MIPS CPU Interrupts handler & ISR

實驗目的

1. 認識MIPS的interrupt機制與MIPS的interrupt處理流程
2. 認識CP0暫存器與存取CP0暫存器的部分指令
3. 實作簡易interrupt handler & ISR

Interrupt

Interrupt Cause

- There are two classes of causes for an interrupt.
  1) Interrupts caused by external signals
  2) Interrupts as result of instruction execution (exceptions)

Interrupts by External Signals

- Hardware Interrupts:

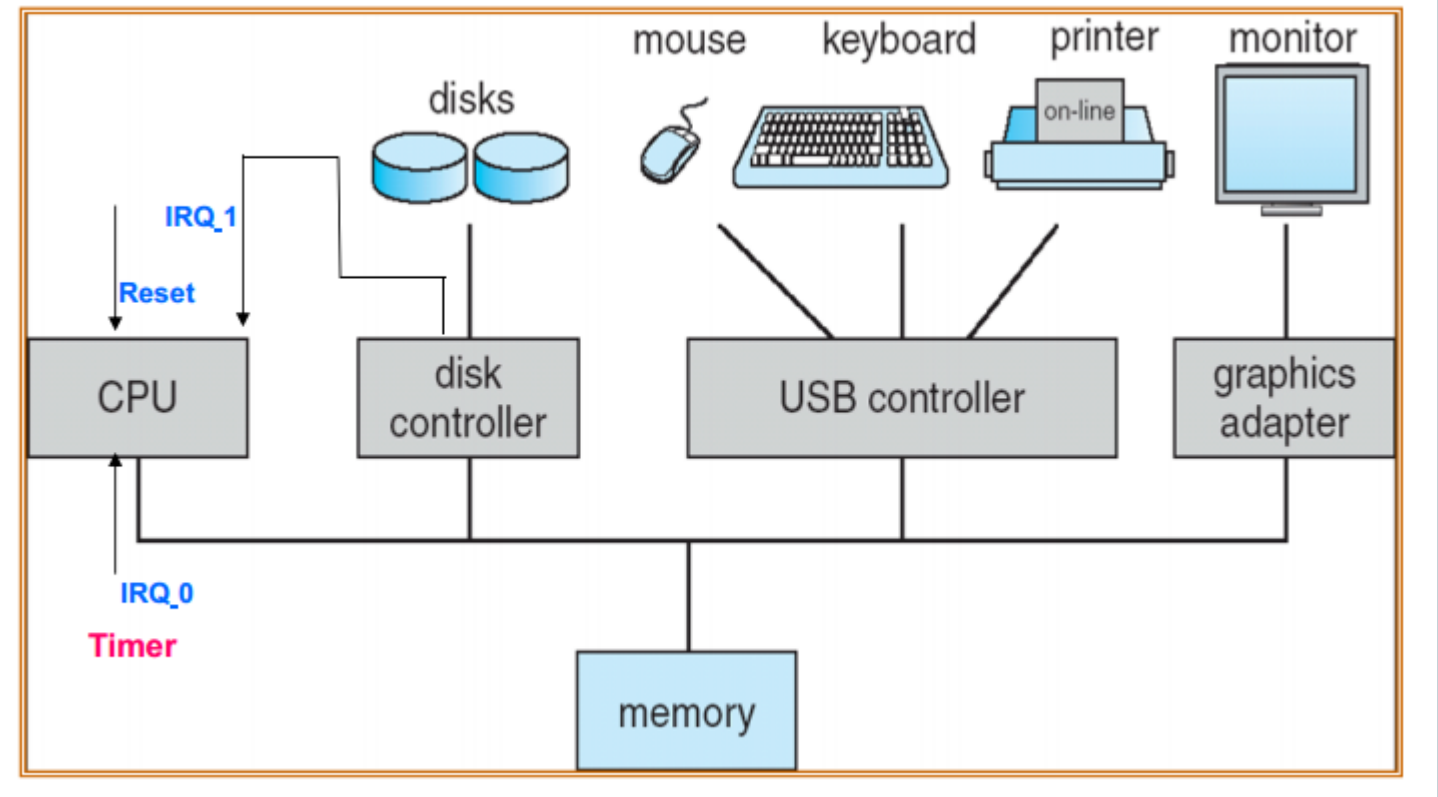
  Interrupt requests made via asserting signal on any of special external pins.

- Interrupt request lines – IRQ's.

- Hardware interrupts can be masked by setting appropriate bits in Status register.

Interrupts by External Signals

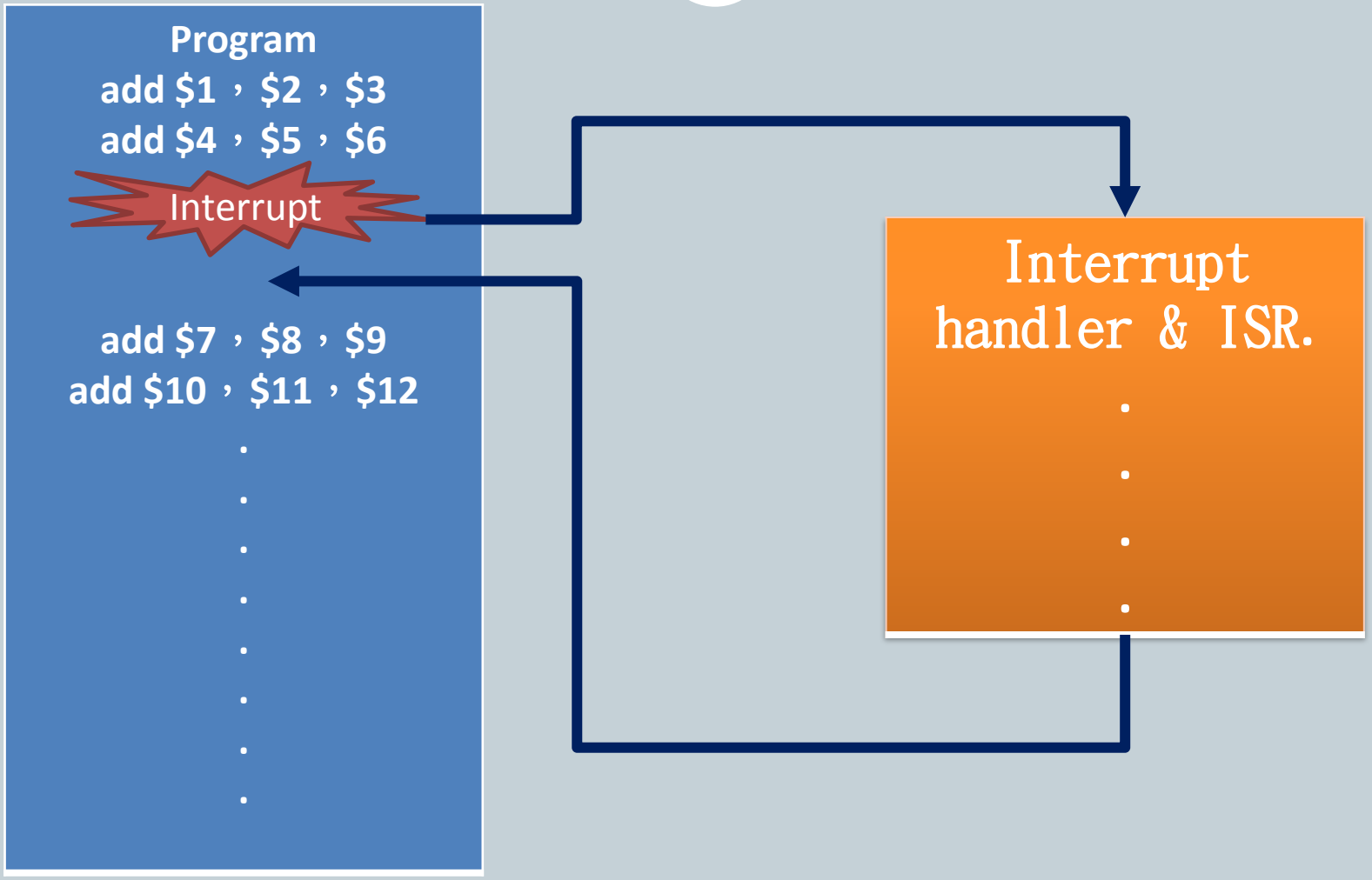
- There are also interrupts caused by hardware failure
  EX. IRQ_0、IRQ_1、……etc.



Interrupts as result of instruction execution

- Address Error
- Reserved Instruction (illegal instructions)
- Overflow
- Floating Point Error
- Syscall or any trap instruction executed

Control Flow



認識 Coprocessor0

➢ The System Coprocessor

- The MIPS ISA provides up to 4 coprocessors. A coprocessor extends the functionality of the MIPS ISA.
- CP0 provides an abstraction of the functions necessary to support an operating system: exception handling 、memory management 、 scheduling and control of critical resources.

認識Coprocessor0部分暫存器

➢ CP0 Registers

- The interface to access CP0 is through various instructions encoded with the CP0 opcode ， including the ability to move and modify data to and from the CP0 registers

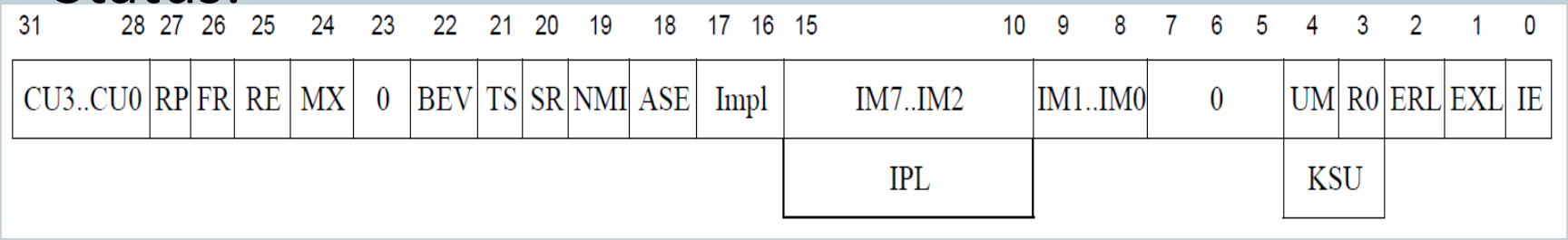
認識Coprocessor0部分暫存器

- CP0 Registers include status、 cause、 EPC … etc
- The processor is running in Kernel Mode or Debug Mode.

Register Number	Register Name	Function
0	..	..
..	..	..
12	Status	Processor status and control
13	Cause	Cause of last general exception
14	EPC	Program counter at last exception
..	..	..
31	..	..

CP0 Registers- Status

➢ Status:



- IM7…IM2 [15:10] : (可知那些中斷被遮蓋)

  ✓ Interrupt Mask:

  代表被遮蓋

IM7	IM6	IM5	IM4	IM3	IM2
1	1	1	0	1	1

  Control the enabling of each of the hardware interrupts.

- IE [0] : (是否接受中斷要求)

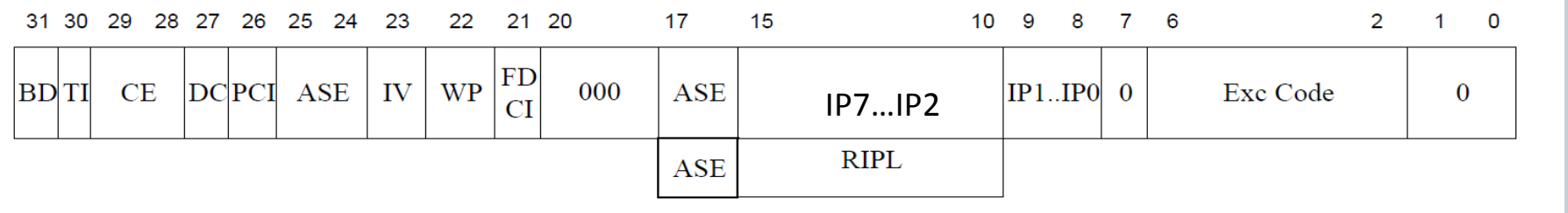
  ✓ Interrupt Enable:

Encoding	Meaning
0	Interrupt request disabled
1	Interrupt request enabled

  Act as the master enable for software and hardware interrupts.

CP0 Registers- Cause

➢ Cause :



- IP7…IM2 [15:10]:

Record the reason for the exception in the Cause register.

Bit	Name	Meaning
15	IP7	Hardware interrupt 5
14	IP6	Hardware interrupt 4
13	IP5	Hardware interrupt 3
12	IP4	Hardware interrupt 2
11	IP3	Hardware interrupt 1
10	IP2	Hardware interrupt 0

Move from Coprocessor 0

➢ Format: MFC0 rt， rd

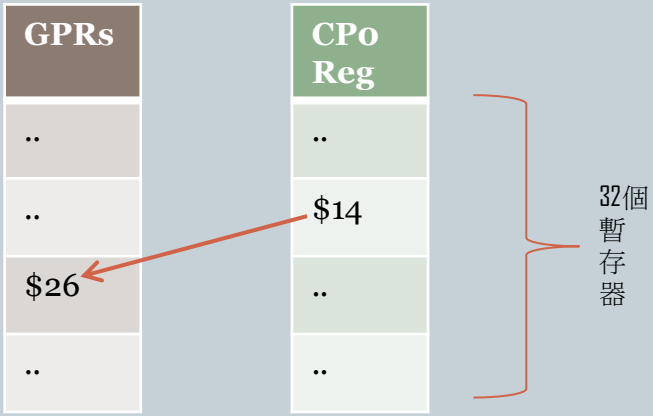
31-26	25-21	20-16	15-11	10-0
010000	00000	rt	rd	00000000000

Purpose: Move from Coprocessor 0

To move the contents of a coprocessor 0 register to a general register.

Ex. mfc0 $26， $14

$14(CP0 register) move to $26(GPRS)



Move to Coprocessor 0

➢ Format: MTC0 rt， rd

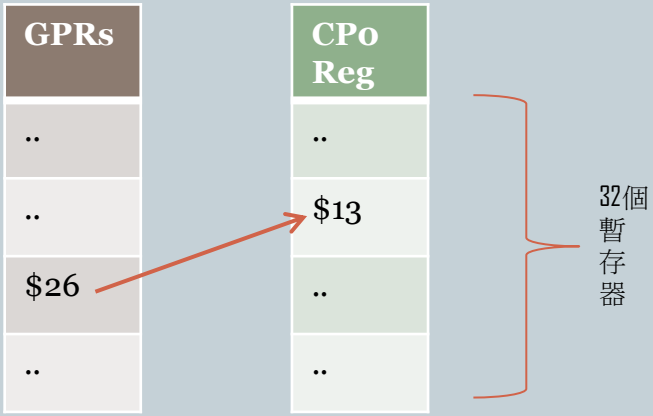
31-26	25-21	20-16	15-11	10-0
010000	00100	rt	rd	00000000000

Purpose: Move to Coprocessor 0

To move the contents of a general register to a coprocessor 0 register.

Ex. mtc0 $26， $13

$ 26(GPRS) move to $13 (CP0 register)



MIPS Interrupt Processing

- MIPS processor performs interrupt processing in five steps.

(The following steps are implemented by the processor hardware)

Step 1. save current PC

Step 2. set state giving the cause of exception

Step 3. change to kernel mode

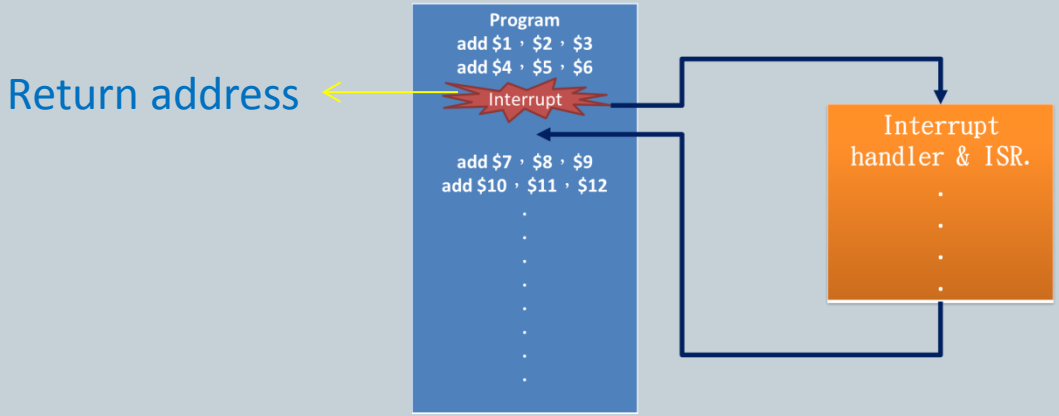
Step 4. disable further interrupts

Step 5. jump to exception handler address

Step 1. Save current PC

➢ EPC <= PC

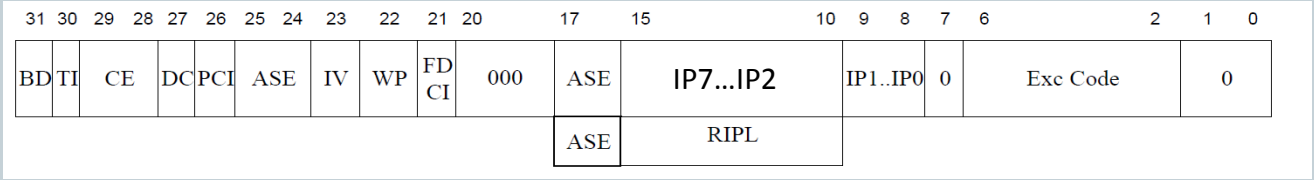
- When interrupts are raised by external causes or by the interrupt causing instructions， we need to give a return address for the interrupted program.
  - ✓ The place(Return address) is where to return to when the exception handling routine is done.
- EPC， coprocessor C0， called the Exception Program Counter.



Step 2. Set state giving cause of exception

➢ Cause <= (cause code for event)

- The processor needs to record the reason for the exception in the Cause register.
- The Cause register is a 32-bit register， but only certain fields in that register will be used.

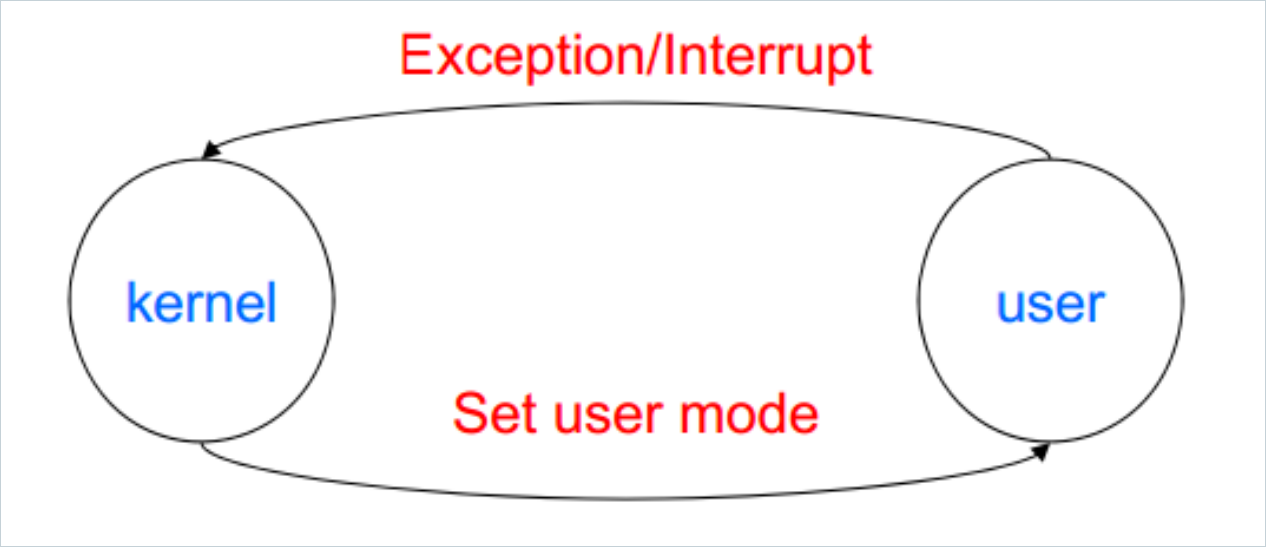


IP7	IP6	IP5	IP4	IP3	IP2
0	0	0	0	0	1

代表這個硬體是中斷的原因

Step 3. change to kernel mode

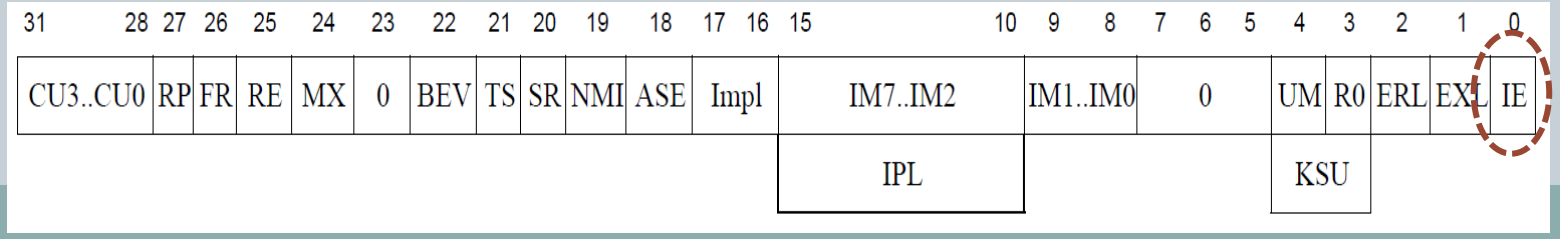
- A CPU mode bit is added to computer hardware to indicate the current CPU mode.
- When an interrupt occurs， the CPU hardware switches to the kernel mode.(kernel mode可直接操縱所有的硬體)



Step 4. Disable further interrupts

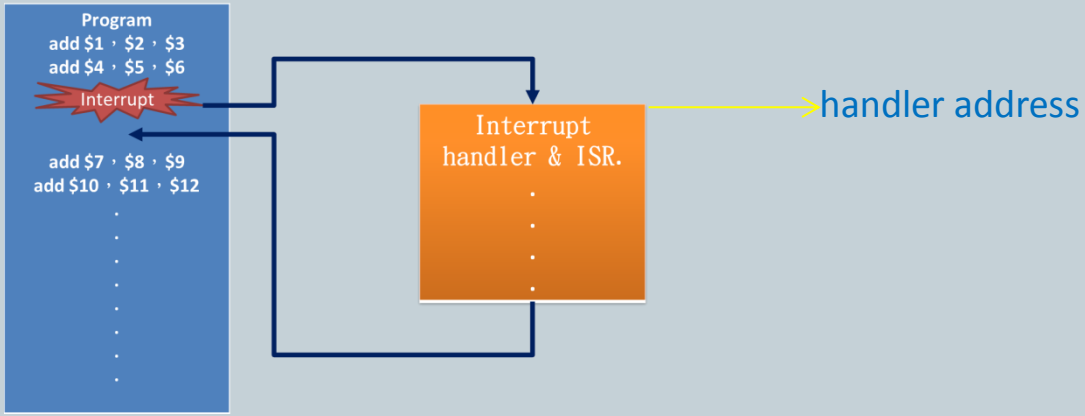
- We require a way to disable interrupts and exceptions. This is necessary to prevent further exceptions and interrupts during this phase.
- Bit 0 of the Status register， the field is called IE (Interrupt Enable)
- Enabled = 1 ，不接受中斷需求。

  Disabled = 0，接受中斷要求。



Step 5. Jump to the exception handler address

- When the processor finished the above steps， it then jumps to a hardwired exception handler address.
- Then， the interrupt service routine along with other system book-keeping codes will be executed.



實作Interrupts handler & ISR

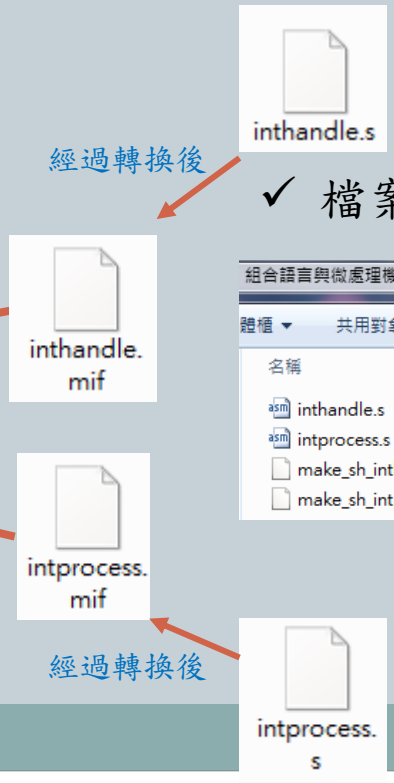
實驗環境

1. Linux
   - 因為我們使用的 MIPS Cross compiler需要在Linux下才能執行
2. MIPS Cross compiler(Compile program)
   - 我們這裡使用的是**MIPS® SDE**
   - 主要是將C轉成ASCII，以便後續的處理
3. Modelsim (Run CPU simulator)
   - 看wave來驗證執行結果是否正確
4. Java
   - 我們後續所使用的covert 和combine是由java來執行

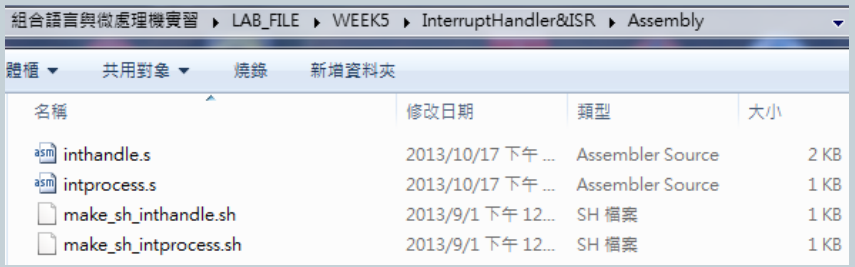
實作Interrupt handler & ISR

- 請以assembly code 完成MIPS CPU的interrupt handler & ISR 的空白部分，並且使用MIPS Cross compiler 編譯完成後放進code.mif讓CPU能夠讀取到編譯完成的指令，請同學依照流程以及規定撰寫，否則不予計分。

```
16 00000000
17 00000000
18 3C08F000
19 01000008
20 00000000
21 424F4F54
22 45584954
23 00000000
24
25 @0400
26
27
28
29 @0500
30
31
32
33
34 @2000
35 00004021
36 00004821
37 200A000B
```

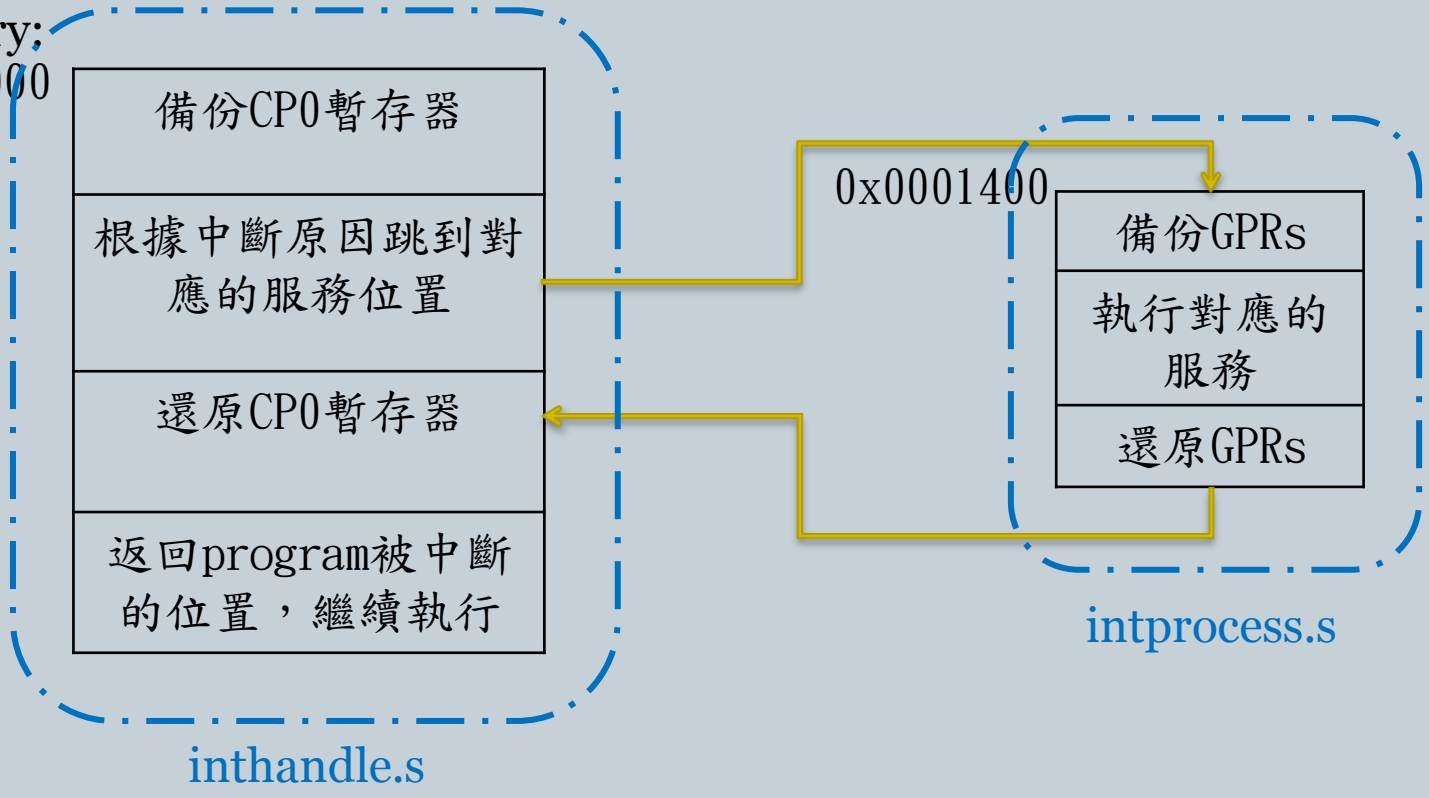


✓ 檔案在Assembly資料夾底下



Interrupt handler & ISR記憶體空間配置與流程

When interrupted，
CPU jumps to
this entry;
0x0001000



interrupt處理過程

- Interrupt handler & ISR處理過程，可以分成兩個部分

第一部分:

根據Cause暫存器，計算要跳到中斷服務程式的指令位址

- 備份CP0暫存器到memory，然後根據造成中斷的原因跳去對應的服務程式，然後執行完對應的服務會再度跳回來，接著還原CP0暫存器、enable interrupt ，最後才跳回program被中斷的位置。

根據被還原的EPC暫存器，可得到program被中斷的指令位址

第二部分:

- 備份GPRs暫存器到memory stack ，處理中斷對應的服務，再從還原先前備份的暫存器，最後跳到第一階段尚未完成的部分。

第一部分

- Step1. 1
  進入對應的服務位置的之前:
  必須先備份CP0_Epc($14)、 CP0_Cause($13) 、CP0_Status($12) 、到memory 中。

- Step2. 2
  根據Cause暫存器，計算要跳到中斷服務程式的指令位址。

- Step3. 3
  從對應的服務位置跳回來之後:
  此時必須要將Epc 、 Cause 、 Status從memory還原到暫存器中，enable interrupt(修改CP0_Status)。

- Step4. 4
  根據被還原的EPC暫存器，可得到program被中斷的指令位址。

第一部分

請同學參考前面說明，並且完成程式碼(inthandle.s)

inthandle.s (~/Desktop/lab6.inthandler) - gedit

File Edit View Search Tools Documents Help

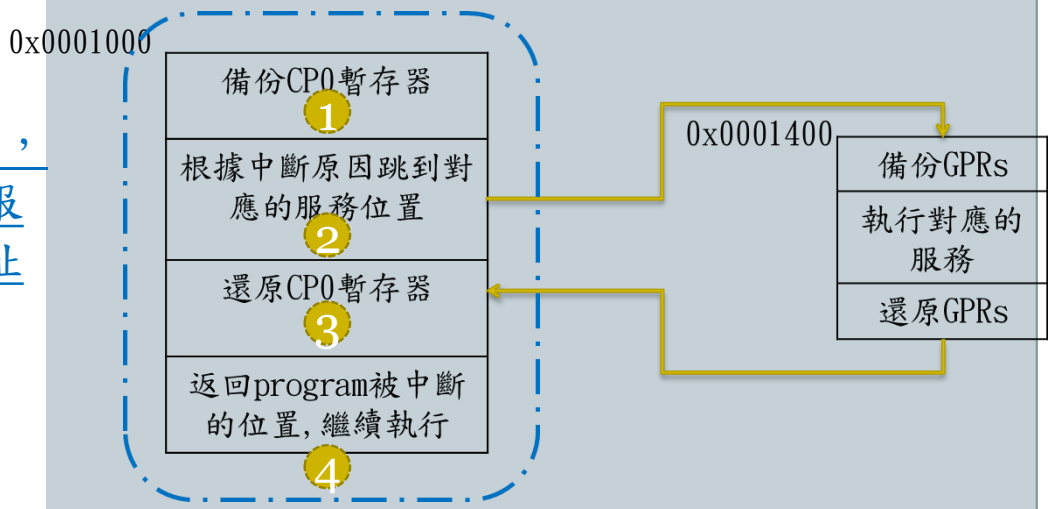
New Open Save Print... Undo Redo Cut Copy Paste Find Replace

inthandle.s

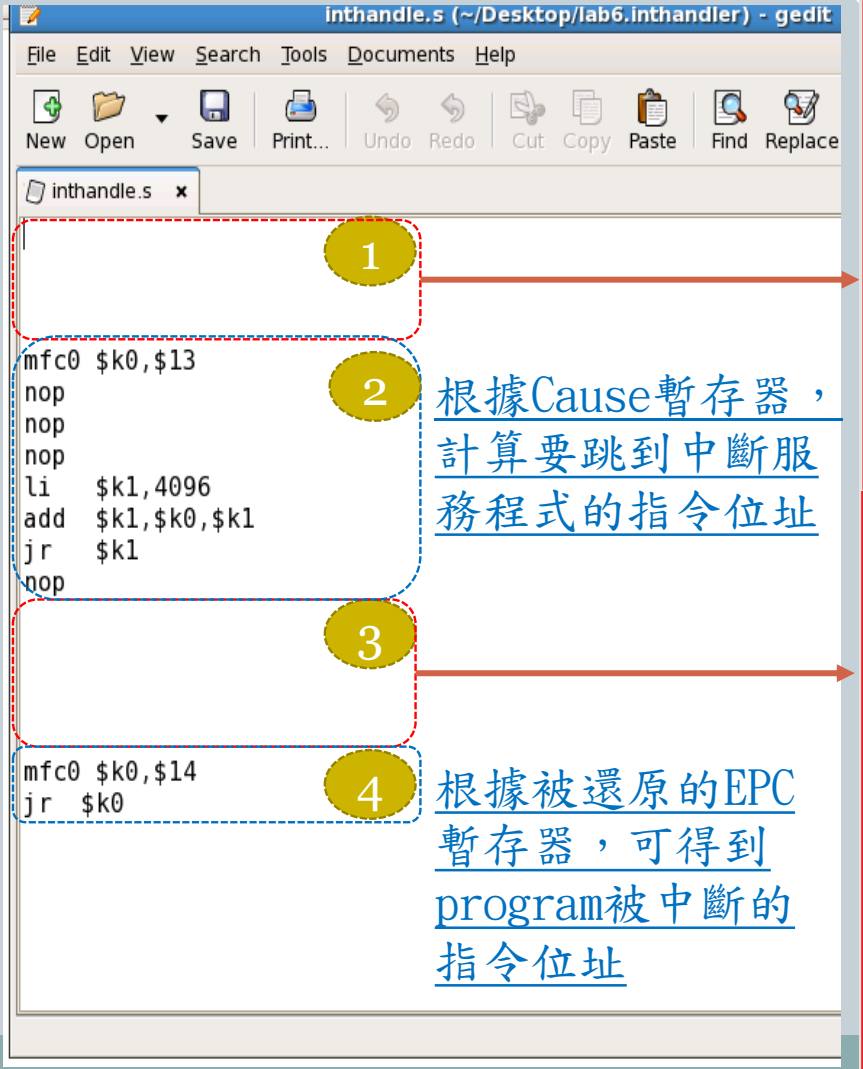
```
1

mfc0 $k0,$13
nop
nop
nop
li   $k1,4096
add  $k1,$k0,$k1
jr   $k1
nop

3

mfc0 $k0,$14
jr  $k0
```

2 根據Cause暫存器，計算要跳到中斷服務程式的指令位址

4 根據被還原的EPC暫存器，可得到program被中斷的指令位址



第一部分(提示)




```
addiu $sp,$sp,-140 #(begin)backup CP0 to MEM Stack
mfc0 $k0,$14     #(begin)backup CP0_Epc($14) to MEM Stack(MEM[$sp+4])
nop
nop
nop
sw   $k0,4($sp) #(end)backup CP0_Epc($14) to MEM Stack(MEM[$sp+4])
                #(begin)backup CP0_Cause($13) to MEM Stack(MEM[$sp+8])
nop
nop
nop
                #(end)backup CP0_Cause($13) to MEM Stack(MEM[$sp+8])
                #(begin)backup CP0_Status($12) to MEM Stack(MEM[$sp+12])
nop
nop
nop
                #(end)backup CP0_Status($12) to MEM Stack(MEM[$sp+12])
addiu $sp,$sp,140
nop
nop
nop
```

```
lw   $k0,4($0)   #(begin)restore CP0_Epc($14) from MEM Stack(MEM[$sp+4])
nop
nop
nop
mtc0 $k0,$14     #(end)restore CP0_Epc($14) from MEM Stack(MEM[$sp+4])
nop
nop
nop
                 #(begin)restore CP0_Cause($13) from MEM Stack(MEM[$sp+8])
nop
nop
nop
                 #(end)restore CP0_Cause($13) from MEM Stack(MEM[$sp+8])
addiu $sp,$sp,-140
nop
nop
nop
                 #(begin)restore CP0_Status($12) from MEM Stack(MEM[$sp+12])
nop
nop
nop
                 #(end)restore CP0_Status($12) from MEM Stack(MEM[$sp+12])
nop
nop
nop
li $k1,64513     #(begin)set CP0_Status($12)= 0000FC01 ,ENABLE IP7~IP2 & BIT 0
                 #(end)set CP0_Status($12)= 0000FC01 ,ENABLE IP7~IP2 & BIT 0
```

編譯第一部分程式

- 在linux下打開Terminal接著輸入以下指令

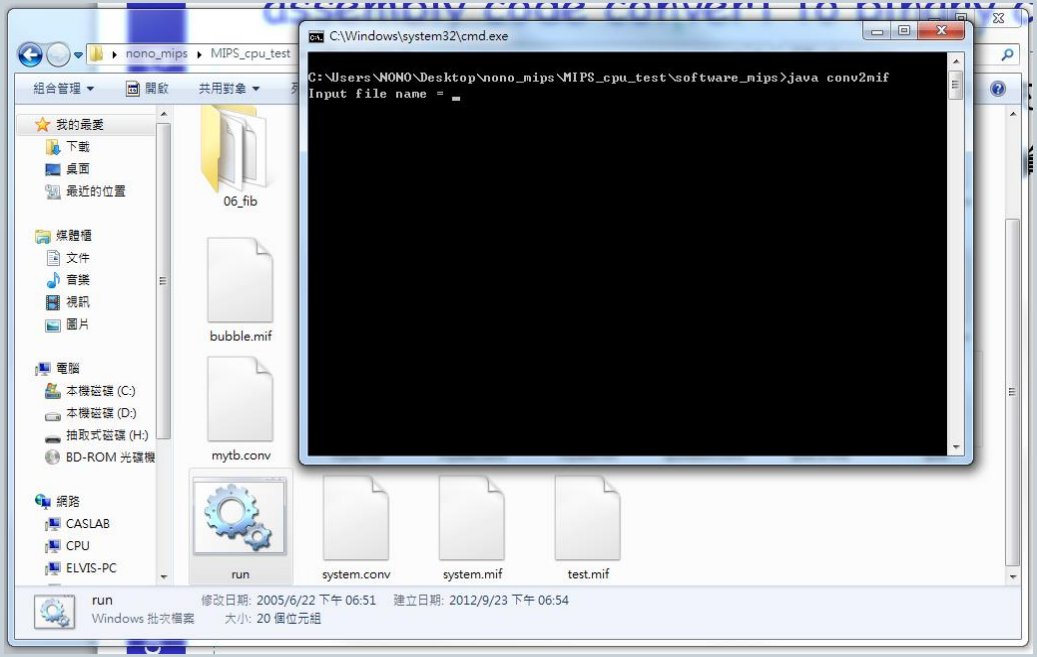
- **sde-as inthandle.s -g -o inthandle.o**
- **sde-ld inthandle.o -Ttext 0x00001000 -o inthandle.image**
- **sde-conv inthandle.image -o inthandle.conv**
- **sde-objdump inthandle.image -D > inthandle.txt**

- **-Ttext 0x00001000表示執行起始位置放在0x00001000**
- **指令詳細介紹請參考LAB1敘述**

（或者執行我們預先寫好的make_sh_inthandle.sh，此檔放在Assembly底下）

放入編譯好的程式

1. 把cross complier出來的inthandle.conv放到converter資料夾
2. 在converter資料夾底下，執行conv2mif.bat，產生inthandle.mif
3. 將inthandle.mif，放到MIPS_CPU_TEST/software_mips底下



放入編譯好的第一部分程式

```
16 00000000
17 00000000
18 3C08F000
19 01000008
20 00000000
21 424F4F54
22 45584954
23 00000000
24
25 @0400
26
27
28
29 @0500
30
31
32
33
34 @2000
35 00004021
36 00004821
37 200A000B
```

打開software_mips資料夾中的code.mif且在紅框放入編譯好的inthandle.mif檔

◎0x0400*4=0x00001000

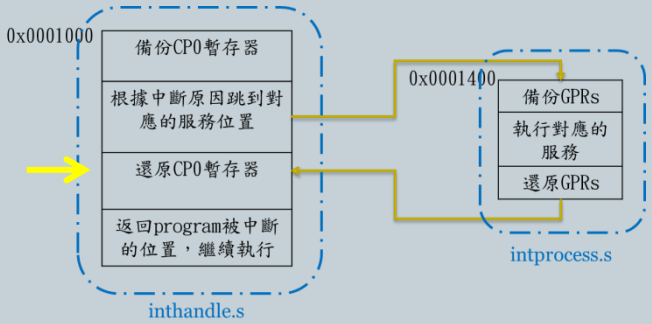
第二部分

- Step1. 1

  對應的服務執行時，需要先備份GPRs暫存器($1~$31) 到memory stack中，當備份完畢，方可使用GPRs暫存器。

- Step2. 2

  執行中斷服務程式。



- Step3. 3

  我們需要還原暫存器，返回尚未執行完的program才不會發生錯誤。

- Step4. 4

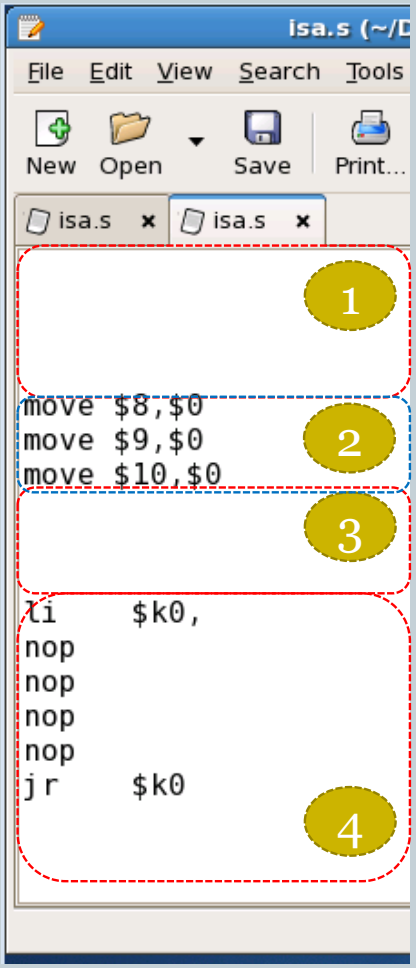
  接下來必須返回剩餘的部分，此時我們需要知道返回的地址(並非0x00001000)，因此我們先打開剛剛編譯完成的inthandle.txt去看跳到第二部分指令的下一條指令位置，舉例如下圖。

```
1048:       401a6800        mfc0    $k0,$cause
    ...
1058:       241b1000        li      $k1,4096
105c:       035bd820        add     $k1,$k0,$k1
1060:       03600008        jr      $k1
1064:       00000000        nop
    ...
1074:       8c1a0004        lw      $k0,4($zero)
```

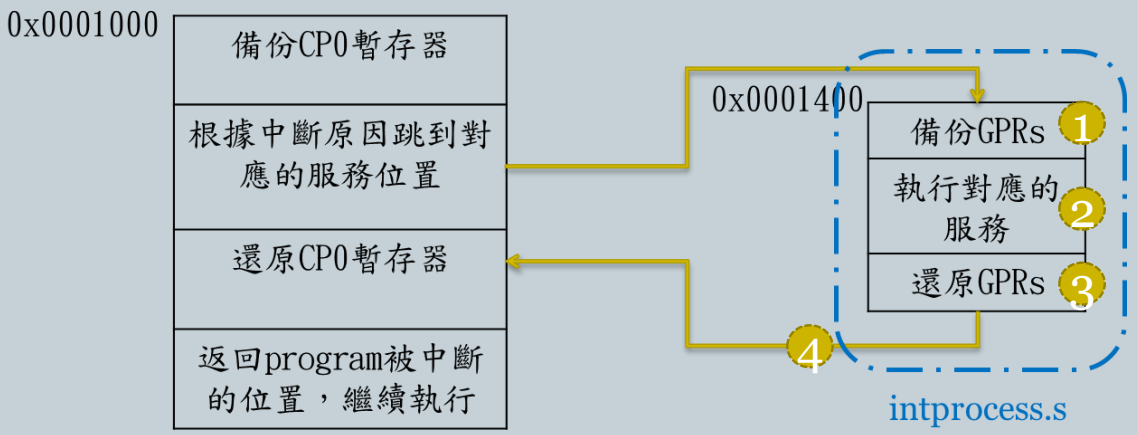
跳到中斷服務程式的指令位址

中斷服務程式執行完成後，要返回的指令位址

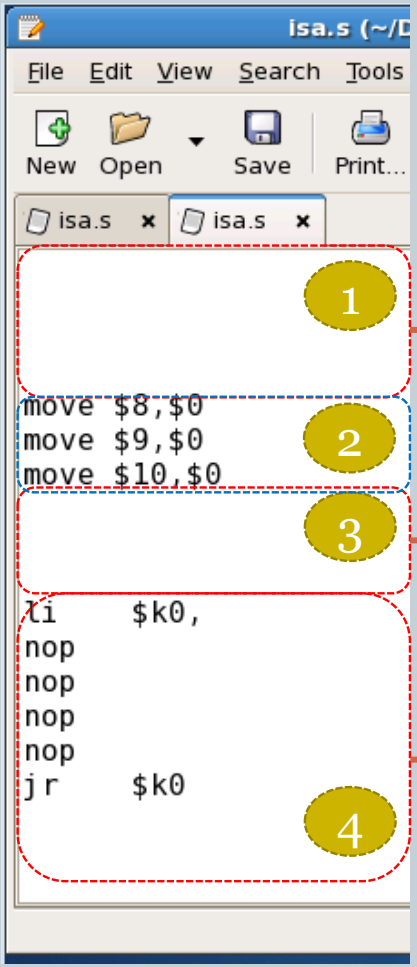
第二部分



請同學參考前面說明，並且完成程式碼(intprocess.s)



第二部分(提示)




```
addiu $sp,$sp,-128  #(begin)backup GPRs($1~$31) to MEM Stack
move  $k0,$1
sw    $k0,0($sp)

      .
      .
      .
      .

move  $k0,$31
sw    $k0,120($sp)  #(end)backup GPRs($1~$31) to MEM Stack
```

```
lw    $1,0($sp)     #(begin)restore GPRs($1~$31) from MEM Stack

      .
      .
      .
      .

lw    $31,120($sp)  #(end)restore GPRs($1~$31) from MEM Stack
```

```
li    $k0,????      #jump return ????(address) that can get from inthandle.txt
nop
nop
nop
nop
jr    $k0
```

```
1048:       401a6800        mfc0    $k0,$cause
    ...
1058:       241b1000        li      $k1,4096
105c:       035bd820        add     $k1,$k0,$k1
1060:       03600008        jr      $k1
1064:       00000000        nop
    ...
1074:       8c1a0004        lw      $k0,4($zero)
```

跳到中斷服務程式的指令位址

中斷服務程式執行完成後，要返回的指令位址

編譯第二部分程式

- 在linux下打開Terminal接著輸入以下指令

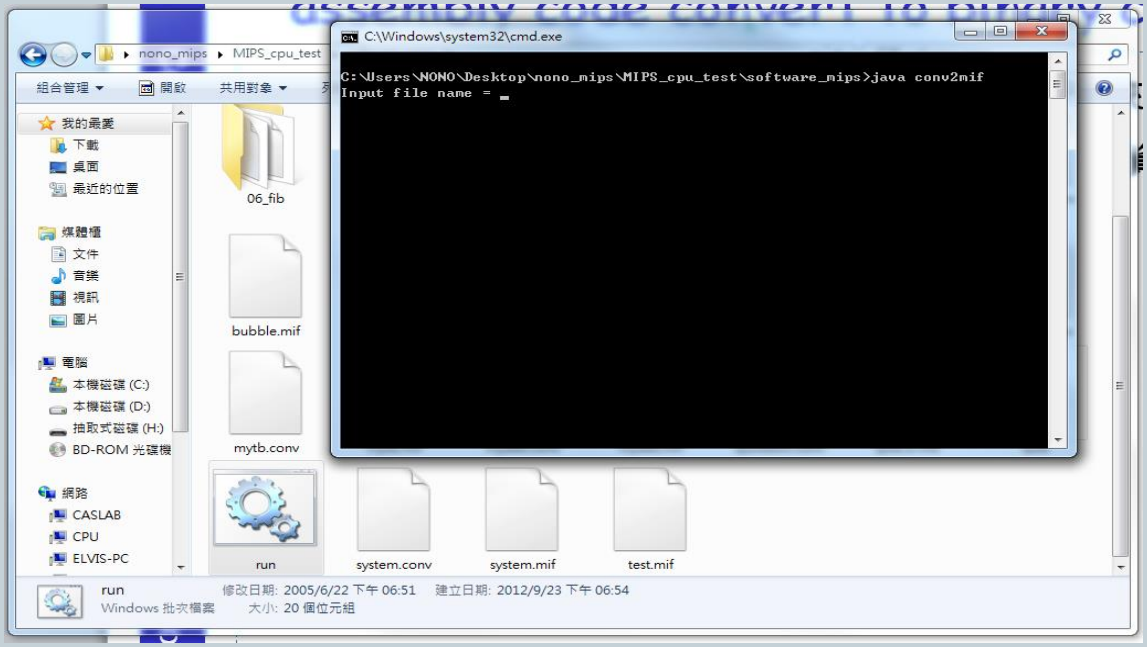
- **sde-as intprocess.s -g -o intprocess.o**
- **sde-ld intprocess.o -Ttext 0x00001400 -o intprocess.image**
- **sde-conv intprocess.image -o intprocess.conv**
- **sde-objdump intprocess.image -D > intprocess.txt**

- **-Ttext 0x00001400表示執行起始位置放在0x00001400**

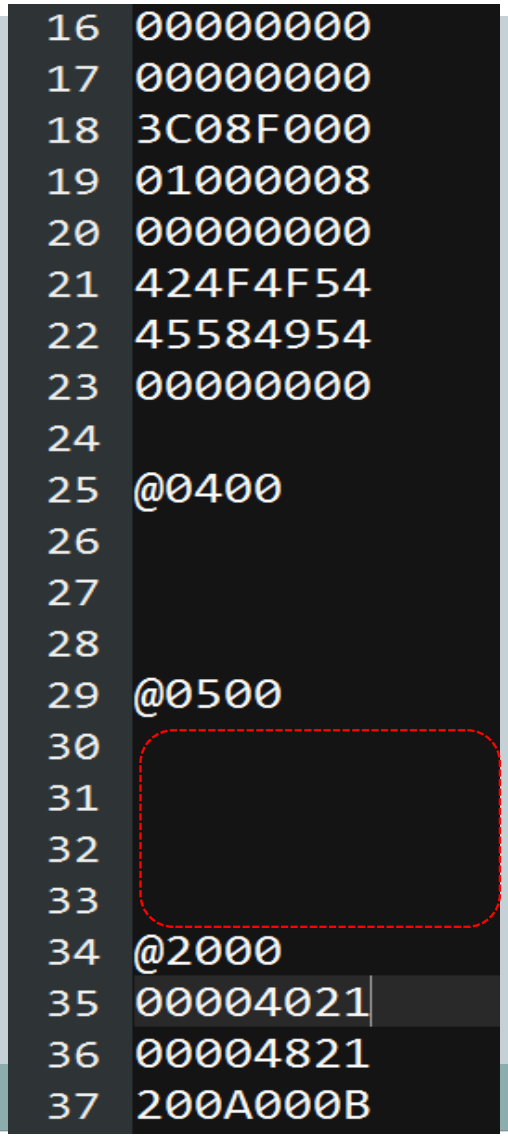
(或者執行我們預先寫好的make_sh_intprocess.sh，此檔放在Assembly底下)

放入編譯好的程式

1. 把cross complier出來的intprocess.conv放到converter資料夾
2. 在converter資料夾底下，執行conv2mif.bat，產生intporcess.mif
3. 將intporcess.mif，放到MIPS_CPU_TEST/software_mips底下



放入編譯好的第二階段程式



打開software_mips資料夾中的code.mif且在紅框放入編譯好的intprocess.mif檔

◎0x0500*4=0x00001400

參考指令

- li:載入常數放入暫存器
- jr:跳到暫存器內存值位置
- mfc0:從cp0 register 取出值存到GPRs
- mfc0:從GPRs 取出值存到cp0 register

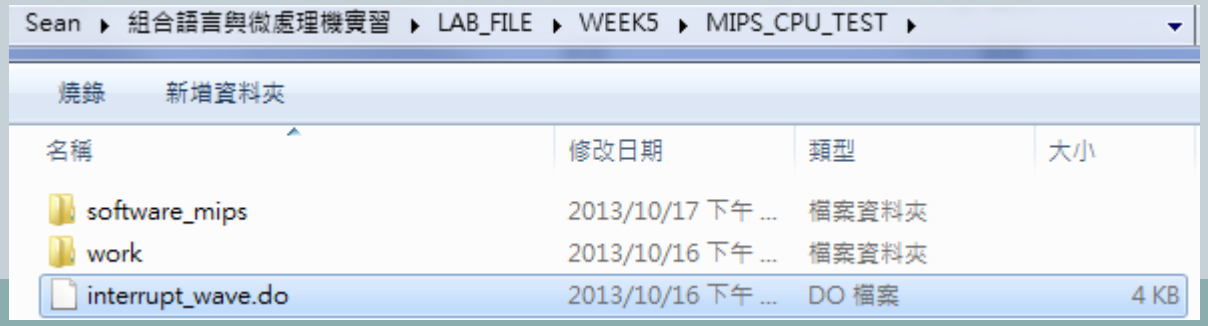
驗證結果

檢查系統是否成功備份資料，並且處理Interrupt handler & ISR後，返回Program，是否會影響原本的結果

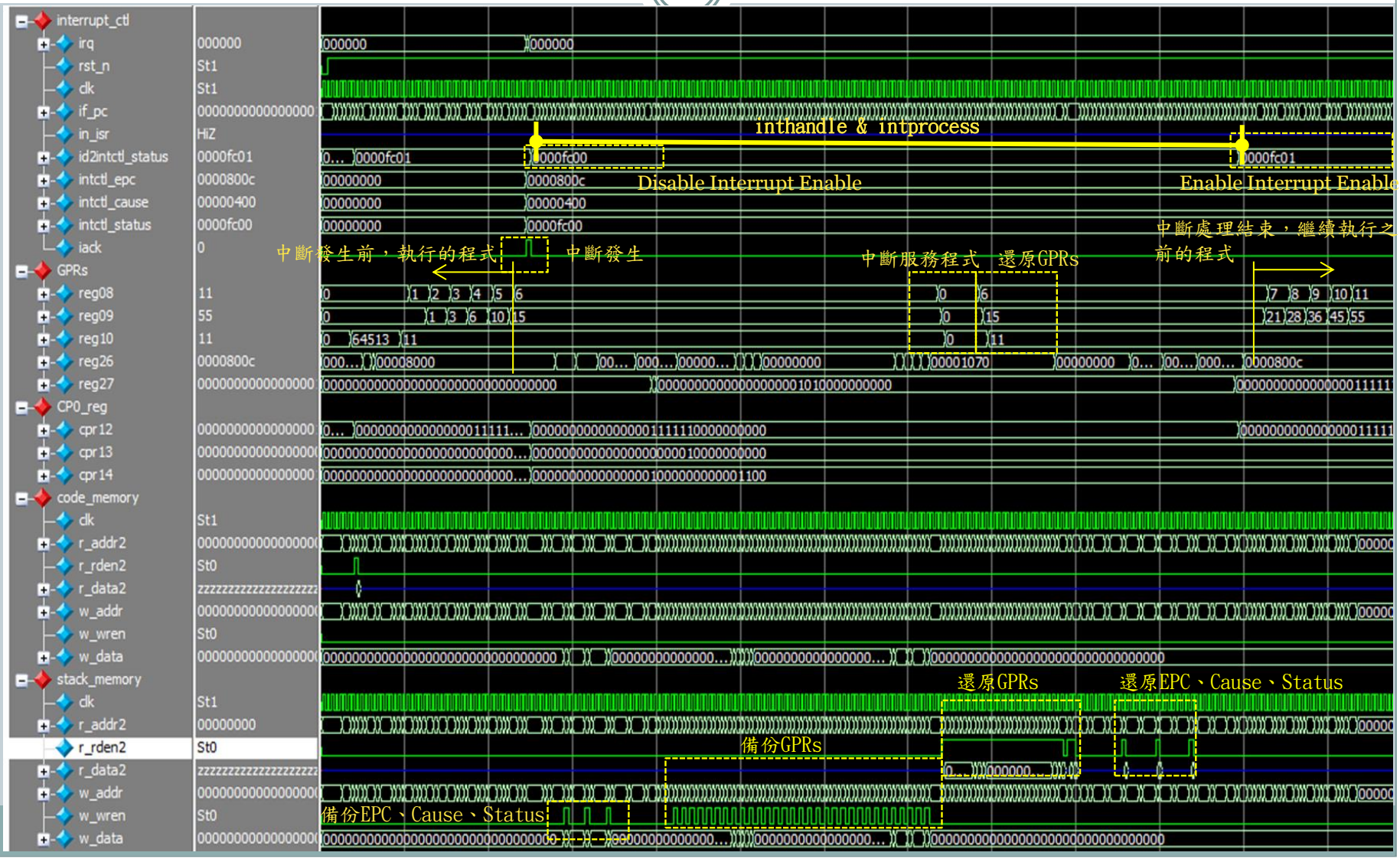
➢ 驗證項目:

1) GPRs是否成功備份、還原
2) EPC， Status， Cause是否成功備份、還原
3) Interrupt handler&ISR是否成功執行
4) 離開中斷程式之前，是否 enable interrupt

✓要觀測的波型圖檔，在MIPS_CPU_TEST底下。



WAVE


interrupt_ctl
irq
rst_n
clk
if_pc
in_isr
id2intctl_status
intctl_epc
intctl_cause
intctl_status
iack
GPRs
reg08
reg09
reg10
reg26
reg27
CP0_reg
cpr12
cpr13
cpr14
code_memory
clk
r_addr2
r_rden2
r_data2
w_addr
w_wren
w_data
stack_memory
clk
r_addr2
r_rden2
r_data2
w_addr
w_wren
w_data
inthandle & intprocess
Disable Interrupt Enable
Enable Interrupt Enable
中斷發生前，執行的程式
中斷發生
中斷服務程式
還原GPRs
中斷處理結束，繼續執行之前的程式
還原GPRs
還原EPC、Cause、Status
備份GPRs
備份EPC、Cause、Status