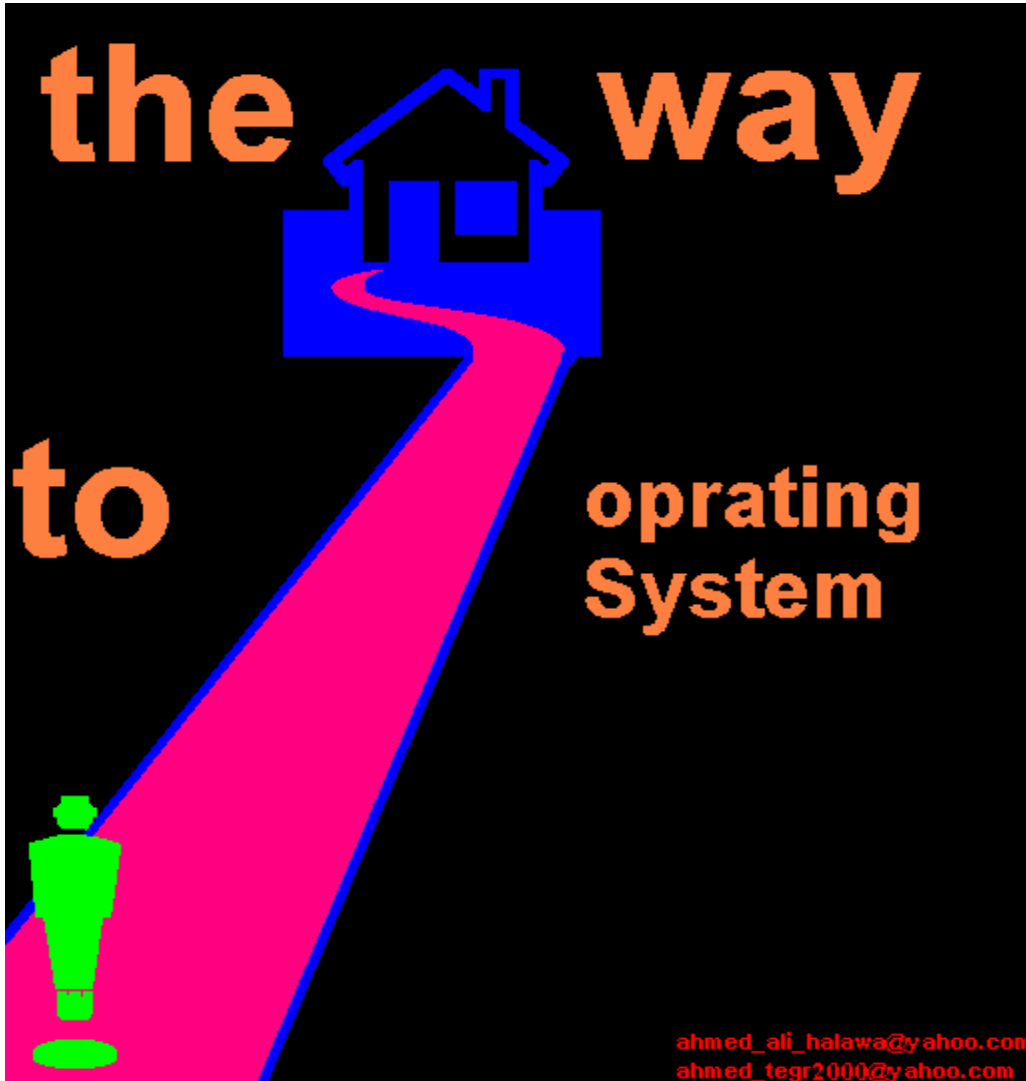


بسم الله الرحمن الرحيم



الدرس الرابع: "الاستثناءات والإنقطاعات"

- ١- مقدمه عن الانقطاعات.....
- ٢- مقدمه عن الاستثناءات.....
- ٣- الانقطاعات الصليه IRQ.....
- ٤- جدول الانقطاعات IDT.....
- ٥- ملف الانقطاعات idt.h.....
- ٦- ملف الانقطاعات idt.cpp.....
- ٧- ملف المدبرات الاولى isrs.asm.....
- ٨- ملف المدبر الاساسي Interrupt.h + isr.cpp.....
- ٩- ملف الاستثناءات exp.h.....
- ١٠- ملف الاستثناءات exp.cpp.....
- ١١- دعم ال timer.....
- ١٢- دعم الاله الكاتيه.....
- ١٣- تجريب النواه.....
- ملحق الدرس برنامج .bochs.....

► مقدمه عن الانقطاعات ◀

اولاً اريد ان اسأل سؤالاً بسيطاً وهو:
ماذا تظن انه يحدث بعد ضغطك علي احد ازرار الاله الكاتبه ☺ ؟
الإجابة :
يتم كتابه الحرف الذي ضغطنا عليه.

اسف ☹ فهذه إجابته خاطئه او شبه صحيحه ولكن الإجابة الصحيحه هو ابتعاد المعالج عن أي شئ يقوم بتنفيذه ويقوم بمناداه نظام التشغيل ويسأله ماذا يفعل في هذا الحرف المضغوط؟ ومن هنا يمكننا تعريف الانقطاعات

بـ:

هي احداث تنبيه المعالج وتخبره بأن يوقف جميع العمليات التي يجريها و يقوم بتنفيذ شئ ما لهذا الحدث الذي تم وتلك الانقطاعات مثل ضغطه زر في الاله الكاتبه او في الماوس وينتج عن ذلك تحول في سير التنفيذ الحالي نحو برنامج اخر يسمى مدبر الانقطاعات (حتي تكون التسميه موحده سوء في الخوارزمي او في كتابي)
Interrupt service routen او Isr وهذا المدبر هو الذي يقوم بطباعه الحرف علي الشاشة في مثالنا السابق وبعد تنفيذه يعود سير البرنامج إلي البرنامج الذي كان يعمل قبل الإنقطاع.

في معالجات انتل توجد ٣ انواع من الانقطاعات وهي كالتالي
١-الاستثناءات:

- تعرف الاستثناءات علي انها اخطاء تحدث وقت التنفيذ مثل عدد ما علي صفر او محاوله احد البرامج للوج إلي منطقه اعلي منه في درجه الحماية
٢-الانقطاعات الصلبه:
من اسمها انقطاعات تحدث بسبب الاجسام الصلبه والمقصر بها هنا هي ال hard ware
من ضغطه مفتاح في الاله الكاتبه إلي اخره
٤ - انقطاعات برمجيه:
تحدث بسبب التعليمات البرمجيّه مثل into – int3 – int

► الاستثناءات ◀

الاستثناءات عباره عن انقطاعات بسبب الاخطاء البرمجيّه مثل القسمه علي صفر
توجد ٣ مصادر للاستثناءات

- ١ - اخطاء مرصوده من قبل المعالج أي ان المعالج هو الذي يقيم باكتشافها
مثال: هناك برنامج يقوم بجعل المستخدم يدخل رقما ثم يقوم بطرح الرقم ٥ من الرقم المدخل ثم يقسم الرقم ٨ علي الناتج ؟ هنا لا يمكن حدوث استثناء إلي في حاله واحده وهي ادخال المستخدم الرقم ٥
- ٢ - استثناءات برمجيه : مثل استخدام البرنامج للتعليمه int3 ومعناها نقطه توقف في البرنامج
- ٣ - استثناءات صلبه : أي انها تحدث نتيجة حدوث أي خطأ في الهارد وير

فلنقل ان هناك خطأ برمجي قد تم ماذا يحدث منذ حدوث الخطأ إلي ان يتم تصليحه؟

اولاً لنفترض الخطأ التالي

Int a = 5/ 0;

عندها سيقوم المعالج بإعلان استثناء القسمة علي صفر
يقوم المعالج بدفع بعض القيم إلي الكومه وهي علي الشكل التالي

Eflags // الاعلام

CS // القسم الذي كان يعمل فيه المعالج

EIP // العنوان الذي يحمل الاستثناء

Error code // رمز الخطأ (في بعض الاستثناءات)

يقوم المعالج بعد ذلك بالبحث في جدول يسمى **جدول الانقطاعات** في هذا الجدول يتم حفظ جميع المعلومات حول المدبر من القسم الذي يوجد فيه المدبر والعنوان بالتحديد درجة حمايه وهل يمكن لبرنامج المستخدم استخدام هذا المدبر عن طريق التعليمات البرمجيه ام لا

في حاله عدم وجود مدبر للخطأ اما ان يقوم المعالج بإعاداه التشغيل او العوده لتنفيذ البرنامج مره اخر في حاله العكس يقوم المعالج بالقفز إلي العنوان الذي يوجد فيه المدبر واستخدام القسم الخاص به يقوم المدبر بتصحيح الخطأ ثم يقوم بالخروج عن طريق تعليمه الخروج iret لان هذه التعليمه تقوم بإستخرج عنوان Eip ورقم القسم من المكس
لاحظ ان رمز الخطأ يتم إزالته من المكس يدوياً أي عن طريق الكود.

السؤال المطروح هنا اين يكمل المعالج البرنامج هل قبل الخطأ ام بعده ام؟؟؟؟
١- في حاله كون الاستثناء عن طريق خطأ يقوم المعالج بحفظ رقم Eip الخاص بتعليمه الخطأ ويعطي للمدبر الفرصه للقيام بتصحيح الخطأ

٢- في حاله كون الاستثناء بسبب مطب أي تعليمه لا يمكن تنفيذها فإن رقم Eip يكون للتعليمه المواليه مثال علي ذلك عندما يعمل برنامج حديث علي جهاز ذا معالج قديم فإن المعالج يري تعليمات لا يعرفها فيضطر إلي ان يكمل بدلاً من التوقف.

٣- الإخفاقات أي استحاله المتابعه رقم Eip غير محدد بدقه.

والجدول التالي يحتوي علي قائمه بالإستثناءات + رمز الخطأ

الوصف	رمز الخطأ
Division by zero Exception	لا
Debug Ecxeption	لا
Non maskable interrupt Exception	لا
Breakoint Exception	لا
into detetcted overflow Exception	لا
Out of bounds Exception	لا
Invalid opcode Exception	لا
No Coprocessor Exception	لا
Double fault Exception	نعم
Coprocessor segment overrun Exception	لا
Bad TSS Exception	نعم
Segment not present Exception	نعم
Stack fault Exception	نعم
General protection fault Exception	نعم
Page fault Exception	نعم
Unkown interrupt Exception	لا

لا	Coprocessor fault Exception
لا	(+Alignment check Exception (only on 486
لا	Machine check Exception (only on (+Pentium/586
لا	Reserved Exceptions

▶ الانقطاعات الصلبة IRQ ◀

تأتي الانقطاعات الصلبة بسبب الأجهزة الموصلة باللوحة تمر هذه الانقطاعات قبلاً علي مايسمي محكم الانقطاعات الصلبة pic حتي لا تحدث مشاكل اثناء التنفيذ لأحد البرمج .

لماذا ذلك ال pic ؟

اولاً احب ان اذكر بعض المعلومات عنه ال pic عباره عن رقاقه تسمي 8257A معظم الاجهزه الشخصيه القديمه تحتوي رقاقتين من هذا النوع الرقاقه الاول اعدت للتوصيل مع ٨ رقاقت ثانويه ولكن هذا يحدث في الاجهزه الضخمه ، الاجهزه الحديثه تحتوي علي ما يسمي APIC هي تقريبا نفس الرقاقه السابقه ولكن متطوره . محكم الانقطاعات يحتوي علي ٨ خطوط كل خط موصل مع فرع موصل باللوحة الام وبما ان الاجهزه الشخصيه القديمه تحتوي علي رقاقتين فبذلك نحن نحوي علي ١٦ خط هناك خط في المحكم الاول او الرقاقه الاولى متصل بالمحكم الثاني لماذا ؟ لان المحكم الاول يتحكم في المحكم الثاني حتي لا يقوم المحكم الثاني بطلب عمل انقطاع صلب من تلقاء نفسه وبالتالي يتضح لنا ان مهمه هذه المحكمات هي عمل انقطاعات لاحد الفروع الموصله باللوحة الام ويتم طلب انقطاع لكل فرع حسب اولويته ويتم تيرتيب الاوليه كما بالشكل التالي

المحكم	الجهاز المستعمل
المحكم الرئيسي	٠ - منبه النظام ١ - لوحة المفاتيح ٢ - موصل مع المحكم الثانوي (الرقاقه ٢) ٣ - serial port 2 ٤ - serial port 1 ٥ - الفرع المتوازي ٢ ٦ - محكم القرص المرن ٧ - الفرع المتوازي ١
المحكم الثانوي	٨ - منبه النظام ٩ - غير مستعمل ١٠ - غير مستعمل ١١ - غير مستعمل ١٢ - الفأره ١٣ - المعالج الرياضي ١٤ - القرص الصلب الرئيسي primary ١٥ - القرص الصلب الثانوي secondary

للتوضيح اكثر ما يمكن عن المحكمات سوف اقوم بعمل مثال لنفرض ان المستخدم ضغط علي زر في لوحة المفاتيح فإن لوحة المفاتيح تطلب من المحكم الاساسي (لانها موصله مع احد فروع) ان يقوم بعمل إنقطاع لها يستقبل المحكم الاساسي عده طلبات ومن بن هذه الطلبات طلب لوحة المفاتيح فيقوم بترتيب الطلبات حسب الجدول السابق لنفرض ان منبه النظام لا يطلب إنقطاع فيكون بالتالي اول طلب انقطاع للوحة بعد ذلك يرسل المحكم طلب

إنقطاع لأحد الخطوط عن طريق إرسال اشارته int فيرد المعالج بعد فتره وجيزه (لاتتعدى الثانيه من الان فصاعداً سنعمل علي مستوي الجزء من الثانيه) بإشارته INTA أي مسموح بإرسال رقم مدير الانقطاع لنفرض اننا نمتلك ١٠ مدبرات وكان المدير الخاص بلوحه الكتابه هو رقم ١ فإن المحكم يرسل الرقم ١ إلي المعالج يقوم المعالج بتنفيذ المدير بعد انتهاء المدير من عمله عليه إرساله إشارته Eio لأحد المحكمين او كلاهما حسب من طلب تشغيله وهذه الاشاره كالتالي

```
Mov ax,0x20
Mov dx,0x20
Out dx,ax
//////////
outp(0x20,0x20);
```

والعوده إلي نفس دوره مع كل طلب إنقطاع

ماذا عن المحكم الثانوي؟ تقريباً مثل المحكم الاساسي والفرق هنا هو ان المحكم الثانوي لا يستطيع طلب انقطاع من نفسه بل لابد من ان يجعل المحكم الاساسي يطلب له انقطاع ثم يقوم بإرسال رقم مدير الانقطاع

سؤال ماذا يحدث في حاله عدم إرسال إشارته EIO ؟
لن نستقبل أي انقطاع من هذا الخط وجميع الخطوط الأخرى ستعمل

◀ جدول الإنقطاعات IDT ▶

جدول مثله مثل ال GDT يختلف في بعض الامور يتكون هذا الجدول من ٢٥٦ مدخل يحتوي كل مدخل علي معلومات عن احد المدبرات مثل عنوانه والقسم الذي يعمل فيه إلخ يتكون الجدول من التالي

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..31 البعد																P	DPL	0	1	1	1	0	0	0	0								
Segment selector محدد القسم																Offset 0..15 البعد																	
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		

البيئات الوصف

من إلي
١٥ ٠

اول ١٦ بت من العنوان وبما اننا نعمل في النمط المحمي فإن طول العنوان = ٣٢ بت
يمكن الحصول عليها عن طريق (address & 0xFFFF)

رقم القسم الذي يعمل في المدير ٣١ ١٦

محفوظه للمعالج يمكننا ان نضع فيها صفراً فقط (في اول مره) ٣٦ ٣٢

دائماً صفر لماذا ؟ انتل اختارت ذلك ولا نعلم لماذا ٣٩ ٣٧

معناها ان هذا المدير خاص بإنقطاع وليس بمطب (بيئات المطب = ٠١١١) ٤٤ ٤٠

درجه الحماية التي نريد ان نستخدم للمدير فمثلاً هناك مدبرات يمكن ان نجعلها تستخدم عن طريق التعليمه Int = ٠ اعلي درجه أي لا يمكن استخدامها عن طريق int ٤٦ ٤٥

البت P معناه هل القسم الذي يعمل فيه المدير موجود في الذاكره ام لا ٤٨ ٤٧

آخر ١٦ بت من العنوان ويمكن الحصول عليها عن طريق ٦٣ ٤٨

(address >> 16) & 0xFFFF

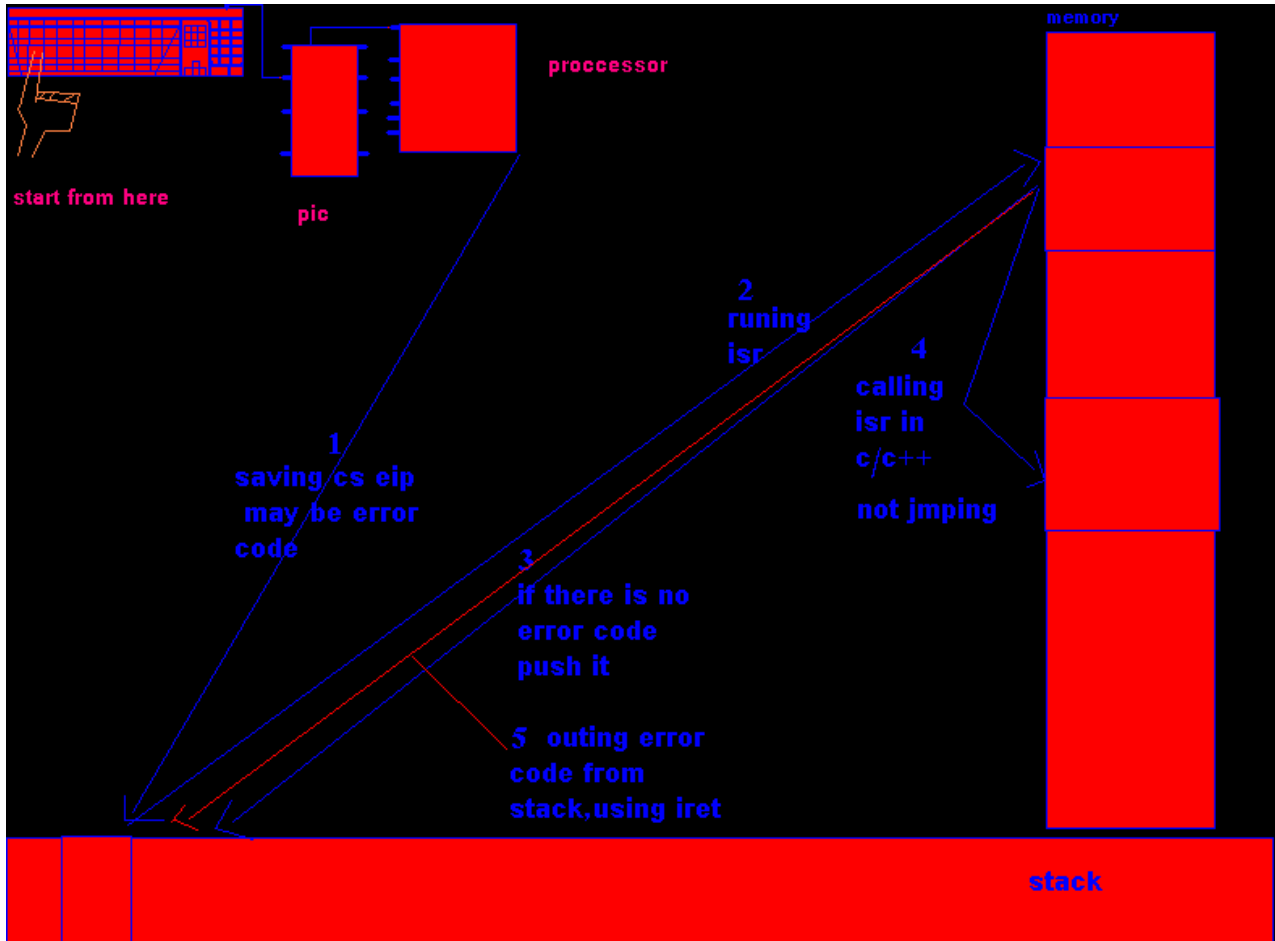
سؤال كيف اقوم بإعلام المعالج بوجود مثل هذا الجدول ؟

اولاً لا بد من عمل متغيرين الاول الاول يحتوي علي طول الجدول والثاني يحتوي علي عنوان الجدول في الذاكره ويتم تحميل عنوان هذين المتغيرين عن طريق التعليمه

LIDT [address]

سؤال هل بعد تحميلي لهذا الجدول سوف تعمل هذه الانقطاعات تلقائياً؟
طبعاً لا فنحن مضطرون إلي تفعيلها يدوياً عن طريق التعليمه STI

قبل البدء اود ان اذكر بعض الاشياء وهي عندما كنت اشرح كنت اكنفي بقولي ويقوم المعالج بتنفيذ المدير ولكن الحقيقه ان هناك تفاصيل اخري تحت هذه الكلمه وهي عندما يستلم المعالج رقم المدير يقوم بالبحث في جدول الانقطاعات (IDT) ما إذا كان هناك مدير لهذا الانقطاع ام لا في حاله عدم وجود مدير يعود ويكمل سير التنفيذ في حاله وجود مدير يقوم بندائه أي انه نادي مدبرنا الخاص لذا سوف اشرح مدبرنا الخاص بما اننا نمتلك استثناءات منها من يقوم بدفع رمز الخطا ومن لا يقوم بذلك لذا فنحن مضطرون إلي صنع رمز خطأ وهمي حتي نحفظ المساحه ونضمن سرعه في التنفيذ وبم اننا سوف نقوم بعمل دفع لبعض القيم لذا سوف احتاج إلي تعليمات الاسمبلي لذا سوف نقسم المدير إلي قسمين الاول مدير اولي يقوم بدفع رمز خطأ إفتراضي ويقوم بمناداه المدير المكتوب بالسي/سي++ ثم يعود يقوك بإزاله رمز الخطا الإفتراضي ويستخدم تعليمه خاصه للخروج تسمي IRET تستخدم هذه التعليمه عندما يكون رقم القسم والعنوان محفوظين في المكس انظر الصوره لتتعرف اكثر



◀ ملف الانقطاعات idt.h ▶

ملف الانقطاعات يحتوي علي الدوال الاساسيه وتكوين الجدول السابق إلخ

IDT.h

```
#ifndef _H_idt_table_
#define _H_idt_table_
#include "system.h"
#include "interrupt.h"
struct idt_entry
{
    ushort offset_1;
    ushort sel;

    uchar res:5;
    uchar zero:3;
    uchar type:5;
    uchar dpl:2;
    uchar pres:1;

    ushort offset_2;
}__attribute__((packed));
struct idt_ptr
{
    ushort limit;
    uint base;
}__attribute__((packed));
/*****/
void idtsetup();
void set_isrs();
void reg_handle(int num,interrupt *handle);
void del_handle(int num);
void picsetup();
void Eio1();
void Eio2();
/*****/
extern "C" void isr0();
extern "C" void isr1();
extern "C" void isr2();
extern "C" void isr3();
extern "C" void isr4();
extern "C" void isr5();
extern "C" void isr6();
extern "C" void isr7();
extern "C" void isr8();
extern "C" void isr9();
extern "C" void isr10();
extern "C" void isr11();
extern "C" void isr12();
extern "C" void isr13();
extern "C" void isr14();
extern "C" void isr15();
extern "C" void isr16();
extern "C" void isr17();
```

```

extern "C" void isr18();
extern "C" void isr19();
extern "C" void isr20();
extern "C" void isr21();
extern "C" void isr22();
extern "C" void isr23();
extern "C" void isr24();
extern "C" void isr25();
extern "C" void isr26();
extern "C" void isr27();
extern "C" void isr28();
extern "C" void isr29();
extern "C" void isr30();
extern "C" void isr31();
extern "C" void isr32();
extern "C" void isr33();
extern "C" void isr34();
extern "C" void isr35();
extern "C" void isr36();
extern "C" void isr37();
extern "C" void isr38();
extern "C" void isr39();
extern "C" void isr40();
extern "C" void isr41();
extern "C" void isr42();
extern "C" void isr43();
extern "C" void isr44();
extern "C" void isr45();
extern "C" void isr46();
extern "C" void isr47();

```

```

/*****/
#endif

```

اول ثلاث اسطر لابد وانهم مفهومان يقوم بعمل بنيه اوليه تحمل محتويات الجدول (IDT) من عنوان وقسم و درجه الصلاحيه حاول ان تقارنهم مع الشرح السابق البنيه الاوليه التاليه تحمل المتغيرين اللذان سيحملان الي المعالج داله تقوم باعداد الجدول داله تقم بملئ الجدول بعناوين المدبرات الاوليه بالاسميلي الداله التاليه تقوم بتخزين عنوان مدبر فعلي بالسبي بلس بلس الداله التاليه تقوم بحذف مدبر فعلي الداله التاليه تقوم بعمل اهم الاشياء بالنسبه للانقطاعات الصلبه

اولاً لابد من ذكر بعض الاشياء المحكمات تكون معده من قبل لان تكون مدبراتها (المحكم الاساسي من ٠ الي ٧ أي ان مدبر الاله الكاتبه = رقم ١) (المحكم الثانوي من ٧٠ ← ٧٧) وهنا نحن نمتلك مشكله وهي الاستثناءات فهي تشغيل المداخل من ٠ الي ٣١ أي انه سوف يحدث تعارض مع مدبرات الانقطاعات الصلبه ومدبرات الاستثناءات لذا قمنا بعمل هذه الداله لتحويل مدبرات المحكمين ليكونو بعد الاستثناءات مباشره

الدالتين التاليين خاصتين بإشاره EIO الداله الاوليه تستخدم إذا كان المدبر تم نداؤه عن طريق المحكم الاساسي ام الاخري تستخدم إذا كان المدبر تم نداؤه عن طريق المحكم الثانوي

بعد ذلك نقوم باستيراد اسماء المدبرات الاوليه بالاسميلي من المدبر ٠ الي المدبر ٤٧ حيث نملك ٣١ مدبر خاص بالاستثناءات و الباقي خاص بالانقطاعات

◀ ملف الانقطاعات idt.cpp ▶

اولاً محتوى الملف

Idt.cpp

```
#include "inc/system.h"
#include "inc/idt.h"
#include "iop.h"

idt_entry idt[256];
idt_ptr idtp;
void idt_gate(int num,uint handle,int selector,int dpl)
{
    if(num < 0 || num > 256)
    {
        return;
    }else if(handle == 0)
    {
        memset((uchar*)&(idt[num]),0,sizeof(idt_entry));
    }else
    {
        idt[num].offset_1 = handle & 0xFFFF;
        idt[num].offset_2 = (handle >> 16) & 0xFFFF;
        idt[num].sel = selector;
        idt[num].pres = 1;
        idt[num].type = 0xE;
        idt[num].zero = 0;
    }
}

void idtsetup()
{
    memset((uchar*)&idt,0,sizeof(idt_entry) * 256);
    idtp.limit = (sizeof(idt_entry) * 256) - 1;
    idtp.base = (uint)&idt;
    asm volatile("lidt %0" :: "m"(idtp));
}

void picsetup()
{
    outb(0x20,0x11);
    outb(0xA0,0x11);

    outb(0x21,0x20);
    outb(0xA1,0x28);

    outb(0x21,4);
    outb(0xA1,2);

    outb(0x21,1);
```

```

    outb(0xA1,1);

    /** disable all irqs */
    outb(0x21,0xF);
    outb(0xA1,0xF);
}
void Eio1()
{
    outb(0x20,0x20);
}
void Eio2()
{
    outb(0x20,0x20);
    outb(0xA0,0x20);
}
/*****/
void set_isrs()
{
    idt_gate(0,(uint)isr0,0x8,0);
    idt_gate(1,(uint)isr1,0x8,0);
    idt_gate(2,(uint)isr2,0x8,0);
    idt_gate(3,(uint)isr3,0x8,0);
    idt_gate(4,(uint)isr4,0x8,0);
    idt_gate(5,(uint)isr5,0x8,0);
    idt_gate(6,(uint)isr6,0x8,0);
    idt_gate(7,(uint)isr7,0x8,0);
    idt_gate(8,(uint)isr8,0x8,0);
    idt_gate(9,(uint)isr9,0x8,0);
    idt_gate(10,(uint)isr10,0x8,0);
    idt_gate(11,(uint)isr11,0x8,0);
    idt_gate(12,(uint)isr12,0x8,0);
    idt_gate(13,(uint)isr13,0x8,0);
    idt_gate(14,(uint)isr14,0x8,0);
    idt_gate(15,(uint)isr15,0x8,0);
    idt_gate(16,(uint)isr16,0x8,0);
    idt_gate(17,(uint)isr17,0x8,0);
    idt_gate(18,(uint)isr18,0x8,0);
    idt_gate(19,(uint)isr19,0x8,0);
    idt_gate(20,(uint)isr20,0x8,0);
    idt_gate(21,(uint)isr21,0x8,0);
    idt_gate(22,(uint)isr22,0x8,0);
    idt_gate(23,(uint)isr23,0x8,0);
    idt_gate(24,(uint)isr24,0x8,0);
    idt_gate(25,(uint)isr25,0x8,0);
    idt_gate(26,(uint)isr26,0x8,0);
    idt_gate(27,(uint)isr27,0x8,0);
    idt_gate(28,(uint)isr28,0x8,0);
    idt_gate(29,(uint)isr29,0x8,0);
    idt_gate(30,(uint)isr30,0x8,0);
    idt_gate(31,(uint)isr31,0x8,0);
}

```

```

idt_gate(32,(uint)isr32,0x8,0);
idt_gate(33,(uint)isr33,0x8,0);
idt_gate(34,(uint)isr34,0x8,0);
idt_gate(35,(uint)isr35,0x8,0);
idt_gate(36,(uint)isr36,0x8,0);
idt_gate(37,(uint)isr37,0x8,0);
idt_gate(38,(uint)isr38,0x8,0);
idt_gate(39,(uint)isr39,0x8,0);
idt_gate(40,(uint)isr40,0x8,0);
idt_gate(41,(uint)isr41,0x8,0);
idt_gate(42,(uint)isr42,0x8,0);
idt_gate(43,(uint)isr43,0x8,0);
idt_gate(44,(uint)isr44,0x8,0);
idt_gate(45,(uint)isr45,0x8,0);
idt_gate(46,(uint)isr46,0x8,0);
idt_gate(46,(uint)isr47,0x8,0);
}

```

اولاً نقوم بضم الملفين الرئيسيين system.h و idt.h
السطر التالي نقوم بتعريف الجدول idt حيث يحتوي على ٢٥٦ مدخل
السطر التالي نقوم بتعريف البنية التي سيتم تحميلها إلى المعالج

الدالة التالية تقوم بملئ احد المداخل حيث رقم المدخل = قيمة المتغير num عنوان المتغير = قيمه
المتغير handle القسم الذي يعمل فيه المتغير او الصلاحيات selector = درجه الحماية dpl

في هذه الدالة نقوم بالتأكد من المدخل المطلوب ينحصر بين ٠ و ٢٥٥ اي انه ليس خارج عن
نطاق الجدول وفي حاله كان المتغير handle = صفر فمعناه اننا نريد محو المدخل رقم num
ثم نقوم بملئ المدخل حسب البيانات المطلوبه

الدالة التالية idtsetup() تقوم بتنصيب الجدول (أي ملئه بصفر) وتحميله للمعالج
asm volatile("lidt %0" :: "m"(idtp)); ربما هذه الدالة هي وحدها التي تحتاج إلى
شرح اولاً نقوم بكتابه الامر asm ومعناه ادخل تعليمه اسمبلي ضمن الكود ثم الكلمه volatile أي
منع التغير في هذه التعليمه بالذات فقد يقوم المترجم بمحوها إذا كان يظن انها تبطل من عمل
البرنامج(المقصود هنا النواه) تعليمه الاسمبلي lidt معناها قم بتحميل الجدول %0 حيث هذا
المتغير يشير إلى عنوان البنية التي صنعناها الحرف m معناها مؤشر إلى
الدالة picsetup الهدف من هذه الدالة هو تغيير ارقام المدبرات المخزنه في المحكمين الاساسي
والثانوي عن طريق ارسال لها بعض الاوامر عن طريق المنفذين (0x20 = لمحكم الاساسي و
0xA0 = للمحكم الثانوي) وإرسال بيانات تلك الاوامر عن طري المنفذين (0x21 = للمحكم
الاساسي و 0xA1 بالنسبه للمحكم الثانوي) ولكي نقوم بتحويل ارقام المدبرات يجب عمل الاتي
سوف نقوم بابع متتاليات أي اربع اوامر ببيانات تلك الاوامر

اولاً المتتاليه الاولى

الجدول التالي يظر البيئات المستعمله

Icw4	P2	٠	١	٠٠٠٠
------	----	---	---	------

Icw4 = اذا كانت = ٠ معناها لن نرسل متتاليه رابعه ١ تعني العكس

P2 = ٠ يوجد محكم ثانوي ١ العكس

أي ان الامر المرسل سيكون ٠٠٠١٠٠٠١ = 0x11

انظر الان الأول الاوامر المرسله للمنفذين 0x20 و 0xA0 وحاول مطابقه الارقام بالبينري
مع الجدول السابق

الكلمة الثانية تستخدم لتحديد البعد الذي ينبغي إضافته علي الخط للحصول علي رقم الانقطاع مثال لنفرض ان الخط ١ حدث له انقطاع واننا اخبرنا المحكم بأن ارقام مدبراته تبدأ من ٤٠ اولا لكي يعرف المحكم رقم المدير المطلوب يقوم بإضافه اقوي ٥ بيتات مع رقم الانقطاع لنري كيف اولا لنقوم بتحميل الرقم ١ إلي البينري = ٠٠٠٠٠٠٠١ ثم الرقم ٤٠ إلي البينري = ٠٠١٠١٠٠٠ نأخذ ال ٣ بتات الاضعف من رقم الانقطاع = ٠٠١ و الخمس الاقوي من رقم المدير = ٠٠١٠١ نضيفهم علي بعضهم = ٠٠١٠١٠٠١ نقوم بالتحويل إلي الارقام العاديه = ٤١ هلي فهمت الان الغرض من هذه الكلمه

فهي هذا الدرس استعملنا المدبرات من ٣٢ إلي ٣٩ للمحكم الاساسي و المدبرات ٤٠ إلي ٤٧ للمحكم الثانوي لذا فنحن مضطرون إلي ارسال رقم البدايه للمدبرات المستعمله للمحكم الاساسي له و رقم البدايه المستعمله للمحكم الثانوي له والذي يساويان بالهكس ٣٢ = 0x20 و ٤٠ = 0x28 الان انظر الملف السابق

نأتي الان للمتاليه الثالثه تستخدم هذه المتتاليه لإرسال رقم الخط المستعمل للاتصال بين المحكم الاساسي والثانوي وبما ان المحكم الاساسي يتصل بالثانوي عن طريق الخط رقم ٢ فإن البتات المرسله تكون علي الشكل التالي
٠٠٠٠٠١٠٠

الرقم ١ معناه الخط رقم ٢ وقد يلاحظ البعض انه بعد تحويله للهكس ان هذا الرقم = ٤ وليس اثنين لماذا ؟ اولا نحن لا يهمنا ما هو الرقم المرسل بالضبط ولكن يهمنا موقع البت ١ المرسل

المتتاليه الرابعه تكون دائماً واحد حسب تعليمات الشركات المنتجه للمحكمات الغرض منها مجهول

في هذه الداله (set_isr) نقوم بملئ الجدول بعناوين المدبرات الاوليه والقسم = 0x8 ودرجه الحمايه = صفر

▶ ملف المدبرات الاولي isrs.asm ◀

اولا وقبل ان اعرض محتويات ملف المدبرات الاولي لابد من ذكر بعض الاشياء اولا سياستي في كتابه مدبرات الانقطاعات تهدف إلي تقليل الكود كلما امكنتني ذلك لضمان السرعه (تذكر اننا نعمل علي مستوي الجزء من الثانيه)

ثانياً قد تلاحظ في الكود السابق انني قد قمت بإستيراد اسماء ال ٤٧ المدير الاول بالاسمبلي فهل سوف اقوم بكتابه كود لل ٤٧ مدير ام لا ؟ الاجابه نعم ولكن بطريق مختصره تعرف بالماكرو ولكن ما هي الماكرو وما مكوناتها؟

اولاً من يعرف هذا الجزء فلينتقل للصفحه التاليه
اولاً سوف اجيب عن الجزء الثاني من الكود مكونات الماكرو كالتالي

```
%macro macro_name ver
; macro code
%endmacro
```

في البدايه تجد الكلمه %macro وعناها عرف الماكرو macro_name (هي اسم الماكرو) وتأخذ عدد من المتغيرات ver السطر التالي نقوم بوضع الكود السطر الاخير نهايه الماكرو

الان لنجيب عن الجزء الاول من السؤال ولذا سوف اضع مثال عسي ان تفهم اكثر بدلاً من الجزء النظري :

مثال نحن نريد عمل ٣ دوال لهم كود واحد وتقريباً اسم واحد ولكننا لا نريد تكرار الكود حسنناً لنستخدم الماكرو ولنقل ان هذا الكود يقوم بوضع قيمه احد المسجلات في المسجل

الأخر الكود كالتالي

Asm code
<pre>%macro switch 1 void_%1: mov ax,bx %endmacro switch 1 switch 2 switch 3</pre>

فلنشرح سطر سطر

السطر الاول مفهوم اسم الماكرو switch عدد المتغيرات = ١
السطر الثاني قمنا بتعريف داله اسمها void_%1 ولكن ما المقصود ب %1 هذه معناه قيمه المتغير او استبدل هذا الرمز بقيمه المتغير الاول السطرين المواليين مفهومين
السطر التالي الكلمه switch 1 معناها عرف ماكرو واجعل قيمه المتغير الاول = ١
السطر التالي الكلمه switch 2 معناها عرف ماكرو واجعل قيمه المتغير الاول = ٢
السطر التالي الكلمه switch 3 معناها عرف ماكرو واجعل قيمه المتغير الاول = ٣
حسنناً المترجم سيقوم بترجمه الكود السابق ليكون كالتالي

Asm code
<pre>Void_1: Mov ax,bx Void_3: Mov ax,bx Void_2: Mov ax,bx</pre>

ربما بعد هذا المثال يفيد بالغرض

.....
.....
.....

حسنناً لنعد الان لموضع المدير الاولي لنعرض الكود

Isrs.asm
<pre>%macro isrn 1 [global _isr%1] _isr%1: push dword 0 push dword %1 jmp c_caller %endmacro ;//////////////////////////////////; %macro isre 1 [global _isr%1] _isr%1: push dword %1 jmp c_caller %endmacro ;//////////////////////////////////; [extern _isrC] c_caller: push eax push ebx</pre>

```
push ecx
push edx
push edi
push esi
push ebp
push es
push ds
push fs
push gs
push ss
mov ax,0x10
mov es,ax
mov ds,ax
mov fs,ax
mov gs,ax
mov ss,ax
call _isrC
pop ss
pop gs
pop fs
pop ds
pop es
pop ebp
pop esi
pop edi
pop edx
pop ecx
pop ebx
pop eax
add esp,8
iret
```

```
;*****;
```

```
isrn 0
isrn 1
isrn 2
isrn 3
isrn 4
isrn 5
isrn 6
isrn 7
isre 8
isrn 9
isre 10
isre 11
isre 12
isre 13
isre 14
isrn 15
isrn 16
isrn 17
```

```

isrn 18
isrn 19
isrn 20
isrn 21
isrn 22
isrn 23
isrn 24
isrn 25
isrn 26
isrn 27
isrn 28
isrn 29
isrn 30
isrn 31
isrn 32
isrn 33
isrn 34
isrn 35
isrn 36
isrn 37
isrn 38
isrn 39
isrn 40
isrn 41
isrn 42
isrn 43
isrn 44
isrn 45
isrn 46
isrn 47

```

في البدايه نقوم بتعرف ٢ ماكرو ولننظر إلي محتواه

```

push dword 0
push dword %1
jmp c_caller

```

اولاً نقوم بدفع رمز خطأ وهمي حتي نجعل الكود متناسقاً (سوف تفهمها فيما بعد)
بعدها نقوم بدفع رقم المدير الذي سيستخدم فيما بعد من قبل المدير بلصي ++سي لمناده المدير
الفعلي حسنناً لنفترض ام المدير قد نودي من قبل مدير يقوم بدفع رمز الخطأ ماذا سنفعل حسنناً
لنقم بصنع ماكرو اخر يقوم بدفع رقم المدير ولا يقوم بدفع رمز الخطأ.

الداله التاليه: c_caller هذه الداله تقوم بدفع القيم إلي المكسد ثم تقوم بمناده المدير الفعلي
المكتوب بـ سي/سي++ والذي يدعي (isrC) وهذه الداله تنادي من جميع من المديرات الاوليه
الاخري والان حاول حاول مطابقه فهمك للماكرو او الشرح السابق مع هذا الكود ومع جدول
الاستثناءات .

◀ ملف المدير الاساسي Interrupt.h + isr.cpp ▶

اولاً ملف المدير الاساسي Interrupt.h

Interrupt.h

```

#ifndef _H_interrupt_
#define _H_interrupt_
#include "system.h"
class interrupt
{
public:
    virtual void handle(regs r)=0;
};
#endif

```

اول ٣ اسطر مفهومه السطر السطر التالي نقوم بتعريف بنيه تسمى **interrupt** هذه البنيه
 نستخدمها عندما نريد تعريف مدبر ما فعلي جميع المدبرات تعريف نفسها علي هذا الشكل السابق
 مثال للآله الكاتبه

```

Calss keyboard:public interrupt
{
public:
    keyboard();
    void handle(regs r);
}

```

سؤال ماسبب استخدام الصفر وكلمه **virtual**
 السبب في ذلك هو اخبار المترجم ان هذه الداله وهميه تستخدم فقط لتعرف دوال اخري

قد تلاحظ ان احد المتغيرات التي اعطيناها للدله **handle** البنيه **regs** هذه البنيه تعمل كمؤشر علي
 القيم التي قمنا بدفعها في داله الاسمبلي **c_caller** الموجوده في الملف **isrs.asm**
 ولكي تعرف محتوى هذه البنيه انظر الملف **system.h**

ناتي الان لدور ملف **isr.cpp**

```

Isr.cpp
#include "inc/system.h"
#include "inc/Interrupt.h"
static interrupt *handles[256];
extern "C" void isrC(regs r)
{
    handles[r.inum]->handle(r);
}
void reg_handle(int num,interrupt *handle)
{
    if(num <0 || num > 256)
    {
        return;
    }else
    {
        handles[num] = handle;
    }
}
void del_handle(int num)
{
}

```

```
handles[num] = (interrupt*)0;
}
```

في البدايه نقوم بضم الملفات الرئيسيه system.h Interrupt.h
ثم نقوم بتعريف المؤشر handles حتي نقوم بحفظ عناوين المدبرات الفعليه المكتوبه بسي/سي++

الداله الاولى isrC هذه الداله قم قمنا باستيرادها من قبل الداله c_caller في الملف isrs.s
الهدف من هذه الداله هو نداء المدبر حسب رقمه المخزن في الفئه r في المتغير inum أي
interrupt number

الداله التاليه تقوم بتسجيل مدبر فعلي تأخذ عاملين الاول رقم المدبر المراد تسجيله ثانياً عنوان
المدبر وتتاكد ان الرقم الذي اعطيناها اياه ليس اقل من الصفر وليس اكبر من ٢٥٦

الداله الاخيريه تقوم بحذف احد المدبرات عن طريق ملئ مكانه بصفر

.....
حان الان لنقوم بأول تجربه للكواد الذي كتبناه سابقاً قم بفتح الملف الرئيسي main.cpp وقم بضم
الملفات الرئيسيه interrupt.h idt.h قم بإضافه الامرين

```
idtsetup();
set_isrs();
picsetup();
```

هذه الامرين الاولين هما من يقوموا ببناء دالتين اعداد جدول الإنقطاعات
الامر الاخير يقوم ببناء داله اعداد المحكم

والان قم بضم الملفات ذات امتداد .cpp و .asm. إلي ملف الربط وقم بفتح القرص الوهمي الخاص
ب Grub ثم قم بتشغيل Cygwin واكتب فيه امر الترجمة

Make -C x:/folder -f Makefile

حيث x:\folder هي مسار ملف ال makefile والملفات التي قمنا بصنعها
Makefile اسم ملف الماك فايل
واخر خطوه هي تجربه ما فاعلنا في برنامج وهمي مثل
Virtual pc bochs إلي اخره

إذا نجحت الخطوه السابقه فإننا الان جاهزون الان لدعم الاستثناءات والاجهزه الصلبه المختلفه

◀ ملف الاستثناءات exp.h ▶

اولاً لنعرض المحتوي

```
Exp.h
#ifndef _H_Exptions_
#define _H_Exptions_

#include "interrupt.h"
class exptions: public interrupt
{
public:
    exptions();
    void handle(regs r);
};
#endif
```

اول ٣ اسطر مفهومه
السطر الرابع نقوم بتعريف فئه اسمه exptions وقد تلاحظ شئ غريب وهو public interrupt
الهدف من هذا السطر إخبار المترجم ان يقوم بمقارنه الفئه
Interrupt بالفئه exptions حتي إذا كان هناك شئ ناقص يقوم بإخبارنا بوجود خطأ

لنأتي الان إلي ملف الاستثناءات exp.cpp

Exp.cpp

```
#include "inc/video.h"
#include "inc/system.h"
#include "inc/exptions.h"

const char *exptxt[18] =
{
    "Division by zero Exception " , "Debug Exception "
    "Non maskable interrupt Exception " , "Breakoint Exception "
    "into detetcted overflow Exception " , "Out of bounds Exception
    "Invalid opcode Exception " , "No Coprocessor Exception"
    "Double fault Exception " , "Coprocessor segment overrun "
    "Bad TSS Exception " , "Bad TSS Exception "
    "Segment not present Exception " , "Stack fault Exception "
    "General protection fault Exception " , "Page fault Exception "
    "Unkown interrupt Exception " , "Coprocessor fault Exception "
    "Alignment check Exception " , "Machine check Exception "
};

exptions::exptions()
{
}

void exptions::handle(regs r)
{
    video v;
    v.printf("\n\n\t");
    v.printf(exptxt[r.inum]);
    while(1);
}
```

في البدايه نقوم بضم الملفات الرئيسيه system.h exptions.h video.h ثم نقوم بتعريف مصفوفه تحمل كافه الاستثناءات حسب ما عرضتها شركه انتل في كتابها

ما يهمننا الان في هذا الكود الداله الاخيره handle لنري كودها اولاً نقوم بتعريف الفئه video بإسم v ثم في السطر التالي نقوم بطباعه سطرين جديدين ثم ٤ مسافات حسب ما حددناها في الدرس السابق

مايهمننا في كل هذا السطر التالي v.printf(exptxt[r.inum]); نقوم في هذا بطباعه نص الاستثناء حسب الرقم الموجود في الفئه regs في المتغير inum ومن هنا سنكتمل لدينا كل الافكار لنتخيل ماذا يحدث من حدوث الاستثناء

قسمه علي صفر ← حدوث الاستثناء ← يقوم المعالج بالبحث في جدول الانقطاعات عن مدبر لهاذا الاستثناء ← وجد المعالج المدبر ← تم نداء مدبرنا الاولي ← في حاله كان الاستثناء الذي نودي مدبرنا من قبله لا يقوم بدفع رمز خطأ نقوم بدفع رمز خطأ وهمي ← دفع رقم المدبر ← نداء الداله c_caller هذه الداله تقوم بدفع قيمه المسجلات gs ssedi edx ecx ebx eax

← تحميل المسجلات es ds gs fs بالقيمة 0x10 أي صلاحيات البيانات ← نداء الدالة isr_c ← تقوم هذه الدالة ببناء المدير الفعلي حسب الرقم المدفوع للمكس ← تم نداء مدير الاستثناء ← قيام المدير بطباعه نص الاستثناء حسب الرقم المدفوع في المكس ← التوقف نهائياً عن العمل .

هل فهمت الان القصه ، الان لنعد إلي موضوعنا ولنقم بتجربه الكود السابق قم بفتح الملف الرئيسي main.cpp وقم بضم ملف الاستثناءات الرئيسي exp.h قم بالذهاب إلي الداله الرئيسي وقم بتعريف البنيه الخاصه بالاستثناء

Exption exp;

لنقم الان بتسجيل هذا المدير كمدير فعلي للاستثناءات لنقم بكتابه الكود التالي

```
For(int i=0;i<= 19;i++)
{
    reg_handle(i,&exp);
}
```

قد يقول البعض لماذا جعلت دوره تتوقف عن الرقم ١٩ ؟ لانني قمت بدعم الاستثناءات في ملف الاستثناءات حتي الرقم ١٩ ولم اكمل باقي الاستثناءات لنمضي الان في طريقنا

لنقم الان بتنشيط الانقطاعات عن طريق اضافته الكود

Asm volatile("sti");

ولنقم بخطأ مقصود وهو القسمة علي صفر

Volatile int zero = 5/0;

ولنقم بتجربه الاستثناءات الان قم بإضافته ملف الاستثناءات إلي ملف الماك فايل ولتقم بالترجمه

اول الاخطاء التي قد تقابلك هي خطأ القسمة علي صفر تجاهله ولنأتي إلي الخطأ التالي وهو عدم تعريف احد الدوال وهذه الداله هي

__cxa_pure_virtual

هذه الداله يحتاجه المترجم هناك عدة طرق لتفاديها والطريقه التي اعرفها هي تعريف الداله وتصدير اسمها

```
extern "C" void __cxa_pure_virtual()
{
};
```

الان عاود الترحمه ولتقم بالتجربه

لا بد ان هناك نص الان مثل Division by zero Exception قد ظهر او أي نص اخر قد ظهر

الان ازل خطأ القسمة علي صفر المقصود وعاود التجربه اكيد لن يظرو أي نص مبروك لقد دعمت الاستثناءات

▶ دعم ال timer ◀

اولاً سوف اقوم بعرض نبذه عن هذا ال timer

يستعمل المنبه او ال timer رقاقه pit رقاقه 8253 او 8254 لإنتاج انقطاعات دوريه في الحاسب تتم هذه الانقطاعات عن طريق بلور متذبذب ينتج نبضات ذات تردد قيمته ١١٩٣١٨٠ أي ١,٨ Hz ولكي نقوم ببرمجه المنبه نستخدم قيمه تسمى القاسم الترددي مثلاً عند إقلاع الحاسب يستعمل قاسم ترددي = ٦٥٥٣٦ بمعني (١١٩٣١٨٠ ÷ ٦٥٥٣٦) = ١٨,٢ نبضه في الدقيقه

يتوفر المنبه علي ٣ مسجلات الاول يستخدم لأجل الانقطاعات والثاني من قبل الهارد وير والثالث يستعمل من قبل system speaker او بوق النظام

في حالتنا سوف نستعمل القناه الاول

المنفذ المرتبط بها هو 0x40 حيث سنضع فيه القاسم الترددي ولكن هناك ملاحظه وهي انه في كل استخداماتنا للمنافذ او الاجهزه الخارجيه كنا مرسل امر التحكم وبعد امر التحكم نرسل البيانات الخاصه بالامر لذا وقبل ارسالنا للقاسم الترددي سوف نقوم بإرسال امر الاعداد إلي المنفذ 0x43 ولننظر إلي المثال التالي

نريد ان نقوم ببرمجه المنبه ليقوم بعمل ٢٠ نبضه في الدقيقه

```
Int qasm = 1193180 / 59659 ;
```

```
Outb(0x43 , 0x34);
```

```
Outb(0x40 , qasm & 0xFF);
```

```
Outb(0x40,qasm >> 8);
```

لنشرح المثال السابق أولاً نقوم بتعريف متغير ليحمل قيمه القاسم الترددي
السطر التالي نقوم بإرسال امر الاعداد = 0x34 (b ٠٠١١٠١٠٠) إلي المنفذ 0x43 ولنقوم بتفسير امر الاعداد (تفسير البتات يبدأ من اليسار)

٠٠ نريد كتابه في المسجل ٠
١١ نريد كتابه البتات الاضعف ثم الاقوي (الاقوي أي الموجود بعد اول ١٦ بت الاضعف أي الموجوده قبل ١٦ بت)

٠١٠ استعمل الوضع rate generator أي انتاج انقطاعات دوريه و منتظمه
٠٠ استعمل الحساب الثنائي مما يتيح استخدام قاسم ترددي اقصاه ٦٥٥٣٦
قد يظن البعض انه بإستخدام الرقم ٥٩٦٥٩ او أي رقم اكبر من ٦٥٥٣٦ فإنه هذا خطأ لا ليس الخطأ وليس المهم الرقم المستعمل ولكن المهم هنا الرقم الناتج

السطر التالي نقوم بإرسال البتات الاضعف ثم السطر الموالي نرسل البتات الاقوي

الان حان الان لنقوم بكتابه الاكود ولنقم بإستعراض كود الملف الاول timer.h

```
Timer.h
#ifndef _H_timer_
#define _H_timer_
#include "interrupt.h"

void tinstall();
class timer: public interrupt
{
public:
    timer();
    void handle(regs r);
};
#endif
```

لا بد ان الكود مفهوم حسب فهمك لكود الاستثناءات هناك ايضاً الداله tinstall الملونه باللون الاحمر هي التي تقوم ببرمجه المنبه

الملف التالي timer.cpp

```
Timer.cpp
#include "inc/timer.h"
#include "inc/system.h"
#include "inc/video.h"
#include "inc/iop.h"
int base = 1193180/20;
int sec=0;
timer::timer()
```

```

{
}
void timer::handle(regs r)
{
    video v;
    if(sec == 150)
    {
        v.printf("\n timer tick");
        sec =0;
    }
    sec++;
    outb(0x20,0x20);
}
void tinstall()
{
    outb(0x43, 0x36);
    outb(0x40, base & 0xff);
    outb(0x40, base >>8);
}

```

اولاً نقوم بتعريف متغيرين احدهما يحمل القاسم الترددي والمتغير الاخر لحفظ عدد الانقطاعات التي تمت من قبل المنبه فإذا مل بلغت ١٥٠ يقوم بطباعه جملة timer tick ثم يعود إلي صفر مره اخرى

الداله handle قد تم شرحها وف نهايه الداله نقوم بإرسال اشاره eio والتي تقوم بإعاده استقبال انقطاعات اخري من نفس الخط

الداله الاخيره تقوم بإعداد المنبه.

لننتقل الان للملف الرئيسي الداله الرئيسي main لنقوم أولاً بتعريف البنيه timer

Timer t;

والان لنقم بتسجيلها بما ان انقطاعات المحكم الاساسي تبدأ من 32 وخط انقطاع المنبه هو رقم ٠ لذا يجب علينا تسجيل المنبه السابق علي الخط 0x32

Reg_handle(32,& t);

قلت في الشرح السابق انه لتنشيط الانقطاعات نستعمل التعليمه Sti ولم اقل كيف كيف اقوم بجعل المحكم يستقبل طلبات انقطاع من خط معين وهل تذكر عندما كتبت هذا الامر في داله اعداد المحكم

outb(0x21,0xF);
outb(0xA1,0xF);

هذا الامر يقوم بمنع المحكمين (بما اننا ارسلنا نفس الامر إلي كل من المحكمين) من استقبال أي طلبات انقطاع ولكي نقوم بجعل المحكم يستقبل طلبات الانقطاع من احد الفروع نستعمل الطريقه التاليه بما ان المحكم يحتوي علي ٨ خطوط فسوف نقوم بتمثيلها بالبتات

١١١١١١١١ ماذا تساوي هذه القيمه بالهكس = 0xFF ولنقل اننا نريد تنشيط خط المنبه
فإن البتات ستساوي
١١١١١١١٠ ماذا تساوي بالهكس = FE الان دور الاله الكاتبه
١١١١١١٠١ ماذا تساوي بالهكس = Fd

من السابق نستنتج انه لكي نقوم بتنشيط خط انقطاع معين نقوم بجعل البت الخاص به = ٠ ثم إرسالها إلي المحكم الذي يوجد هذا الخط فيه

اضف امر تنشيط الانقطاع إلي الداله الرئيسيه

```
Outb(0x21 , 0xfE);
```

الان قم بالتجربه وفي اثناء تجربتك للكود سوف تلاحظ ان النظام يكتب timer tick تقريباً كل ثانيه

► دعم الاله الكاتبه ◀

هذا الجزء سوف تقوم بكتابته من نفسك ونرجو المحاوله ثم النظر في الكود المرفق

اولا ماذا يحدث عند ضغطك للاله الكاتبه ؟

اولا يتم إرسال طلب انقطاع.....نفس دوره..... تشغيل المدير
وفي نفس الوقت تقوم الاله الكاتبه بإرسال رمز الحرف المضغوط ثم يقوم الحاسب
بتخزينه في المنفذ 0x60

لذا فإن فكره الكود قائمه علي إدخال رمز الحرف المضغوط ومن ثم اختيار الحرف المقابل له ثم
طباعته
مثال

```
Char opp_char [10]= // مصفوفه تحمل الحروف المقابله  
{  
    0,0,'1','2','3','4','5','6','7','8','9','0','-','=','\','\b'  
}  
Int scan_code;
```

```
Scan_code = inb(0x60); // دخال رمز الحرف المضغوط ثم تخزينه في المتغير  
// scan_code  
video.putc(opp_char[scan_code]); // طباعه الحرف المقابل حسب الرمز المخزن في  
// scan_code
```

فكره الداله gets هي اخذ عنوان المتغير المراد حفظ النص فيه ثم استدعاء الداله strepy الذي
قمنا بكتابته كودها في الدرس السابق ونسخ النص الموجود في المتغير الذي يحفظ نص المتغير إلي
المتغير المقصود

بعد دعمك للاله الكاتبه يمكنك الان الكاتبه في نظامك كما تشاء

◀ Bochs ▶

المحاكي Bochs او المصحح Bochs كلها اسماء تشير إلى كاشف اخطاء انظمه التشغيل التي في قيد التطوير فهذا المصحح لا يكشف إلا الأخطاء الخاصة بجدول واوامر المعالج ولا يكشف الأخطاء البرمجية فمن يكشفها فهو المترجم .

اولاً وقبل أي شئ ينبغي علينا تحميل المحاكى Bochs ويمكنك تحميله من الموقع التالي ملحوظه عليك الحصول علي النسخه الخاصه ب windows لانها من ساقوم بشرحها مع ان النسختين لا يختلفان ولكن نسخه اللينكس لا اعرف كيف يتم تنصيبها

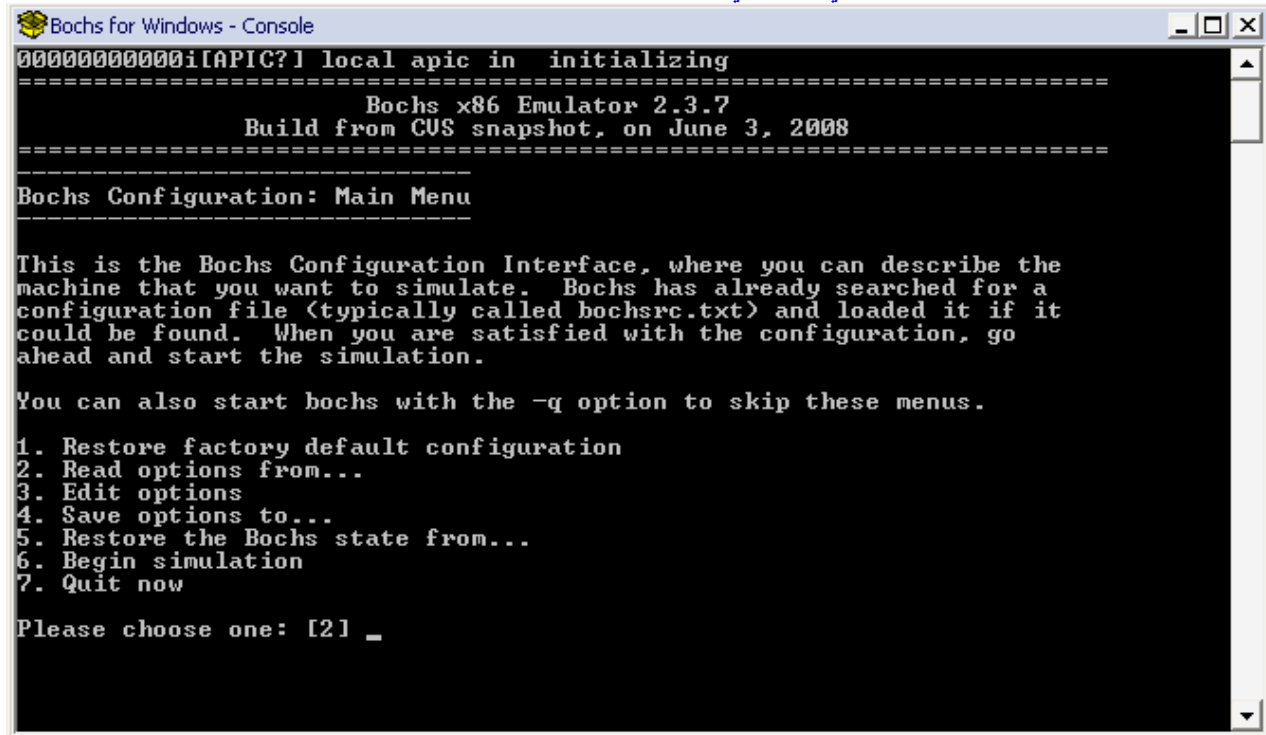
<http://bochs.sourceforge.net/getcurrent.html>

ثم تنصيه كاي برنامج عادي

قم بفتح برنامج Bochs عن طريق الاتي

Start → programs → Bochs → Bochs

سوف تظهر نافذه شكلها كالتالي تقريبا ربما الاصدارات الاقدم لا تتضمن اشياء موجوده في اصدارتي وربما الاحدث تتضمن اشياء غير موجوده في اصدارتي



```
Bochs for Windows - Console
000000000000i[APIC?] local apic in  initializing
=====
                Bochs x86 Emulator 2.3.7
            Build from CVS snapshot, on June 3, 2008
=====

Bochs Configuration: Main Menu
=====

This is the Bochs Configuration Interface, where you can describe the
machine that you want to simulate.  Bochs has already searched for a
configuration file <typically called bochsrc.txt> and loaded it if it
could be found.  When you are satisfied with the configuration, go
ahead and start the simulation.

You can also start bochs with the -q option to skip these menus.

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [2] _
```

قم بالضغط الرقم ٣ ثم enter (انظر في الصورة لتري النص الذي يساويه حتي اذا اختلف رقمها في اصدارات تكون عالماً لما نقصده) انظر الصورة التاليه

```
Bochs for Windows - Console

1. Restore factory default configuration
2. Read options from...
3. Edit options
4. Save options to...
5. Restore the Bochs state from...
6. Begin simulation
7. Quit now

Please choose one: [2] 3

-----
Bochs Options Menu
-----
0. Return to previous menu
1. Logfile options
2. Log options for all devices
3. Log options for individual devices
4. CPU options
5. Memory options
6. Clock & CMOS options
7. PCI options
8. Bochs Display & Interface options
9. Keyboard & Mouse options
10. Disk options
11. Serial & Parallel port options
12. Network card options
13. Sound Blaster 16 options
14. Other options

Please choose one: [0]
```

اضغط الرقم ١٠ ثم enter ستظهر الصورة التالية

```
Bochs for Windows - Console

9. Keyboard & Mouse options
10. Disk options
11. Serial & Parallel port options
12. Network card options
13. Sound Blaster 16 options
14. Other options

Please choose one: [0] 10

-----
Bochs Disk Options
-----
0. Return to previous menu
1. Floppy Disk 0
2. Floppy Disk 1
3. ATA channel 0
4. First HD/CD on channel 0
5. Second HD/CD on channel 0
6. ATA channel 1
7. First HD/CD on channel 1
8. Second HD/CD on channel 1
9. ATA channel 2
10. First HD/CD on channel 2 <disabled>
11. Second HD/CD on channel 2 <disabled>
12. ATA channel 3
13. First HD/CD on channel 3 <disabled>
14. Second HD/CD on channel 3 <disabled>
15. Boot Options

Please choose one: [0]
```

اضغط ١ ثم enter

```

Bochs for Windows - Console
Bochs Disk Options
-----
0. Return to previous menu
1. Floppy Disk 0
2. Floppy Disk 1
3. ATA channel 0
4. First HD/CD on channel 0
5. Second HD/CD on channel 0
6. ATA channel 1
7. First HD/CD on channel 1
8. Second HD/CD on channel 1
9. ATA channel 2
10. First HD/CD on channel 2 <disabled>
11. Second HD/CD on channel 2 <disabled>
12. ATA channel 3
13. First HD/CD on channel 3 <disabled>
14. Second HD/CD on channel 3 <disabled>
15. Boot Options

Please choose one: [0] 1

-----
Floppy Disk 0
-----

Enter new filename, or 'none' for no disk: [A:] A:

What type of floppy disk? <auto=detect> [1.44M] 1.44M

Is the floppy inserted or ejected? [inserted]

```

اكتب رمز القرص الوهمي الخاص بك ثم **enter** ثم حجم القرص الوهمي (المعتاد 1.44 M لا بد وكتابه حرف ال M كبير وليس بصغير) ثم **enter**

اضغط ٠ ثم **enter** ثم ٠ ثم **enter** لنعود إلى الشاشة الرئيسية اضغط الرقم ٤ ثم **enter** اكتب اسم ومسار الملف الذي تريد تخزين فيه المعلومات الخاصة بـ **Bochs** مثال

C:\windows\bochs.ini

ثم **enter** استخدمت الخاصية ٤ لأنه عند فتحك للمحاكي **Bochs** لا بد من كتابتك لكل ما سبق وسوف تعرف لاحقاً كيف نستغل هذه الخاصية
افت ملف جديد عن طريق ال **notepad** ثم اكتب التالي

bochs.exe -q -f C:\windows\bochs.ini

pause

cls

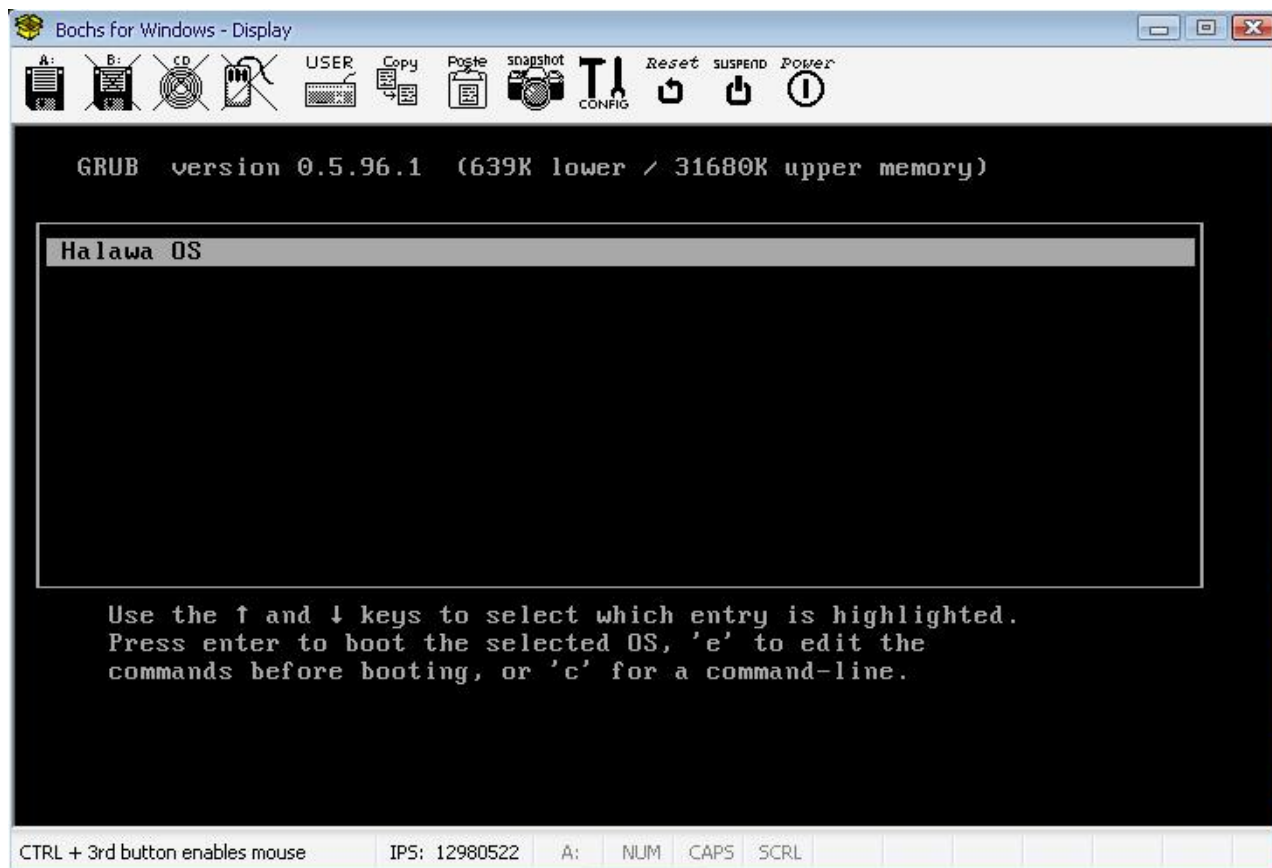
bochs.bat

استبدل C:\windows\bochs.ini باسم الملف الذي قمت بحفظه عند الخاصية ٤
قم بحفظ هذا الملف في المسار التالي

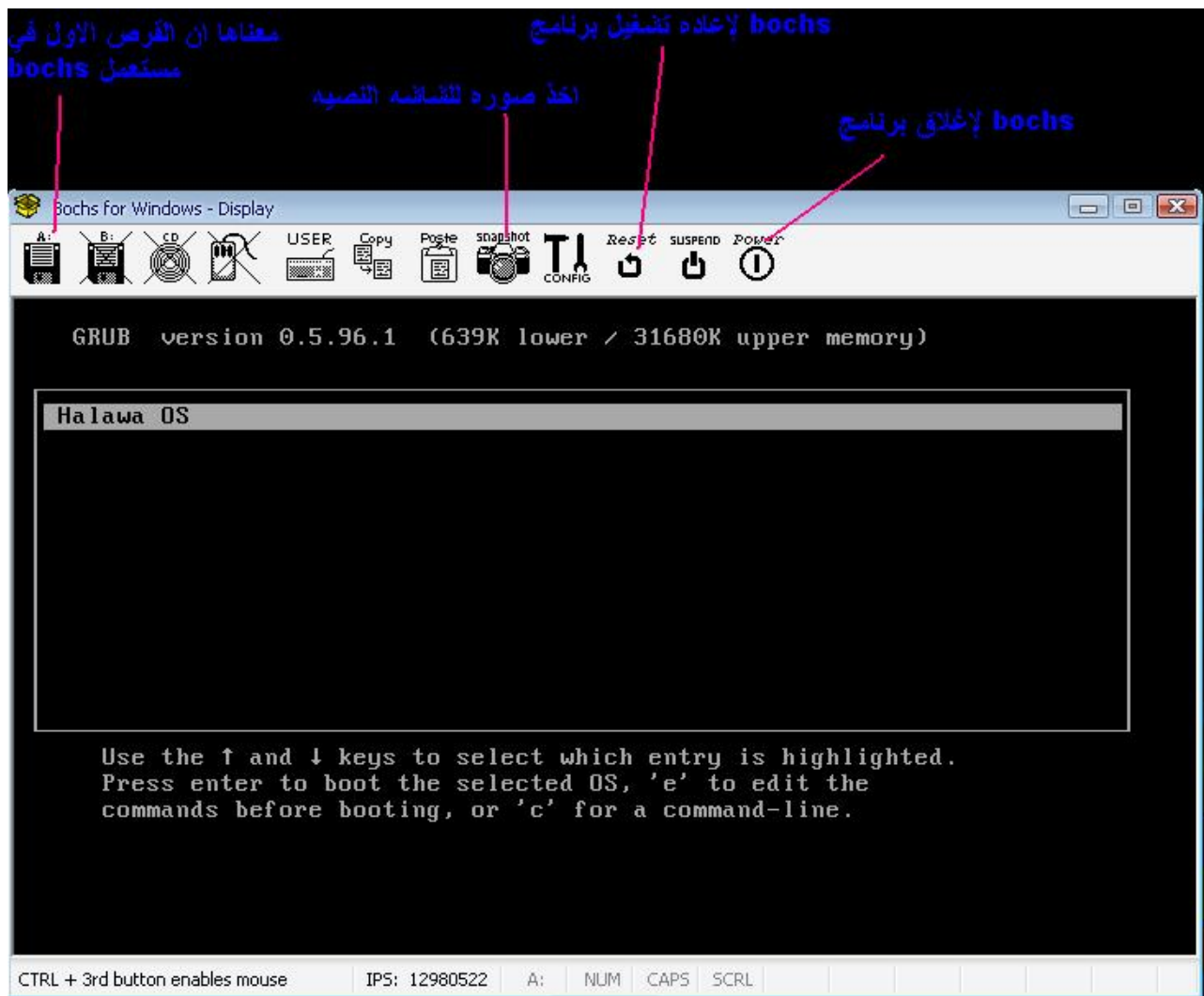
C:\program files\bochs

أي بجانب برنامج **Bochs** ولا بد أن يكون نوع الملف الذي تقوم بحفظه = **.bat**
استبدل هذا النص باسم الملف الذي قمت بحفظه ونوعه
ثم قم بعمل اختصار لهذا الملف لاي مكان تريده بشرط أن يكون قريب منك حتي تقوم بتشغيله
بسرعة

الهدف من الملف السابق هو استعمال احد ملفات الخاصية بـ **Bochs** مثل الذي حفظناها عند الخاصية ٤ وبدلاً من فتحك للملف السابق لكثير من مره سوف تضغط أي زر في لوحه المفاتيح ليعود الملف السابق إلي تشغيل نفسه وفتح برنامج **Bochs** من جديد
لنقم الآن بتجربه برنامج **Bochs** قم بفتح القرص الوهمي الخاص بـ **grub** وقم بترجمه النواه السابقه ثم قم بفتح الملف السابق سوف تري ان النواه تعمل



لنري الان ادوات التحكم في Bochs



قم بفتح برنامج bochs عن طريق زر التحكم الخاص به لنرى النص الذي ظهر لك

```

00003632000p[WGUI] >>PANIC<< POWER button turned off.
00003632000i[CPU0] CPU is in real mode (halted)
00003632000i[CPU0] CS.d_b = 16 bit
00003632000i[CPU0] SS.d_b = 16 bit
00003632000i[CPU0] EFER = 0x00000000
00003632000i[CPU0] RAX=0000000000000000 RBX=0000000000000000
00003632000i[CPU0] RCX=0000000000000000 RDY=0000000000000000
00003632000i[CPU0] RSP=0000000000000000 RBP=0000000000000000
00003632000i[CPU0] RSI=0000000000000000 RDI=0000000000000000
00003632000i[CPU0] R8=0000000000000000 R9=0000000000000000
00003632000i[CPU0] R10=0000000000000000 R11=0000000000000000
00003632000i[CPU0] R12=0000000000000000 R13=0000000000000000
00003632000i[CPU0] R14=0000000000000000 R15=0000000000000000
00003632000i[CPU0] IOPL=0 id vip vif ac vm rf nt of df IF tf SF zf f
00003632000i[CPU0] SEG selector base limit G D
00003632000i[CPU0] SEG sltr(index|ti|rp1) base limit G D
00003632000i[CPU0] CS:f000< 0004: 0! 0) 000f0000 000ffff 0 0
00003632000i[CPU0] DS:0000< 0005: 0! 0) 00000000 000ffff 0 0
00003632000i[CPU0] SS:0000< 0005: 0! 0) 00000000 000ffff 0 0
00003632000i[CPU0] ES:0000< 0005: 0! 0) 00000000 000ffff 0 0
00003632000i[CPU0] FS:0000< 0005: 0! 0) 00000000 000ffff 0 0
00003632000i[CPU0] GS:0000< 0005: 0! 0) 00000000 000ffff 0 0
00003632000i[CPU0] MSR_FS_BASE:0000000000000000
00003632000i[CPU0] MSR_GS_BASE:0000000000000000
00003632000i[CPU0] RIP=0000000000000000 <0000000000000000>
00003632000i[CPU0] CR0=0x60000010 CR1=0x0 CR2=0x0000000000000000
00003632000i[CPU0] CR3=0x00000000 CR4=0x00000000
00003632000i[CPU0] >> mov ax, 0x040c : 88b604
00003632000i[CMOS] /Last time is 1221211458 (Fri Sep 12 12:24:18 2008)
# In bx_win32 qui c::exit(void)!

```

التمط الذي كان يعمل فيه برنامج bochs قبل حدوث الخطأ أو إغلاق البرنامج

قيم المسجلات وقت حدوث الخطأ أو إغلاق برنامج bochs

قيم مسجلات التحكم قبل حدوث
ال.....
ما بهما فيه المسجل rip

سبب الخطأ عند حدوث خطأ

bochs كطبي برنامج

ملحوظة عند حدوث خطأ فقم بغلق البرنامج لان هذا النص ستم محوه كلما اعداد Bochs التشغيل
وقت حدوث الخطأ يظهر نصان يشبهان النص السابق النص الاول يظهر معلومات عن الخطأ
والثاني يظهر المعلومات الي كانت موجوده قبل إعداده التشغيل

وفي النهايه اتمني ان اكون قد وفيت في الشرح ولمن يريد دعم المزيد من الهارد وير فعليه بتحميل
برنامج يمسي help pc ويمكنك إيجاده في هذا الموقع

<http://www.osdever.net>