CSCD260 Computer Talk on May 23, 2019

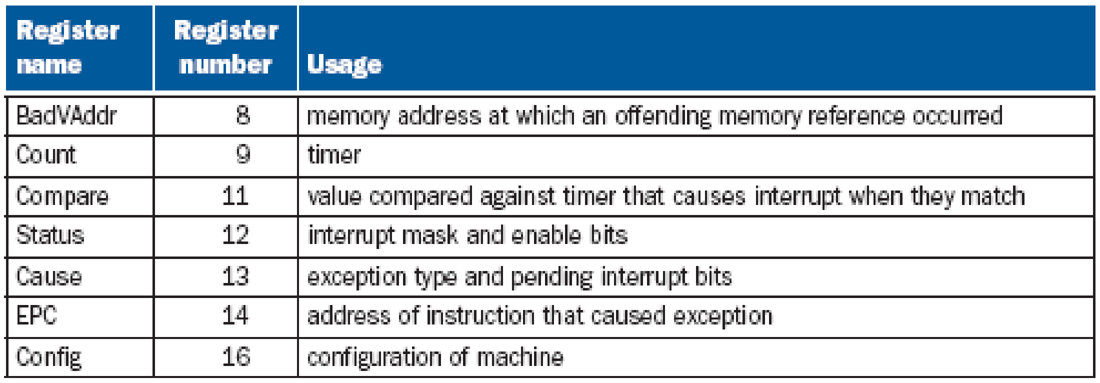
Configure QtSpim: mapped I/O and no exception handler load.
Click Simulator -> Settings -> MIPS

Now, we are writing code that makes a computer tick.

What is the interrupt?

1. When external devices need CPU’s attention, they raise interrupt requests.
2. CPU sets EPC = current PC + 4.
3. CPU stops instruction fetch and finishes executing instructions already in the pipe.
4. When 3 is completed, CPU starts fetching instructions from 0x80000180.
5. The eret (exception return) instruction restores PC from EPC.
6. CPU resumes the program previously running.

Interrupt Service Routine Programming



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

1. 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.

Make sure that CPU's interrupt enable bit is 0. This is to prevent an accident by accidentally running ISR before system setup is completed.

Startup QtSpim status register: 0011 0000 0000 0000 1111 1111 0001 0000
This shows up the QtSpim simulator’s left upper corner.
Anytime you want to disable the interrupt enable bit;

```
mfc0 $a0, $12         # a0 = xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
lui  $t0, 0xFFFF      # t0 = 1111 1111 1111 1111 0000 0000 0000 0000
ori  $t0, $t0,0xFFFE  # t0 = 1111 1111 1111 1111 1111 1111 1111 1110
and  $t0, $a0,$t0     # t0 = xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxx0
mtc0 $t0, $12         # Status reg bit-0 set 0 without disturbing other bits.
```

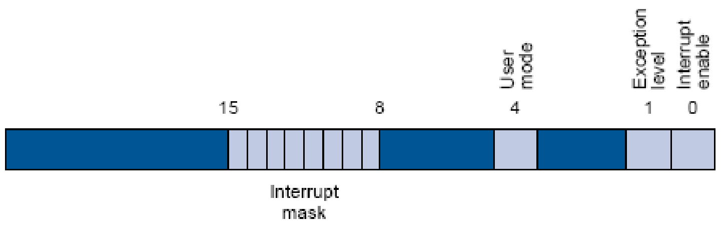


FIGURE A.7.1 The Status register.

2. Initialize devices.

Program the keyboard so that it can raise an interrupt request.

```
lui $t0, 0xFFFF  # t0 = 0xFFFF0000      // control register address
addi $t1, $0, 2  # t1 = 0000 ··· ··· 0010
sw  $t1, 0($t0)  # Receiver (keyboard) control = 0000 ··· 0010
# now interrupt enable bit is enabled (can raise interrupt request).
```

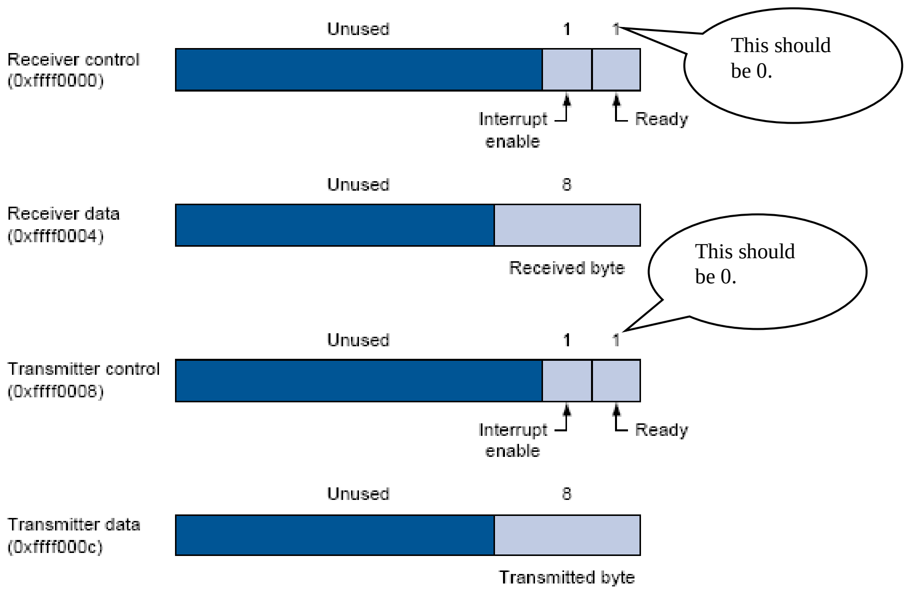


FIGURE A.8.1 The terminal is controlled by four device registers, each of which appears as a memory location at the given address. Only a few bits of these registers are actually used. The others always read as 0s and are ignored on writes.

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

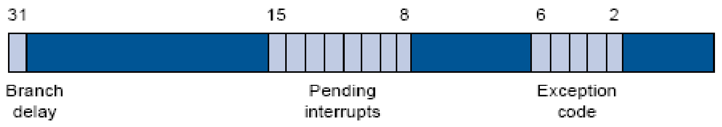


FIGURE A.7.2 The Cause register.

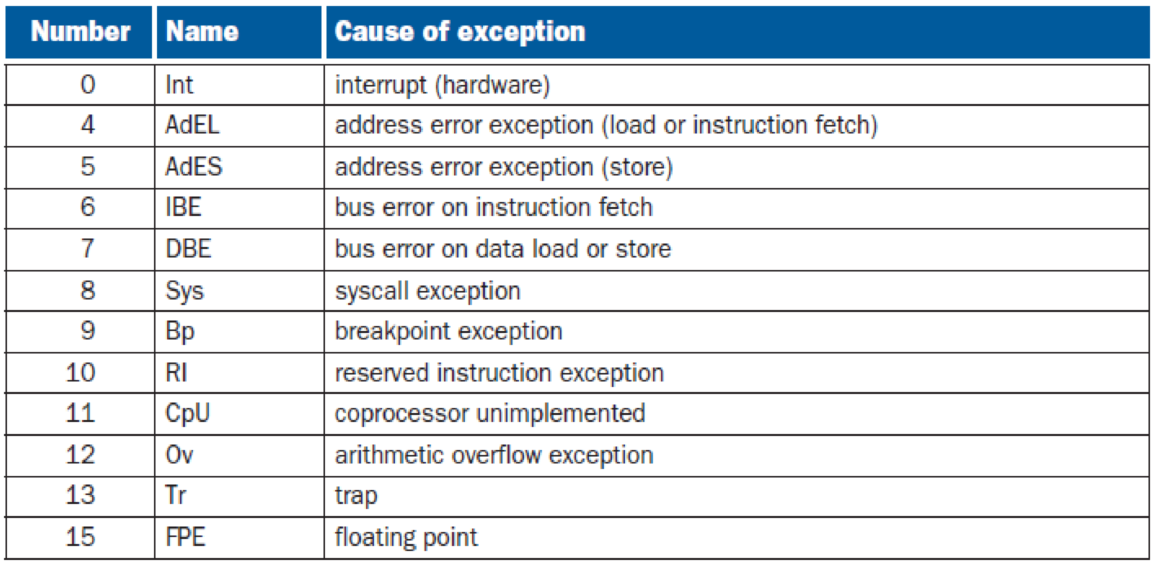
Now, ISR writing:
The job of MIPS ISR is to find what kind of interrupt and which device requested it.
The cause register is $13 on coprocessor-0 (Do not worry about the branch delay bit).

Check out what type of exception it is by reading the cause register $13 on the coprocessor 0:

```
mfc0 $a0, $13        # a0 = xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
addi $t0, $0, 0x7C   # t0 = 0000 0000 0000 0000 0000 0000 0111 1100
and  $t0, $a0, $t0   # t0 = 0000 0000 0000 0000 0000 0000 0xxx xx00
and  $t0, $a0,$t0    # t0 = 0000 0000 0000 0000 0000 0000 0xxx xx00
```

# now you can read the exception code.

The exception code is:
```
srl $t0, $t0, 2  # t0 = t0>>2;
                 # t0 = 0000 0000 0000 0000 0000 0000 000x xxxx
```
if (xxxxx == 0) hardware interrupt
else if (xxxxx == 0) Instruction fetch (address) error
else …. See below for exception code.



Number	Name	Cause of exception
0	Int	interrupt (hardware)
4	AdEL	address error exception (load or instruction fetch)
5	AdES	address error exception (store)
6	IBE	bus error on instruction fetch
7	DBE	bus error on data load or store
8	Sys	syscall exception
9	Bp	breakpoint exception
10	RI	reserved instruction exception
11	CpU	coprocessor unimplemented
12	Ov	arithmetic overflow exception
13	Tr	trap
15	FPE	floating point

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 QtSpim!
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
If Exception code is 00000, look at Pending interrupts to find out who interrupted.

For instance to see if it is a key hit interrupt;
(0000 1000 0000 0000) is a keyboard interrupt pending bit:

```
mfc0 $a0, $13        # a0 = xxxx xxxx xxxx xxxx xxxx xxxx xxxx xxxx
addi $t0, $a0,0x0800 # t0 = 0000 0000 0000 0000 0000 1000 0000 0000
and  $t0, $a0,$t0    # t0 = 0000 0000 0000 0000 0000 x000 0000 0000
```

If it is a keyboard interrupt, then read data from the keyboard.
Store it somewhere so a user program can retrieve it.

Before returning from exception,

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

```
         mtc0 $0,  $13
         mfc0 $k0, $12       # read status reg
         andi $k0, 0xfffd    # 1111 1111 1111 1101 //bit1 reset
         ori  $k0, 0x1       # enable interrupt
         mtc0 $k0, $12
        eret
```

“eret” is exception return, there are a few version of this: jr, rfe, depending on MIPS versions.

Let’s try our first test ISR, which is the minimal to prove interrupt happened and we captured by our program. To do so,

ISR at 0x80000180
  Well, let’s die here!
  If we die here, we are in good shape, because CPU executed this code.
  That means, interrupt happened, and our ISR ran and died.
  One way to die is forever loop.

__start
Let’s set up CPU and initialize the keyboard.
Since the status register comes up default interrupt disabled, let’s initialize the keyboard.
That is, just set the keyboard control register interrupt enable bit.
Now the system is ready to run, let’s set CPU interrupt enable bit on.
All set, jump to the main, which is the user program.

So, your first program is like:

```
    .kdata
    temp0: .space 4   # <- v0
    temp1: .space 4   # <- a0
    temp2: .space 4   # <- t0 etc, etc, ....

    .ktext 0x80000180  # ISR starting address

die: j die     ### we sleep here forever……
```

```
###############################
######    start main   ########
###############################

    .text
    .globl __start
        .globl main
__start:
lui  $t0, 0xFFFF  # t0 = 0xFFFF0000      // control register address
addi $t1, $0, 2   # t1 = 0000 … … 0010
sw   $t1, 0($t0)  # Receiver (keyboard) control = 0000 … 0010

mfc0 $t0, $12         # I read 0x3000ff10
ori  $t0, $t0, 0x0001 # Turn ON Interrupt enable bit
mtc0 $t0, $12         # write it to the status register
j main                # start main
nop
nop


main:
   # this is where you have user code.
done:  nop
       nop
    j done
```