراسة مدير الذاكرة الفيزيائية والتخيلية وكيفية برمجتهم كذلك سيتم عرض بعض الطرق لحساب المساحة الكلية للذاكرة وعرض مقاطع الذاكرة المستخدمة من قبل النظام.

١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management

الذاكرة الفيزيائية (Physical Memoery) هي كتلة ذاكرية توجد بداخل شريحة الذاكرة الرئيسية RAM وتختلف طريقة حفظ البيانات فيها بحسب نوع الذاكرة (فمثلا تستخدم المكثفات والترانزستورات لحفظ البتات في ذاكرة DRAM) ، ويتم التحكم في هذه الكتلة بواسطة شريحة داخل اللوحة الأم تسمى بمتحكم الذاكرة أو بالجسر الشمالي (North Bridge) حيث يقوم هذا المتحكم بعملية إنعاش المكثفات بداخل الذاكرة حتى تحافظ على محتوياتها من الضياع كذلك يعنون هذا المتحكم الذاكرة الفيزيائية بداية من أول ٨ بتات حيث تأخذ العنوان الفيزيائي 0x0 ويليها العنوان 0x1 للثمانية بتات التالية ، وهكذا تتم عنونة الذاكرة من قبل المتحكم. وعند وصول طلب قراءة أو كتابة الى متحكم الذاكرة فان المتحكم يقوم بقراءة العنوان من مسار العناوين وبالتالي يتمكن من تفسير العنوان وتوجيهه الى المكان الصحيح حيث غالبا ما توجد أكثر من شريحة رام بداخل الحاسب. وتشكل مجموعة كل العناوين في كل شرائح الذاكرة مساحة عناوين فيزيائية (Physical Address Space) .لكن قد يكون هناك عددا من هذه العناوين غير مستخدمة فعليا وذلك في حالة وجود شريحة ذاكرة في المنفذ الأول والثالث وعدم وجود شريحة ذاكرة في المنفذ الثاني وفي هذه الحالة اذا كان حجم كل شريحة ذاكرة هي n فان العناوين من n-1 الى 2n-1 هي عناوين غير موجودة حقيقة وتسمى فتحات (Holes) ، وعملية الكتابة في هذه الفتحات لا تؤثر شيء على النظام بينما عند القراءة من هذه الفتحات فان البيانات الموجودة في مسار البيانات هي التي يتم قرائتها على الرغم من أنها غير صحيحة. وقبل الخوض في برمجة مدير الذاكرة يجب توفير عددا من المعلومات مثل حجم الذاكرة الكلي ومناطق الذاكرة المستخدمة وغير المستخدمة و الفتحات الموجودة ان كانت هناك فتحات ، وهذا حتى يتمكن مدير الذاكرة من إدارتها على النحو المطلوب.

١.١.٧. حساب حجم الذاكرة

توجد عدة طرق لحساب حجم الذاكرة بعضها تعتمد على النظام والأخرى لا تعتمد ، وفي بداية إقلاع الحاسب يقوم نظام البايوس بالتخاطب مع متحكم الذاكرة وأخذ حجم الذاكرة ويقوم بحفظها في أحد العناوين في منطقة بيانات البايوس في الذاكرة. وتعتبر هذه الطريقة هي الأكثر دقة في الحصول على حجم الذاكرة ولكن تبقى مشكلة أن مقاطعات البايوس لا يمكن استدعائها بداخل النواة والتي تعمل في النمط المحمي. لذلك سيتم استخدام المقاطعة الخاصة بجلب حجم الذاكرة قبل أن تبدأ النواة في عملها وتمرير هذا الحجم مع بعض المعلومات الاخرى و تسمى معلومات الإقلاع المتعدد الى النواة.

المقاطعة int 0x12

تقوم هذا المقاطعة بجلب حجم الذاكرة من منطقة بيانات البايوس (بالتحديد من العنوان 0x413) لكنها لا تستخدم حاليا نظرا لأنها تجلب حجم ٦٤ كيلوبايت كحد أقصى وبالتالي اذا كانت الذاكرة لديك أكبر من هذا الحجم فانها لا تجلب الحجم الصحيح ، لذلك لا تستخدم هذه المقاطعة الان. والمثال ٧.١ يوضح كيفية استخدامها

Example ٧.١: Using int 0x12 to get size of memory

```
١
٢ ;————————————————————————
٣ ; get_conventional_memory_size
٤ ; ret ax=KB size from address 0
٥ ;————————————————————————
٦ get_conventional_memory_size:
٧   int 0x12
٨   ret
```

المقاطعة 0x15 الدالة 0xe801

تجلب هذه المقاطعة الحجم الصحيح وتستخدم دائما لهذا الغرض ، وتعود بالقيم:

- العلم CF: صفر في حالة نجاح عمل المقاطعة.
- المسجل EAX: عدد الكيلوبايتات من العنوان 1 MB الى 16 MB.
- المسجل EBX: عدد الوحدات المكونة من ٦٤ كيلوبايت بدئا من العنوان 16 MB، ويجب ضربها لاحقا بالعدد ٦٤ حتى يتم تحويلها الى عدد الكيلوبايتات.

وفي بعض الأنظمة يستخدم البايوس المسجلين ECX و EDX بدلا من المسجلين EAX و EBX. والمثال ٧.٢ يوضح كيفية استخدام هذه المقاطعة.

Example ٧.٢: Using int 0x15 Function 0xe801 to get size of memory

```
١
٢ ; ——————————————————————
٣ ; get memory size:
٤ ; get a total memory size.except the first MB.
٥ ; return:
٦ ;   ax=KB between 1MB and 16MB
٧ ;   bx=number of 64K blocks above 16MB
٨ ;   bx=0 and ax= —1 on error
٩ ; ——————————————————————
١٠ get_memory_size:
١١
١٢     push ecx
١٣     push edx
١٤     xor ecx,ecx
١٥     xor edx,edx
١٦
١٧     mov ax,0x801    ; BIOS get memory size
١٨     int 0x15
١٩
٢٠     jc .error
٢١     cmp ah,0x86
٢٢     je .error
٢٣     cmp ah,0x80
٢٤     je .error
٢٥
٢٦     jcxz .use_eax
٢٧
٢٨     mov ax,cx
٢٩     mov bx,dx
٣٠
٣١   .use_eax:
٣٢     pop edx
٣٣     pop ecx
٣٤     ret
٣٥
٣٦   .error:
```

```
٣٧      mov ax,-1
٣٨      mov bx,0
٣٩      pop edx
٤٠      pop ecx
٤١      ret
```

ويجدر ملاحظة أن هذا المقاطعة لا تحسب أول ميجا بايت من الذاكرة ، لذلك عند استخدام هذه الدالة يجب زيادة الحجم الكلي بمقدار ١٠٢٤ كيلوبايت (الزيادة تكون في مسجل ax) ، كذلك يجب ضرب المسجل bx بالعدد ٦٤ نظرا لان القيمة التي تضعها الدالة هي عدد الوحدات المكونة من ٦٤ كيلوبايت ، وهذا يعني ان كان العدد هو ٢ مثلا فان النتيجة يجب أن تكون 64*2 وتساوي 128 كيلوبايت.

٧.١.٢. خريطة الذاكرة Memory Map

بعد أن تم حساب حجم الذاكرة يجب الانتباه الى أن عددا منها محجوز لبعض الأغراض (راجع فقرة خريطة الذاكرة في فصل إقلاع الحاسب) وهنا يأتي دور خريطة الذاكرة التي تعرف وتحدد لنا مقاطع الذاكرة المستخدمة والغير مستخدمة. ويمكن الحصول على خريطة الذاكرة بواسطة مقاطعة البايوس int 0x15 الدالة e820 التي تأخذ المدخلات التالية:

- العلم CF: صفر في حالة نجاح عمل المقاطعة.
- المسجل eax: القيمة 0x534d4150 وتساوي SMAP بترميز ASCII، وفي حالة حدوث خطأ فان رقم الخطأ سيحفظ في المسجل ah.
- المسجل ebx: عنوان السجل التالي أو صفر في حالة الإنتهاء.
- المسجل ecx: طول المقطع بالبايت.
- المسجلان es:di: سجل واصفة المقطع.

وتعود بالمخرجات:

- المسجل eax: رقم الدالة وهي 0xe820.
- المسجل ebx: عنوان البداية.
- المسجل ecx: حجم الذاكرة المؤقتة (buffer) وتساوي ٢٠ بايت على الأقل.
- المسجل edx: القيمة 0x534d4150 وتساوي SMAP بترميز ASCII.
- المسجلان es:di: عنوان الذاكرة buffer التي ستحفظ بها النتائج.

وبعد تنفيذ المقاطعة فان الذاكرة Buffer يتم ملئها بأول سجل وهو بطول ٢٤ بايت ويتم تكرار استدعاء المقاطعة الى أن تكون قيمة المسجل ebx مساوية للصفر. ومحتويات كل سجل يوضحها المثال ٧.٣.

Example ٧.٣: Memory Map Entry Structure

```
١
٢ ; Memory Map Entry Structure
٣ struc memory_map_entry
٤   .base_addr  resq  1
٥   .length   resq  1
٦   .type   resd  1
٧   .acpi_null  resd  1
٨ endstruc
```

وهي توصف مقطع الذاكرة حيث تحوي عنوان بداية المقطع وطوله ونوع المقطع وأخيرا بيانات محجوزة. ونوع المقطع يحدد ما إذا كان المقطع مستخدما أو محجوزا ويأخذ عدة قيم:

- القيمة 1: تدل على أن المقطع متوفرا.
- القيمة 2: تدل على أن المقطع محجوزا.
- القيمة 3: ACPI Reclaim Memory وهي منطقة محجوزة لكي يستخدمها النظام.
- القيمة 4:ACPI NVS Memory كذلك هي منقطة محجوزة للنظام.
- بقية القيم الأخرى تدل على أن المقطع غير معرف أو غير موجود.

والمثال ٧.٤ يوضح كيفية جلب مقاطع الذاكرة لكي يستفيد منها مدير الذاكرة لاحقا.

Example ٧.٤: Get Memory Map

```
١ ; ------------------------------------------------
٢ ; get_memory_map:
٣ ; Input:
٤ ;   EAX = 0x0000E820
٥ ;   EBX = continuation value or 0 to start at beginning of map
٦ ;   ECX = size of buffer for result (Must be >= 20 bytes)
٧ ;   EDX = 0x534D4150h ('SMAP')
٨ ;   ES:DI = Buffer for result
٩ ;
١٠ ; Return:
١١ ;   CF = clear if successful
```

```
;   EAX = 0x534D4150h ('SMAP')
;   EBX = offset of next entry to copy from or 0 if done
;   ECX = actual length returned in bytes
;   ES:DI = buffer filled
;   If error, AH containes error code
; ------------------------------------------------
get_memory_map:

    pushad
    xor ebx,ebx
    xor bp,bp
    mov edx,'PAMS'    ; 0x534D4150
    mov eax,0xe820
    mov ecx,24
    int 0x15      ; BIOS get memory map.

    jc .error
    cmp eax,'PAMS'
    jne .error

    test ebx,ebx
    je .error

    jmp .start

  .next_entry:
    mov edx,'PAMS'    ; 0x534D4150
    mov eax,0xe820
    mov ecx,24
    int 0x15      ; BIOS get memory map.

  .start:
    jcxz .skip_entry

    mov ecx,[es:di + memory_map_entry.length]
    test ecx,ecx
    jne short .good_entry
    mov ecx,[es:di + memory_map_entry.length+4]
    jecxz .skip_entry

  .good_entry:
```

```
٥٣      inc bp
٥٤      add di,24
٥٥
٥٦    .skip_entry:
٥٧      cmp ebx,0
٥٨      jne .next_entry
٥٩      jmp .done
٦٠
٦١    .error:
٦٢      stc         ; set carry flag
٦٣
٦٤    .done:
٦٥      popad
٦٦      ret
٦٧
٦٨ endstruc
```

٣.١.٧. مواصفات الإقلاع المتعدد

العديد من محملات النظام (Bootloader) تدعم الإقلاع المتعدد لمختلف أنظمة التشغيل وذلك عبر مواصفات ومقاييس محددة يجب أن يلتزم بها محمل النظام عند تحميل نواة النظام. ومن ضمن هذا المقياس تمرير بيانات الإقلاع المتعدد (Multiboot Information) من محمل النظام الى نواة نظام التشغيل. وما يهمنا حاليا هو تمرير حجم الذاكرة وخريطة الذاكرة الى النواة حتى يتمكن مدير الذاكرة من إدارتها بناءا على خريطة الذاكرة وحجمها. حيث ذكرنا سابقا أنه أثناء عمل النواة لا توجد طريقة مبسطة لتحديد حجم الذاكرة ومخطط المقاطع فيها ، لذلك تم اللجوء الى مقاطعات البايوس والإستفادة من خدماته ومن ثم تمرير النتائج الى نواة النظام عن طريق هيكلة قياسية[١].

حالة الحاسب

من ضمن هذه المقاييس أيضا توفر حالة معينة للحاسب (Machine State) ،وهي تنص على أنه عند تحميل نواة أي نظام تشغيل فان بعض المسجلات يجب أن تأخذ قيما محددة كالاتي:

- المسجل eax: يجب أن يأخذ الرقم 0x2BADB002 وهي إشارة لنواة النظام بأن محمل النظام يدعم الإقلاع المتعدد.
- المسجل ebx: تحتوي على عنوان بداية هيكلة الإقلاع المتعدد.

[١]على الرغم من أنه يمكن تمرير هذه البيانات بأي طريقة إلا أن الإلتزام بهيكلة قياسية سيفيد لاحقا عند دعم الإقلاع المتعدد.

- المسجل cs: واصفة الشفرة يجب أن تكون ٣٢ بت قراءة/تنفيذ بدءا من العنوان 0x0 وانتهاءا بالعنوان 0xffffffff.
- المسجلات ds,es,fs,gs,ss: يجب أن تكون مقاطع القراءة والكتابة ٣٢ بت وتبدأ من العنوان 0x0 وتنتهي بالعنوان 0xffffffff.
- يجب تفعيل بوابة a20.
- مسجل التحكم cr0: البت 0 يجب أن يفعل (تفعيل النمط المحمي) والبت ٣١ يجب أن يعطل (تعطيل التصفيح).

معلومات الإقلاع المتعدد

تعتبر هيكلة معلومات المتعدد من أهم الهياكل التي يجب تمريرها الى النواة ، ويتم حفظ عنوانها في المسجل ebx وهي الطريقة القياسية التي يتم بها تمرير الهيكلة الى النواة ، لكن بسبب أننا في هذه المرحلة لا ندعم الإقلاع المتعدد فيمكن أن نمرر هذه البيانات بأي شكل كان كدفع عنوانها الى المكدس (stack). والمثال ٧.٥ يوضح هيكلة معلومات الإقلاع المتعدد.

Example ٧.٥: Multiboot Inforamtion Structure

```
١ boot_info:
٢
٣ istruc multiboot_info
٤   at multiboot_info.flags,        dd 0
٥   at multiboot_info.mem_low,        dd 0
٦   at multiboot_info.mem_high,       dd 0
٧   at multiboot_info.boot_device,      dd 0
٨   at multiboot_info.cmd_line,       dd 0
٩   at multiboot_info.mods_count,     dd 0
١٠  at multiboot_info.mods_addr,      dd 0
١١  at multiboot_info.sym0,         dd 0
١٢  at multiboot_info.sym1,         dd 0
١٣  at multiboot_info.sym2,         dd 0
١٤  at multiboot_info.mmap_length,      dd 0
١٥  at multiboot_info.mmap_addr,      dd 0
١٦  at multiboot_info.drives_length,    dd 0
١٧  at multiboot_info.drives_addr,      dd 0
١٨  at multiboot_info.config_table,     dd 0
١٩  at multiboot_info.bootloader_name,    dd 0
٢٠  at multiboot_info.apm_table,      dd 0
```

```
٢١   at multiboot_info.vbe_control_info,   dd 0
٢٢   at multiboot_info.vbe_mode_info,    dw 0
٢٣   at multiboot_info.vbe_interface_seg,  dw 0
٢٤   at multiboot_info.vbe_interface_off,  dw 0
٢٥   at multiboot_info.vbe_interface_len,  dw 0
٢٦ iend
```

ويحدد المقياس استخدام المتغير flags لتحديد البيانات المستخدمة في هيكلة معلومات الإقلاع المتعدد من غير المستخدمة ، فمثلا في حالة كانت قيمة البت flags[0] هي ١ فان المتغيرات mem_low و mem_high يمكن استخدامها وهكذا.وحاليا لن يتم التركيز على المتغير flags وسيتم وضع قيم للمتغيرات mem_low و mem_high والتي تحوي حجم الذاكرة الفيزيائية والتي تم جلبها بواسطة البايوس ، وكذلك المتغيرات mmap_addr و mmap_length والتي توضح عنوان بداية مخطط الذاكرة الذي تم جلبه أيضا بواسطة البايوس وكذلك طولها.والمثال ٧.٦ يوضح الأوامر التي تم إضافتها الى المرحلة الثانية من محمل النظام لدعم الإقلاع المتعدد وكيفية حفظ معلومات الإقلاع المتعدد وارسالها الى نواة نظام التشغيل.

Example ٧.٦: snippet from stage2 bootloader

```
١
٢ ...
٣ ; when stage2 begin started, BIOS put drive number where stage1 are
      loaded from in dl
٤       mov [boot_info+multiboot_info.boot_device],dl
٥ ...
٦
٧ ;-----------------
٨ ; Get Memory Size
٩ ;-----------------
١٠      xor eax,eax
١١      xor ebx,ebx
١٢      call get_memory_size
١٣
١٤      mov [boot_info+multiboot_info.mem_low],ax
١٥      mov [boot_info+multiboot_info.mem_high],bx
١٦
١٧ ...
١٨
١٩ ;-----------------------------------------------
٢٠ ; Pass MultiBoot Info to the Kernel
٢١ ;-----------------------------------------------
٢٢      mov eax,0x2badb002
```

```
٢٣      mov ebx,0
٢٤      mov edx,[kernel_size]
٢٥      push dword boot_info
٢٦
٢٧      call ebp     ; Call Kernel
```

وعند استدعاء النواة فان البيانات التي تم دفعها الى المكدس تعتبر كوسيط للدالة ، وتم انشاء هيكلة بلغة السي (بداخل النواة) بنفس حجم الهيكلة التي تم دفعها الى المكدس وذلك لقراءة محتوياتها بشكل مبسط ومقروء.والمثال ٧.٧ يوضح كيفية استقبال النواة لهذه الهيكلة.

Example ٧.٧: Kernel Entry

```
١
٢ void _cdecl kernel_entry (multiboot_info* boot_info)
٣ {
٤
٥ #ifdef i386
٦   _asm {
٧     cli
٨
٩     mov ax, 10h   // select data descriptor in GDT.
١٠    mov ds, ax
١١    mov es, ax
١٢    mov fs, ax
١٣    mov gs, ax
١٤  }
١٥ #endif
١٦
١٧  // Execute global constructors
١٨  init_ctor();
١٩
٢٠  // Call kernel entry point
٢١  main(boot_info);
٢٢
٢٣  // Cleanup all dynamic dtors
٢٤  exit();
٢٥
٢٦ #ifdef i386
٢٧  _asm {
٢٨    cli
٢٩    hlt
```

```
٣٠    }
٣١ #endif
٣٢
٣٣    for(;;);
٣٤ }
```

٤.١.٧. مدير الذاكرة الفيزيائية

Example ٧.٨: Physical Memory Manager Interface

```
١
٢ #ifndef PMM_H
٣ #define PMM_H
٤
٥ #include <stdint.h>
٦
٧ extern void pmm_init(size_t size,uint32_t base);
٨ extern void pmm_init_region(uint32_t base,size_t size);
٩ extern void pmm_deinit_region(uint32_t base,size_t size);
١٠ extern void* pmm_alloc();
١١ extern void* pmm_allocs(size_t size);
١٢ extern void pmm_dealloc(void*);
١٣ extern void pmm_deallocs(void*,size_t);
١٤ extern size_t pmm_get_mem_size();
١٥ extern uint32_t pmm_get_block_count();
١٦ extern uint32_t pmm_get_used_block_count();
١٧ extern uint32_t pmm_get_free_block_count();
١٨ extern uint32_t pmm_get_block_size();
١٩ extern void pmm_enable_paging(bool);
٢٠ extern bool pmm_is_paging();
٢١ extern void pmm_load_PDBR(uint32_t);
٢٢ extern uint32_t pmm_get_PDBR();
٢٣
٢٤ #endif // PMM_H
```

Example ٧.٩: Physical Memory Manager Implemetation

```
١
```

```
٢ #include <string.h>
٣ #include "pmm.h"
٤ #include "kdisplay.h"
٥
٦ #define PMM_BLOCK_PER_BYTE  8
٧ #define PMM_PAGE_SIZE   4096
٨ #define PMM_BLOCK_SIZE    PMM_PAGE_SIZE
٩ #define PMM_BLOCK_ALIGN   PMM_BLOCK_SIZE
١٠
١١ static uint32_t _pmm_mem_size = 0;
١٢ static uint32_t _pmm_max_blocks = 0;
١٣ static uint32_t _pmm_used_blocks = 0;
١٤ static uint32_t* _pmm_mmap = 0;
١٥
١٦ inline void mmap_set(int bit) {
١٧   _pmm_mmap[bit/32] = _pmm_mmap[bit/32] | (1 << (bit%32)) ;
١٨ }
١٩
٢٠ inline void mmap_unset(int bit) {
٢١   _pmm_mmap[bit/32] = _pmm_mmap[bit/32] & ~(1 << (bit%32));
٢٢ }
٢٣
٢٤ inline bool mmap_test(int bit) {
٢٥   return _pmm_mmap[bit/32] & (1 << (bit%32));
٢٦ }
٢٧
٢٨ int mmap_find_first() {
٢٩
٣٠   for (uint32_t i=0;i<pmm_get_free_block_count()/32;++i) {
٣١     if (_pmm_mmap[i] != 0xffffffff) {
٣٢       for (int j=0;j<32;++j) {
٣٣         int bit = 1 << j;
٣٤         if (!(_pmm_mmap[i] & bit))
٣٥           return i*32+j;
٣٦       }
٣٧     }
٣٨   }
٣٩
٤٠   return -1;
٤١ }
٤٢
```

```
int mmap_find_squence(size_t s) {
  if (s == 0)
    return -1;

  if (s == 1)
    return mmap_find_first();

  for (uint32_t i=0;i<pmm_get_free_block_count()/32;++i) {
    if (_pmm_mmap[i] != 0xffffffff) {
      for (int j=0;j<32;++j) {
        int bit = 1 << j;
        if (!(_pmm_mmap[i] & bit)) {

          int start_bit = i*32 + bit;
          uint32_t free_bit = 0;

          for (uint32_t count=0;count<=s;++count) {
            if (!(mmap_test(start_bit+count)))
              free_bit++;

            if (free_bit == s)
              return i*32+j;
          }
        }
      }
    }
  }

  return -1;
}


void pmm_init(size_t size,uint32_t mmap_base) {
  _pmm_mem_size = size;
  _pmm_mmap = (uint32_t*)mmap_base;
  _pmm_max_blocks = _pmm_mem_size*1024 / PMM_BLOCK_SIZE;
  _pmm_used_blocks = _pmm_max_blocks;  // all memory used by default

  memset(_pmm_mmap,0xf,_pmm_max_blocks/PMM_BLOCK_PER_BYTE);
}

```

```
٨٤ void pmm_init_region(uint32_t base,size_t size) {
٨٥   int align = base/PMM_BLOCK_SIZE;
٨٦   int blocks = size/PMM_BLOCK_SIZE;
٨٧
٨٨   for (;blocks >=0;blocks--) {
٨٩     mmap_unset(align++);
٩٠     _pmm_used_blocks--;
٩١   }
٩٢
٩٣   mmap_set(0);
٩٤ }
٩٥
٩٦ void pmm_deinit_region(uint32_t base,size_t size) {
٩٧   int align = base/PMM_BLOCK_SIZE;
٩٨   int blocks = size/PMM_BLOCK_SIZE;
٩٩
١٠٠   for (;blocks >=0;blocks--) {
١٠١     mmap_set(align++);
١٠٢     _pmm_used_blocks++;
١٠٣   }
١٠٤ }
١٠٥
١٠٦ void* pmm_alloc() {
١٠٧   if (pmm_get_free_block_count() <= 0)
١٠٨     return 0;
١٠٩
١١٠   int block = mmap_find_first();
١١١
١١٢   if (block == -1)
١١٣     return 0;
١١٤
١١٥   mmap_set(block);
١١٦
١١٧   uint32_t addr = block * PMM_BLOCK_SIZE;
١١٨   _pmm_used_blocks++;
١١٩
١٢٠   return (void*) addr;
١٢١ }
١٢٢
١٢٣ void* pmm_allocs(size_t s) {
١٢٤   if (pmm_get_free_block_count() <= s)
```

```
١٢٥     return 0;
١٢٦
١٢٧   int block = mmap_find_squence(s);
١٢٨
١٢٩   if (block == -1)
١٣٠     return 0;
١٣١
١٣٢   for (uint32_t i=0;i<s;++i)
١٣٣     mmap_set(block+i);
١٣٤
١٣٥   uint32_t addr = block * PMM_BLOCK_SIZE;
١٣٦   _pmm_used_blocks += s;
١٣٧
١٣٨   return (void*) addr;
١٣٩ }
١٤٠
١٤١ void pmm_dealloc(void* p) {
١٤٢   uint32_t addr = (uint32_t)p;
١٤٣   int block = addr / PMM_BLOCK_SIZE;
١٤٤   mmap_unset(block);
١٤٥   _pmm_used_blocks--;
١٤٦ }
١٤٧
١٤٨ void pmm_deallocs(void* p,size_t s) {
١٤٩   uint32_t addr = (uint32_t)p;
١٥٠   int block = addr / PMM_BLOCK_SIZE;
١٥١
١٥٢   for (uint32_t i=0;i<s;++i)
١٥٣     mmap_unset(block+i);
١٥٤
١٥٥   _pmm_used_blocks -= s;
١٥٦ }
١٥٧
١٥٨ size_t pmm_get_mem_size() {
١٥٩   return _pmm_mem_size;
١٦٠ }
١٦١
١٦٢ uint32_t pmm_get_block_count() {
١٦٣   return _pmm_max_blocks;
١٦٤ }
١٦٥
```

```
١٦٦ uint32_t pmm_get_used_block_count() {
١٦٧   return _pmm_used_blocks;
١٦٨ }
١٦٩
١٧٠ uint32_t pmm_get_free_block_count() {
١٧١   return _pmm_max_blocks - _pmm_used_blocks;
١٧٢ }
١٧٣
١٧٤ uint32_t pmm_get_block_size() {
١٧٥   return PMM_BLOCK_SIZE;
١٧٦ }
١٧٧
١٧٨ void pmm_enable_paging(bool val) {
١٧٩ #ifdef _MSC_VER
١٨٠   _asm {
١٨١
١٨٢     mov eax,cr0
١٨٣     cmp [val],1
١٨٤     je enable
١٨٥     jmp disable
١٨٦
١٨٧   enable:
١٨٨     or eax,0x80000000  // set last bit
١٨٩     mov cr0,eax
١٩٠     jmp done
١٩١
١٩٢   disable:
١٩٣     and eax,0x7fffffff // unset last bit
١٩٤     mov cr0,eax
١٩٥
١٩٦   done:
١٩٧   }
١٩٨
١٩٩ #endif
٢٠٠ }
٢٠١
٢٠٢ bool pmm_is_paging() {
٢٠٣   uint32_t val = 0;
٢٠٤ #ifdef _MSC_VER
٢٠٥   _asm {
٢٠٦     mov eax,cr0
```

```
٢٠٧     mov [val],eax
٢٠٨   }
٢٠٩
٢١٠   if ( val & 0x80000000 )
٢١١     false;
٢١٢   else
٢١٣     true;
٢١٤ #endif
٢١٥ }
٢١٦
٢١٧ void pmm_load_PDBR(uint32_t addr) {
٢١٨ #ifdef _MSC_VER
٢١٩   _asm {
٢٢٠     mov eax,[addr]
٢٢١     mov cr3,eax
٢٢٢   }
٢٢٣ #endif
٢٢٤ }
٢٢٥
٢٢٦ uint32_t pmm_get_PDBR() {
٢٢٧ #ifdef _MSC_VER
٢٢٨   _asm {
٢٢٩     mov eax,cr3
٢٣٠     ret
٢٣١   }
٢٣٢ #endif
٢٣٣ }
```

٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management

Example ٧.١٠: Virtual Memory Manager Interface

```
١
٢ #ifndef VMM_H
٣ #define VMM_H
٤
٥ #include <stdint.h>
٦
٧ #include "vmm_pte.h"
٨ #include "vmm_pde.h"
```

```
٩
١٠ #define PTE_N 1024
١١ #define PDE_N 1024
١٢
١٣ struct page_table {
١٤   uint32_t pte[PTE_N];
١٥ };
١٦
١٧ struct page_dir_table {
١٨   uint32_t pde[PDE_N];
١٩ };
٢٠
٢١ extern void vmm_init();
٢٢ extern bool vmm_alloc_page(uint32_t* e);
٢٣ extern void vmm_free_page(uint32_t* e);
٢٤ extern bool vmm_switch_page_dir_table(page_dir_table*);
٢٥ extern page_dir_table* vmm_get_page_dir_table();
٢٦ extern void vmm_flush_tlb_entry(uint32_t addr);
٢٧ extern void vmm_page_table_clear(page_table* pt);
٢٨ extern uint32_t vmm_vir_to_page_table_index(uint32_t addr);
٢٩ extern uint32_t* vmm_page_table_lookup_entry(page_table* pt,uint32_t
       addr);
٣٠ extern uint32_t vmm_vir_to_page_dir_table_index(uint32_t addr);
٣١ extern void vmm_page_dir_table_clear(page_dir_table* pd);
٣٢ extern uint32_t* vmm_page_dir_table_lookup_entry(page_dir_table* pd,
       uint32_t addr);
٣٣
٣٤
٣٥ #endif // VMM_H
```

Example ٧.١١: Virtual Memory Manager Implemetation

```
١
٢ #include "vmm.h"
٣
٤ #include <string.h>
٥ #include "vmm.h"
٦ #include "pmm.h"
٧
٨ #define PAGE_SIZE         4096
```

```
٩ #define PT_ADDRESS_SPACE_SIZE 0x400000
١٠ #define PD_ADDRESS_SPACE_SIZE 0x100000000
١١
١٢ page_dir_table* _cur_page_dir = 0;
١٣ uint32_t  _cur_page_dir_base_register = 0;
١٤
١٥ void vmm_init() {
١٦   page_table* pt = (page_table*) pmm_alloc();
١٧   if (!pt)
١٨     return ;
١٩
٢٠   vmm_page_table_clear(pt);
٢١
٢٢   for (int i=0, frame =0; i < 1024 ; ++i, frame += 4096) {
٢٣     uint32_t page = 0;
٢٤     pte_add_attrib(&page,I386_PTE_PRESENT);
٢٥     pte_set_frame(&page,frame);
٢٦
٢٧     pt->pte[vmm_vir_to_page_table_index(frame)] = page;
٢٨   }
٢٩
٣٠   page_dir_table* pd = (page_dir_table*) pmm_allocs(3);
٣١   if (!pd)
٣٢     return ;
٣٣
٣٤   vmm_page_dir_table_clear(pd);
٣٥
٣٦   uint32_t* pde = vmm_page_dir_table_lookup_entry(pd,0);
٣٧   pde_add_attrib(pde,I386_PDE_PRESENT);
٣٨   pde_add_attrib(pde,I386_PDE_WRITABLE);
٣٩   pde_set_frame(pde,(uint32_t)pt);
٤٠
٤١   _cur_page_dir_base_register = (uint32_t) &pd->pde;
٤٢   vmm_switch_page_dir_table(pd);
٤٣
٤٤   pmm_enable_paging(true);
٤٥
٤٦ }
٤٧
٤٨ bool vmm_alloc_page(uint32_t* e) {
٤٩   void* p = pmm_alloc();
```

```
٥٠   if (!p)
٥١     return false;
٥٢
٥٣   pte_set_frame(e,(uint32_t)p);
٥٤   pte_add_attrib(e,I386_PTE_PRESENT);
٥٥
٥٦   return true;
٥٧ }
٥٨
٥٩ void vmm_free_page(uint32_t* e) {
٦٠   void* p = (void*)pte_get_frame(*e);
٦١
٦٢   if (p)
٦٣     pmm_dealloc(p);
٦٤
٦٥   pte_del_attrib(e,I386_PTE_PRESENT);
٦٦
٦٧ }
٦٨
٦٩
٧٠ bool vmm_switch_page_dir_table(page_dir_table* pd) {
٧١   if (!pd)
٧٢     return false;
٧٣
٧٤   _cur_page_dir = pd;
٧٥   pmm_load_PDBR(_cur_page_dir_base_register);
٧٦   return true;
٧٧ }
٧٨
٧٩ page_dir_table* vmm_get_page_dir_table() {
٨٠   return _cur_page_dir;
٨١ }
٨٢
٨٣ void vmm_flush_tlb_entry(uint32_t addr) {
٨٤ #ifdef _MSC_VER
٨٥   _asm {
٨٦     cli
٨٧     invlpg addr
٨٨     sti
٨٩   }
٩٠ #endif
```

```
٩١ }
٩٢
٩٣ void vmm_page_table_clear(page_table* pt) {
٩٤   if (pt)
٩٥     memset(pt,0,sizeof(page_table));
٩٦ }
٩٧
٩٨ uint32_t vmm_vir_to_page_table_index(uint32_t addr) {
٩٩   if ( addr  >= PT_ADDRESS_SPACE_SIZE )
١٠٠    return 0;
١٠١  else
١٠٢    return addr / PAGE_SIZE;
١٠٣ }
١٠٤
١٠٥ uint32_t* vmm_page_table_lookup_entry(page_table* pt,uint32_t addr) {
١٠٦  if (pt)
١٠٧    return &pt->pte[vmm_vir_to_page_table_index(addr)];
١٠٨  else
١٠٩    return 0;
١١٠ }
١١١
١١٢ uint32_t vmm_vir_to_page_dir_table_index(uint32_t addr) {
١١٣  if ( addr >= PD_ADDRESS_SPACE_SIZE )
١١٤    return 0;
١١٥  else
١١٦    return addr / PAGE_SIZE;
١١٧ }
١١٨
١١٩ void vmm_page_dir_table_clear(page_dir_table* pd) {
١٢٠  if (pd)
١٢١    memset(pd,0,sizeof(page_dir_table));
١٢٢ }
١٢٣
١٢٤ uint32_t* vmm_page_dir_table_lookup_entry(page_dir_table* pd,uint32_t
       addr) {
١٢٥  if (pd)
١٢٦    return &pd->pde[vmm_vir_to_page_table_index(addr)];
١٢٧  else
١٢٨    return 0;
١٢٩ }
```

٨. مشغلات الاجهزة Device Driver

١.٨. برمجة مشغل لوحة المفاتيح Keyboard Driver

Example ٨.١: Keybaord Driver Interface

```
١
٢ #ifndef KEYBOARD_H
٣ #define KEYBOARD_H
٤
٥ #include <stdint.h>
٦
٧ enum KEY_CODE{
٨   KEY_SPACE             = ' ',
٩   KEY_0                 = '0',
١٠  KEY_1                 = '1',
١١  KEY_2                 = '2',
١٢  KEY_3                 = '3',
١٣  KEY_4                 = '4',
١٤  KEY_5                 = '5',
١٥  KEY_6                 = '6',
١٦  KEY_7                 = '7',
١٧  KEY_8                 = '8',
١٨  KEY_9                 = '9',
١٩  KEY_A                 = 'a',
٢٠  KEY_B                 = 'b',
٢١  KEY_C                 = 'c',
٢٢  KEY_D                 = 'd',
٢٣  KEY_E                 = 'e',
٢٤  KEY_F                 = 'f',
٢٥  KEY_G                 = 'g',
٢٦  KEY_H                 = 'h',
٢٧  KEY_I                 = 'i',
٢٨  KEY_J                 = 'j',
٢٩  KEY_K                 = 'k',
```

```
٣٠   KEY_L                = 'l',
٣١   KEY_M                = 'm',
٣٢   KEY_N                = 'n',
٣٣   KEY_O                = 'o',
٣٤   KEY_P                = 'p',
٣٥   KEY_Q                = 'q',
٣٦   KEY_R                = 'r',
٣٧   KEY_S                = 's',
٣٨   KEY_T                = 't',
٣٩   KEY_U                = 'u',
٤٠   KEY_V                = 'v',
٤١   KEY_W                = 'w',
٤٢   KEY_X                = 'x',
٤٣   KEY_Y                = 'y',
٤٤   KEY_Z                = 'z',
٤٥   KEY_RETURN           = '\r',
٤٦   KEY_ESCAPE           = 0x1001,
٤٧   KEY_BACKSPACE        = '\b',
٤٨   KEY_UP               = 0x1100,
٤٩   KEY_DOWN             = 0x1101,
٥٠   KEY_LEFT             = 0x1102,
٥١   KEY_RIGHT            = 0x1103,
٥٢   KEY_F1               = 0x1201,
٥٣   KEY_F2               = 0x1202,
٥٤   KEY_F3               = 0x1203,
٥٥   KEY_F4               = 0x1204,
٥٦   KEY_F5               = 0x1205,
٥٧   KEY_F6               = 0x1206,
٥٨   KEY_F7               = 0x1207,
٥٩   KEY_F8               = 0x1208,
٦٠   KEY_F9               = 0x1209,
٦١   KEY_F10              = 0x120a,
٦٢   KEY_F11              = 0x120b,
٦٣   KEY_F12              = 0x120b,
٦٤   KEY_F13              = 0x120c,
٦٥   KEY_F14              = 0x120d,
٦٦   KEY_F15              = 0x120e,
٦٧   KEY_DOT              = '.',
٦٨   KEY_COMMA            = ',',
٦٩   KEY_COLON            = ':',
٧٠   KEY_SEMICOLON        = ';',
```

```
٧١   KEY_SLASH             = '/',
٧٢   KEY_BACKSLASH         = '\\',
٧٣   KEY_PLUS              = '+',
٧٤   KEY_MINUS             = '—',
٧٥   KEY_ASTERISK          = '*',
٧٦   KEY_EXCLAMATION       = '!',
٧٧   KEY_QUESTION          = '?',
٧٨   KEY_QUOTEDOUBLE       = '\"',
٧٩   KEY_QUOTE             = '\'',
٨٠   KEY_EQUAL             = '=',
٨١   KEY_HASH              = '#',
٨٢   KEY_PERCENT           = '%',
٨٣   KEY_AMPERSAND         = '&',
٨٤   KEY_UNDERSCORE        = '_',
٨٥   KEY_LEFTPARENTHESIS   = '(',
٨٦   KEY_RIGHTPARENTHESIS  = ')',
٨٧   KEY_LEFTBRACKET       = '[',
٨٨   KEY_RIGHTBRACKET      = ']',
٨٩   KEY_LEFTCURL          = '{',
٩٠   KEY_RIGHTCURL         = '}',
٩١   KEY_DOLLAR            = '$',
٩٢   KEY_POUND             = '£',
٩٣   KEY_EURO              = '$',
٩٤   KEY_LESS              = '<',
٩٥   KEY_GREATER           = '>',
٩٦   KEY_BAR               = '|',
٩٧   KEY_GRAVE             = '`',
٩٨   KEY_TILDE             = '~',
٩٩   KEY_AT                = '@',
١٠٠  KEY_CARRET            = '^',
١٠١  KEY_KP_0              = '0',
١٠٢  KEY_KP_1              = '1',
١٠٣  KEY_KP_2              = '2',
١٠٤  KEY_KP_3              = '3',
١٠٥  KEY_KP_4              = '4',
١٠٦  KEY_KP_5              = '5',
١٠٧  KEY_KP_6              = '6',
١٠٨  KEY_KP_7              = '7',
١٠٩  KEY_KP_8              = '8',
١١٠  KEY_KP_9              = '9',
١١١  KEY_KP_PLUS           = '+',
```

```
١١٢   KEY_KP_MINUS          = '—',
١١٣   KEY_KP_DECIMAL        = '.',
١١٤   KEY_KP_DIVIDE         = '/',
١١٥   KEY_KP_ASTERISK       = '*',
١١٦   KEY_KP_NUMLOCK        = 0x300f,
١١٧   KEY_KP_ENTER          = 0x3010,
١١٨   KEY_TAB               = 0x4000,
١١٩   KEY_CAPSLOCK          = 0x4001,
١٢٠   KEY_LSHIFT            = 0x4002,
١٢١   KEY_LCTRL             = 0x4003,
١٢٢   KEY_LALT              = 0x4004,
١٢٣   KEY_LWIN              = 0x4005,
١٢٤   KEY_RSHIFT            = 0x4006,
١٢٥   KEY_RCTRL             = 0x4007,
١٢٦   KEY_RALT              = 0x4008,
١٢٧   KEY_RWIN              = 0x4009,
١٢٨   KEY_INSERT            = 0x400a,
١٢٩   KEY_DELETE            = 0x400b,
١٣٠   KEY_HOME              = 0x400c,
١٣١   KEY_END               = 0x400d,
١٣٢   KEY_PAGEUP            = 0x400e,
١٣٣   KEY_PAGEDOWN          = 0x400f,
١٣٤   KEY_SCROLLLOCK        = 0x4010,
١٣٥   KEY_PAUSE             = 0x4011,
١٣٦   KEY_UNKNOWN,
١٣٧   KEY_NUMKEYCODES
١٣٨ };
١٣٩
١٤٠ extern bool keyboard_get_scroll_lock();
١٤١ extern bool keyboard_get_caps_lock();
١٤٢ extern bool keyboard_get_num_lock();
١٤٣ extern bool keyboard_get_alt();
١٤٤ extern bool keyboard_get_ctrl();
١٤٥ extern bool keyboard_get_shift();
١٤٦ extern void keyboard_ignore_resend();
١٤٧ extern bool keyboard_check_resend();
١٤٨ extern bool keyboard_get_diagnostic_res();
١٤٩ extern bool keyboard_get_bat_res();
١٥٠ extern bool keyboard_self_test();
١٥١ extern uint8_t keyboard_get_last_scan();
١٥٢ extern KEY_CODE keyboard_get_last_key();
```

```
١٥٣ extern void keyboard_discard_last_key();
١٥٤ extern void keyboard_set_leds(bool nums,bool caps,bool scroll);
١٥٥ extern char keyboard_key_to_ascii(KEY_CODE k);
١٥٦ extern void keyboard_enable();
١٥٧ extern void keyboard_disable();
١٥٨ extern bool keyboard_is_disabled();
١٥٩ extern void keyboard_reset_system();
١٦٠ extern void keyboard_install(int);
١٦١
١٦٢ #endif // KEYBOARD_H
```

Example ٨.٢: Keybaord Driver Implemetation

```
١
٢ void keyboard_install(int irq) {
٣   // Install  interrupt handler (irq 1 uses interrupt 33)
٤   set_vector(irq, i386_keyboard_irq);
٥
٦   // assume BAT test is good. If there is a problem, the IRQ handler
        where catch the error
٧   _keyboard_bat_res = true;
٨   _scancode = 0;
٩
١٠  // set lock keys and led lights
١١  _numlock = _scrolllock = _capslock = false;
١٢  keyboard_set_leds (false, false, false);
١٣
١٤  // shift, ctrl, and alt keys
١٥  _shift = _alt = _ctrl = false;
١٦ }
١٧
١٨
١٩ //  keyboard interrupt handler
٢٠ void _cdecl i386_keyboard_irq () {
٢١
٢٢  _asm add esp, 12
٢٣  _asm pushad
٢٤  _asm cli
٢٥
٢٦  static bool _extended = false;
```

```
٢٧
٢٨   int code = 0;
٢٩
٣٠   // read scan code only if the keyboard controller output buffer is
        full (scan code is in it)
٣١   if (keyboard_ctrl_read_status () & KEYBOARD_CTRL_STATS_MASK_OUT_BUF
        ) {
٣٢
٣٣     // read the scan code
٣٤     code = keyboard_enc_read_buf ();
٣٥
٣٦     // is this an extended code? If so, set it and return
٣٧     if (code == 0xE0 || code == 0xE1)
٣٨       _extended = true;
٣٩     else {
٤٠
٤١       // either the second byte of an extended scan code or a single
            byte scan code
٤٢       _extended = false;
٤٣
٤٤       // test if this is a break code (Original XT Scan Code Set
            specific)
٤٥       if (code & 0x80) {  //test bit 7
٤٦
٤٧         // covert the break code into its make code equivelant
٤٨         code -= 0x80;
٤٩
٥٠         // grab the key
٥١         int key = _keyboard_scancode_std [code];
٥٢
٥٣         // test if a special key has been released & set it
٥٤         switch (key) {
٥٥
٥٦           case KEY_LCTRL:
٥٧           case KEY_RCTRL:
٥٨             _ctrl = false;
٥٩             break;
٦٠
٦١           case KEY_LSHIFT:
٦٢           case KEY_RSHIFT:
٦٣             _shift = false;
```

```
٦٤              break;
٦٥
٦٦            case KEY_LALT:
٦٧            case KEY_RALT:
٦٨              _alt = false;
٦٩              break;
٧٠          }
٧١        }
٧٢        else {
٧٣
٧٤          // this is a make code — set the scan code
٧٥          _scancode = code;
٧٦
٧٧          // grab the key
٧٨          int key = _keyboard_scancode_std [code];
٧٩
٨٠          // test if user is holding down any special keys & set it
٨١          switch (key) {
٨٢
٨٣            case KEY_LCTRL:
٨٤            case KEY_RCTRL:
٨٥              _ctrl = true;
٨٦              break;
٨٧
٨٨            case KEY_LSHIFT:
٨٩            case KEY_RSHIFT:
٩٠              _shift = true;
٩١              break;
٩٢
٩٣            case KEY_LALT:
٩٤            case KEY_RALT:
٩٥              _alt = true;
٩٦              break;
٩٧
٩٨            case KEY_CAPSLOCK:
٩٩              _capslock = (_capslock) ? false : true;
١٠٠             keyboard_set_leds (_numlock, _capslock, _scrolllock);
١٠١             break;
١٠٢
١٠٣           case KEY_KP_NUMLOCK:
١٠٤             _numlock = (_numlock) ? false : true;
```

```
١٠٥             keyboard_set_leds (_numlock, _capslock, _scrolllock);
١٠٦             break;
١٠٧
١٠٨           case KEY_SCROLLLOCK:
١٠٩             _scrolllock = (_scrolllock) ? false : true;
١١٠             keyboard_set_leds (_numlock, _capslock, _scrolllock);
١١١             break;
١١٢         }
١١٣       }
١١٤     }
١١٥
١١٦     // watch for errors
١١٧     switch (code) {
١١٨
١١٩       case KEYBOARD_ERR_BAT_FAILED:
١٢٠         _keyboard_bat_res = false;
١٢١         break;
١٢٢
١٢٣       case KEYBOARD_ERR_DIAG_FAILED:
١٢٤         _keyboard_diag_res = false;
١٢٥         break;
١٢٦
١٢٧       case KEYBOARD_ERR_RESEND_CMD:
١٢٨         _keyboard_resend_res = true;
١٢٩         break;
١٣٠     }
١٣١   }
١٣٢
١٣٣   // tell hal we are done
١٣٤   int_done(0);
١٣٥
١٣٦   // return from interrupt handler
١٣٧   _asm sti
١٣٨   _asm popad
١٣٩   _asm iretd
١٤٠ }
```

الخاتمة

وهكذا نصل الى نهاية المطاف لهذا البحث المتواضع والذي نرجو أن يكون إضافة ولو يسيرة للمكتبة العربية وللقارئ العربي في مجال برمجة أنظمة التشغيل. ونسأل الله تعالى أن ينفعنا بما علمنا وأن ينفع بنا الإسلام والمسلمين وأن يعيننا على تعلم العلم وتبليغه انه ولي ذلك والقادر عليه. وأخيرا ما كان من خطأ فمن نفسي والشيطان وما كان من صواب فمن الله عز وجل.

وصلى اللهم وسلم على نبينا محمد وعلى آله وصحبه أجمعين . وآخر دعوانا أن **الحمد لله رب العالمين**.

النتائج

التوصيات

حتى يصل البحث لمرحلة مناسبة فانه يجب الحديث عن عددا من الموضيع التي تم تجاهلها لسبب أو لآخر ، فبداية من الفصل الأول والذي يعتبر مقدمة عامة الى أنظمة التشغيل حيث يمكن أن تضاف له العديد من المعلومات عن الأنظمة الحالية وكذلك مزيدا من التوضيحات عن ماهية أنظمة التشغيل. أما الفصل الثاني فلم يتحدث سوى عن القشور في معمارية الحاسب حيث أنه يتطلب كتابا متكاملا لشرح كل جزئية ولكن تم الإيجاز والحديث عن المواضيع التي يكثر تناولها عند برمجة أنظمة التشغيل. وقبل الإنتقال الى الفصل الثالث يمكن كتابة فصل جديد عن لغة التجميع بحيث يشرح أساسيات اللغة مع مترجم NASM لانه المترجم المستخدم في عملية تجميع البرامج حيث أنه يحوي أوامر خاصة به يجب الوقوف عندها وتوضيحها بشكل جيد. أما الفصل الثالث والرابع فتم الحديث فيها عن محمل النظام الذي تم برمجته لتحميل نظام إقرأ ويمكن هنا الحديث عن أي من المحملات المفتوحة المصدر (مثل محمل GRUB والذي يعتبر من أشهر المحملات وأكثرهم استخداما) وكيفية تحميل نواة النظام مباشرة من خلالهم. والفصل الخامس تحدث عن النواة وطرق تصميمها بشكل مختصر و لم يتناول الاختلاف بينهم وطرق برمجة كل منهم على حدة ويعتبر هذا الفصل مجالا كاملا للبحث فيه . أما الفصل السادس فهو مكتمل تقريبا ويمكن اضافة المزيد من العلومات ، بينما الفصل السابع لم يتحدث عن خوارزميات ادراة الذاكرة وتم تطبيق خوارزمية First Fit للبحث عن أماكن شاغرة في الذاكرة ويمكن تطبيق أي من الخوارزميات الشهيرة ، كذلك يمكن تطبيق القائمة المتصلة لمتابعة الأماكن الشاغرة في الذاكرة بدلا من استخدام Bit Map. أما الفصل الثامن فهو عن مشغلات الأجهزة لكن نظرا لضيق الوقت فقد تم الحديث عن متحكم لوحة المفاتيح وكيفية قراءة الأحرف المدخلة منه ، ويمكن في هذا الفصل اضافة العديد من مشغلات الأجهزة مثل مشغل القرص المرن والقرص الصلب ومشغل لكرت الشبكة

وكرت الصوت والعديد من المشغلات الضرورية. أخيرا يمكن إضافة فصل للحديث عن أنظمة الملفات وكيفية برمجتهم مثل FAT12,FAT16,FAT32,EXT3,EXT4,...etc وفصل لبرمجة المجدول لدعم تعدد المهام.

ومن ناحية نظام **إقرأ** فانه بحاجة الى العديد من الإضافات والتحسينات:

- دعم واجهة POSIX.
- دعم لأنظمة الملفات FAT12,FAT16,FAT32,EXT3,EXT4,...etc.
- دعم لتعدد المهام.
- نقل (port) مترجم gcc,g++ وواجهة x11 الى نظام **إقرأ**.

Bibliography

[١] William Stallings, *Operating System: Internals and Design Principles*. Prentice Hall, 5th Edition, 2004.

[٢] Andrew S. Tanenbaum ,Albert S Woodhull, *Operating Systems Design and Implementation*. Prentice Hall, 3rd Edition, 2006.

[٣] Michael Tischer, Bruno Jennrich, *PC Intern: The Encyclopedia of System Programming*. Abacus Software, 6th Edition, 1996.

[٤] Hans-Peter Messmer, *The Indispensable PC Hardware Book*. Addison-Wesley Professional, 4th Edition, 2001.

[٥] Andrew S. Tanenbaum, *Structured Computer Organization*. Prentice Hall, 4th Edition, 1998.

[٦] Ytha Yu,Charles Marut, *Asssembly Language Programming and Organization IBM PC*. McGraw-Hill/Irwin, 1st Edition, 1992.

[٧] Intel® Manuals, *Intel® 64 and IA-32 Architectures Software Developer's Manuals*. http://www.intel.com/products/processor/manuals/

[٨] *OSDev*: http://wiki.osdev.org

[٩] *brokenthorn*: http://brokenthorn.com

[١٠] *Computer Sciense Student's Community in Sudan*: http://sudancs.com

[١١] *Wikipedia*: http://wikipedia.org

١. ترجمة وتشغيل البرامج

لتطوير نظام التشغيل يجب استخدام مجموعة من الادوات واللغات التي تساعد وتيسير عملية التطوير وفي هذا الفصل سيتم عرض هذه الأدوات وكيفية استخدامها.

اعداد مترجم فيجوال سي++ لبرمجة النواة.

١.١. نظام ويندوز

١.٢. نظام لينوكس

ب. شفرة نظام إقرأ

كود النظام