

بَرْمَجَة وَ تَصْمِيمُ نِظَامِ تَشْغِيلِ "نِظَامِ إِقْرَأْ"

أحمد بن عصام عبد الرحيم أحمد

٢٣ مايو ٢٠١٠

المحتويات

I. الأساسيات Basics ١

٣ ١. مقدمة عن أنظمة التشغيل وتاريخ ظهورها

٥ ٢. معمارية حواسيب x86

٦	١.٢ معمارية النظام
٦	١.١.٢ مسار النظام System Bus
٨	٢.١.٢ متحكم الذاكرة
٨	٣.١.٢ متحكم الإدخال والإخراج
٩	٢.٢ المعالج
١٠	١.٢.٢ دورة تنفيذ التعليمات
١٠	٢.٢.٢ أنماط عمل المعالج CPU Modes
١٢	٣.٢.٢ النمط الحقيقي Real Mode
١٣	٤.٢.٢ النمط المحمي Protected Mode
١٥	٥.٢.٢ النمط الغير حقيقي والنمط التخيلي
١٥	٦.٢.٢ معمارية معالجات x86

II. إقلاع الحاسب Booting ٢٣

٢٥ ٣. إقلاع الحاسب ومحمل النظام Bootloader

٢٥	١.٣ إقلاع الحاسب
٢٦	٢.٣ محمل النظام Bootloader
٢٨	٣.٣ مخطط الذاكرة
٢٨	٤.٣ برمجة محمل النظام
٢٩	١.٤.٣ عرض رسالة ترحيبية
٣٢	٢.٤.٣ معلومات قطاع الإقلاع

٤٠	٣.٤.٣	تحميل قطاع من القرص باستخدام المقاطعة 0x13 int
٤٢	٥.٣	مقدمة الى نظام FAT12
٤٣	١.٥.٣	قيود نظام FAT12
٤٤	٢.٥.٣	هيكلية نظام FAT12 على القرص
٤٧	٣.٥.٣	هيكلية القرص المرن
٤٧	٤.٥.٣	القراءة و الكتابة من نظام FAT12
٥٩	٤	برمجة محمل النظام - المرحلة الثانية
٥٩	١.٤	الانتقال الى النمط المحمي
٦٠	١.١.٤	جدول الواصفات العام Global Descriptor Table
٦٤	٢.١.٤	العنونة في النمط المحمي PMode Memory Addressing
٦٤	٣.١.٤	الانتقال الى النمط المحمي
٦٦	٢.٤	تفعيل البوابة A20
٦٦	١.٢.٤	متحكم لوحة المفاتيح 8042 والبوابة A20
٦٧	٢.٢.٤	طرق تفعيل البوابة A20
٧٣	٣.٤	أساسيات ال VGA
٧٤	١.٣.٤	عنونة الذاكرة في متحكمات VGA
٧٥	٢.٣.٤	طباعة حرف على الشاشة
٨٠	٣.٣.٤	طباعة السلاسل النصية strings
٨١	٤.٣.٤	تحديث المؤشر Hardware Cursor
٨٤	٥.٣.٤	تنظيف الشاشة Clear Screen
٨٥	٤.٤	تحميل النواة
٩٣	III	النواة Kernel
٩٥	٥	مقدمة حول نواة نظام التشغيل
٩٥	١.٥	نواة نظام التشغيل
٩٦	١.١.٥	مستويات التجريد
٩٧	٢.٥	وظائف نواة النظام
٩٧	١.٢.٥	إدارة الذاكرة
٩٨	٢.٢.٥	إدارة العمليات
٩٨	٣.٢.٥	نظام الملفات
٩٨	٣.٥	هيكلية وتصميم النواة
٩٨	١.٣.٥	النواة الضخمة Monolithic Kernel
٩٩	٢.٣.٥	النواة المصغرة MicroKernel
٩٩	٣.٣.٥	النواة الهجينة Hybrid Kernel

٩٩	٤.٥ . برمجة نواة النظام
١٠٠	١.٤.٥ . تحميل وتنفيذ نواة PE
١٠٣	٢.٤.٥ . تطوير بيئة التشغيل للغة سي++
١٠٨	٣.٤.٥ . نقل التنفيذ الى النواة
١١٠	٥.٥ . نظرة على شفرة نظام إقرأ
١١٠	٦.٥ . مكتبة السي القياسية
١١٥	٧.٥ . دالة طباعة المخرجات للنواة
١١٧	٦ . المقاطعات Interrupts
١١٧	١.٦ . المقاطعات البرمجية Software Interrupts
١١٧	١.١.٦ . المقاطعات في النمط الحقيقي
١١٩	٢.١.٦ . المقاطعات في النمط المحمي
١٢١	٣.١.٦ . أخطاء المعالج
١٢٣	٤.١.٦ . إنشاء جدول الواصفات العام GDT
١٢٧	٥.١.٦ . إنشاء جدول المقاطعات IDT
١٢٧	٢.٦ . Programmable Interrupt Controller
١٢٧	٣.٦ . Programmable Interval Timer
١٢٧	٤.٦ . المقاطعات العنادية Hardware Interrupts
١٢٩	٧ . إدارة الذاكرة
١٢٩	١.٧ . إدارة الذاكرة الفيزيائية Physical Memory Management
١٢٩	٢.٧ . إدارة الذاكرة التخيلية Virtual Memory Management
١٣١	٨ . مشغلات الاجهزة Device Driver
١٣١	١.٨ . برمجة مشغل لوحة المفاتيح Keyboard Driver
١٣١	٢.٨ . برمجة مشغل القرص المرن Floppy Disk Driver
١٣١	٣.٨ . برمجة متحكم DMAC
١٣٣	٩ . أنظمة الملفات
١٣٥	١ . ترجمة وتشغيل البرامج
١٣٥	١.١ . نظام ويندوز
١٣٥	٢.١ . نظام لينوكس
١٣٩	ب . المراجع
١٣٩	١.١ . المراجع

١٤٣	ج. شفرة نظام إقرأ
١٤٥	د. إتفاقية ترخيص المستندات الحرة GNU FDL

الأمثلة التوضيحية

٢٨	Bootloader Smallest	.١.٣
٢٩	World Hello	.٢.٣
٣٢	Block Parameter Bios	.٣.٣
٣٣	example BPB	.٤.٣
٣٦	bootloader of value Hex	.٥.٣
٣٧	Code Some	.٦.٣
٤١	Code Some	.٧.٣
٤٢	Code Some	.٨.٣
٤٨	Code Some	.٩.٣
٤٩	Code Some	.١٠.٣
٥٠	Code Some	.١١.٣
٥١	Code Some	.١٢.٣
٥٢	Code Some	.١٣.٣
٥٣	Code Some	.١٤.٣
٥٤	Code Some	.١٥.٣
٥٤	Code Some	.١٦.٣
٥٦	Code Some	.١٧.٣
٦١	Code Some	.١.٤
٦٣	Code Some	.٢.٤
٦٥	Code Some	.٣.٤
٦٧	Code Some	.٤.٤
٦٨	Code Some	.٥.٤
٦٩	Code Some	.٦.٤
٧٠	Code Some	.٧.٤
٧٢	Code Some	.٨.٤
٧٥	Code Some	.٩.٤
٧٧	Code Some	.١٠.٤
٨٠	Code Some	.١١.٤
٨٢	Code Some	.١٢.٤
٨٤	Code Some	.١٣.٤
٨٥	Code Some	.١٤.٤
٨٧	Code Some	.١٥.٤

١٠١	operator new/delete Global	.١.٥
١٠٣	operator new/delete Global	.٢.٥
١٠٤	operator new/delete Global	.٣.٥
١٠٤	operator new/delete Global	.٤.٥
١٠٥	operator new/delete Global	.٥.٥
١٠٥	operator new/delete Global	.٦.٥
١٠٧	operator new/delete Global	.٧.٥
١٠٨	operator new/delete Global	.٨.٥
١١١	C++ and C in NULL of null.h:Definition	.٩.٥
١١٢	C/C++ in t_size of t.h:Definition_size	.١٠.٥
١١٢	type data stdint.h:typedef	.١١.٥
١١٣	type data typedef cstdint:C++	.١٢.٥
١١٣	type character ctype.h:determine	.١٣.٥
١٢٠	descriptor interrupt of Example	.١.٦
١٢٠	IDTR in put to Value	.٢.٦
١٢٣	Interface Layer Abstraction include/hal.h:Hardware	.٣.٦
١٢٥	GDT hal/gdt.cpp:Install	.٤.٦
١٣٥	Code Some	.١.١

قائمة الأشكال

٥	معمارية حواسيب x86	١.٢
٦	المسارات في الحواسيب الشخصية x86	٢.٢
٨	الجسر الشمالي	٣.٢
١٤	تداخل المقاطع في النمط الحقيقي	٤.٢
١٥	حلقات المعالج	٥.٢
٩٢	محمل النظام أثناء العمل	١.٤
٩٢	بدء تنفيذ النواة	٢.٤

القسم I.

الأساسيات Basics

١ . مقدمة عن أنظمة التشغيل وتاريخ ظهورها

٢. معمارية حواسيب x86

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

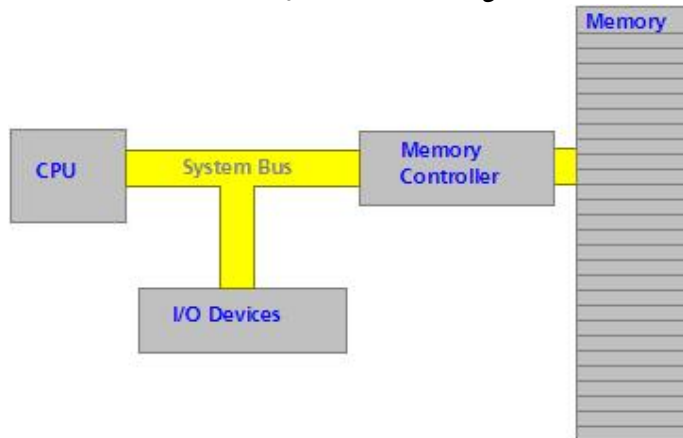
١. معالج أو وحدة معالجة مركزية (Central Processing Unit).

٢. ذاكرة (Memory).

٣. أجهزة إدخال وإخراج (I/O Devices).

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

شكل ١.٢.١: معمارية حواسيب x86

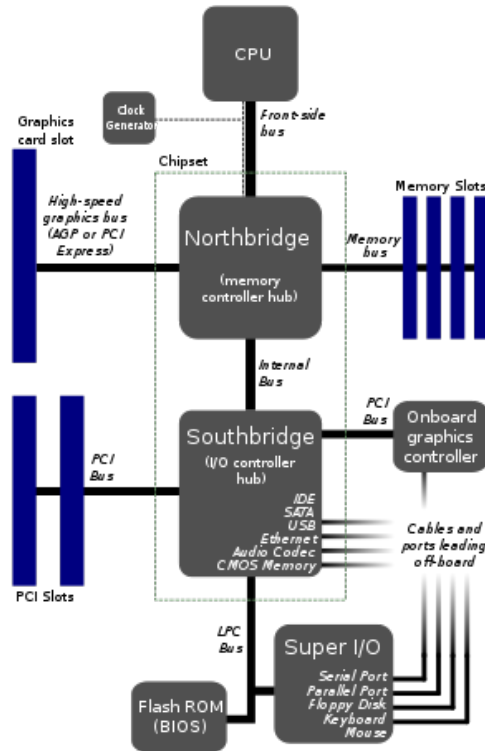


١.٢. معمارية النظام

١.١.٢. مسار النظام System Bus

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

شكل ٢.٢: المسارات في الحواسيب الشخصية x86



^١ ويسمى أيضا Front-side Bus.

مسار البيانات Data Bus

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

مسار العناوين Address Bus

يستخدم مسار العناوين في نقل عنوان الذاكرة المراد استخدامه سواءاً للقراءة منه أو الكتابة عليه ، ويحدد حجم مسار العناوين أكبر عنوان يمكن الوصول اليه في الذاكرة وبالتالي يحدد لنا حجم الذاكرة التي يستطيع الحاسب التعامل معها . وفي الأجهزة التي تستخدم معالجات انتل 8086 كان حجم هذا المسار هو 20-bit وبالتالي فان أقصى ذاكرة يتعامل معها هذا المعالج هي 1 MB^٢ أما في معالجات 80286/80386 فان حجم هذا مسار هو 24-bit وفي المعالجات التي تليها تم زيادة هذا الحجم الى 32-bit وبالتالي يمكن تنصيب ذاكرة بحجم 4 GB ، وفي المعالجات الحديثة تم زيادة هذا الحجم ، ولكننا سنقتصر في هذا البحث على المعالجات التي تدعم مسار عناوين بطول 32-bit بسبب انتشارها وسيطرتها لمدة من الزمن على أجهزة الحواسيب الشخصية.

مسار التحكم Control Bus

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

^٢ ناتجة من حساب 2 مرفوع للقوة 20.

٢.١.٢. متحكم الذاكرة

قبل أن نذكر وظيفة هذا المتحكم يجب إعطاء نبذة عن ماهية المتحكمات (Controllars) في جهاز الحاسب. ويُعرف **المتحكم** بأنه شريحة تتحكم بعتاد ما تحوي العديد من المسجلات الداخلية وظيفتها هو استقبال الأوامر وتنفيذها على العتاد. ويمكن أن نعرفها بأنها شريحة للربط ما بين الأوامر البرمجية إلى أوامر تنفيذ على عتاد ما. وأي متحكم يحوي العديد من المسجلات سواءاً كانت لإرسال واستقبال البيانات أو للأوامر، وأي مسجل يجب أن يأخذ رقم فريد يميزه عن بقية المسجلات الموجودة في هذا المتحكم أو في أي متحكم آخر وذلك حتى تتمكن من التعامل معه برمجياً، هذا الرقم يعرف باسم المنفذ (Port) وسنطلع عليه لاحقاً. وعمل المتحكم يبدأ عندما يُرسل أمر إليه حيث يبدأ المتحكم في تنفيذ هذا الأمر ومن ثم يضع النتيجة في أحد مسجلاته ويرسل إشارة (Interrupt) إلى المعالج لكي يقوم بقراءة القيمة. نعود إلى متحكم الذاكرة الرئيسية والذي يتواجد غالباً على متحكم الجسر الشمالي (NorthBridge) إنظر الشكل ?? . حيث تكمن وظيفته الأساسية في استقبال الأوامر المرسلة إلى الذاكرة وتنفيذها، ويقوم هذا المتحكم بتوجيه العناوين المرسلة إلى أي من شرائح الذاكرة كذلك يقوم بإعادة تنعش (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 إلى المتحكم المطلوب.



٣.١.٢. متحكم الإدخال والإخراج

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

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

المنافذ Ports

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

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

٢.٢. المعالج

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

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

جدول ١.٢: مخطط الذاكرة لحواسيب x86

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

١.٢.٢. دورة تنفيذ التعليمات

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

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

ثانياً مرحلة تفسير البيانات (Decode).

ثالثاً مرحلة تنفيذ البيانات (Execute).

رابعاً مرحلة حفظ النتائج (Write back).

٢.٢.٢. أنماط عمل المعالج CPU Modes

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

جدول ٢.٢: منافذ الإدخال والإخراج لحواسيب x86

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

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

٣.٢.٢. النمط الحقيقي Real Mode

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

عنوانة المقطع:الإزاحة (Segment:Offset Addressing)

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

٤ هذا ناتج من حساب 2^{16} .

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

$$physical_address = segment * 0x10 + offset$$

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

مشكلة تداخل المقاطع

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

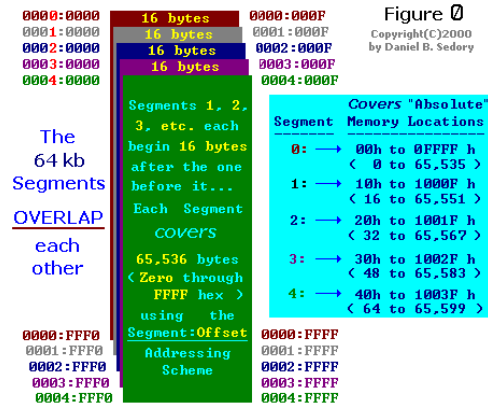
٢.٢.٤ . النمط المحمي Protected Mode

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

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

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

شكل ٤.٢: تداخل المقاطع في النمط الحقيقي



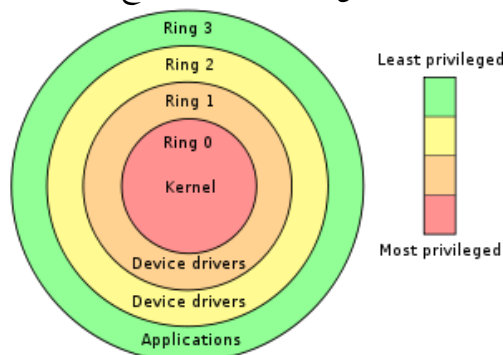
من الوصول العشوائي من قبل برامج المستخدم وذلك عن طريق وصف مقاطع الذاكرة وصلاحيات الوصول إليها في جداول تسمى جداول الوصفات (Descriptor Table). المعالج 80386 هو أول معالج ٣٢ بت يستخدم مسجلات بحجم ٣٢ بت وحجم مسار البيانات أيضا بنفس الحجم مما سمح بإمكانية التعامل مع ذاكرة بحجم ٤ جيجابايت. كذلك تم إضافة دعم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

حلقات المعالج CPU Rings

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

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

شكل ٥.٢: حلقات المعالج



٥.٢.٢ النمط الغير حقيقي والنمط التخيلي

٦.٢.٢ معمارية معالجات x86

أي معالج يتعرف على مجموعة من الأوامر تسمى Instruction Set بعضاً منها تتطلب صلاحية معينة (الحلقة صفر) لكي يقوم المعالج بتنفيذها (انظر الجدول ٣.٢ لمعرفة هذه الأوامر) وإلا فإن هذا سيتسبب في حدوث خطأ من المعالج يسمى الخطأ العام (General Protection Fault) والذي ان لم تتوفر دالة تتعامل معه (Exception Handler) فإن هذا يؤدي إلى توقف النظام عن العمل.

وتحتوي معالجات x86 العديد من المسجلات منها ما يستخدم للأغراض العامة (General Registers) ومنها ما يستخدم لحفظ العناوين وأرقام المقاطع (Segments Registers) وتوجد أيضاً مسجلات لا يمكن استخدامها إلا في برامج الحلقة صفر (أي النواة) حيث أن التغيير فيها يؤثر على عمل النظام وأخيراً هناك مجموعة من المسجلات الداخلية للمعالج والتي لا يمكن الوصول لها برمجياً. والقائمة التالية توضح هذه المسجلات :

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

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

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

المسجلات العامة General Purpose Registers

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

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

مسجلات المقاطع Segment Registers

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

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

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

مسجلات الإزاحة Offset Registers

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

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

مؤشر التعليمات Instruction Pointer

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

مسجل الأعلام FLAGS Register

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

ويحدد البتين IOPL مستوى الحماية المطلوب لتنفيذ مجموعة من الأوامر (مثل الأوامر CLI,STI,IN,OUT) حيث لن يتم تنفيذ مثل هذه التعليمات إلا في حالة كان مستوى الحماية الحالي Current Privilege Level أعلى من أو مساوياً للقيمة الموجودة في البتين IOPL^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 CRO, CR1, CR2, CR3, CR4, CR8 ، ونظراً لخطورة التعامل معها فان هذه المسجلات لا يمكن الوصول لها إلا عند العمل في نمط النواة (Kernel Moder/Ring0) ولا يُمكن لبرمجيات المستخدم الوصول الى هذه المسجلات والتعامل معها. وفي الوقت الحالي يهملنا فقط أول مسجل تحكم وهو CRO حيث من خلاله يمكن أو نقوم بعملية تحويل نمط المعالج من النمط الحقيقي الى النمط المحمي (PMode) وكذلك يمكن أن نقوم بتفعيل خاصية الصفحات (Paging) ، والتركيبية التالية توضح محتويات كل بت في مسجل التحكم CRO وهو مسجل بحجم ٣٢ بت.

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

^٧أعلى مستوى حماية هو الحلقة صفر (Ring0) ويليهما الحلقة ١ ثم ٢ و ٣.

جدول ٤.٢: مسجل الأعلام EFLAGS

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

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

القسم II.

إقلاع الحاسب Booting

٣. إقلاع الحاسب ومحمل النظام Bootloader

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

١.٣. إقلاع الحاسب

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

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

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

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

٢.٣. محمل النظام Bootloader

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

١. حجم البرنامج يجب أن يكون 512 بايت بالضبط.
٢. أن يتواجد على القطاع الأول في القرص : القطاع رقم 1 ، الرأس 0 ، المسار 0 ، وأن يحمل التوقيع المعروف.
٣. أن يحوي شفرة تحميل النواة ونقل التنفيذ اليها.
٤. أن يكون البرنامج object code خالي من أي إضافات (header,symbol table,...etc) وهو ما يعرف أيضا بـ Flat Binary .

الشرط الأول يُقيد وظيفة المحمل وقدرته على توفير خصائص متقدمة^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 وانتهاءً بتغيير نمط المعالج الى النمط المحمي والقفز الى النواة للمباشرة في تنفيذها . كل ذلك يحتاج الى

^٢مثل خاصية ال Safe Mode

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

• القطاع رقم 1

• المسار رقم 0

• الرأس رقم 0

وتحقيق هذا الشرط ليس بالأمر المعقد خصوصا مع توفر العديد من الادوات التي تساعد على نسخ مقطع من قرص ما الى مقطع في قرص آخر ، أما الشق الثاني من الشرط فهو متعلق بتمييز القطاع الاول كقطاع قابل للإقلاع من غيره ، حيث حتى يكون القطاع قابلا للإقلاع فانه يجب أن يحمل التوقيع 0xAA55 في البايت رقم 510 و 511 . وبدون هذا التوقيع فان البايوس (وتحديدا مقاطعة رقم 0x19) لن تعرف على هذا القطاع كقطاع قابل للإقلاع. أما الشرط الثالث فهو شرط اختياري وليس اجباري ، فمن الممكن أن تكون وظيفة المحمل هي عرض رسالة ترحيب فقط ! ولكن في أغلب الحالات الواقعية يجب أن تُحمل النواة وتنفذ عن طريق هذا المحمل. وقد أسلفنا وذكرنا أن تحميل نواة 32-bit يختلف عن تحميل نواة 16-bit ، حيث في الاولى يجب تجهيز الطريق أمام النواة وتفعيل بعض الخصائص لذلك وجب تقسيم مهمة محمل النظام الى مرحلتين - كما سنرى ذلك - ، أما في حالة كانت النواة 16-bit 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 قابلا للتنفيذ وليس معلومات أو هيدر عن الملف - كما في حالة الصيغ السابق ذكرها- . لذلك يجب أن تكون صيغة المحمل هي عبارة عن الصيغة الثنائية المقابلة للأوامر الموجودة فيه بدون أي اضافات أي Object Code او Flat Binary.

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

٣.٣. مخطط الذاكرة

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

٣.٤. برمجة محمل النظام

المثال ١.٣ يوضح أصغر محمل للنظام يمكن كتابته وتنفيذه ، باستخدام المجمع ^٣NASM وهو مجمع متعدد المنصات ويوفر ميزة إنتاج ملفات ثنائية object code .

Listing ٣١.: Smallest Bootloader

```
;Simple Bootloader do nothing.

bits 16          ; 16-bit real mode.

start:           ; label are pointer.
    cli          ; clear interrupt.
    hlt          ; halt the system.

times 510-($-$$) db 0    ; append zeros.

; $ is the address of first instruction (should be 0
; x07c00).
; $$ is the address of current line.
; $-$$ means how many byte between start and current.

; if cli and hlt take 4 byte then time directive will
; fill
; 510-4 = 506 zero's.

; finally the boot signature 0xaa55
db 0x55          ; first byte of a boot signature.
db 0xaa          ; second byte of a boot signature.
```

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

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

١.٤.٣. عرض رسالة ترحيبية

طالما ما زلنا نعمل في النمط الحقيقي فان ذلك يمكننا من استخدام مقاطعات البايوس ، وفي المثال ٢.٣ تم عرض رسالة باستخدام مقاطعة البايوس 0x10 int الدالة 0xe .

Listing ٣٢.: Hello World

```
;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.

; *****
; data
; *****

hello_msg    db    "Welcome to eqraOS, Coded by Ahmad Essam"
              ,0xa,0xd,0

; *****
; puts16: prints string using BIOS interrupt
; input:
```

```

;      es: pointer to data segment.
;      si: point to the string
; *****

puts16:

    lodsb      ; read character from ds:si to al ,and
                increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe   ; print character routine number.
    int 0x10     ; call BIOS.

    jmp puts16   ; continue prints until 0 is found.

end_puts16:

    ret

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

main:

    ;-----
    ; init registers
    ;-----

    ; because bootloader are loaded at 0x07c00 we can
    ; refrence this location with many different
    ; combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x:07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
    ; for offset.

    mov ax,0x07c0

```

```

mov ds,ax
mov es,ax

mov si,hello_msg
call puts16

cli      ; clear interrupt.
hlt      ; halt the system.

times 510-($-$$) db 0 ; append zeros.

; finally the boot signature 0xaa55
db 0x55
db 0xaa

```

النتيجة :

الشيء الملاحظ في المثال السابق هو أن مقطع الكود code segment ومقطع البيانات data segment متواجدان في نفس المكان على الذاكرة (داخل ال 512 بايت) لذلك يجب تعديل قيم مسجلات المقاطع للإشارة إلى المكان الصحيح. و بداية نذكر أن البايوس عندما ينقل التنفيذ إلى برنامج محمل النظام الذي قمنا بكتابته فإنه في حقيقة الأمر يقوم بعملية far jump والتي ينتج منها تصحيح قيم ال cs:ip لذلك لا داعي للقلق حول هذين المسجلين، لكن يجب تعديل قيم مسجلات المقاطع الأخرى مثل ds, es, ss, fs, gs. وكما نعلم أن العنوان الفيزيائي لمحمل النظام هو 0x07c00 يمكن الوصول إليه بأكثر من 4000 طريقة مختلفة، لكن سوف نقتصر على استخدام 0x0:0x07c0 أو 0x07c00:0x0 نظراً لأن هذه هي القيم الفعلية التي تستخدمها البايوس.

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

٣.٤.٢. معلومات قطاع الإقلاع

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 الإقلاع منه ، هذه المعلومات تحوي معرف OEM وتحوي بيانات BIOS Parameter Block (تختصر ب BPB) ويجب أن تبدأ كل هذه البيانات من البايت رقم 3^٤. وسوف يتم استخدام هذه البيانات بكثرة أثناء تطوير محمل النظام كذلك أحد فوائد هذه البيانات هو تعرف أنظمة التشغيل على نظام الملفات المستخدم في القرص.

Listing ٣٣.: Bios Parameter Block

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

bytes_per_sector dw      0x200          ; 512 byte per
sector.

sectors_per_cluster db      0x1          ; 1 sector per
cluster.

reserved_sectors dw      0x1            ; boot sector is
reserved.

total_fats        db      0x2            ; two fats.
root_directory    dw      0xe0           ; root dir has
224 entries.

total_sectors     dw      0xb40          ; 2880 sectors
in the volume.

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.
```

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

٤.٣. برمجة محمل النظام

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

المثال ٤.٣ يوضح شفرة المحمل بعد اضافة بيانات OEM and BPB.

Listing ٣٤: 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.

bytes_per_sector dw          0x200        ; 512 byte per
sector.
sectors_per_cluster db      0x1          ; 1 sector per
cluster.
reserved_sectors dw          0x1          ; boot sector is
reserved.
total_fats       db          0x2          ; two fats.
```

```

root_directory    dw      0xe0      ; root dir has
                  224 entries.
total_sectors     dw      0xb40     ; 2880 sectors
                  in the volume.
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 format the disk.
volume_label      db      "MOS FLOPPY " ; 11 byte.
system_id         db      "fat12  "  ; 8 byte.

; *****
; data
; *****

hello_msg db      "Welcome to eqraOS, Coded by Ahmad Essam"
          ,0xa,0xd,0

; *****
; puts16: prints string using BIOS interrupt
; input:
;     es: pointer to data segment.
;     si: point to the string
; *****

```

```
puts16:

    lodsb      ; read character from ds:si to al ,and
                increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe   ; print character routine number.
    int 0x10     ; call BIOS.

    jmp puts16   ; continue prints until 0 is found.

end_puts16:

    ret

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

main:

    ; _____
    ; intit registers
    ; _____

    ; because bootloader are loaded at 0x07c00 we can
    ; refrence this location with many different
    ; combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
    ; for offset.

    mov ax,0x07c0
    mov ds,ax
    mov es,ax

    mov si,hello_msg
```

```
call puts16

cli      ; clear interrupt.
hlt      ; halt the system.

times 510-($-$$) db 0 ; append zeros.

; finally the boot signature 0xaa55
db 0x55
db 0xaa
```

و المخرج ١.١ يوضح الشفرة السابقة في حالة عرضها بأي محرر سادس عشر Hex Editor حيث كما نلاحظ أن بيانات المحمل متداخلة مع الشفرة التنفيذية (تعليمات المعالج) لذلك يجب أن يتم القفز فوق هذه البيانات حتى لا تُنفذ كتعليمات خاطئة ، كذلك يجب التأكد من آخر بايتين وأنها تحمل التوقيع الصحيح.

Listing ٣٥.: 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.@...
00000018	12	00	02	00	00	00	00	00
00000020	00	00	00	00	00	00	29	00).
00000028	00	00	00	4D	4F	53	20	46	...MOS F
00000030	4C	4F	50	50	59	20	66	61	LOPPY fa
00000038	74	31	32	20	20	20	57	65	t12 We
00000040	6C	63	6F	6D	65	20	74	6F	lcome to
00000048	20	65	71	72	61	4F	53	2C	eqraOS,
00000050	20	43	6F	64	65	64	20	62	Coded b
00000058	79	20	41	68	6D	61	64	20	y Ahmad
00000060	45	73	73	61	6D	0A	0D	00	Essam...
00000068	AC	3C	00	74	07	B4	0E	CD	٢<.t.´.Í
00000070	10	E9	F4	FF	C3	B8	C0	07	.Ăéôÿ,À.
00000078	8E	D8	8E	C0	BE	3E	00	E8	.Ø.À¼>.è
00000080	E6	FF	FA	F4	00	00	00	00	æýúô....
00000088	00	00	00	00	00	00	00	00
									...
									...

```
000001F0  00 00 00 00 00 00 00 00  .....
000001F8  00 00 00 00 00 00 55 AA  .....Ua
```

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

Listing ٣٦.: Some Code

```
;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.

bytes_per_sector dw      0x200        ; 512 byte per
sector.
sectors_per_cluster db      0x1        ; 1 sector per
cluster.
reserved_sectors dw      0x1          ; boot sector is
reserved.
total_fats       db      0x2          ; two fats.
```

^٥ في حالة لم تتمكن من الوصول الى ملف المصدر source code.

```

root_directory    dw      0xe0      ; root dir has
                  224 entries.
total_sectors     dw      0xb40     ; 2880 sectors
                  in the volume.
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 format the disk.
volume_label      db      "MOS FLOPPY " ; 11 byte.
system_id         db      "fat12  "  ; 8 byte.

; *****
; data
; *****

hello_msg db      "Welcome to eqraOS, Coded by Ahmad Essam"
          ,0xa,0xd,0

; *****
; puts16: prints string using BIOS interrupt
; input:
;     es: pointer to data segment.
;     si: point to the string
; *****

```

```
puts16:

    lodsb      ; read character from ds:si to al ,and
                increment si if df=0.

    cmp al,0    ; check end of string ?
    je end_puts16 ; yes jump to end.

    mov ah,0xe   ; print character routine number.
    int 0x10     ; call BIOS.

    jmp puts16   ; continue prints until 0 is found.

end_puts16:

    ret

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

main:

    ; _____
    ; intit registers
    ; _____

    ; because bootloader are loaded at 0x07c00 we can
    ; refrence this location with many different
    ; combination
    ; of segment:offset addressing.

    ; So we will use either 0x0000:0x7c000 or 0x07c0:0x0000
    ; and in this example we use 0x07c0 for segment and 0x0
    ; for offset.

    mov ax,0x07c0
    mov ds,ax
    mov es,ax

    mov si,hello_msg
```



```

call puts16

cli      ; clear interrupt.
hlt      ; halt the system.

times 510-($-$$) db 0 ; append zeros.

; finally the boot signature 0xaa55
db 0x55
db 0xaa

```

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

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

إعادة القرص المرن

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

- المسجل `ah` : `0x0`.

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

النتيجة:

- المسجل `ah` : الحالة.

- `CF` : `0x1` إذا حدث خطأ ، `0x0` إذا تمت العملية بنجاح.

مثال:

Listing ٣٧.: Some Code

```
reset_floppy:

    mov ah,0x0    ; reset floppy routine number.
    mov dl,0x0    ; drive number

    int 0x13      ; call BIOS

    jc reset_floppy ; try again if error occur.
```

قراءة المقاطع sectors

أثناء العمل في النمط الحقيقي فإننا 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:

Listing ٣.٨.: Some Code

```
read_sectors:

reset_floppy:

    mov ah,0x0    ; reset floppy routine number.
    mov dl,0x0    ; drive number

    int 0x13      ; call BIOS

    jc reset_floppy ; try again if error occur.

    ; init buffer.
    mov ax,0x1000
    mov es,ax
    xor bx,bx

read:

    mov ah,0x2    ; routine number.
    mov al,1      ; how many sectors?
    mov ch,1      ; cylinder or track number.
    mov cl,2      ; sector number "fisrt sector is 1 not 0",
                    now we read the second sector.
    mov dh,0      ; head number "starting with 0".
    mov dl,0      ; drive number ,floppy drive always zero.

    int 0x13      ; call BIOS.
    jc read       ; if error, try again.

    jmp 0x1000:0x0 ; jump to execute the second sector.
```

٥.٣. مقدمة الى نظام FAT12

نظام الملفات هو برنامج يساعد في حفظ الملفات على القرص بحيث ينشئ لنا مفهوم الملف وخصائصه والعديد من البيانات المتعلقة به من تاريخ الانشاء والوقت ، كذلك يحتفظ بقائمة بجميع الملفات وأماكن تواجدها في القرص ، أيضاً أحد أهم فوائد أنظمة الملفات هي متابعة الأماكن الغير المستخدمة في القرص

والأماكن التي تضررت بسبب أو لآخر bad sectors ، كذلك أنظمة الملفات الجيدة تقوم بعمل تجميع الملفات المبعثرة على القرص Defragmentation حتى تستفيد من المساحات الصغيرة التي ظهرت بسبب حذف ملف موجود أو تخزين ملف ذو حجم أقل من المساحة الخالية. وبدون أنظمة الملفات فإن التعامل مع القرص سيكون مستحيلاً ! حيث لن نعرف ماهي المساحات الغير مستخدمة من الأخرى ولن نستطيع ان نقوم بقراءة ملف طلبه المستخدم لعرضه على الشاشة ! وبشكل عام فإن نظام الملفات يتكون من:

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

١.٥.٣ . قيود نظام FAT12

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

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

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

٣.٥.٢. هيكلية نظام FAT12 على القرص

عند تهيئة القرص المرن^٦ (Format) بنظام FAT12 فإن تركيبة القرص تكون على الشكل التالي:

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

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

• القيمة 0x00: تدل على أن الكلستر خالي.

• القيمة 0x01: تدل على أن الكلستر محجوز.

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

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

ويمكن النظر الى جدول FAT بأنه مصفوفة من القيم أعلاه ، وعندما نريد تحميل ملف فاننا سنأتي بعنوان أول كلستر له من جدول Root Directory (سنأتي عليها لاحقا) وبعدها نستخدم عنوان الكلستر ك index الى جدول FAT ونقرأ القيمة المقابلة للكلستر ، فاذا كانت القيمة بين 0x02 الى 0xfef فانها تدل على الكلستر التالي للملف ، ومن ثم سنستخدم هذه القيمة أيضا ك index ونقرأ القيمة الجديدة ، ونستمر على هذا الحال الى أن نقرأ قيمة تدل على نهاية الملف. هذا الجدول FAT يبدأ من المقطع المنطقي ١^٧ 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 بالإضافة الى شرح مبسط عن هيكلية القرص المرن وكيفية حفظه للبيانات . وبعد جدول FAT توجد نسخة أخرى من هذا الجدول وتستخدم كنسخة احتياطية backup وهي بنفس حجم وخصائص النسخة الاولى ، وبعدها يأتي دليل الجذر Root Directory وهو مصفوفة من 224 سجل كل سجل بطول 32 بايت ، وظيفية هذا الدليل هي حفظ أسماء الملفات الموجودة على القرص المرن بالإضافة الى العديد من المعلومات التي تخص وقت الانشاء والتعديل وحجم الملف وعنوان أول كلستر للملف ، عنوان الكلستر هو أهم معلومة لكي نستطيع تحميل الملف كاملا ، حيث كما ذكرنا أن هذا العنوان سيعمل ك index في جدول FAT وبعدها سنحدد ما اذا كانت توجد كلسترات أخرى يجب تحميلها أم أن الملف يتكون من كلستر واحد. والجدول التالي يوضح محتويات السجل الواحد في دليل ال root directory بدءاً من البايث الاول الى الاخير:

- 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.
 - البت 3: اسم القرص Volume Label.

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

- البت 4: الملف هو مجلد فرعي.
 - البت 5: أرشيف.
 - البت 6: جهاز.
 - البت 7: غير مستخدم.
 - البايت 12: غير مستخدم.
 - البايت 13: وقت الانشاء بوحدة MS.
 - البايتات 14-15: وقت الانشاء بالترتيب التالي:
 - البتات 0-4: الثواني (0-29).
 - البتات 5-10: الدقائق (0-59).
 - البتات 11-15: الساعات (0-23).
 - البايتات 16-17: سنة الانشاء بالترتيب التالي:
 - البتات 0-4: السنة (0=1980; 127=2107).
 - البتات 5-8: الشهر (1=يناير; 12=ديسمبر).
 - البتات 9-15: الساعة (0-23).
 - البايتات 18-19: تاريخ آخر استخدام (تتبع نفس الترتيب السابق).
 - البايتات 20-21: EA index.
 - البايتات 22-23: وقت آخر تعديل (تتبع نفس ترتيب البايتات 14-15).
 - البايتات 24-25: تاريخ آخر تعديل (تتبع نفس ترتيب البايتات 16-17).
 - البايتات 26-27: عنوان أول كلستر للملف.
 - البايتات 28-29: حجم الملف.
- ويجب ملاحظة أن حجم السجلات هو ثابت Fixed Length Record فمثلا اسم الملف يجب ان يكون بطول 8 بايت وفي حالة زاد على ذلك فان هذا سوف يحدث ضرراً على هذا الدليل ، أيضا في حالة كان الاسم بحجم أقل من المطلوب فانه يجب تكلمة العدد الناقص من الحروف بحرف المسافة Space.

٣.٥.٣. هيكلية القرص المرن

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

٣.٥.٤. القراءة و الكتابة من نظام FAT12

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

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

إنشاء المرحلة الثانية من محمل النظام

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

Listing ٣٩.: Some Code

```
; Second Stage Bootloader.
; loaded by stage1.bin at address 0x050:0x0 (0x00500).

bits 16      ; 16-bit real mode.
org 0x0      ; offset to zero.

start:  jmp stage2

; data and variable
hello_msg db "Welcome to eqraOS Stage2",0xa,0xd,0

; include files:
%include "stdio.inc"      ; standard i/o routines.

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

stage2:

    push cs
    pop ds      ; ds = cs.
```

```
mov si,hello_msg
call puts16

cli      ; clear interrupt.
hlt      ; halt the system.
```

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

تحميل ال Root Directory الى الذاكرة

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

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

Listing ٣١٠.: Some Code

```
;-----
; 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 إذا كان الملف غير موجود فنصدر رسالة خطأ ونوقف النظام عن العمل. والشفرة التالية توضح ذلك.

Listing ٣١١.: Some Code

```

;-----
; 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 الجدول بها . والشفرة التالية توضح ذلك.

Listing ٣١٢.: Some Code

```

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

Listing ٣١٣.: Some Code

```

; *****
; cluster_to_lba: convert cluster number to LBA
;   input:
;       ax: Cluster number.
;   output:
;       ax: lba number.
; *****
cluster_to_lba:

    ; lba = (cluster - 2)* sectors_per_cluster
    ; the first cluster is always 2.

    sub ax,2

```

```

xor cx,cx
mov cl, byte[sectors_per_cluster]
mul cx

add ax,word[data-region]    ; cluster start from data
                             area.
ret

```

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

Listing ٣١٤.: Some Code

```

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

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

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

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

```

ret

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

Listing ٣١٥.: Some Code

```

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

xor bx,bx
mov ax,0x0050
mov es,ax

load_cluster:

mov ax,word[cluster-number] ; ax = cluster number
call cluster_to_lba ; convert cluster number to
                    LBA addressing.

xor cx,cx
mov cl,byte[sectors_per_cluster] ; cx = 1 sector

call read_sectors_bios ; load cluster.

```

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

Listing ٣١٦.: Some Code

```

; *****
; read_sectors_bios: load sector from floppy disk
; input:
;     es:bx : Buffer to load sector.
;     ax:   first sector number ,LBA.
;     cx:   number of sectors.
; *****
read_sectors_bios:

```

```
begin:
    mov di,5      ; try 5 times to load any sector.

load_sector:

    push ax
    push bx
    push cx

    call lba_to_chs

    mov ah,0x2      ; load sector routine number.
    mov al,0x1      ; 1 sector to read.
    mov ch,byte[absolute_track] ; absolute track number.
    mov cl,byte[absolute_sector] ; absolute sector number.
    mov dh,byte[absolute_head] ; absolute head number.
    mov dl,byte[drive_number] ; floppy drive number.

    int 0x13      ; call BIOS.

    jnc continue ; if no error jmp.

    ; reset the floppy and try read again.

    mov ah,0x0      ; reset routine number.
    mov dl,0x0      ; floppy drive number.
    int 0x13      ; call BIOS.

    pop cx
    pop bx
    pop ax

    dec di
    jne load_sector

    ; error.
    int 0x18

continue:

    mov si,progress_msg
    call puts16
```



```

pop cx
pop bx
pop ax

add ax,1 ; next sector
add bx,word[bytes_per_sector] ; point to next empty
    block in buffer.

loop begin ; cx time

ret

```

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

وباختصار، لقراءة القيمة المقابلة لرقم كلستر ما فيجب أولاً تطبيق القانون :

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

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

Listing ٣١٧.: Some Code

```

read_cluster_fat_entry:

    mov ax,word[cluster_number]

; Every FAT entry are 12-bit long( byte and half one) .
; so we must map the cluster number to this entry.
; to read cluster 0 we need to read fat[0].
; cluster 1 -> fat[1].

```

```

; cluster 2 -> fat[3],...etc.

mov cx,ax    ; cx = cluster number.
shr cx,1     ; divide cx by 2.
add cx,ax    ; cx = ax + (ax/2).
mov di,cx
add di,0x0200
mov dx,word[di] ; read 16-bit form FAT.

; Now, because FAT entry are 12-bit long, we should
; remove 4 bits.
; if the cluster number are even, we must mask the last
; four bits.
; if it odd, we must do four right shift.

test ax,1
jne odd_cluster

even_cluster:

    and dx,0x0fff
    jmp next_cluster

odd_cluster:

    shr dx,4

next_cluster:
    mov word[cluster_number],dx    ; next cluster to load.

    cmp dx,0x0fff0                ; check end of file, last cluster?
    jnb load_cluster              ; no, load the next cluster.

; yes jmp to end
jmp end_of_first_stage

find_fail:

    mov si, fail_msg

```

```
    call puts16

    mov ah,0x0
    int 0x16      ; wait keypress.
    int 0x19      ; warm boot.

end_of_first_stage:

    ; jump to stage2 and begin execute.
    push 0x050    ; segment number.
    push 0x0      ; offset number.

    retf          ; cs:ip = 0x050:0x0

times 510-($-$$) db 0 ; append zeros.

; finally the boot signature 0xaa55
db 0x55
db 0xaa
```

٤. برمجة محمل النظام – المرحلة الثانية

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

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

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.

١.١.٤. جدول الوصفات العام Global Descriptor Table

عند الانتقال الى النمط المحمي PMode فان أي عملية وصول الى الذاكرة تتم عن طريق هذا الجدول GDT ، هذا الجدول يعمل على حماية الذاكرة وذلك بفحص العنوان المراد الوصول اليه والتأكد من عدم مخالفته لبيانات هذا الجدول. هذه البيانات تحدد القسم الذي يمكن أن ينفذ كشفرة (Code) والقسم الذي لا ينفذ (Data) كذلك تحدد هذه البيانات العديد من الخصائص كما سنراها الان.

وعادة يتكون جدول GDT من ثلاث واصفات Descriptors (حجم كل منها هو 64 بت) وهم:

- Null Descriptor: تكون فارغة في العادة.
 - Code Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي ينفذ كشفرة Code.
 - Data Descriptor: تصف خصائص المقطع أو القسم من الذاكرة الذي لا ينفذ ويحوي بيانات Data.
- بيانات أي واصفة Descriptor تأخذ الجدول التالي:
- البتات 0-15: تحوي أول بايتين (من بت 0 -15) من حجم المقطع.
 - البتات 16-39: تحوي أول ثلاث بايتات من عنوان بداية المقطع Base Address.
 - البت 40: بت الوصول Access Bit (يستخدم مع الذاكرة الظاهرية Virtual Memory).
 - البتات 41-43: نوع الواصفة Descriptor Type:
 - البت 41: القراءة والكتابة:
 - * Data Descriptor: القيمة 0 للقراءة فقط والقيمة 1 للقراءة والكتابة.
 - * Code Descriptor: القيمة 0 للتنفيذ فقط execute والقيمة 1 للقراءة والتنفيذ.
 - البت 42: Expansion direction (Data segments), conforming (Code Segments).
 - البت 43: قابلية التنفيذ:
 - * 0: اذا كان المقطع عبارة عن بيانات.
 - * 1: اذا كان المقطع عبارة عن شفرة.
 - البت 44: Descriptor Bit:
 - System descriptor: 0
 - Code or Data Descriptor: 1
 - البتات 45-46: مستوى الحماية Privilege Level:
 - 0: Highest (Ring 0).

– 3: Lowest (Ring 3).

• البت 47: Segment is in memory (Used with Virtual Memory).

• البتات 48-51: تحوي البت 16-19 من حجم المقطع.

• البت 52: محجوزة.

• البت 53: محجوزة.

• البت 54: 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 وللبينات Code and Data Descriptor، بحيث يمكن القراءة و الكتابة من أول بايت في الذاكرة الى آخر الذاكرة 0xffffffff.

Listing ٤١: Some Code

```

;*****
; 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.

; Data Descriptor: start at 0x10.

dw 0xffff     ; limit low.
dw 0x0        ; base low.
db 0x0        ; base middle.
db 10010010b ; access byte.
db 11001111b ; granularity byte.
db 0x0        ; base high.

end_of_gdt:

```

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

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

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

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

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

- البت 7: تستخدم مع الذاكرة الظاهرية لذا تم اهمالها.
- 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 0xffff أي 1 ميغا من الذاكرة ، ولاحقاً عندما يتم تفعيل بوابة A20 ستمكن من الوصول حتى 4 جيجا من الذاكرة.
- البتات 4-5: محجوزة للنظام لذا تم اهمالها.
- البت 6: تم اختيار القيمة 1 دلالة على هذا المقطع هو 32 بت.
- البت 7: باختيار القيمة 1 سيتم إحاطة المقطع ب 4 KB.
- 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 تماماً لوصفة الكود باستثناء البت رقم 43 حيث يحدد ما اذا كان المقطع كود أم بيانات.
- وبعد إنشاء هذا الجدول (GDT) في الذاكرة ، يجب أن يحمل المسجل **gdtr** على حجم هذا الجدول ناقصاً واحد وعلى عنوان بداية الجدول، ويتم ذلك عن طريق إنشاء مؤشر الى جدول **GDT** ومن ثم استخدام الامر **lgdt** (وهو أمر يعمل فقط في الحلقة صفر **Ring0**) ، والشفرة التالية توضح ذلك.

Listing ٤٢.: Some Code

```
bits 16      ; real mode.

;*****
; load_gdt: Load GDT into GDTR.
;*****

load_gdt:

    cli          ; clear interrupt.
    pusha        ; save registers
    lgdt [gdt_ptr] ; load gdt into gdtr
    sti          ; enable interrupt
```



```

    popa                ; restore registers.

    ret

;*****
; gdt_ptr: data structure used by gdt
;*****

gdt_ptr:

    dw end_of_gdt - begin_of_gdt - 1    ; size -1
    dd begin_of_gdt                     ; base of gdt

```

٢.١.٤. العنوان في النمط المحمي PMode Memory Addressing

في النمط الحقيقي يستخدم المعالج عنوان Segment:Offset وذلك بأن تكون أي من مسجلات المقاطع (Segments Registers) تحوي عنوان بداية المقطع ، ومسجلات العناوين تحوي العنوان داخل مقطع ما ، ويتم ضرب عنوان المقطع بالعدد 0x10 وجمع ال offset اليه للحصول على العنوان النهائي والذي سيمر بداخل مسار العنوان Address Bus.

أما النمط المحمي PMode فانه يستخدم عنوان Descriptor:Offset وذلك بأن تكون مسجلات المقاطع تحوي عنوان أحد الواصفات التي قمنا ببنائها (مثلا مسجل CS يحوي العنوان 0x8 ومسجل البيانات DS يحوي العنوان 0x10) ، وال offset سيتم جمعها الى عنوان بداية المقطع Base Address والذي قمنا بتحديدده في جدول الواصفات كذلك سيتم التأكد من أن هذا العنوان لا يتجاوز حجم المقطع Segment Limit أيضا سيتم التأكد من مستوى الصلاحية وأنه يمكن الوصول للعنوان المطلوب. ونظراً لأن في النمط المحمي يمكن استخدام مسجلات 32-bit فانه يمكن عنوانة 4 جيجا من الذاكرة^١.

٣.١.٤. الانتقال الى النمط المحمي

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

^١ يفرض أن بوابة A20 تم تفعيلها.

١.٤ . الانتقال الى النمط المحمي

وبعد الانتقال الى النمط المحمي فان يجب تعيين الوصفة 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.

Listing ٤٣.: Some Code

```
;  
; Load gdt into gtr.  
;  
  
    call load_gdt  
  
;  
; Go to PMode.  
;  
    ; just set bit 0 from cr0 (Control Register 0).  
  
    cli          ; important.  
    mov eax, cr0  
    or  eax, 0x1  
    mov cr0, eax    ; entering pmode.  
  
;  
; Fix CS value  
;  
    ; select the code descriptor  
    jmp 0x8:stage3  
  
;*****  
; entry point of stage3  
;*****  
  
bits 32          ; code now 32-bit  
  
stage3:  
  
    ;  
    ; Set Registers.  
    ;
```

```

mov ax,0x10      ; address of data descriptor.
mov ds,ax
mov ss,ax
mov es,ax
mov esp,0x90000  ; stack begin from 0x90000.

;-----;
; Hlat the system.
;-----;
cli      ; clear interrupt.
hlt      ; halt the system.

```

٢.٤. 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 A31-A20 واغلاقها.

هذه البوابة مرتبطة مع متحكم 8042 وهو متحكم لوحة المفاتيح (Keyboard Controller)، وعند تفعيل البت رقم 1 في منفذ خروج البيانات (output data port) التابع لمتحكم لوحة المفاتيح فان هذا يفتح بوابة A20 وبهذا نستطيع الوصول الى 4 جيجا من الذاكرة، ابتداءً من العنوان 0x0-0xffffffff وعند اقلاع الحاسب فان البايوس يقوم بتفعيل هذه البوابة لأغراض حساب حجم الذاكرة واختبارها ومن ثم يقوم بغلقها مجدداً للحفاظ على التوافقية مع الاجهزة القديمة. وتوجد العديد من الطرق لتفعيل هذه البوابة، العديد منها يعمل على أجهزة معينة لذلك سيتم ذكر العديد من الطرق واستخدام أكثر الطرق محمولية على صعيد الاجهزة المختلفة.

١.٢.٤. متحكم لوحة المفاتيح 8042 والبوابة A20

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

^٢توجد البوابة تحديداً على خط العناوين رقم 20

متحكم لوحة المفاتيح فانه لا بد من التعامل مع هذا المتحكم لتفعيل البوابة ، وهذا يتم عن طريق استخدام أوامر المعالج in والامر out.

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

وفي الفصل السادس سيتم مناقشة متحكم لوحة المفاتيح بالتفصيل ، وسنكتفي هنا فقط بتوضيح الأجزاء المتعلقة بتفعيل بوابة A20.

٢.٢.٤ طرق تفعيل البوابة A20

بواسطة System Control Port 0x92

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

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

والمثال التالي يوضح طريقة تفعيل البوابة .

Listing ٤.٤: Some Code

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

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

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

بواسطة البايوس

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

Listing ٤.٥: Some Code

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

بواسطة متحكم لوحة المفاتيح

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

• البت 0: حالة ال Output Buffer:

- القيمة 0: ال Output Buffer خالي (لا توجد نتيجة ، لا تقرأ الان).
- القيمة 1: ال Output Buffer ممتلئ (توجد نتيجة ، قم بالقراءة الان).

• البت 1: حالة ال Input Buffer:

- القيمة 0: ال Input Buffer خالي (لا توجد أوامر غير منفذة ، يمكن الكتابة الان).
- القيمة 1: ال Input Buffer ممتلئ (توجد أوامر غير منفذة ، لا تكتب الان).

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

Listing ٤٦.: Some Code

```

;*****
; wait_output: wait output buffer to be full.
;*****

wait_output:

    in al,0x64      ; read status
    test al,0x1     ; is output buffer is empty?
    je wait_output  ; yes, hang.

    ret             ; no,there is a result.

```

```

;*****
; wait_input: wait input buffer to be empty.
               command executed already.
;*****

wait_input:

    in al,0x64    ; read status
    test al,0x2   ; is input buffer is full?
    jne wait_input ; yes, hang.

    ret          ; no,command executed.

```

ولإرسال أوامر الى 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 تمناها 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.

Listing ٧٠.: Some Code

```

;*****
; enable_a20_keyboard_controller:
;   Enable A20 with command 0xdd
;*****

```

٢.٤ . تفعيل البوابة A20

```
enable_a20_keyboard_controller:
```

```
    ;cli
    push ax      ; save register.

    mov al,0xdd  ; Enable A20 Keyboard Controller Command.
    out 0x64,al

    pop ax      ; restore register.
    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.

Listing ٤٨: Some Code

```

;*****
; enable_a20_keyboard_controller_output_port:
;   Enable A20 with write to keyboard output port.
;*****

enable_a20_keyboard_controller_output_port:

    cli
    pusha    ; save all registers

    call wait_input    ; wait last operation to be finished.

    ;-----
    ; Disable Keyboard
    ;-----
    mov al,0xad    ; disable keyboard command.
    out 0x64,al
    call wait_input

    ;-----
    ; send read output port command
    ;-----
    mov al,0xd0    ; read output port command
    out 0x64,al
    call wait_output    ; wait output to come.
    ; we don't need to wait_input because when output came
    ; we know that operation are executed.

    ;-----
    ; read input buffer
    ;-----
    in al,0x60
    push eax    ; save data.
    call wait_input

    ;-----
    ; send write output port command.
    ;-----
    mov al,0xd1    ; write output port command.

```

```
out 0x64,al
call wait_input

;-----
; enable a20.
;-----
pop eax
or al,2 ; set bit 2.
out 0x60,al
call wait_input

;-----
; Enable Keyboard.
;-----
mov al,0xae ; Enable Keyboard command.
out 0x64,al
call wait_input

popa ; restore registers
sti

ret
```

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

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 نظراً لانه تم تطويره كشريحة واحدة single chip حيث استبدلت العديد من الشرائح والتي كانت تستخدم في مقاييس اخرى مثل MDA و CGA و EGA

، ويتكون ال VGA من Video Buffer , Video DAC , CRT Controller , Sequencer unit , Graphics Controller و Attribute Controller^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 بدءاً من العنوان 0xa0000 كذاكرة للشاشة وفي حالة تم الكتابة الى هذه الذاكرة فان هذا سوف يغير في الشاشة ، هذا الربط يسمى Memory Mapping ، أما ال Graphics Controller فهو الذي يقوم بتحديث محتويات الشاشة بناءً على البيانات الموجودة في ال Video buffer .

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

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

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

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

٤.٣.١. عنوان الذاكرة في متحكمات VGA

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

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

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

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

Listing ٤٩: Some Code

```

#define VIDEO_MEMORY  0xb8000    ; Base Address of Mapped
    Video Memory.
#define CHAR_ATTRIBUTE  0x7        ; White chracter on black
    background.

mov edi, VIDEO_MEMORY

mov [edi], 'A'                ; print A
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 تتمكن من طباعة العديد من الحروف فانه يجب انشاء متغيران (x,y) لحفظ المكان الحالي للصف والعمود ومن ثم تحويل هذا المكان الى عنوان في ال Video Memory. وفي البداية ستكون قيم (x,y) هي (0,0) أي ان الحرف سيكون في الجزء الاعلي من اليسار في الشاشة ويجب ارسال هذا الحرف الى عنوان بداية 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.

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

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

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

• البتات 0-2: لون الحرف:

- البت 0: أحمر.

- البت 1: أخضر.

- البت 2: أزرق.

• البت 3: كثافة لون الحرف (0 غامق ، 1 فاتح).

• البت 4-6: لون خلفية الحرف:

- البت 0: أحمر.

- البت 1: أخضر.

- البت 2: أزرق.

• البت 7: كثافة لون خلفية الحرف (0 غامق ، 1 فاتح).

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

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

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

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

Listing ٤.١٠: Some Code

```
; *****
; putch32: print character in protected mode.
;   input:
;       bl: character to print.
; *****

bits    32

%define VIDEO_MEMORY    0xb8000    ; Base Address of Mapped
    Video Memory.
%define COLUMNS        80          ; text mode (mode 7) has 80
    columns,
%define ROWS            25          ; and 25 rows.
%define CHAR_ATTRIBUTE  31          ; white on blue.

x_pos    db    0          ; current x position.
y_pos    db    0          ; current y position.

putch32:

    pusha                ; Save Registers.

    ;-----
    ; Check if bl is new line ?
```

```
;-----  
  
    cmp bl,0xa      ; if character is newline ?  
    je new_row      ; yes, jmp at end.  
  
;-----  
; Calculate the memory offset  
;-----  
; because in text mode every character take 2 bytes: one  
;   for the character and one for the attribute, we must  
;   calculate the memory offset with the following  
;   formula:  
; offset = x_pos*2 + y_pos*COLUMNS*2  
  
xor eax,eax  
  
mov al,2  
mul byte[x_pos]  
push eax          ; save the first section of formula.  
  
xor eax,eax  
xor ecx,ecx  
  
mov ax,COLUMNS*2    ; 80*2  
mov cl,byte[y_pos]  
mul ecx  
  
pop ecx  
add eax,ecx  
  
add eax,VIDEO_MEMORY ; eax = address to print the  
                      ; character.  
  
;-----  
; Print the chracter.  
;-----  
  
mov edi,eax  
  
mov byte[edi],bl      ; print the character,  
mov byte[edi+1],CHAR_ATTRIBUTE ; with respect to the  
                      ; attribute.
```

```
;  
; Update the postions.  
;  
  
inc byte[x-pos]  
cmp byte[x-pos],COLUMNS  
je new_row  
  
jmp putch32_end  
  
new_row:  
  
mov byte[x-pos],0      ; clear the x-pos.  
inc byte[y-pos]        ; increment the y-pos.  
  
putch32_end:  
  
popa      ; Restore Registers.  
  
ret
```

وتبدأ هذه الدالة بفحص الحرف المراد طباعته (موجود في المسجل b1) مع حرف الانتقال الى السطر الجديد 0xa وفي حالة التساوي يتم نقل التنفيذ الى آخر جسم الدالة والذي يقوم بتصفير قيمة x وزيادة قيمة y دلالة على الانتقال الى السطر الجديد. أما في حالة كان الحرف هو أي حرف آخر فانه يجب حساب العنوان الذي يجب ارسال الحرف اليه حتى يمكن طباعته ، وكما ذكرنا أن النمط النصي Mode 7 يستخدم بايتين لكل حرف لذا سيتم استخدام العلاقة التالية للتحويل ما بين (x,y) الى العنوان المطلوب.

$$\text{videomemory} = 0xb0000$$
$$\text{videomemory} += x * 2 + y * 80 * 2$$

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

Listing ٤١١.: Some Code

```

; *****
; puts32: print string in protected mode.
;   input:
;       ebx: point to the string
; *****

bits    32

puts32:

    pusha        ; Save Registers.

    mov edi,ebx

@loop:
    mov bl,byte[edi] ; read character.

    cmp bl,0x0      ; end of string ?
    je  puts32_end  ; yes, jmp to end.

    call putch32     ; print the character.

    inc edi          ; point to the next character.

    jmp @loop

puts32_end:

;-----
; Update the Hardware Cursor.
;-----
; After print the string update the hardware cursor.

```

```

mov bl,byte[x-pos]
mov bh,byte[y-pos]

call move_cursor

popa      ; Restore Registers.

ret

```

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

٤.٣.٤ . تحديث المؤشر Hardware Cursor

عند طباعة حرف أو سلسلة نصية فإن مؤشر الكتابة لا يتحرك من مكانه الا عند تحديده يدوياً ، وهذا يتم عن طريق التعامل مع متحكم CRT Controller . هذا المتحكم يحوي العديد من المسجلات ولكننا سوف نركز على مسجل البيانات Data Register ومسجل نوع البيانات Index Register . ولإرسال بيانات الى هذا المتحكم ، فيجب أولاً تحديد نوع البيانات وذلك بإرسالها الى مسجل Index Register ومن ثم إرسال البيانات الى مسجل البيانات Data Register ، وفي حواسيب x86 فإن مسجل البيانات يأخذ العنوان 0x3d5 ومسجل Index Register يأخذ العنوان 0x3d4 . والجدول التالي يوضح القيم التي يمكن إرسالها الى مسجل نوع البيانات Index Register .

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

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

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

$$videomemory = x + y * 80$$

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

Listing ٤١٢.: Some Code

```
; *****
; move_cursor: Move the Hardware Cursor.
;   input:
;       bl: x pos.
;       bh: y pos.
```

```
; *****

bits 32

move_cursor:

    pusha        ; Save Registers.

;-----
; Calculate the offset.
;-----
    ; offset = x-pos + y-pos*COLUMNS

    xor ecx,ecx
    mov cl,byte[x-pos]

    mov eax,COLUMNS
    mul byte[y-pos]

    add eax,ecx
    mov ebx,eax

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

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

    mov al,b1
    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

popa      ; Restore Registers.

ret

```

٥.٣.٤. تنظيف الشاشة Clear Screen

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

Listing ٤١٣.: Some Code

```

; *****
; clear_screen: Clear Screen in protected mode.
; *****

bits 32

clear_screen:

    pusha      ; Save Registers.
    cld

    mov edi,VIDEO_MEMORY      ; base address of video memory.
    mov cx,2000      ; 25*80
    mov ah,CHAR_ATTRIBUTE      ; 31 = white character on blue
                                background.
    mov al,' '

    rep stosw

    mov byte[x_pos],0
    mov byte[y_pos],0

```

```
popa      ; Restore Registers.
```

```
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 ، وابتداءً من الفصل القادم سنترك لغة التجميع جانباً ونبدأ العمل بلغة السي والسي++ .

وبما أننا نعمل في النمط المحمي PMode فلا يمكننا أن نستخدم مقاطعة البايوس 0x13 int لتحميل مقاطعات النواة الى الذاكرة ، ويجب أن نقوم بكتابة درايفر لمحرك القرص المرن أو نقوم بتحميل النواة الى الذاكرة قبل الانتقال الى النمط المحمي وهذا ما سنفعله الان ، وسنترك جزئية برمجة محرك القرص المرن لاحقاً.

وحيث أن النمط المحمي يسمح باستخدام ذاكرة حتى 4 جيجا ، فإن النواة سنقوم بتحميلها على العنوان 0x100000 أي عند 1 ميغا من الذاكرة . لكن علينا التذكر بأن النمط الحقيقي لا يدعم الوصول الى العنوان 0x100000 لذلك سنقوم بتحميل النواة أولاً في أي عنوان خالي وليكن 0x3000 وعند الانتقال الى النمط المحمي سنقوم بنسخها الى العنوان 0x100000 ونقل التنفيذ والتحكم اليها. والشفرة التالية توضح نواة ترحيبية.

Listing ٤.١: Some Code

```
org      0x100000      ; kernel will load at 1 MB.

bits    32              ; PMode.

jmp kernel_entry

#include "stdio.inc"
```

^٤ الفصل التالي سيتناول موضوع النواة وكيفية برمجتها بالتفصيل.

```
kernel_message db 0xa,0xa,0xa,"                      eqraOS
v0.1 Copyright (C) 2010 Ahmad Essam"
db 0xa,0xa,"                      University of Khartoum
- Faculty of Mathematical Sceinces.",0
```

```
logo_message db 0xa,0xa,0xa,"                      --- ---
----- - / -- \ / --/"                      / --) - ` / --/ -
db 0xa,"                      \ / / / - / / - \ \ "
db 0xa,"                      \ -- / \ -, / - / \ -, - /
db 0xa,"                      \ ---- // ---- / "
db 0xa,"                      / - /
",0
```

```
;*****
; Entry point.
;*****
```

kernel_entry:

```
;-----
; Set Registers
;-----

mov ax,0x10      ; data selector.
mov ds,ax
mov es,ax
mov ss,ax
mov esp,0x90000  ; set stack.

;-----
; Clear Screen and print message.
;-----

call clear_screen

mov ebx,kernel_message
call puts32

mov ebx,logo_message
call puts32
```

```

;-----
; Halt the system.
;-----

cli
hlt

```

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

Listing ٤.١٥.: Some Code

```

bits 16 ; 16-bit real mode.
org 0x500

start: jmp stage2

;*****
; include files:
;*****
#include "stdio.inc" ; standard I/O routines.
#include "gdt.inc" ; GDT load routine.
#include "a20.inc" ; Enable A20 routines.

```

^٥راجع الفصل السابق لمعرفة التفاصيل.

^٦جميع شفرات الملفات مرفقة مع البحث في مجلد /example/ch3/boot/ وشفرة المحمل النهائية ستكون ملحقه في نهاية البحث.


```
%include "fat12.inc"      ; FAT12 driver.
%include "common.inc"     ; common declarations.

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

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

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

stage2:

;-----
; Set Registers.
;-----

cli

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

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

sti

;-----
; Load gdt into gdtr.
;-----

call load_gdt
```

```

;-----
; Enable A20.
;-----
    call enable_a20_keyboard_controller_output_port

;-----
; Display Message.
;-----
    mov si,hello_msg
    call puts16

;-----
; Load Root Directory
;-----
    call load_root

;-----
; Load Kernel
;-----
    xor ebx,ebx
    mov bp,KERNEL_RMODE_BASE    ; bx:bp buffer to load
                                kernel

    mov si,kernel_name
    call load_file

    mov dword[kernel_size],ecx
    cmp ax,0
    je enter_stage3

    mov si,fail_message
    call puts16

    mov ah,0
    int 0x16    ; wait any key.
    int 0x19    ; warm boot.
    cli        ; cannot go here!
    hlt

;-----
; Go to PMode.

```

```
;-----  
  
enter_stage3:  
  
    ; just set bit 0 from cr0 (Control Register 0).  
  
    cli                ; important.  
    mov eax,cr0  
    or  eax,0x1  
    mov cr0,eax        ; entering pmode.  
  
;-----  
; Fix CS value  
;-----  
    ; select the code descriptor  
    jmp CODE_DESCRIPTOR:stage3  
  
;*****  
; entry point of stage3  
;*****  
  
bits 32        ; code now 32-bit  
  
stage3:  
  
;-----;  
; Set Registers.  
;-----;  
  
    mov ax,DATA_DESCRIPTOR    ; address of data  
                                descriptor.  
    mov ds,ax  
    mov ss,ax  
    mov es,ax  
    mov esp,0x90000    ; stack begin from 0x90000.  
  
;-----  
; Clear Screen and print message.  
;-----;
```

```
    call clear_screen

    mov ebx, stage2_message
    call puts32

    mov ebx, logo_message
    call puts32

;-----
; Copy Kernel at 1 MB.
;-----
    mov eax, dword[kernel_size]
    movzx ebx, word[bytes_per_sector]
    mul ebx
    mov ebx, 4
    div ebx

    cld

    mov esi, KERNEL_RMODE_BASE
    mov edi, KERNEL_PMODE_BASE
    mov ecx, eax
    rep movsd

;-----
; Execute the kernel.
;-----
    jmp CODE_DESCRIPTOR:KERNEL_PMODE_BASE

;-----;
; Hlat the system.
;-----;
    cli      ; clear interrupt.
    hlt      ; halt the system.
```

النتيجة:

شكل ١.٤: محمل النظام أثناء العمل

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

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

Bochs VBE Display Adapter enabled

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

Press F12 for boot menu.

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

شكل ٢.٤: بدء تنفيذ النواة

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

eagraOS
```

القسم III.

النواة Kernel

٥. مقدمة حول نواة نظام التشغيل

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

١.٥. نواة نظام التشغيل

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

١.١.٥. مستويات التجريد

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

المستوى الأول: مستوى العتاد

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

المستوى الثاني: مستوى برامج العتاد Firmware

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

المستوى الثالث: مستوى النواة (الحلقة صفر)

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

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

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

المستوى الخامس: مستوى برامج المستخدم (الحلقة ٣)

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

٢.٥. وظائف نواة النظام

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

١.٢.٥. إدارة الذاكرة

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

^١المقصود أنها برامج تعمل في الحلقة ٣.

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

٢.٢.٥. إدارة العمليات

٣.٢.٥. نظام الملفات

٣.٥. هيكلية وتصميم النواة

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

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

١.٣.٥. النواة الضخمة Monolithic Kernel

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

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

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

٢.٣.٥ . النواة المصغرة MicroKernel

الأنوية MicroKernel هي الأكثر ثباتا واستقرارا ومرونة والأسهل في الصيانة والتعديل والتطوير وذلك نظرا لان النواة تكون أصغر ما يمكن ، حيث أن الوظائف الأساسية فقط تكون ضمن النواة وهي ادارة الذاكرة وادارة العمليات (مجدول العمليات، أساسيات IPC)، أما بقية برامج النظام مثل نظام الملفات ومشغلات الأجهزة وغيرها تتبع لبرامج المستخدم وتعمل في نمط المستخدم (الحلقة ٣) ، وهذا يعني في حالة حدوث خطأ في هذه البرامج فان النظام لن يتأثر كذلك يمكن تغيير هذه البرامج (مثلا تغيير نظام الملفات) دون الحاجة الى اعادة تشغيل الجهاز حيث أن برامج النظام تعمل كبرامج المستخدم . والشكل التالي يوضح مخطط هذه الأنوية.

المشكلة الرئيسية لهذا التصميم هو بطيء عمل النظام وذلك بسبب أن برامج النظام عليها أن تتخاطب مع بعضها البعض عن طريق تمرير الرسائل (Message Passing) أو مشاركة جزء من الذاكرة (Shared Memory) وهذا ما يعرف ب Interprocess Communication . وأشهر مثال لنظام تشغيل يتبع هذا التصميم هو نظام مينكس الاصدار الثالث.

٣.٣.٥ . النواة الهجينة Hybrid Kernel

هذا التصميم للنواة ما هو إلا مزيج من التصميمين السابقين ، حيث تكون النواة MicroKernel لكنها تطبق ك Monolithic Kernel ، ويسمى هذا التصميم Hybrid Kernel أو Modified MicroKernel . والشكل التالي يوضح مخطط لهذا التصميم.

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 Assembly) حيث أن النواة كثيرا ما يجب عليها التعامل المباشر مع أوامر لغة التجميع (مثلا عند تحميل جدول الواصفات العام وجدول المقاطعات وكذلك عند غلق المقاطعات وتفعيلها وغيرها).

الشيء الآخر الذي يجب وضعه في الحسبان هو أنه لا يمكن استخدام لغة 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 كلياً على نظام التشغيل، أيضاً دوال الادخال والاخراج تعتمد كلياً على النظام).

لذلك غالباً تستخدم لغة السي والسي++ لبرمجة أنوية أنظمة التشغيل نظراً لما تتمتع به اللغتين من ميزات فريدة تميزها عن باقي اللغات ، وتنتشر لغة السي بشكل أكبر لاسباب كثيرة منها هو أنها لا تحتاج الى مكتبة وقت التشغيل (RunTime Library) حتى تعمل البرامج المكتوبة بها على عكس لغة سي++ والتي تحتاج الى (RunTime Library) لدعم الكثير من الخصائص مثل الاستثناءات و دوال البناء والهدم.

وفي حالة استخدام لغة سي أو سي++ فإنه يجب إعادة تطوير اجزاء من مكتبة السي والسي++ القياسية (Standard C/C++ Library) وهي الأجزاء التي تعتمد على نظام التشغيل مثل دوال printf و scanf و دوال حجز الذاكرة malloc/new وتحريرها free/delete.

ونظراً لاننا بصدد برمجة نظام 32 بت ، فإن النواة أيضاً 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) ، وإنما يضاف على الشفرة الثنائية العديد من الأشياء Headers,...etc. ولتحميل مثل هذه البرامج فإنه 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 وإزالة أي اعتمادية على مكتبات أو ملفات وقت التشغيل.

وسنعيد كتابة النواة التي قمنا ببرمجتها بلغة التجميع في الفصل السابق ولكن بلغة السي والسي++ ، وسناقش كيفية تحميل وتنفيذ هذه النواة حيث أن المخرج من مترجم فيجوال سي++ هو ملف تنفيذي (Portable Executable) ولديه صيغة محددة يجب التعامل معها حتى تتمكن من تنفيذ الدالة الرئيسية للنواة (main()) ، كذلك سنبدأ في تطوير ملفات وقت التشغيل للغة سي++ وذلك حتى يتم دعم بعض خصائص اللغة والتي تحتاج الى دعم وقت التشغيل مثل دوال البناء والهدم والدوال الظاهرية (Pure Virtual Function) ، وفي الوقت الحالي لا يوجد دعم للإستثناءات (Exceptions) في لغة السي++ .

٥.٤.١. تحميل وتنفيذ نواة PE

بما أننا سنستخدم مترجم فيجوال سي++ والذي يخرج لنا ملف تنفيذي (Portable Executable) فإنه 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. ويمكن أن نصف هذه الاضافات بلغة السي++ كالتالي.

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

Listing ٥١.: Global new/delete operator

```

// header information format for PE files

typedef struct _IMAGE_DOS_HEADER { // DOS .EXE header
    unsigned short e_magic; // Magic number (Should be MZ
    unsigned short e_cblp; // Bytes on last page of file
    unsigned short e_cp; // Pages in file
    unsigned short e_crlc; // Relocations
    unsigned short e_cparhdr; // Size of header in
        paragraphs
    unsigned short e_minalloc; // Minimum extra paragraphs
        needed
    unsigned short e_maxalloc; // Maximum extra paragraphs
        needed
    unsigned short e_ss; // Initial (relative) SS value
    unsigned short e_sp; // Initial SP value
    unsigned short e_csum; // Checksum
    unsigned short e_ip; // Initial IP value
    unsigned short e_cs; // Initial (relative) CS value
    unsigned short e_lfarlc; // File address of relocation
        table
    unsigned short e_ovno; // Overlay number
    unsigned short e_res[4]; // Reserved words
    unsigned short e_oemid; // OEM identifier (for
        e_oeminfo)
    unsigned short e_oeminfo; // OEM information; e_oemid
        specific
    unsigned short e_res2[10]; // Reserved words
    long e_lfanew; // File address of new exe header
} IMAGE_DOS_HEADER, *PIMAGE_DOS_HEADER;

// Real mode stub program

typedef struct _IMAGE_FILE_HEADER {
    unsigned short Machine;
    unsigned short NumberOfSections;
    unsigned long TimeDateStamp;
    unsigned long PointerToSymbolTable;
    unsigned long NumberOfSymbols;

```

```

        unsigned short  SizeOfOptionalHeader;
        unsigned short  Characteristics;
    } IMAGE_FILE_HEADER, *PIMAGE_FILE_HEADER;

typedef struct _IMAGE_OPTIONAL_HEADER {
    unsigned short  Magic;
    unsigned char   MajorLinkerVersion;
    unsigned char   MinorLinkerVersion;
    unsigned long   SizeOfCode;
    unsigned long   SizeOfInitializedData;
    unsigned long   SizeOfUninitializedData;
    unsigned long   AddressOfEntryPoint;    // offset of
        kernel_entry
    unsigned long   BaseOfCode;
    unsigned long   BaseOfData;
    unsigned long   ImageBase;    // Base address of
        kernel_entry
    unsigned long   SectionAlignment;
    unsigned long   FileAlignment;
    unsigned short  MajorOperatingSystemVersion;
    unsigned short  MinorOperatingSystemVersion;
    unsigned short  MajorImageVersion;
    unsigned short  MinorImageVersion;
    unsigned short  MajorSubsystemVersion;
    unsigned short  MinorSubsystemVersion;
    unsigned long   Reserved1;
    unsigned long   SizeOfImage;
    unsigned long   SizeOfHeaders;
    unsigned long   CheckSum;
    unsigned short  Subsystem;
    unsigned short  DllCharacteristics;
    unsigned long   SizeOfStackReserve;
    unsigned long   SizeOfStackCommit;
    unsigned long   SizeOfHeapReserve;
    unsigned long   SizeOfHeapCommit;
    unsigned long   LoaderFlags;
    unsigned long   NumberOfRvaAndSizes;
    IMAGE_DATA_DIRECTORY DataDirectory[DIRECTORY_ENTRIES];
} IMAGE_OPTIONAL_HEADER, *PIMAGE_OPTIONAL_HEADER;

```

ما نريد الحصول عليه هو عنوان الدالة الرئيسية للنواة (`kernel entry()`) والتي سيبدأ تنفيذ النواة

منها ، هذا العنوان موجود في أحد المتغيرات في آخر إضافة (header) وهي IMAGE OPTIONAL HEADER ، وحتى نحصل على عنوان هذه الأضافة يجب أن نبدأ من أول إضافة وذلك بسبب أن الإضافة الثانية ذات حجم متغير وليست ثابتة مثل بقية الإضافات.

وبالنظر إلى أول إضافة IMAGE DOS HEADER وبالتحديد إلى المتغير e Ifanew حيث يحوي عنوان الإضافة الثالثة IMAGE FILE HEADER والتي هي اضافة ثابتة الحجم ، ومنها نصل إلى 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 إلى الدالة بواسطة الأمر call. والشفرة التالية توضح طريقة ذلك (ويتم تنفيذها في المرحلة الثانية من محمل النظام مباشرة بعدما يتم تحميل النواة إلى الذاكرة على العنوان (KERNEL PMODE BASE).

Listing ٥٢.: Global new/delete operator

```
mov ebx, [KERNEL.PMODE.BASE+60]
add ebx, KERNEL.PMODE.BASE ; ebx = _IMAGE_FILE_HEADER

add ebx, 24 ; ebx = _IMAGE_OPTIONAL_HEADER

add ebx, 16 ; ebx point to AddressOfEntryPoint

mov ebp, dword[ebx] ; epb = AddressOfEntryPoint

add ebx, 12 ; ebx point to ImageBase

add ebp, dword[ebx] ; epb = kernel_entry

cli

call ebp
```

٢.٤.٥. تطوير بيئة التشغيل للغة سي++

حتى تتمكن من استخدام جميع خصائص لغة سي++ فانه يجب كتابة بعض الشفرات التشغيلية (startup) والتي تمهد وتعرف العديد من الخصائص في اللغة ، وفي هذا الجزء سيتم تطوير مكتبة وقت التشغيل للغة سي++ (C++ Runtime Library) وذلك نظراً لأننا قد الغينا الاعتماد على مكتبة وقت التشغيل التي تأتي مع المترجم المستخدم في بناء النظام (النظام الخاص بنا) حيث أن هذه المكتبة تعتمد على نظام التشغيل المستخدم في عملية التطوير مما يسبب مشاكل استدعاء دوال ليست موجودة.

وبدون تطوير هذه المكتبة فلن يمكن تهيئة الكائنات العامة (Global Object) وحذف الكائنات ، وكذلك لن يمكن استخدام بعض المعاملات (new, delete) و RTTI والاستثناءات (Exceptions).

المعاملات العامة Global Operator

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

Listing ٥٣.: Global new/delete operator

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

Pure virtual function call handler

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

Listing ٥٤.: Global new/delete operator

```
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.

^٥ عند تعريف دالة بألمة Pure virtual داخل أي فئة فإن هذا يدل على أن الفئة مجردة (Abstract) ويجب إعادة تعريف الدالة (Override) في الفئات المشتقة من الفئة التي تحوي هذه الدالة، والا ستكون الفئة المشتقة .
^٦ هذه الدالة يقوم مترجم الفيجوال سي++ باستدعائها. وقد تختلف من مترجم لآخر.

Listing ٥٥.: Global new/delete operator

```
extern "C" long __declspec (naked) _ftol2_sse() {
    int a;
#ifdef i386
    _asm {
        fistp [a]
        mov ebx, a
    }
#endif
}

extern "C" int _fltused = 1;
```

تهيئة الكائنات العامة والساكنة

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

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

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

Listing ٥٦.: Global new/delete operator

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

```
// Function pointer typedef for less typing
typedef void (__cdecl *_PVFV)(void);

// __xc_a points to beginning of initializer table
#pragma data_seg(".CRT$XCA")
_PVFV __xc_a[] = { 0 };

// __xc_z points to end of initializer table
#pragma data_seg(".CRT$XCZ")
_PVFV __xc_z[] = { 0 };

// Select the default data segment again (.data) for the
// rest of the unit
#pragma data_seg()

// 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 فإنه يضيف مؤشراً الى دالة الهدم بداخل هذه المصفوفة وذلك حتى يتم استدعائها لاحقاً (عند استدعاء الدالة (exit())، ويجب تعريف الدالة atexit حيث أن مترجم الفيجوال سي++ يقوم باستدعائها عندما يجد أي كائن ، وظيفة هذه الدالة هي اضافة مؤشر لدالة هدم الكائن الى مصفوفة المؤشرات ، وبخصوص مصفوفة المؤشرات فإنه يمكن حفظها في أي مكان على الذاكرة . والشفرة التالية توضح ما سبق.

Listing ٥٧.: Global new/delete operator

```

/*! function pointer table to global deinitializer table
static _PVFV * pf_atexitlist = 0;

// Maximum entries allowed in table. Change as needed
static unsigned max_atexitlist_entries = 32;

// Current amount of entries in table
static unsigned cur_atexitlist_entries = 0;

//! initialize the de-initializer function table
void __cdecl _atexit_init(void) {

    max_atexitlist_entries = 32;

    // Warning: Normally, the STDC will dynamically allocate
    this. Because we have no memory manager, just choose
    // a base address that you will never use for now
    pf_atexitlist = (_PVFV *)0x5000;
}

//! adds a new function entry that is to be called at
shutdown
int __cdecl atexit(_PVFV fn) {

```

```

    ///! Insure we have enough free space
    if (cur_atexitlist_entries >= max_atexitlist_entries)
        return 1;
    else {

        ///! Add the exit routine
        *(pf_atexitlist++) = fn;
        cur_atexitlist_entries++;
    }
    return 0;
}

///! shutdown the C++ runtime; calls all global de-
initializers
void _cdecl exit () {

    ///! Go through the list, and execute all global exit
    routines
    while (cur_atexitlist_entries--) {

        ///! execute function
        (*(--pf_atexitlist)) ();
    }
}

```

٣.٤.٥. نقل التنفيذ الى النواة

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

Listing ٥٨.: Global new/delete operator

```

extern void _cdecl main ();    // main function.
extern void _cdecl init_ctor(); // init constructor.
extern void _cdecl exit ();    // exit.

```

```
void _cdecl kernel_entry ()
{
#ifdef i386
    _asm {
        cli

        mov ax, 10h          // select data descriptor in GDT.
        mov ds, ax
        mov es, ax
        mov fs, ax
        mov gs, ax
        mov ss, ax           // Set up base stack
        mov esp, 0x90000
        mov ebp, esp         // store current stack pointer
        push ebp
    }
#endif

    // Execute global constructors
    init_ctor();

    // Call kernel entry point
    main();

    // Cleanup all dynamic dtors
    exit();

#ifdef i386
    _asm cli
#endif
    for(;;);
}
```

وتعريف الدالة main() حالياً سيكون حالياً.

٥.٥. نظرة على شفرة نظام إقرأ

أهم الخصائص التي يجب مراعاتها أثناء برمجة نواة نظام التشغيل هي خاصية المحمولية على صعيد الأجهزة والمنصات^٨ وخاصية قابلية توسعة النواة (Expandability) 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 تتمكن 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 هيكلية المشروع ومعماريته مثلما تؤثر الواجهة .

• eqraOS:

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

٦.٥. مكتبة السي القياسية

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

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

تعريف NULL

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

Listing ٥٩.: null.h:Definition of NULL in C and C++

```
#ifndef NULL_H
#define NULL_H

#if defined (_MSC_VER) && (_MSC_VER >= 1020)
#pragma once
#endif

#ifdef NULL
#undef NULL
#endif

#ifdef __cplusplus
extern "C"
{
#endif

/* C++ NULL definition */
#define NULL 0

#ifdef __cplusplus
}
#else

/* C NULL definition */
#define NULL (void*)0

#endif

#endif //NULL_H
```

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

تعريف `size_t`

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

Listing ٥١٠.: `size_t.h`: Definition of `size_t` in C/C++

```
#ifndef SIZE_T_H
#define SIZE_T_H

#ifdef __cplusplus
extern "C"
{
#endif

/* Standard definition of size_t */
typedef unsigned size_t;

#ifdef __cplusplus
}
#endif

#endif //SIZE_T_H
```

إعادة تعريف أنواع البيانات

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

Listing ٥١١.: `stdint.h`: typedef data type

```
#ifndef STDINT_H
#define STDINT_H`

#define __need_wint_t
#define __need_wchar_t

/* Exact-width integer type */
```

```
typedef char int8_t;
typedef unsigned char uint8_t;
typedef short int16_t;
typedef unsigned short uint16_t;
typedef int int32_t;
typedef unsigned int uint32_t;
typedef long long int64_t;
typedef unsigned long long uint64_t;
```

```
// to be continue..
```

```
#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.

Listing ٥١٢.: `cstdint`: C++ typedef data type

```
#ifndef CSTDINT_H
#define CSTDINT_H

#include <stdint.h>

#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).

Listing ٥١٣.: `ctype.h`: determine character type

```
#ifndef CTYPE_H
#define CTYPE_H
```

^٩ملفات الرأس للغة سي++ تتبع نفس هذا الأسلوب لذلك لن يتم ذكرها مجدداً وسنكتفي بذكر ملفات الرأس للغة سي.

```
#ifndef _MSC_VER
#pragma warning (disable:4244)
#endif

#ifdef __cplusplus
extern "C"
{
#endif

extern char _ctype[];

/* constants */

#define CT_UP      0x01 // upper case
#define CT_LOW     0x02 // lower case
#define CT_DIG     0x04 // digit
#define CT_CTL     0x08 // control
#define CT_PUN     0x10 // punctuation
#define CT_WHT     0x20 // white space (space,cr,lf,tab).
#define CT_HEX     0x40 // hex digit
#define CT_SP      0x80 // sapce.

/* macros */

#define isalnum(c)  ( (_ctype+1)[(unsigned)(c)] & (CT_UP|
    CT_LOW|CT_DIG) )
#define isalpha(c) (( _ctype + 1)[(unsigned)(c)] & (CT_UP
    | CT_LOW))
#define iscntrl(c) (( _ctype + 1)[(unsigned)(c)] & (
    CT_CTL))

// to be continue..

#ifdef __cplusplus
}
#endif

#endif // CTYPE_H
```

دعم الدوال بعدد غير محدود من الوسائط

٧.٥ . دالة طباعة المخرجات للنواة

٦. المقاطعات Interrupts

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

٦.١. المقاطعات البرمجية Software Interrupts

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

٦.١.١. المقاطعات في النمط الحقيقي

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

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

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

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

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

٢.١.٦. المقاطعات في النمط المحمي

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

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

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

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

Listing ٦١.: Example of interrupt descriptor

```
idt_descriptor:
    baseLow      dw    0x0
    selector     dw    0x8
    reserved     db    0x0
    flags        db    0x8e          ; 010001110
    baseHi       dw    0x0
```

المتغير الأول baseLow هو أول 16 بت من عنوان دالة معالجة المقاطعة IR ويكمل الجزء الآخر من العنوان المتغير baseHi وفي هذا المثال العنوان هو 0x0. بمعنى أن دالة تخدم المقاطعة ستكون في العنوان 0x0. وبما أن دالة معالجة (تخدم) المقاطعة تحوي شفرة برمجية للتنفيذ وليست بيانات (Data) فإن قيمة المتغير selector يجب أن تكون 0x8 للإشارة إلى ناخب الشفرة (Code Selector) في جدول الوصفات العام (GDT). أما المتغير flags فإن قيمته هي 010001110b دلالة على أن الوصفة هي 32-bit وأن مستوى الحماية هو الحلقة صفر (Ring0). وبعد أن يتم انشاء أغلب الوصفات 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 الوصفات في المتغير idt_end ومن ثم انشاء مؤشر يسمى idt_ptr والذي يجب أن يكون في صورة معينة بحيث يحفظ عنوان بداية الجدول ونهايته :

Listing ٦٢.: Value to put in IDTR

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

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

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

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

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

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

٣.١.٦. أخطاء المعالج

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

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

والجدول التالي يوضح أخطاء المعالج والمقاطعات التي يقوم بتوليدها.

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

ويجدر بنا الوقوف على ملاحظة كئنا قد ذكرناها في الفصول السابقة وهي إلغاء المقاطعات (بواسطة الأمر cli) عند الانتقال الى النمط المحمي حتى لا يتسبب في حدوث خطأ General Protection Fault وبالتالي توقف النظام عن العمل وسبب ذلك هو أن عدم تنفيذ الأمر cli يعني أن المقاطعات العادية مفعلة وبالتالي أي عتاد يمكنه أن يرسل مقاطعة الى المعالج لكي ينقل التنفيذ الى دالة تخدمها . 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 Prprogrammable Interval Timer وتختصر بـ PIT وهي تمثل ساعة النظام System Timer بحيث ترسل مقاطعة بشكل دائم الى المعالج والذي بدوره ينقل التنفيذ الى دالة تخدم هذه المقاطعة . وبسبب أن جدول المقاطعات غير متواجد

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

وفيما يلي سيتم إنشاء جدول المقاطعات (IDT) باستخدام لغة السي وتوفير ال 256 دالة لمعالجة المقاطعات وحاليا سيقصر عمل الدوال على طباعة رسالة ، وقبل ذلك سنقوم بإنشاء جدول الواصفات العام (GDT) مجدداً (أي سيتم الغاء الجدول الذي قمنا بإنشائه في مرحلة الاقلاع) وبعد ذلك سنبدأ في برمجة متحكم PIC وإعادة 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 المحمولىة على صعيد المنصات ، وهذا ما أدى الى اعتماد فكرة طبقة HAL والتي يقبع تحتها كل ما يتعلق بعتاد الحاسب وادارته وكل ما يجعل النظام معتمداً على معمارية معينة أيضاً نجده تحت طبقة HAL ، و جدول الواصفات العام - كما ذكرنا في الفصول السابقة- يحدد ويقسم لنا الذاكرة الرئيسية كأجزاء قابلة للتنفيذ وأجزاء تحوي بيانات وغيرها ، ونظراً لأن إنشاء هذا الجدول يعتمد على معمارية المعالج والأوامر المدعومة فيه فانه يجب ان يقبع 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 تماماً عن التطبيق حتى يتمكن أي مطور من إعادة تطبيقها لاحقاً على معمارية حاسوب آخر.

Listing ٦٣: include/hal.h: Hardware Abstraction Layer Interface

```
#ifndef HAL_H
#define HAL_H

#ifndef i386
#error "HAL is not implemented in this platform"
#endif

#include <stdint.h>

#ifdef _MSC_VER
```

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

```

#define interrupt __declspec(naked)
#else
#define interrupt
#endif

#define far
#define near

/* Interface */

extern int _cdecl hal_init();
extern int _cdecl hal_close();
extern void _cdecl gen_interrupt(int);

#endif // HAL_H

```

و حالياً واجهة طبقة HAL مكونة من ثلاث دوال تم الإعلان عنها بأنها extern وهذا يعني أن أي تطبيق (Implementation) لهذه الواجهة يجب أن يُعرّف هذه الدوال. الدالة الأولى هي hal_init() والتي تقوم بتهيئة العتاد وجدول المعالج بينما الدالة الثانية hal_close() تقوم بعملية الحذف والتحرير وأخيراً الدالة gen_interrupt() والتي تم وضعها لغرض تجربة إرسال مقاطعة برمجية والتأكد من أن دالة معالجة المقاطعة تعمل كما يرام.

نعود بالحديث إلى 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 إلى 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 إلى الواصفة المطلوبة.
- الدالة gdt_install: تقوم بتحميل المؤشر الذي يحوي حجم الجدول وعنوان بدايته إلى المسجل GDTR.

والشفرة التالية توضح كيفية انشاء الجدول ^٥.

^٣راجع ١.١.٤.

^٤لغرض التنظيم والتقسيم لا أكثر ولا أقل.

^٥راجع شفرة النظام لقراءة ملف الرأس hal/gdt.h.

Listing ٦٤.: hal/gdt.cpp:Install GDT

```
#include <string.h>
#include "gdt.h"

static struct gdt_desc _gdt[MAX_GDT_DESC];
static struct gdtr _gdtr;

static void gdt_install();

static void gdt_install() {
#ifdef _MSC_VER
    _asm lgdt [_gdtr];
#endif
}

extern void i386_gdt_set_desc(uint32_t index, uint64_t base,
    uint64_t limit, uint8_t access, uint8_t grand) {

    if ( index > MAX_GDT_DESC )
        return;

    // clear the desc.
    memset((void*)&_gdt[index], 0, sizeof(struct gdt_desc));

    // set limit and base.
    _gdt[index].low_base = uint16_t(base & 0xffff);
    _gdt[index].mid_base = uint8_t((base >> 16) & 0xff);
    _gdt[index].high_base = uint8_t((base >> 24) & 0xff);
    _gdt[index].limit = uint16_t(limit & 0xffff);

    // set flags and grandularity bytes
    _gdt[index].flags = access;
    _gdt[index].grand = uint8_t((limit >> 16) & 0x0f);
    _gdt[index].grand = _gdt[index].grand | grand & 0xf0;
}

extern gdt_desc* i386_get_gdt_desc(uint32_t index) {
    if ( index >= MAX_GDT_DESC )
```

```
        return 0;
    else
        return &_gdt[index];
}

extern int i386_gdt_init() {

    // init _gdt
    _gdt.limit = sizeof(struct gdt_desc) * MAX_GDT_DESC - 1;
    _gdt.base = (uint32_t)&_gdt[0];

    // set null desc.
    i386_gdt_set_desc(0,0,0,0,0);

    // set code desc.
    i386_gdt_set_desc(1,0,0xffffffff,
        I386_GDT_CODE_DESC | I386_GDT_DATA_DESC | I386_GDT_READWRITE
        | I386_GDT_MEMORY,    // 10011010
        I386_GDT_LIMIT_HI | I386_GDT_32BIT | I386_GDT_4K
        // 11001111
    );

    // set data desc.
    i386_gdt_set_desc(2,0,0xffffffff,
        I386_GDT_DATA_DESC | I386_GDT_READWRITE | I386_GDT_MEMORY,
        // 10010010
        I386_GDT_LIMIT_HI | I386_GDT_32BIT | I386_GDT_4K    //
        11001111
    );

    // install gdt
    gdt_install();

    return 0;
}
```

٥.١.٦ . إنشاء جدول المقاطعات IDT

٢.٦ . Programmable Interrupt Controller

٣.٦ . Programmable Interval Timer

٤.٦ . المقاطعات العادية Hardware Interrupts

٧. إدارة الذاكرة

١.٧. إدارة الذاكرة الفيزيائية Physical Memory Management

٢.٧. إدارة الذاكرة التخيلية Virtual Memory Management

٨. مشغلات الاجهزة Device Driver

٨.١. برمجة مشغل لوحة المفاتيح Keyboard Driver

٨.٢. برمجة مشغل القرص المرن Floppy Disk Driver

انظر الى شفرة النظام.

٨.٣. برمجة متحكم DMAC

انظر الى شفرة النظام.

٩ . أنظمة الملفات

انظر الى شجرة النظام.

١. ترجمة وتشغيل البرامج

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

١.١. نظام ويندوز

٢.١. نظام لينوكس

Listing ١١.: Some Code

The following is a list of the standard BIOS interrupts used in a typical BIOS.

%<http://www.bioscentral.com/misc/biosservices.htm>

Interrupt	Address	Type	Description
00h	0000:0000h	Processor	Divide by zero
01h	0000:0004h	Processor	Single step
02h	0000:0008h	Processor	Non maskable interrupt (NMI)
03h	0000:000Ch	Processor	Breakpoint
04h	0000:0010h	Processor	Arithmetic overflow
05h	0000:0014h	Software	Print screen
06h	0000:0018h	Processor	Invalid op code
07h	0000:001Ch	Processor	Coprocessor not available
08h	0000:0020h	Hardware	System timer service routine
09h	0000:0024h	Hardware	Keyboard device service routine

0Ah	0000:0028h	Hardware	Cascade from 2nd programmable interrupt controller
0Bh	0000:002Ch	Hardware	Serial port service – COM post 2
0Ch	0000:0030h	Hardware	Serial port service – COM port 1
0Dh	0000:0034h	Hardware	Parallel printer service – LPT 2
0Eh	0000:0038h	Hardware	Floppy disk service
0Fh	0000:003Ch	Hardware	Parallel printer service – LPT 1
10h	0000:0040h	Software	Video service routine
11h	0000:0044h	Software	Equipment list service routine
12h	0000:0048h	Software	Memory size service routine
13h	0000:004Ch	Software	Hard disk drive service
14h	0000:0050h	Software	Serial communications service routines
15h	0000:0054h	Software	System services support routines
16h	0000:0058h	Software	Keyboard support service routines
17h	0000:005Ch	Software	Parallel printer support services
18h	0000:0060h	Software	Load and run ROM BASIC
19h	0000:0064h	Software	DOS loading routine
1Ah	0000:0068h	Software	Real time clock service routines
1Bh	0000:006Ch	Software	CRTL – BREAK service routines
1Ch	0000:0070h	Software	User timer service routine
1Dh	0000:0074h	Software	Video control parameter table
1Eh	0000:0078h	Software	Floppy disk parameter routine
1Fh	0000:007Ch	Software	Video graphics character routine
20h–3Fh	0000:0080f – 0000:00FCh	Software	DOS interrupt points
40h	0000:0100h	Software	Floppy disk revector routine
41h	0000:0104h	Software	hard disk drive C:

parameter table			
42h	0000:0108h	Software	EGA default video driver
43h	0000:010Ch	Software	Video graphics characters
44h	0000:0110h	Software	Novel Netware API
45h	0000:0114h	Software	Not used
46h	0000:0118h	Software	Hard disk drive D:
parameter table			
47h	0000:011Ch	Software	Not used
48h		Software	Not used
49h	0000:0124h	Software	Not used
4Ah	0000:0128h	Software	User alarm
4Bh-63h	0000:012Ch	Software	Not used
64h		Software	Novel Netware IPX
65h-66h		Software	Not used
67h		Software	EMS support routines
68h-6Fh	0000:01BCh	Software	Not used
70h	0000:01c0h	Hardware	Real time clock
71h	0000:01C4h	Hardware	Redirect interrupt cascade
72h-74h	0000:01C8h - 0000:01D0h	Hardware	Reserved -
Do not use			
75h	0000:01D4h	Hardware	Math coprocessor exception
76h	0000:01D8h	Hardware	Hard disk support
77h	0000:01DCh	Hardware	Suspend request
78h-79h	0000:01E0h	Hardware	Not used
7Ah		Software	Novell Netware API
78h-FFh	0000:03FCh	Software	Not used

ب. المراجع

ب.١. المراجع

Bibliography

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

ج. شفرة نظام إقرأ

كود النظام

د. إتفاقية ترخيص المستندات الحرة GNU FDL