المرحلة الرابعة: تدبير الانقطاعات و الاستثناءات

في الدرس السابق رأينا كيفية استعمال لغة سي++ من أجل تطوير نواة محملة بكروب، كما اطلعنا أيضا على بعض مبادئ البرمجة المتفرقة.

في هذا الدرس سنتعرف على أنواع و كيفية تدبير الانقطاعات في الوضع المحمي ل x86. سنرى كذلك مثالا عبر المنبه.

ملحوظة: في نهاية الدرس تجدون ملحقا يوضح كيفية استعمال المحاكي Bochs مع صورة للقرص المرن من أجل تجريب الشفرة المطورة في هذا الدرس أو الدروس السابقة.

1- نظرة عن الانقطاعات في الحاسبات الشخصية

تعرف الانقطاعات على أنها أحداث تنبئ المعالج بوقوع أمر يسترعي انتباهه، سواء في البرنامج الذي في طور التنفيذ أو في النظام عامة. ينتج عن ذلك تحويل سير التنفيذ من التعليمات الحالية المنفذة نحو برنامج آخر يسمى مدبر الانقطاع interrupt handler (يسمى أيضا إجراء خدمة الانقطاع interrupt service routine – ISR). بعد تنفيذ هذا الأخير يعود التحكم للبرنامج الذي تم قطعه ليواصل تنفيذه. مثلا، عندما يقوم المستعمل بضغط زر في لوحة المفاتيح، يوقف المعالج سير التعليمات الجاري، ليقوم بتنفيذ شفرة مدبر الانقطاع الخاص بلوحة المفاتيح، بعد الانتهاء من تنفيذ المدبر، يعود المعالج لمواصلة تنفيذ البرنامج السابق.

في معالجات x86 نميز بين ثلاث أنواع من الانقطاعات:

1- **الاستثناءات Exceptions**: تقع عند رصد خطأ أثناء تنفيذ التعليمة الحالية، مثل قسمة عدد على صفر. قد يكون الخطأ أيضا ناتجا عن خرق لحماية الذاكرة (عبر محاولة الولوج إلى قسم في الذاكرة درجة أفضليته أعلى من درجة الأفضلية الحالية التي يتم بها التنفيذ). تعرف هذه الانقطاعات بأنها متزامنة Sychnronous ذلك لأنها وقوعها مرتبط بسير التعليمات الحالي.

2- **الانقطاعات الصلبة IRQ** : تأتي هذه الانقطاعات من الفروع المثبتة على البطاقة الأم (مثل لوحة المفاتيح، تكة المنبه...). هذه الانقطاعات غير مرتبطة بسير التعليمات الجاري ويمكن أن تحدث في أي وقت بمعزل عن هذا الأخير. لذلك فهي تعرف بأنها انقطاعات لامتزامنة Asynchronous.

3- **الانقطاعات البرمجية Software interrupts**: يمكن أيضا التسبب في انقطاعات عبر تعليمات برمجية مثل int, int3, into, bound. إذا سبق لك البرمجة بالأسمبلر في الوضع الحقيقي real mode فلا بد أنك استعملت التعليمة int لاستعمال خدمات البيوس أو الدوس.

في معالجات x86، يتم تمييز كل انقطاع بعدد من بايت واحد يسمى متجهة الانقطاع. عبر هذا العدد يستطيع المعالج استنتاج موقع مدبر الانقطاع الخاص به، لكن طريقة استنتاج هذا الموقع تختلف حسب وضع العنونة الذي يوجد فيه المعالج:

- في الوضع الحقيقي: تكون عناوين مدبرات الانقطاعات مخزنة في جدول يبدأ من الموقع 0 في الذاكرة و يسمى جدول متجهات الانقطاعات Interrupt Vector Table (IVT). كل عنوان يتشكل من 4 بايت: اثنان لتخزين عنوان القسم و اثنان لتخزين البعد داخل القسم. لاستخراج عنوان المدبر من الجدول يستعمل المعالج متجهة الانقطاع كمؤشر داخل الجدول. مثلا المتجهة 0 تكافئ المدخل 0 في

الجدول أي العنوان 0، المتجهة 1 تكافئ المدخل رقم 1 أي العنوان 4. المتجهة 2 تكافئ العنوان 8 ...إلخ. للحصول على العنوان نكتفي بضرب المتجهة في 4.

- **في الوضع المحمي:** تستعمل المتجهة كمؤشر لكن هذه المرة في جدول آخر يسمى **جدول موصفات الانقطاعات Interrupt Descriptor Table (IDT)**. داخل كل موصف للانقطاع نجد من بين معلومات أخرى محددا للقسم (بما أننا في الوضع المحمي نحتاج لمحدد القسم الذي ليس بدوره سوى مؤشر على موصف القسم في جدول الموصفات العام GDT) الذي يوجد فيه مدبر الانقطاع بالإضافة إلى البعد داخل القسم. لاحقا في هذا الدرس سنرى كيف يمكن صنع هذا الجدول و استعماله من طرف المعالج.

بما أن متجهة الانقطاع تحدد على بايت واحد فهذا يجعل مجال الانقطاعات الممكنة محصورا بين 0 و 255. المتجهات من 0 إلى 31 تستعمل من طرف المعالج لإعلان الاستثناءات المرصودة. بينما تبقى المتجهات من 32 إلى 255 لاستغلالها من طرف الفروع (الانقطاعات الصلبة) و نظام التشغيل (الانقطاعات البرمجية التي تستعمل غالبا للخدمات التي يود النظام توفيرها لبرامج المستعمل).

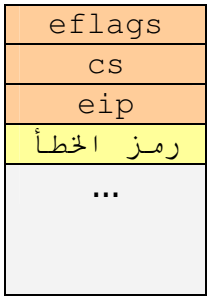
فيما يلي سنتحدث بالتفصيل عن تدبير الاستثناءات و الانقطاعات الصلبة.

أ- الاستثناءات

هناك ثلاث مصادر للاستثناءات:

1- الأخطاء المرصودة من طرف المعالج: أخطاء في التنفيذ مثل القسمة على 0.
2- الاستثناءات البرمجية: يمكن إعلان بعض الاستثناءات عبر تعليمات برمجية : مثلا التعليمة int3 تقوم باعلان الاستثناء رقم 3 و الذي ينتج نقطة وقوف Breakpoint في البرنامج الجاري. التعليمة into تقوم باعلان الاستثناء رقم 4 والمسمى Overflow Exception.
3- الاستثناء الناتج في حالة رصد أخطاء في عمل الهاردوير و يسمى Machine Check Exception.

قلنا أنه عند وقوع استثناء يقوم المعالج بالانتقال إلى تنفيذ شفرة مدبر الاستثناء. لكن قبل ذلك يقوم بتخزين قيمة بعض المسجلات الحيوية للبرنامج الحالي حتى يتمكن من مواصلة تنفيذه بطريقة صحيحة بعد العودة من مدبر الاستثناء. المسجلات المخزنة تشمل eflags, cs, eip. يتم تخزين هذه المسجلات عبر دفعها في الكومة على غرار التعليمة push. في بعض الاستثناءات يتم أيضا دفع عدد إضافي يشكل رمز الخطأ المرتكب error code. هكذا عندما تعطى اليد لشفرة مدبر الاستثناء تكون الكومة على الشكل التالي



فقط في حالة بعض الاستثناءات

بعد أن ينهي مدبر الاستثناء عمله يعود لكن ليس بالتعليمة المعتادة `ret` بل بتعليمة خاصة iret. هذه الأخيرة تقوم بإرجاع قيمة المسجلات الثلاث eflags, cs, eip من الكومة على غرار التعليمة pop (لاحظ أنه يتوجب علينا قبلا إزاحة رمز الخطأ من الكومة بأنفسنا).

السؤال المطروح هنا هو أين يكمل البرنامج المقطع تنفيذه بالضبط؟ هل في التعليمة التي كان المعالج ينفذها ساعة إعلان الاستثناء أو التعليمة الموالية لها؟ الجواب على هذا السؤال يقودنا إلى تمييز ثلاث مجموعات من الاستثناءات:

أ- **الأخطاء Faults**: يتم تخزين قيمة **التعليمة الحالية** لeip. هكذا تعطى الفرصة للنظام بتصحيح الخطأ. و في حالة العودة يتم الرجوع إلى بدأ التعليمة التي تسببت في الخطأ.

ب- **المطبات Traps**: يتم إعلانها دائما بعد نهاية التعليمة الحالية. مما يعني أن القيمة المخزنة لeip هي **التعليمة الموالية**.

ت- **الإخفاقات Aborts**: تستعمل لإعلان أخطاء فادحة. **القيمة المخزنة لeip غير محددة بدقة** مما يعني استحالة مواصلة البرنامج المقطع بطريقة صحيحة.

الجدول التالي يعدد استثناءات معالجات x86.

متجهة الاستثناء	الوصف	رمز الخطأ في الكومة؟
0	Division by zero Exception	لا
1	Debug Ecxeption	لا
2	Non maskable interrupt Exception	لا
3	Breakoint Exception	لا
4	into detetcted overflow Exception	لا
5	Out of bounds Exception	لا
6	Invalid opcode Exception	لا
7	No Coprocessor Exception	لا
8	Double fault Exception	نعم
9	Coprocessor segment overrun Exception	لا
10	Bad TSS Exception	نعم
11	Segment not present Exception	نعم
12	Stack fault Exception	نعم
13	General protection fault Exception	نعم
14	Page fault Exception	نعم
15	Unkown interrupt Exception	لا
16	Coprocessor fault Exception	لا
17	Alignement check Exception (only on 486+)	لا
18	Machine check Exception (only on Pentium/586+)	لا
31-19	Reserved Exceptions	لا

لمعرفة المزيد عن هذه الاستثناءات، انصح القارئ بالاطلاع على كتاب الانتل (الجزء الثالث).

ب- الانقطاعات الصلبة IRQ

هذه الانقطاعات تأتي كما قلنا سابقا من الفروع المتصلة بالبطاقة الأم و تسمى بطلبات الانقطاع Interrupt Requests.

عندما يقع حدث في أحد الفروع مما يسترعي انتباه المعالج (مثل ضغطة زر في لوحة المفاتيح – التوصل بحزمة بيانات عبر الشبكة)، يقوم الفرع بطلب انقطاع من المعالج. على مستوى الهاردوير، فإن طلبات الانقطاع الصلبة تأتي للمعالج عبر دبابيس (Pins) LINT[1:0] (الدبابيس موجودة بأسفل المعالج و عن طريقها يثبت على البطاقة الأم).

بما أن عدد الفروع يفوق بكثير قدرة الخط المرصود من طرف المعالج (الخط الثاني مرصود لانقطاع خاص يسمى الانقطاع غير المحتجب Non Maskable Interrupt). فإن طلبات الانقطاع تمر قبلا على رقاقة تمتلك أكثر من خط تسمى **محكم الانقطاعات Programmable) Interrupt Controller - PIC)**. تلعب هذه الرقاقة دور الحكم بين طلبات الانقطاع الوافدة من الفروع: يتم ترتيبها حسب أولوية كل فرع ثم تقديمها واحدا بواحد للمعالج حتى تتم معالجتها.

<u>ملحوظة</u>: الPIC كان يستعمل في الحاسبات القادمة، لكن مع ظهور معالجات بنتيوم، تم التخلي عن هذا المحكم. و بغرض دعم أنظمة متعددة المعالجات (multiprocessor)، أصبحت هذه المعالجات تتوفر على رقاقة مدمجة في المعالج تسمى Advanced Programmable Interrupt Controller أو APIC ويسمى أيضا ب Local APIC لأنه يستعمل بالتعاون مع رقاقة أخرى على البطاقة الأم تسمى IO APIC. لكن الحاسبات ظلت تدعم الPIC للتوافق مع البرامج القديمة. في درسنا سنتحدث فقط عن برمجة الPIC. وقد تكون لنا عودة إلى الAPIC في درس قادم إن شاء الله.

محكم الانقطاعات PIC عبارة عن رقاقة تدعى 8259A، و يتوفر على 8 خطوط لتلقي طلبات الانقطاع (لاحقا سنقول خطوط الانقطاع الصلبة) مما يعني احتماله ل 8 فروع على الأكثر. من أجل ربط المزيد من الفروع يستعمل محكم ثان من نفس النوع لربط فروع أخرى، و يربط تسلسليا مع المحكم الأول عبر خطوط خاصة (3 line cascade bus → CAS0:CAS3). هكذا يصبح لدينا محكمان: - محكم أساسي و آخر ثانوي (التسمية الإنجليزية تتحدث عن المحكم السيد و العبد Master & Slave، شخصيا لا أحبذ هذه المصطلحات). المحكم الأساسي هو الذي يستطيع إعلان طلب الانقطاعات للحاسب مباشرة بينما على المحكم الثانوي أن يمر قبلا عبر المحكم الأساسي. لإعلان طلب الانقطاع من المحكم الثانوي تُرصد أحد خطوط الانقطاع في المحكم الأساسي لهذا الغرض – بالتحديد الخط رقم 2 (الخط الثالث لأن الخطوط تبدأ من 0).

لفهم طريقة عمل الـ8259A، لنتصور الحالتين التاليتين:
- <u>المحكم الأساسي</u> يتلقى طلب انقطاع (أو أكثر) من أحد الفروع على أحد خطوطه الثمانية. على إثر ذلك يقوم بإرسال إشارة INT عبر ناقلة التحكم (Control bus) إلى المعالج، المعالج يرد بإرسال إشارة INTA (INT Aknowledge) إلى المحكم، بعد فترة وجيزة (يتاح خلالها للمحكم اختيار خط الانقطاع الذي له الأولوية)، يرسل المعالج إشارة ثانية INTA تخبر المحكم أنه على استعداد لتلقي متجهة الانقطاع، فيرسل المحكم متجهة الانقطاع عبر ناقلة البيانات (سنرى لاحقا كيف يستخلص المحكم متجهة الانقطاع انطلاقا من رقم خط الانقطاع). بعد تنفيذ مدبر الانقطاع ينتظر المحكم إشارة EOI (End Of Interrupt)، لكي يقوم بعدها بتحديث مسجلاته. لاحظ أن إشارة EOI لا ترسل آليا من طرف المعالج بل على مدبر الانقطاع القيام بذلك (اللهم إذا قمنا بتنشيط الوضع AEOI أي Automatic EOI الأمر الذي لم ألحظه في أي من الأمثلة عن معالج x86).

- بالنسبة <u>للمحكم الثانوي</u> الأمر مختلف قليلا كونه لا يستطيع إرسال إشارة INT مباشرة للمعالج لأن المحكم الأساسي يتحكم في عملياته عبر الخطوط CAS0:CAS3، هكذا عند تلقيه لطلب انقطاع من أحد الفروع، يقوم أولا بتنشيط خط الانقطاع 2 في المحكم الرئيسي. هذا الأخير يعرف أن هذا الخط مرصود للمحكم الثانوي و ليس لفرع عادي (لاحقا سنرى كيف يمكن تزويد المحكم بهذه المعلومة)، فيقوم بإرسال إشارة INT للمعالج، و حال تلقيه لأول INTA يقوم بتمكين المحكم الثانوي من إرسال متجهة الانقطاع. في هذه الحالة يتوجب على مدبر الانقطاع إرسال إشارتي EOI، واحدة لكل محكم.

بما أنه لا يمكن استعمال خط الانقطاع 2 في المحكم الرئيسي من طرف الفروع، فهذا يعني أن عدد الخطوط المتاحة في هذه المنظومة يصبح 15. في الحاسبات الشخصية ترصد خطوط الانقطاع عادة على النحو التالي

رقم خط الانقطاع	الفرع المستعمل للخط
المحكم الرئيسي	
0	منبه النظام System timer
1	لوحة المفاتيح Keyboard
2	موصل مع المحكم الثانوي
3	الفرع التسلسلي Serial port 2
4	الفرع التسلسلي Serial port 1
5	الفرع الموازي Paralle port 2
6	محكم القرص المرن Floppy drive
7	الفرع الموازي Paralle port 1
المحكم الثانوي	
8	منبه CMOS Real Time Clock
9	غير مستعمل
10	غير مستعمل
11	غير مستعمل
12	الفأرة Mouse
13	المعالج الرياضي Math Coprocessor
14	محكم القرص الصلب الرئيسي Primary Ide
15	محكم القرص الصلب الثانوي Secondary Ide

عند انطلاق الحاسب (في الوضع الحقيقي) يقوم البيوس بتهيئة المحكمين بحيث تتطابق خطوط الانقطاع في المحكم الرئيسي 0-7 مع متجهات الانقطاع 0-7 (أي أن طلب انقطاع من الخط 0 يؤدي إلى تنشيط متجهة الانقطاع 0 ....الخط 7 ينشط المتجهة 7) في حين تتطابق خطوط المحكم الثانوي مع المتجهات 70-77 (الخط 8 -> المتجهة 70 ... الخط 15-> المتجهة 77). المشكل أنه كما رأينا في الوضع المحمي تخصص المتجهات من 0 إلى 31 لاستثناءات المعالج مما يعني وقوع نزاع Conflict بين إعلان الاستثناءات من 0 إلى 7 و المتجهات المتولدة عن طلبات الانقطاع في المحكم الرئيسي. هذا يحتم علينا إعادة المطابقة بين خطوط الانقطاع و المتجهات بحيث نتفادى أي نزاع بين المتجهات.

الطريقة الوحيدة لهذا الغرض هي إعادة تهيئة المحكم من الأول عبر برمجة مسجلاته.

كيف نبرمج محكم الانقطاعات؟

في الحاسبات الشخصية تخصص البوابتين الفرعيتين 0x20-0x21 للمحكم الرئيسي، و البوابتين 0xA0-0xA1 للمحكم الثانوي. البوابات 0x20-0xA0 تسمى بوابات التحكم Command Ports و البوابات 0x21-0xA1 تسمى بوابات البيانات Data Ports. تتم البرمجة عبر كتابة بايتات في هذه البوابات: تسمى هذه البايتات كلمات التحكم Command Words. في الـ8259A نميز بين نوعين:

- كلمات التحكم بالتهيئة Initialization Command Words (ICWs): يجب إرسالها قبل أي استعمال للمحكم و هي التي تحدد طريقة عمله فيما بعد.
- كلمات التحكم بالعمليات Operation Command Words (OCWs): يمكن إرسالها في أي وقت بعد إتمام التهيئة و تتحكم في أوضاع الانقطاعات: حجب بعض الخطوط – إعادة ترتيب الأولويات...

في حالتنا نحن نحتاج فقط لإعادة مطابقة المتجهات لذلك سنكتفي بالتهيئة عبر إرسال ICWs. لإتمام ذلك في الحاسبات الشخصية نحتاج لإرسال متتالية من أربع بايتات على التوالي، تسمى متتالية الكلمات هذه Initialization sequence (ICW1, ICW2, ICW3, ICW4)

الكلمة الأولى ICW1 تبدأ متتالية التهيئة و ترسل للبوابة 0x20 (0xA0 بالنسبة للمحكم الثانوي)، الجدول يظهر فقط البيتات المستعملة في x86.

-	-	-	1	LTIM	-	SNGL	IC4

IC4 - 1: المتتالية تتضمن ICW4 - 0: لا تتضمن ICW4.
SNGL - 1: لا يوجد محكم ثانوي - 0: يوجد
LTIM - 1: Level triggered mode - 0: Edge triggered mode

(لإشباع رغبة الفضوليين ☺ هذا العلم flag يتحكم في طريقة رصد طلبات الانقطاع على دبابيس الـ8259A، Edge detection ترصد الانقطاع بتغير وضع أحد خطوط الانقطاع من أسفل low إلى أعلى high بينما level detection ترصد الانقطاع بوجود مستوى الدبوس في أعلى)

باقي كلمات التهيئة التي ترسل كلها للبوابة 0x21 (0xA1 في المحكم الثانوي)

الكلمة الثانية ICW2 تحدد *البعد* الذي ينبغي إضافته على خط الانقطاع للحصول على متجهة الانقطاع، في الواقع و من أجل استخلاص متجهة الانقطاع من رقم الخط، كل ما يقوم به المحكم هو وضع رقم الانقطاع في البيتات الثلاث الأضعف ثم يضع الخمس بيتات الأقوى من *البعد* المذكور سابقا في يسار المتجهة، إذا حددنا البعد في 72 (بالثنائي = 1001000) وكان خط الانقطاع هو 3 (011) فإن المتجهة تستخلص كما يلي

1	0	0	1	0	0	1	1
الـ5بيتات الأقوى في البعد					رقم الخط		

هذا يفرض علينا أن نحدد البعد في قيمة تكون قابلة للقسمة على ثمانية حتى تكون البيتات الثلاث الأضعف فيه 0، إذ أن الـ8259A يتجاهل هذه البيتات في معالجات x86.

الكلمة الثالثة ICW3 تستعمل لتحديد الخط الذي يرصده المحكم الرئيسي لتلقي طلبات الانقطاع من المحكم الثانوي. تأويل الكلمة يختلف بين المحكمين:
في المحكم الرئيسي نثبت البيت المكافئ للخط على 1 (الـ8259A صمم في الأصل لتحمل الربط مع ثمانية محكمات ثانوية بحيث كل محكم ثانوي يستعمل أحد خطوط الانقطاع) في الحاسبات الشخصية يوجد محكم ثانوي وحيد و يستعمل الخط الثالث (رقم 2) مما يجعل الـICW3 على الشكل التالي

0	0	0	0	0	1	0	0

في المحكم الثانوي تؤول الكلمة على أنها رقم الخط المستعمل (حيث أن المحكم الثانوي لا يمكنه الارتباط سوى بمحكم رئيسي واحد). لذلك تكون قيمة الكلمة 2.

الكلمة الرابعة ICW4 تكون قيمتها دائما 1 في معالجات x86.

فيما يلي مثال لبرمجة الـ8259A:

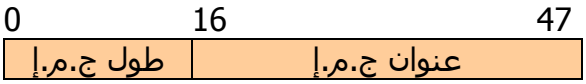
```
const int pic1= 0x20;
const int pic2= 0xA0;
```

```
//ICW1: need ICW4, pic2 present, edge detection
outb(pic1, 0x11);
outb(pic2, 0x11);

//ICW2: remap pic1 to use vectors from 0x20 (32)
//      remap pic2 to use vectors from 0x28 (40)
outb(pic1+1, 0x20);
outb(pic2+1, 0x28);

//ICW3: pic2 uses IRQ2
outb(pic1+1, 4);
outb(pic2+1, 2);

//ICW4
outb(pic1, 1);
outb(pic2, 1);
```

أعتقد أن الشفرة مفهومة على ضوء ما سبق، في البدء نرسل ICW1 إلى البوابة 0x20 (0xA0 للمحكم الثانوي) وهي 0x11 (00010001) بمعنى: المتتالية تتضمن ICW4، المحكم الثانوي موصول ثم استعمال edge detection. بعد ذلك الكلمة الثانية من المتوالية و فيها نحدد البعد الذي سيضاف على خط الانقطاع لاستخلاص متجهة الانقطاع، في الشفرة متجهات الانقطاع في المحكم الرئيسي تبدأ من 0x20 (32) أي بعد الاستثناءات مباشرة، بينما تأتي متجهات المحكم الثانوي انطلاقا من 0x28 (40). في ICW3 نحدد الخط المستعمل بين المحكمين و هو 2. و أخيرا ICW4 دائما 1 في الx86.

2- تدبير الانقطاعات: جدول موصفات الانقطاعات IDT

كما رأينا في أول الدرس، تستعمل متجهة الانقطاع في الوضع المحمي كمؤشر على جدول موصفات الانقطاعات (ج.م.إ). تتضمن مداخل هذا الجدول معلومات عن كيفية تدبير الانقطاع: مثل عنوان مدبر الانقطاع و بعض المعلومات الأخرى. على غرار جدول الموصفات العام، يمتد كل موصف على 8 بايت، لكن بخلاف ج.م.ع. ليس من الضروري أن يكون المدخل الأول فارغا. يخزن عنوان هذا الجدول في المسجل IDTR، بنية هذا الأخير مماثلة للبنية المستعملة في المسجل GDTR الذي يحتوي عنوان ج.م.ع.



لتحميل هذه البنية إلى المسجل IDTR نستعمل التعليمة lidt.

موصف الانقطاعات IDT Descriptor

هناك ثلاث أنواع من الموصفات يمكن ل ج.م.إ أن يتضمنها:

1- موصف لمدخل انقطاع Interrupt-gate Descriptor
2- موصف لمدخل مطبّ Trap-gate Descriptor
3- موصف لمدخل مهمة Task-gate Descriptor

النوع الأخير يستعمل بغرض تدبير المهام المختلفة Task management. فيما يلي سنقتصر على النوعين الأولين (لكننا سنستعمل فقط النوع الأول في الشفرة).

الشكل التالي يوضح بنية موصفات الانقطاعات و المطبات. البنيتان متشابهتان فيما عدا البيت رقم 40.

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48	47	46 45	44 43 42 41 40	39 38 37	36 35 34 33 32
Offset 16..31البعد	P	DPL	0 1 1 1 0	0 0 0	
Segment selectorمحدد القسم	Offset 0..15البعد				

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

مدخل انقطاع

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48	47	46 45	44 43 42 41 40	39 38 37	36 35 34 33 32
Offset 16..31البعد	P	DPL	0 1 1 1 1	0 0 0	
Segment selectorمحدد القسم	Offset 0..15البعد				

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

مدخل مطب

P- 1: القسم موجود 0: القسم غير موجود
DPL- درجة أفضلية الموصف، تستعمل لحماية الأقسام ذات الأفضلية العليا (أي الأقل رقميا). هذا العلم يستعمل فقط في الانقطاعات البرمجية و يُتجاهل في الانقطاعات الصلبة و الاستثناءات المعلنة من طرف المعالج. لتبسيط الأمور نقول مثلا بإمكان برنامج يعمل في درجة أفضلية 3 أن ينادي مدبر انقطاع عندما تكون قيمة DPL تساوي 0. لكن العكس يشكل انتهاكا لقواعد الحماية و يؤدي إلى إعلان استثناء #GP أي خطأ حماية عام General Protection Fault.

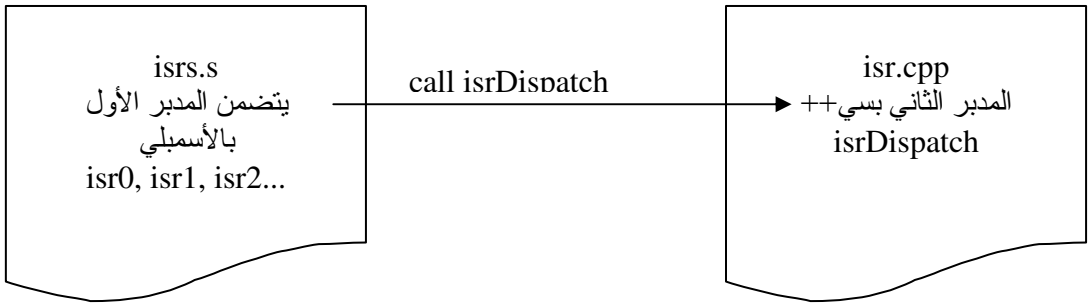
الفرق بين موصف مطب و موصف انقطاع يكمن فقط في كيفية تثبيت المعالج للعلم IF في المسجل eflags بعد تخزين هذا الأخير في الكومة وقبل مناداة مدبر الانقطاع. في موصف انقطاع يقوم بتثبيت IF على 0 (مكافئ للتعليمة cli) مما يعني أن المعالج لن يتلقى أيا من الانقطاعات القابلة للحجب حتى العودة من مدبر الانقطاع بواسطة iret التي تعيد قيمة المسجل eflags المخزنة قبلا و معها القيمة الأصلية ل IF. الأمر يختلف في موصف مطب حيث لا يغير المعالج قيمة IF.

3- كتابة الشفرة

عمليا، و من أجل تنشيط الانقطاعات نحتاج لإنجاز المهام الآتية:

1- إعادة برمجة محكم الانقطاعات 8259A و تحويل المتجهات الخاصة بخطوط الانقطاع IRQ لتفادي النزاع مع متجهات الاستثناءات.
2- بناء جدول موصفات الانقطاعات و ملئه بعناوين مدبري الانقطاعات.
3- تنشيط الانقطاعات عبر التعليمة sti.

في هذا الدرس، سننجز مدبرا على مرحلتين: عندما نتلقى انقطاعا من المعالج، سنقوم أولا بتنفيذ دالة بالأسمبلي مطابقة للانقطاع المعلن (الانقطاع 0 -> الدالة isr0 ...). دالة الأسمبلي ليست سوى غلاف wrapper للمدبر الحقيقي الذي سينجز بسي++. وظيفة دالة الأسمبلي تتلخص أساسا في تخزين المسجلات في الكومة، نداء المدبر الفعلي المنجز بسي++ isrDispatch، بعد العودة من هذا الأخير يتم استرجاع قيمة المسجلات التي سبق تخزينها في الكومة ثم تنفيذ التعليمة iret التي تنهي مدبر الانقطاع.



عندما تتسلم الدالة isrDispatch اليد من دالة الأسمبلي، تستخدم متجهة الانقطاع كمؤشر على جدول يحتوي على عناصر من فئة خاصة Interrupt.

```
class Interrupt {
      public:
            virtual void handle(regs *r, int vector, int errorCode)=0;
};
```

لاحظ استعمال 0= في آخر السطر. سنجعل من Interrupt فئة معنوية abstract class. و على الفئات المشتقة عنها تعريف هذه الطريقة.

بعد استخلاص العنصر المسجل لمعالجة الانقطاع، تقوم الدالة بنداء الطريقة handle لهذا العنصر.

هذا يعني أننا سنضيف على شفرة الدرس السابق ثلاث ملفات:

pic.cpp	- يتضمن شفرة برمجة المحكم 8259A
isrs.s	- يتضمن شفرةالمدبر الأول بالأسمبلي
isr.cpp	- يتضمن شفرة المدبر الثاني بسي++.

الملف pic.cpp يتضمن أربع دوال لبرمجة محكمي الانقطاعات

```
#include "pic.h"

ثابتين لتعريف بوابتي المحكمين
const int pic1= 0x20;
const int pic2= 0xA0;
تقوم بتهيئة المحكمين عبر إرسال ICWs، أنظر المثال السابق

void initPIC() {

      //ICW1: need ICW4, pic2 present, edge detetction
      outb(pic1, 0x11);
      outb(pic2, 0x11);

      /*ICW2: remap pic1 to use vectors from 0x20 (32)
      ***         remap pic2 to use vectors from 0x28 (40)
```

```
	* */
	outb(pic1+1, 0x20);
	outb(pic2+1, 0x28);

	//ICW3: pic2 uses IRQ2
	outb(pic1+1, 4);
	outb(pic2+1, 2);

	//ICW4
	outb(pic1+1, 1);
	outb(pic2+1, 1);
```

حجب جميع خطوط الانقطاع (أنظر التنجيز تحت) لأننا لا نتوفر بعد على مدبرين لها

```
	//mask all irqs initially
	setPIC1Mask(0xF);
	setPIC2Mask(0xF);
}
```

المعامل mask يتضمن بيتات الحجب، كل بيت يكافئ أحد خطوط الانقطاع، 1:تعطل خط الانقطاع، 0:تنشطه

```
// disable irqs specified in mask
void setPIC1Mask(BYTE mask) {
```

لتحديد بيتات الحجب The mask، يجب إرسال كلمة التحكم بالبيانات رقم 1 OCW إلى البوابة 0x21. كل بيت يحدد وضع خط الانقطاع المقابل له

```
	outb(pic1+1, mask);
}
```

مثل الدالة السابقة لكن بالنسبة للمحكم الثانوي

```
void setPIC2Mask(BYTE mask) {
	outb(pic2+1, mask);
}
```

عند الانتهاء من تدبير الانقطاع، يلزمنا إرسال إشارة "نهاية الانقطاع" EOI إلى المحكم. انتبه الخط يبقى معطلا حتى يتم إرسال هذه الاشارة.

```
void acknowledgePIC1() {
```

ترسل إشارة EOI عبر كتابة كلمة التحكم بالبيانات OCW2 في البوابة 0x20 (في حالتنا الكلمة هي 0x20 أي Non specific EOI)

```
	/* EOI is sent by writing the OCW2 (=0x20
	 * non specific EOI in our case)to 0x20 port*/
	outb(pic1, 0x20);
}
```

في حالة المحكم الثانوي يلزمنا إرسال إشارة للمحكم الرئيسي أيضا

```
void acknowledgePIC2() {
	// we must send EOI ti both controllers
	outb(pic1, 0x20);
	outb(pic2, 0x20);
}
```

الملف isr.cpp يتضمن الدوال الخاصة بتدبير جدول موصفات الانقطاعات و إضافة مدبرات handlers هذه الاخيرة

```
#include "system.h"
#include "interrupt.h"
```

تعريف بنية الموصف، انظر الجدول أعلاه

```
struct IDTEntry {
      WORD offset_1;          // offset 0..15
      WORD segment;           // segment selector
      BYTE reserved:5;
      BYTE flags:3;           // 0 for interrupt/trap gates
      BYTE type:5;            // type of gate
      BYTE dpl:2;
      BYTE present:1;
      WORD offset_2;
} __attribute__((packed));
```

البنية التي ستحمل إلى المسجل IDTR

```
struct IDTR {
    WORD limit;
    DWORD base;
} __attribute__((packed));


#define MAX_ENTRIES 256
```

جدول الموصفات

```
IDTEntry idt[MAX_ENTRIES];
```

كما قلنا من قبل، يتم تدبير الانقطاعات على ثلاث مراحل: 1- مثلا عند إعلان متجهة الانقطاع 0، ينقل المعالج التحكم إلى المدبر المسجل في المدخل 0 في ج.م.إ IDT، الذي هو في حالتنا دالة الأسمبلي isr0، هذه الأخيرة بعد تخزين المسجلات في الكومة تقوم بنداء دالة بسي++ isrDispatch . 2- isrDispatch تبحث في الجدول المعرف أسفله handlers عن عنصر من فئة Interrupt (سنراها فيما بعد) في المدخل 0 . ثم تسلمه التحكم عبر نداء الطريقة handle. 3- يقوم بتدبير الانقطاع، بعد الانتهاء يعود إلى isrDispatch التي تعود بدورها إلى isr0. هذه الأخيرة تقوم باسترجاع قيم المسجلات من الكومة و "تنظيفها" عبر التعليمة iret.

```
static Interrupt* handlers[MAX_ENTRIES];
IDTR idtr;
```

تقوم بتثبيت ج.م.إ عبر ملئه ب0 و تحميله إلى المسجل IDTR

```
void IDTSetup() {

      memset((BYTE*)&idt, 0, sizeof(IDTEntry)*MAX_ENTRIES);

      idtr.base = (DWORD)&idt;
      idtr.limit = sizeof(IDTEntry)*MAX_ENTRIES-1;
      asm volatile ("lidt %0" :: "m"(idtr));
}
```

تقوم بملء المدخل vector بعنوان المدبر handler و الأفضلية المطلوبة لنداء هذا الأخير

```
extern "C" void IDTsetHandler(int vector, DWORD handler, int dpl) {
```

vector يجب أن تكون محصورة بين 0 و 255

```
      if(vector < 0 || vector >= MAX_ENTRIES) {
            return;
```

إذا كان العنوان handler يساوي 0 فهذا يعني ملء المدخل ب0 حتى يعرف المعالج أن المدبر غير موجود (عبر البيت present أنظر الجدول أعلاه)

```
	} else if(!handler) {
		memset((BYTE*)&(idt[vector]), 0, sizeof(IDTEntry));
	} else {
		idt[vector].segment = 8;
		idt[vector].offset_1 = handler & 0xFFFF;
		idt[vector].offset_2 = (handler>>16) & 0xFFFF;
		idt[vector].type = 14; //gate's type = interrupt gate
		idt[vector].dpl = dpl & 3;
		idt[vector].present = 1;
	}
}
```

مدير الانقطاعات سي++، كما قلنا أعلاه، ينادى من طرف جميع المدبرين بالأسمبلي. وهو بدوره يقوم بنداء المدبر المسجل في الجدول handlers. دوال الأسمبلي تقوم بدفع العناصر التالية في الكومة: 1- رمز الخطأ (0 في حالة عدم وجوده)، 2- رقم المتجهة و 3- قيمة المسجلات eax, ebx... هذه الأخيرة سنلج إليها بطريقة غير مباشرة عبر البنية regs التي تمت إضافتها إلى الملف system.h (المزيد من الشرح في ملف isrs.s)

```
extern "C" void isrDispatch(regs r, int vector, int errorCode) {
	if(vector < 0 || vector >= MAX_ENTRIES)
		return;
```

نستعمل vectorكمؤشر على الجدول handlers، بعد استعادة العنصر Interrupt ننادي طريقته handle.

```
	if(handlers[vector])
		handlers[vector]->handle(&r, vector, errorCode);
}
```

دالة مساعدة لتسجيل مدبر من فئة Interrupt لانقطاع vector عبر وضعه في الجدول handlers

```
void registerHandler(BYTE vector, Interrupt *interrupt) {
	handlers[vector] = interrupt;
}
```

تقوم بإزاحة مدبر Interrupt من الجدول handlers

```
void unregisterHandler(BYTE vector) {
	handlers[vector] = (Interrupt*)null;
}
```

الملف isrs.s يتضمن شفرة الأسمبلي اللازمة لتدبير الانقطاعات في المرحلة الأولى. الملف يستعمل العديد من المُركّبات macros حتى لا نضطر لتكرار الشفرة نفسها. جزء من هذه الأخيرة معرف في الملف asm.inc

قبل المرور إلى شفرة الملف سنرى بسرعة تنجيز هذه المُركّبات.
المُركّبة PUSHS تقوم بدفع عدة عناصر في الكومة باستعمال تعليمات push متتالية، مثلا

PUSHS eax, ebx, ecx

```
%macro PUSHS 1-*
	%rep %0
		push %1
		%rotate 1
	%endrep
%endmacro
```

%macro تبدأ تعريف المُركّبة و%endmacro تنهيه. بعد %macro نحدد الاسم ثم عدد العوامل التي تأخذها: *-1 توضح أن المُركّبة PUSHS تأخذ على الأقل عاملا واحدا و على الأكثر عددا غير محدد من العوامل (مما يعني أننا نستطيع أن نناديها بأي عدد من العوامل ابتداء من 1).

%rep...%endrep أشبه بالحلقة (..)for في لغة سي، أي أن التعليمات داخل الحلقة ستعاد %0 مرة. %0 كناية عن عدد العوامل التي مدت بها المُركّبة. %1 كناية عن قيمة العامل الأول. مثلا في حال كتبنا PUSHS eax, ebx, ecx %0تساوي 3 و %1 يساوي eax.

%rotate 1 تقوم بإزاحة العوامل خطوة إلى اليسار بحيث يصبح العامل 2 مكان العامل 1، العامل 3 مكان 2 و هكذا. العامل الأول يزاح إلى آخر الصف. في المثال السابق بعد %rotate 1 تصبح العوامل eax,ebx,ecx على شاكلة ebx,ecx,eax. على ضوء ما سبق المُركّبة PUSHS تقوم بالمرور على جميع العوامل و تنتج لكل عامل x الشفرة push x. في المثال السابق المُركّبة تنتج الشفرة التالية

```
push eax
push ebx
push ecx
```

بالمقابل، المُركّبة POPS تنتج لكل عامل x الشفرة pop x

```
%macro POPS 1-*
      %rep %0
            %rotate -1
            pop %1
      %endrep
%endmacro
```

لاحظ استعمال %rotate -1، هذه التعليمة تقوم بإزاحة العناصر من اليسار إلى اليمين. و استعمالها داخل %rep يمكننا من المرور على جميع العوامل لكن في الاتجاه المقلوب أي من الأخير إلى الأول. هذا يعني في حال كتبنا POPS eax,ebx,ecx فإن الشفرة المنتجة تصبح

```
pop ecx
pop ebx
pop eax
```

المُركّبة proc تمكننا من انتاج شفرة تعريف دالة في الأسمبلي قابلة للاستعمال من ملفات سي++

```
%macro proc 1
global %1, _%1
%1:
_%1:
%endmacro
```

لتسهيل القراءة نضيف المُركّبة endproc

```
%define endproc
```

نعرف أيضا المُركّبتين c_call لمناداة دالة من سي++ و c_extern لتعريف رموز خارجية

```
%macro c_call 1
      %ifdef LEADING_USCORE
            call _%1
      %else
            call %1
      %endif
%endmacro
```

```
%macro c_extern 1-*
%rep %0
      %ifdef LEADING_USCORE
            extern _%1
      %else
            extern %1
      %endif
      %rotate 1
%endrep
%endmacro
```

الآن لنمر إلى شفرة الأسمبلي. تحدثنا فيما قبل عن مدبرات الانقطاعات الأولى بالأسمبلي و هذه شفرة-مثال لمدبر المتجهة 0.

```
isr0:
      push dword 0
      push dword 0
      jmp isrCommon
```

السطر الأول يدفع رمز خطأ "مزور" في الكومة. تذكر أنه توجد استثناءات تدفع برمز للخطأ في الكومة قبل مناداة المدبر وأخرى لا تفعل. لكننا نريد للكومة أن تظل متجانسة في جميع الحالات، لذلك نقوم بدفع 0 أولا قبل أن ندفع رقم المتجهة في السطر الثاني. في السطر الثالث نقفز إلى isrCommon و هو كود مشترك لجميع لمدبرات كما سنرى فيما بعد.

شفرة المدبرات الأخرى التي لا تدفع رمز الخطأ مماثلة

```
isr1:
      push dword 0
      push dword 1
      jmp isrCommon
```

بالنسبة للمدبرات التي تدفع رمز الخطأ كالمتجهة 8

```
isr8:
      push dword 8
      jmp isrCommon
```

الآن يمكننا كتابة جميع المدبرات الأخرى لكننا سنكرر نفس الشفرة 47 مرة؟؟ بما أنني مولع بالمُركّبات ☺ كما رأيتم سأستعملها هنا أيضا

```
%macro INTR 1
isr%1:
      push dword 0            ; pushes dummy error code
      push dword %1           ; push the vector
      jmp isrCommon
%endmacro

;; interrupts that push error code
%macro INTR_EC 1
isr%1:
      push dword %1           ; push the vector
      jmp isrCommon
%endmacro
```

والآن هل سنكتفي بهذه المُركّبات و نكتب تباعا

```
INTR 0
INTR 1
...
INTR_EC 8
...
INTR 41
```

47 مرة؟ أم نكتب مُركّبة أخرى تقوم بهذا العمل؟ أظنكم عرفتم الجواب ☺

```
%macro MAKE_INTR 2
%assign i %1
%rep (%2-%1+1)
      INTR i
      %assign i i+1
%endrep
%endmacro
```

الآن يكفي أن نكتب

```
MAKE_INTR 0,7

INTR_EC 8
INTR 9
INTR_EC 10
INTR_EC 11
INTR_EC 12
INTR_EC 13
INTR_EC 14

MAKE_INTR 15, 47
```

عرفنا أن جميع المدبرات تدفع رمز خطأ و رقم المتجهة في الكومة ثم تقفز إلى isrCommon.
لنر الآن ماذا تفعل هذه الأخيرة

```
isrCommon:
```

أولا نخزن المسجلات في الكومة

```
      PUSHS eax,ebx,ecx,edx,edi,esi,ebp,ds,es,fs,gs,ss
```

ثم للتأكد نحمل مسجلات الأقسام بالقيمة الصحيحة. حاليا لا يوجد ما يستدعي ذلك لكن في دروس قادمة قد نحتاج لهذه التعليمات

```
            mov ax, 0x10
            mov ds, ax
            mov es, ax
            mov fs, ax
            mov gs, ax
```

ننقل التحكم لمدبر سي++ isrDispatch، المزيد من الشرح أسفله

```
           c_call isrDispatch
```

بعد العودة من مدبر سي++، نعيد قيمة المسجلات

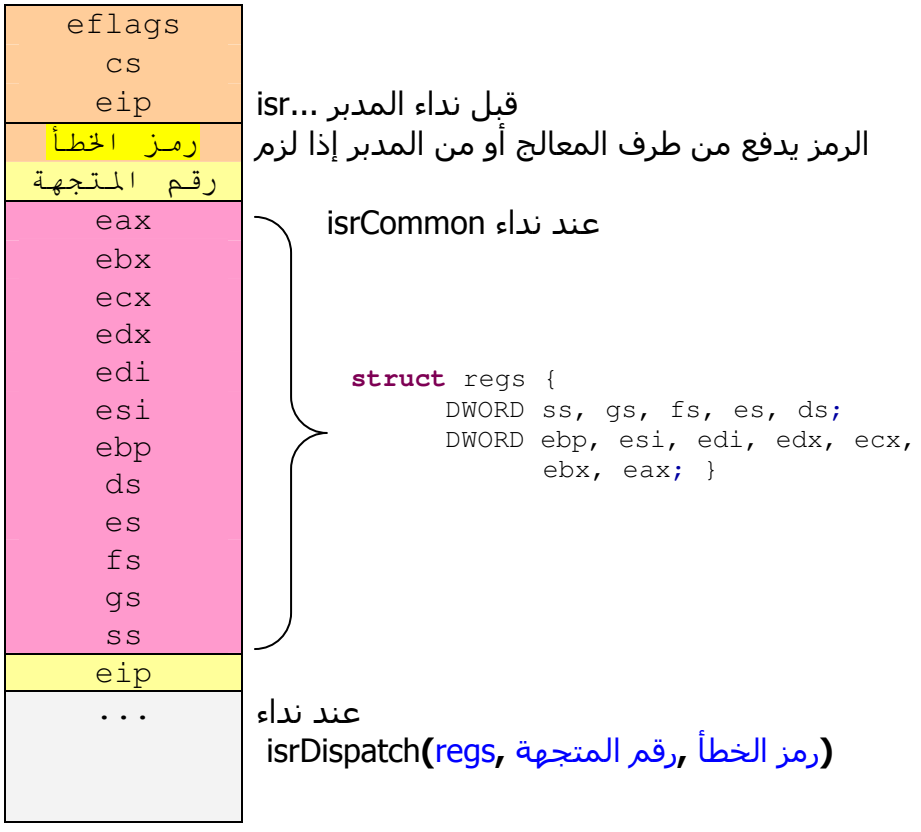
```
      POPS  eax,ebx,ecx,edx,edi,esi,ebp,ds,es,fs,gs,ss
```

```
و ننظف الكومة بإزاحة رمز الخطأ و رقم المتجهة التي تم دفعهما قبلا
        add esp, 8
للعودة من مدبر انقطاع نستعمل iret بدل ret لإزاحة eflags, cs, eip من الكومة دفعة واحدة
        iret
```

لفهم هذه الشفرة جيدا، لنبدأ من الصفر و نتصور شكل الكومة (المفتاح يكمن في فهم عمل الكومة) بعد أن يتسلم مدبر الانقطاع التحكم للتو. المعالج دفع قبلا eflags, cs, eip – و ربما رمز الخطأ – بينما يدفع المدبر : - رمز الخطأ إذا لزم - ، رقم المتجهة (...isr0)، ثم قيمة المسجلات. ثم ننادي على دالة سي++ isrDispatch لنتذكر تعريف هذه الدالة

```
void isrDispatch(regs r, int vector, int errorCode);
```

و الآن لنقارن مع شكل الكومة مع الأخذ بعين الاعتبار أن العوامل تمرر للدوال عبر الكومة و تدفع انطلاقا من آخر عامل



بعد أن أنجزنا مدبرات الانقطاعات بالأسمبلي، يتبقى لنا وضع عناوينها في المدخل المناسب في جدول موصفات الانقطاعات: وضع عنوان isr0 في المدخل 0، isr1 في المدخل 1 ...

لأجل هذا الغرض أنشئنا الدالة setHandler المعرفة في ملف isr.cpp، نذكر هنا بتعريفها

```
void IDTsetHandler(int vector, DWORD handler, int dpl)
```

هذه الدالة تأخذ كعوامل: dpl الأفضلية الأصغر اللازمة لنداء المدبر، handler عنوان المدبر و vector رقم المدخل الذي سيوضع فيه العنوان. سنقوم بنداء هذه الدالة انطلاقا من ملف الأسمبلي. مثلا لتخزين عنوان isr5

```
push 0
push isr5
push 5
call IDTsetHandler
add esp, 12
```

في شفرتنا استعملنا المركّبة setHandler

```
%macro setHandler 1
      push dword 0            ;; dpl=0
      push dword isr%1        ;; handler = isr%1 {isr0, isr1...}
      push dword %1           ;; interrupt vector
      c_call IDTsetHandler
      add esp, 12
%endmacro
```

لاستعمالها مثلا في المدخل 5 : `setHandler 5`

دالة الأسمبلي initISRS تقوم باستعمال هذه المركّبة لملء جميع المداخل من 0 إلى 47 (سنقوم بنداء هذه الدالة من main.cpp لذلك استعملنا المركّبة proc التي تجعلها GLOBAL)

```
proc  initISRS
      push ebp
      mov ebp, esp

      %assign i 0
      %rep 48
            setHandler i
      %assign i i+1
      %endrep

      leave
      ret
endproc
```

أصبح لدينا نظام مكتمل لدعم الانقطاعات. لتفعيله نضيف التعليمات الآتية إلى الدالة main.cp

```
extern "C" void kmain(DWORD magic, multiboot_info *mbi) {
      GDTSetup();

      IDTSetup();
      initPIC();
      initISRS();
      sti();

      Video v;
      v.clear();

      printMemInfo(v, magic, mbi);

      while(1);
}
```

لاحظ بعد نداء initISRS تنشيط الانقطاعات في المعالج عبر الدالة sti. هذه الأخيرة ليست سوى غلاف لتعليمة الأسمبلي sti (تم إضافة cli() و sti() إلى system.h).

الدالة printMemInfo تضم شفرة الدرس السابق التي تطبع معلومات الذاكرة على الشاشة.

3- مثال لتدبير الاستثناءات

فيما يلي سنطور مثالا بسيطا عن تدبير استثناءات المعالج: المدبر يقم فقط عند تلقي استثناء بطباعة وصف قصير للاستثناء، محتوى المسجلات ثم يوقف البرنامج عبر while(1)

أولا لنتذكر ماذا تفعل isrDispatch عند تسلم التحكم: تقوم بالبحث في جدول عناصر من فئة Interrupt. لتسجيل مدبرنا علينا أولا تنجيز فئة مشتقة عن Interrupt ثم تسجيله عبر الدالة المعرفة في isr.cpp

```
void registerHandler(BYTE vector, Interrupt *interrupt)
```

لاشتقاق فئة من Interrupt كل ما نحتاجه هو تنجيز الطريقة handle. فيما يلي نعرض تعريف الفئة Exception :

```
class Exception : public Interrupt
{     Video* out;

public:
      Exception(Video *v);
      void handle(regs* r, int vector, int errorCode);
};
```

و هذا هو تنجيزها:

```
const char *messages[] =
      {
          "Division By Zero",
          "Debug",
          "Non Maskable Interrupt",
            ...
      };

Exception::Exception(Video* v)
{
      out = v;
}


void Exception::handle(regs* r, int vector, int errorCode) {
      out->printf("Exception %s\n", messages[vector]);
      out->printf("eax: %x, ebx: %x, ecx: %x, edx: %x\n",
            r->eax, r->ebx, r->ecx, r->edx);

      while(1);
}
```

مصنع الفئة يقبل عاملا Video *v يتم تخزينه في out. سنستعمله لطباعة جملنا.

الطريقة handle تقوم باستعمال رقم المتجهة vector كمؤشر على الجدول messages الذي يضم النص الخاص بوصف كل استثناء. بعد استخراج النص المناسب يقوم بطباعته مع قيم بعض المسجلات ثم ينهي تنفيذ النواة ب while(1). لاحظ أنه إذا واصلنا التنفيذ فإن المعالج بعد تدبير الانقطاع سيعيد تنفيذ نفس التعليمة التي تسببت في الاستثناء مما سيعيد تنفيذ المدبر مرة أخرى مما سيعطينا عددا لا متناهيا من الجمل على الشاشة.

لاستعمال المدبر نضيف التعليمات الآتية إلى الدالة kmain:

```
...

printMemInfo(v, magic, mbi);

Exception exc(&v);
for (int i = 0; i < 32; ++i) {
      registerHandler(i, &exc);
}
volatile int a = 5/0;
```

هنا نقوم بصنع عنصر من Exception ثم تسجيله كمدبر لكل متجهات الاستثناء من 0 إلى 31.

بعد ذلك نقوم بخطأ مقصود عبر قسمة 5 على 0 مما سيتسبب في استثناء Div by 0. الشيء الذي سيؤدي لتنفيذ مدبر الاستثناء. لاحظ استعمال الكلمة volatile حتى نمنع المترجم من محاولة تصحيح الخطأ بإزاحة التعليمة.

للترجمة نستعمل نفس makefile الدرس السابق مع إضافة الملفات الجديدة إلى القائمة OBJS

```
CPPFLAGS = -c -fno-builtin -fno-rtti -fno-exceptions

.PHONY:  clean

OBJS = multiboot.o main.o video.o gdt.o isr.o isrs.o exception.o pic.o


all: kernel.elf

kernel.elf: $(OBJS) link.lds
      ld -T link.lds $(OBJS) -o kernel.tmp
      objcopy -O elf32-i386 kernel.tmp kernel.elf
      $(RM) kernel.tmp


clean:
      $(RM) $(OBJS)


%.o: %.cpp
      g++ $(CPPFLAGS) $< -o $@


%.o: %.s
      nasm -f elf  -DLEADING_USCORE $< -o $@
```

لا تنسى إزاحة السطور بالأحمر) إذا كنت تشتغل في لينوكس (استعمل DLEADING_USCORE فقط إذا كان مترجمك يضيف الحرف _ على بداية أسماء الرموز.

عند الترجمة، سيعطي المترجم تنبيها عندما يصادف عملية القسمة على 0 لكن الخطأ في حالتنا طبعا مقصود.

بعد الترجمة. انقل الملف المنتج kernel.elf إلى أصل root صورة القرص المرفقة مع الدرس (انظر الملحق) .
الصورة التالية تعطينا نتيجة التنفيذ:

```
Assalamou Alaikoum from grub
Lower memory = 639KB
Upper memory = 31744KB
Boot device = floppy A

Exception Division By Zero
eax: 0x5, ebx: 0x2bdc0, ecx: 0x15, edx: 0x0
```

- مثال لتدبير الانقطاعات الصلبة: المنبه Programmable Interval Timer PIT

سنوضح هنا مثالا بسيطا لتدبير الانقطاعات الصلبة عبر تدبير انقطاعات المنبه.

يستعمل المنبه (رقاقة الPIT نموذج 8253 أو 8254) لإنتاج انقطاعات دورية في الحاسب. يتم ذلك عبر بلور متذبذب ينتج نبضات ذات تردد قيمته 1193180 نبضة في الثانية (1.19MHz).

لبرمجة المنبه نستعمل قيمة تسمى قاسم التردد frequence divisor. مثلا عند إقلاع الحاسب يستعمل البيوس قاسما قيمته 65536 مما يعطينا 1193180/65536=18.2. أي 18نبضة في الثانية. يتوفر المنبه على ثلاث مسجلات لتخزين ثلاث قيم مختلفة. هذه المسجلات مرتبطة بثلاث قنوات: القناة (والمسجل) 0 مرتبط بخط الانقطاع 0 و يمكننا استعمالها كما نشاء. القناة 1 تستعمل من طرف الهاردوير (DMA Controller) و القناة 2 مرتبطة ببوق النظام System Speaker.

فيما يخصنا سنستعمل القناة 0. المسجل المرتبط بها يوجد في البوابة الفرعية 0x40.

قبل وضع القاسم المراد في المسجل، يجب علينا إعداد المنبه قبلا عبر مسجل التحكم الموجود في البوابة 0x43.

فيما يلي شفرة تدبير المنبه عبر تعريف الفئة Timer المشتقة عن Interrupt

التعريف:

```
class Timer: public Interrupt
{     WORD _phase;
      Video* out;

public:
      Timer(Video *v);
```

```
	void handle(regs* r, int vector, int errorCode);
	void setPhase(WORD phase);
}
```

و التنجيز:

```
nb لتخزين عدد تكات المنبه و sec لتخزين عدد الثواني التي مرت

static long long nb=0;
static int sec=0;

التردد القاعدي للمنبه

static int base = 1193180;

Timer::Timer(Video* v)
{
	out = v;
_phase التردد الذي سنستخدمه في مثالنا

	_phase = 20;

الطريقة setPhase تبرمج ذبذبة المنبه، انظر أسفله

	setPhase(_phase);
}

void Timer::setPhase(WORD phase) {

	_phase = phase;

	base/divisor=phase => divisor=base/phase  للحصول على قاسم التردد divisor. البرهان الريلضي

	int divisor = base / phase;

أولا نعطل الانقطاعات

	cli();

إعداد المنبه بكتابة البوابة 0x43. 0x34 (00110100) تعني:
:00  نريد الكتابة في المسجل 0
:11  نريد كتابة البايت الأضعف متبوعا بالبايت الأقوى
:010  استعمل الوضع rate Generator مما ينتج انقطاعات متكررة و منتظمة
:0  استعمل الحساب الثنائي (يتيح استخدام قاسم أقصاه 65536)

لمعرفة المزيد أنظر الـdatasheet الخاصة ب 8253

	outb(0x43, 0x34);

كتابة البايت الأضعف

	outb(0x40, divisor & 0xff);

ثم البايت الأقوى

	outb(0x40, divisor>>8);

نعيد تنشيط الانقطاعات

	sti();
}
```

```
void Timer::handle(regs* r, int vector, int errorCode) {
```

المنبه ينتج عدد انقطاعات في الثانية مساو ل _phase. نخزن عدد "تكّات" المنبه في nb

```
	nb++;
```

عندما نكمل ثانية واحدة

```
	if((nb  % _phase) == 0) {
```

نخزن إحداثيات الشاشة الحالية، نكتب عدد الثواني التي مرت في الموقع (0,24) في الشاشة ثم نعود للموقع السابق

```
		int x = out->x();
		int y = out->y();
		out->moveTo(0,24);
		out->printf("%u secondes elapsed", ++sec);
		out->moveTo(x,y);
	}
```

انتبه، علينا إرسال EOI إلى المحكم الرئيسي و إلا لن نتوصل أبدا بانقطاعات أخرى من هذا الخط

```
	acknowledgePIC1();
}
```

لتسجيل Timer كمدبر لخط الانقطاع 0 (أي متجهة الانقطاع 32). نضيف التعليمات الآتية لشفرة الدالة kmain

```
...
Exception exc(&v);
for (int i = 0; i < 32; ++i) {
	registerHandler(i, &exc);
}
//volatile int a = 5/0;
```

ننشئ عنصرا Timer ثم نسجله كمدبر لمتجهة الانقطاع 32 (أي خط الانقطاع 0 في محكم الانقطاعات الرئيسي)

```
Timer t(&v);
registerHandler(32,&t);
```

ننشط خط الانقطاع 0 وحده (0xE = 11111110) كل الخطوط المثبتة على 1 معطلة

```
setPIC1Mask(0xFE);

while(1);
```

و هذا ملف ال makefile

```
CPPFLAGS = -c -fno-builtin -fno-rtti -fno-exceptions

.PHONY:  clean

OBJS = multiboot.o main.o video.o gdt.o isr.o isrs.o exception.o pic.o
timer.o


all: kernel.elf

kernel.elf: $(OBJS) link.lds
```

```
	ld -T link.lds $(OBJS) -o kernel.tmp
	objcopy -O elf32-i386 kernel.tmp kernel.elf
	$(RM) kernel.tmp
	"C:\Program Files\WinImage\winimage.exe" floppy.img /i kernel.elf /h
	 objdump.exe -D kernel.elf > log.txt

clean:
	$(RM) $(OBJS)


%.o: %.cpp
	g++ $(CPPFLAGS) $< -o $@

video.cpp: system.h video.h
main.cpp: system.h
exception.cpp: exception.h system.h
timer.cpp: timer.h system.h pic.h
isr.cpp: isr.h system.h


%.o: %.s

nasm -f elf  -DLEADING_USCORE $< -o $@
```

بعد التنفيذ سنرى النتيجة الآتية

ملحوظة: التردد الذي اخترناه تقريبي و لا يكافئ الثانية الواحدة مائة بالمائة.

تلخيص لما قمنا به:

في هذا الدرس تعرفنا إلى الانقطاعات في الx86 و كيف يتم تدبيرها في الوضع المحمي.
تعرضنا أيضا لبرمجة محكم الانقطاعات 8255A و المنبه 8253.

رأينا أيضا كيف يمكن استعمال الأسمبلي مع سي++، و كيف يتك تدبير الكومة لهذا الشأن.

بالمقارنة مع نواة أول درس فقد تقدمنا خطوات لا بأس بها. لكنها البداية فقط بإذن الله.

في الدرس القادم سنبدأ دراسة مجال مهم جدا و هو تدبير الذاكرة.

ياسين الوافي 25/01/2006 23h 35mn

ملحق : استعمال المحاكي Bochs مع صورة للقرص لتجريب الشفرة

فيما يلي توضيح لكيفية استعمال محاكي للحاسب الشخصي بدل حاسب حقيقي لتجريب الشفرة. اخترت هن المحاكي Bochs و هو محاكي جيد لأغراض مبرمج النواة بالإضافة إلى أنه مجاني و مفتوح المصدر.
أفترض هنا أن مستعملي ويندوز يتوفرون على cygwin مثبت قبلا.
المزيد من المعلومات في

http://www.mega-tokyo.com/osfaq2/index.php/Working%20with%20Disk%20Images

أولا سأوضح كيف تستعمل صورة للقرص المرن بدل قرص حقيقي (جيد بالنسبة لمن يتوفر على قارئ للقرص المرن). سأتحدث عن ويندوز ثم لينوكس.

استعمال صور القرص في ويندوز:

البرنامج VFD (http://chitchat.at.infoseek.co.jp/vmware/vfd.html#download) يمكننا من استعمال صور القرص من ويندوز كما لو كانت حقيقية. حمل الملف vfd21-050404.zip ثم بعد استخلاص الملفات:

1- نفذ البرنامج VFDWIN.EXE، ثم في الزاوية Driver (الزاوية الثالثة) اختر Start type : auto ثم اضغط على التوالي Install و start.
2- في الزاوية Shell انقر الخانة الأولى .... Context menu, drag & drop
3- في الزاوية Association اختر الخانة flp. اضغط Apply (أسفل النافذة).
4- الآن لفتح الملف في ويندوز و نقل الملفات يكفي ببساطة أن تنقر نقرة مزدوجة على ملف الصورة المرفق مع الدرس و استعماله كأي قارئ عادي في ويندوز (القارئ يأخذ – مسبقا- الحرف B و رغم أن ويندوز يكتب أمام القرص "5 إلا أن القارئ يستعمل من طرف VFD بطريقة صحيحة).

استعمال صورة القرص في لينوكس:

نستعمل الواجهة loopback devices

1- اكتب التعليمة

```
losetup /dev/loop0 floppy.flp
```

مع floopy.flp اسم الملف المرفق مع الدرس

2- تركيب الصورة

```
mount -t msdos /dev/loop0 /mnt/myfloppy
```

3- الآن يمكنك استعمال المجلد /mnt/floppy بطريقة عادية.

استعمال Bochs:

1- التثبيت: حمل الشفرة الأخيرة من هذه الصفحة
http://bochs.sourceforge.net/getcurrent.html

بعد استخراج الملفات المضغوطة، ادخل المجلد الذي استخرجت فيه الملفات و اكتب التعليمات الآتية:

```
./configure --enable-x86-debugger --enable-gdb-stub --enable-apic  --enable-vbe
make
make install
```

إذا تم كل شيء على ما يرام، فقد تمت الترجمة و نقل الملفات إلى /usr/local/. الجدول التالي يوضح شجرة الملفات انطلاقا من /usr/local

bin	binary executables (bochs, bxcommit, bximage)
lib/bochs/plugins	plugins (if present)
man/man1	manpages for installed binaries
man/man5	manpage for the config file (bochsrc)
share/bochs	BIOS images, VGABIOS images, keymaps
share/doc/bochs	HTML docs, license, readme, changes, bochsrc sample

بعد التثبيت عليك بتعريف متغير نظام جديد environment variable. في لينوكس يمكنك إضافة التعليمة الآتية إلى ملف الshell الخاص بك

```
export BXSHARE=/usr/local/ share/bochs
```

في ويندوز، على سطح المكتب انقر بيمين الفأرة على أيقونة My Computer و اختر Properties
في الزاوية Advanced اضغط Environment variables. ثم في النافذة التي تظهر لك. انقر على new ثم أدخل اسم و قيمة المتغير كالتالي

```
Name : BXSHARE
Value : C:\cygwin\usr\local\share\bochs
```

بفرض أن cygwin مثبت في المجلد C:\cygwin

التهيئة: ادخل إلى المجلد share/doc/bochs و انقل الملف bochsrc-sample.txt إلى المجلد الذي تشتغل فيه. ثم بدل اسمه إلى bochs-ar.txt (الملف النهائي مرفق مع الدرس). الملف يضم عدة مداخل – كل مدخل عبارة عن مجموعة سطور متتالية لتهيئة أحد الفروع مثلا -

ابحث عن المدخل الخاص بالأقراص المرنة و اجعله كما يلي

```
#floppya: 1_44=/dev/fd0, status=inserted
floppya: image="path/floppy.flp", status=inserted
#floppya: 1_44=/dev/fd0H1440, status=inserted
#floppya: 1_2=../1_2, status=inserted
#floppya: 1_44=a:, status=inserted
#floppya: 1_44=a.img, status=inserted
#floppya: 1_44=/dev/rfd0a, status=inserted
```

(الخيارات التي تبدأ ب # معطلة).

لا تنس إبدال path/floppy.flp باسم ملف صورة القرص الذي تود الاقلاع منه. مثلا (بفرض أن طريق الملف هو /home/yassin/tuts/floppy.flp).

```
floppya: image=/home/yassin/tuts/floppy.flp, status=inserted
...
```

مثال آخر (طريق الملف هو C:\yassine\tuts\floppy.flp)

```
#floppya: 1_44=/dev/fd0, status=inserted
floppya: image=C:\yassin\tuts\floppy.flp, status=inserted
...
```

علينا أيضا تعطيل اختيارات القرص الصلب لأننا لن نستعمله، ابحث عن المداخل ata و تأكد من تعطيلها جميعا بوضع enabled=0

```
ata0: enabled=1, ioaddr1=0x1f0, ioaddr2=0x3f0, irq=14
ata1: enabled=0, ioaddr1=0x170, ioaddr2=0x370, irq=15
ata2: enabled=0, ioaddr1=0x1e8, ioaddr2=0x3e0, irq=11
ata3: enabled=0, ioaddr1=0x168, ioaddr2=0x360, irq=9
```

بعد ذلك ابحث عن المدخل :ata0-master و عطله بإضافة # في أوله.

```
#ata0-master: type=disk, mode=flat, …
#...
```

يجب أيضا إعداد المنبه. –مسبقا- منبه Bochs لا ينتج نبضات بالتردد الصحيح لأجل ذلك ابحث عن المدخل clock و جعله كالآتي

```
# Default value are sync=none, time0=local
#=========================================
=========================
clock: sync=realtime, time0=local
```

أخيرا، ابحث عن المدخل boot و اجعله كالآتي

```
boot: floppy
#boot: disk
```

الآن للإقلاع باستعمال الملف bochs-ar.txt انتقل إلى المجلد الذي فيه و اكتب التعليمة

```
bochs –q –f bochs-ar.txt
```