1. QtSpim: Menu -> Simulator -> setting: Uncheck load exception handler.

2. Start_up routine:

The SPIM's start_up address on reset is at '__start'. Do the following before you run a user program sitting at main.

First thing first. Disable CPU's interrupt enable bit. This is to prevent an accident by accidentally running ISR before system setup is completed. At the same time, set the interrupt mask to be all 1 to accept every interrupt of the whole world. And user mode bit is permanently 1 on the simulator. That is how QtSPIM sets**. QtSPIM comes up with the status to be 3000FF10 as default. That is 0011 0000 0000 0000 1111 1111 0001 0000**

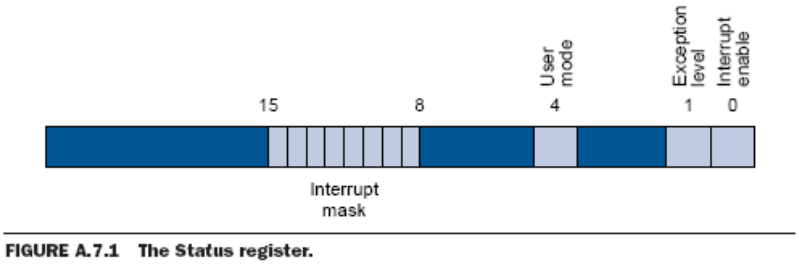

FIGURE A.7.1 The Status register.

That means, QtSpim's interrupu mask is **all 1**, thus **everyone can interrupt**. The user mode bit 4 is 1. Interrput disabled.

3. Enable the devices' interrupt enable bit.
4. Enable CPU's interrupt bit to accept interrupts.
5. Now, everything is ready, so run a user program by jumping to main.

ISR(or exception handler):

1. Locate your ISR or kernel code at 0x80000180. Syntax for that is ".ktext 0x80000180". Store regitsters you are going to use if any. $k0 and $k1 can be used freely.
Do not use pseudo instruction here unless you save and restore $at register, because pseudo instructions are translated often using $at.

2. Check the cause register exception code, and if it is hardware interrupt, then it has to be timer, keyboar input, or terminal output. Because, we do not have anything else on SPIM! So, find out who interrupted by checking Pentding interrupts bits:

 # c0$13 = 32768 (1000 0000 0000 0000) timer
 # = 2048 (0000 1000 0000 0000) key hit
 # = 1024 (0000 0100 0000 0000) terminal output

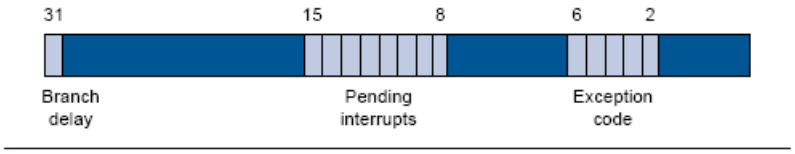

FIGURE A.7.2 The Cause register.

I could not find the description on Pending interrupt bits on SPIM. So, I did experiment. If Exception code is 00000, then look at Pending interrupts to find out who interrupted.

4. Before returning from exception,

4.1

If it is software interrupt, then advance EPC by 4. This is because if it is syscall, EPC = PC. If hardware interrupt, current instruction will be completed and EPC is set to the next instruction address. BD is on, EPC is on branch instruction of that branch delay slot.

4.2

Clear cause the register by; mtc0 $0, $13

4.3

Restore registers if you modified them

Clear Exception level bit of the status register. This bit is set to 1 when exception occurs. If 1, then interrupt is momentarily disabled until reset. Thus reset at before returning from exception handling.

```
mfc0 $k0, $12
andi $k0, 0xfffd
ori    $k0, 0x1
mtc0 $k0, $12
```

Then, **eret** /* exception return, there are a few version of this: jr, rfe, */

User program:

1. Disable interrupt via status register. This is to prevent queue pointers overwritten by ISR.

2. Check the buffer to see if any characters are deposited.

3. If yes, retrieve it and update tail pointer.

4. Enable interrupt.

5. Display the character.

6. Simulate some other jobs by writing an empty loop with a big loop counter.

7. Goto 1.

Data receiver from a Keyboard and data transmitter to a Monitor.

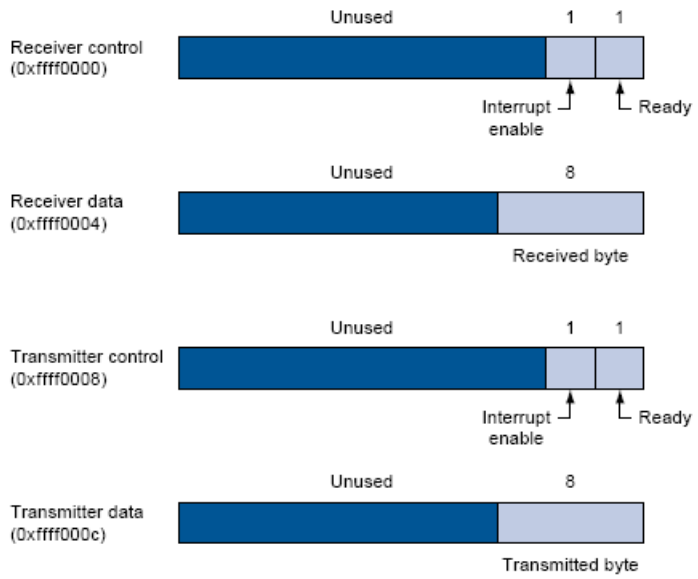

FIGURE A.8.1 The terminal is controlled by four device registers, each of which appears as a memory location at the given address. Only a few bits of these registers are actually used. The others always read as 0s and are ignored on writes.

character input into $v0

```
  lui $t0,0xffff  #ffff0000
waitloop:
  lw  $t1,0($t0)  #control
  andi    $t1,$t1,0x0001
  beq $t1,$0,waitloop
```

output of character in $a0

```
  lui $t0,0xffff  #ffff0000
waitloop1:
  lw    $t1,8($t0)  #control
  andi   $t1,$t1,0x0001
  beq    $t1,$0,waitloop
```

```
  lw  $v0,4($t0)  #data            sw    $a0,12($t0) #data
```

Register name	Register number	Usage
BadVAddr	8	memory address at which an offending memory reference occurred
Count	9	timer
Compare	11	value compared against timer that causes interrupt when they match
Status	12	interrupt mask and enable bits
Cause	13	exception type and pending interrupt bits
EPC	14	address of instruction that caused exception
Config	16	configuration of machine

Cause Register & Interrput mask bit locations.

coprocessor 0 register $13 cause register bit description
#
c0$13 = 32768 (1000 0000 0000 0000) timer
= 2048 (0000 1000 0000 0000) key hit
= 1024 (0000 0100 0000 0000) terminal output

```
#------------------------
#This is how it works.
#
#Data Structure:
#
# queue: This tutorial does not implement a queue. Just one byte
#(or word) location.
# so, there is no head or tail business.
#
#Programs:
#
#  ISR:
#  installed at ktext segment 0x80000180
#  This program is invoked when key hit, and
#  Reads keyboard
#  Stores the data in queue.
#  then exception return
#  End of ISR
#
#  __Start: // simulated power on
#  initializes the CPU status register
#  initializes the keyboard to enable interrupt.
```

```
#  set CPU status register interrupt enable
#  jump to main (user program)
#
#  main:
#  loop
#    print queue
#  endloop



#-------------------------
#   CODE
#-------------------------

# This is terribly minimal, but ......

   .kdata
   .ktext 0x80000180


       # read keyboard // in real, check keyboard data ready.
       # and store it

       # clear cause reg
       # read status reg
       # enable intrpt
       # write to status reg

    eret  # ret from exception (PC <- EPC)


   .text
   .globl __start
       .globl main
__start:

# device interrupt enable
# CPU interrupt enable
# QtSpim default status register: 0x3000FF10
#
# Program keyboard: enable interrupt request
lui $s0, 0xFFFF   # control register
li  $t0, 0x02     # turn on enable bit
sw  $t0, 0($s0)   # write to keyboard


# Now, system initialization done, Let's go!
```

```
mfc0 $t0, $12     # read status register
ori  $t0, $t0, 0x0001 # set interrupt enable bit on
mtc0 $t0, $12     # write to status register

jal main
nop

main:
loop:
      # read keyboard data from ch in .data section.
      # print ch
      # for(i=0; i<100000; i++); let's slow down
j loop


.data
ch: .space 4

.align 2
count: .space 4
head:  .space 4
tail:  .space 4
```