INDIAN INSTITUTE OF TECHNOLOGY GOA

COMPUTER ARCHITECTURE LAB (CS 211)

Faculty : Dr. Sharad Sinha
Teaching Assistant: Prachi
1st Contributor: Pavitra P. Bhade

LAB 03

In this course, you will learn the exception handling mechanism in MIPS processor and also write Extended Assembly level programs using the QtSpim MIPS32 simulator.

We will be splitting this lab exercise in to two parts:

PART A:

In this part, you will be introduced to the concept of Exception Handling and Extended Assembly codes.

1. Exception Handling:

Branches and jumps provide ways to change the control flow in a program. Exceptions can also change the control flow in a program. The MIPS architecture calls an exception any unexpected change in control flow, regardless of its source. When an exception happens, control is transferred to an exception handler, written specifically for the purpose of dealing with exceptions. After executing the exception handler, control is returned to the program. The program continues as if nothing happened. The exception handler appears as a procedure called suddenly in the program with no parameters and no return value.

The MIPS processor operates either in user or kernel mode. User programs (applications) run in user mode. The CPU enters the kernel mode when an exception happens.

MIPS exceptions are handled by a peripheral device to the CPU called *coprocessor 0* (cp0). Coprocessor 0 contains a number of registers, but QtSPIM does not implement all of these registers, since they are not of much use in a simulator. However, it does provide the following:

1. BadVAddr: Contains the invalid memory address caused by load, store, or fetch.
2. Status: Contains the interrupt mask and enable bits.
3. Cause: Contains the type of exception and any pending bits.
4. EPC: Contains the address of the instruction when the exception occurred.

The QtSpim tool shows the values of these registers, in the register area as shown in Fig 1.

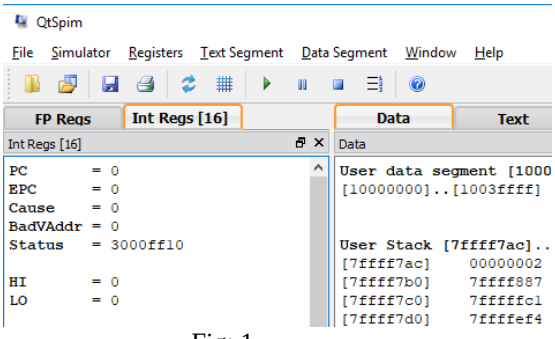

Fig: 1

There is a default exception handler in QtSpim which can be loaded as shown in Fig 3.

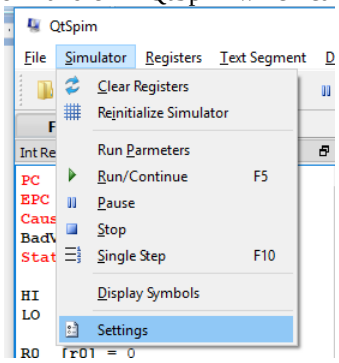

Fig 2

Simulator->Settings
This opens the window as soon in Fig 3.

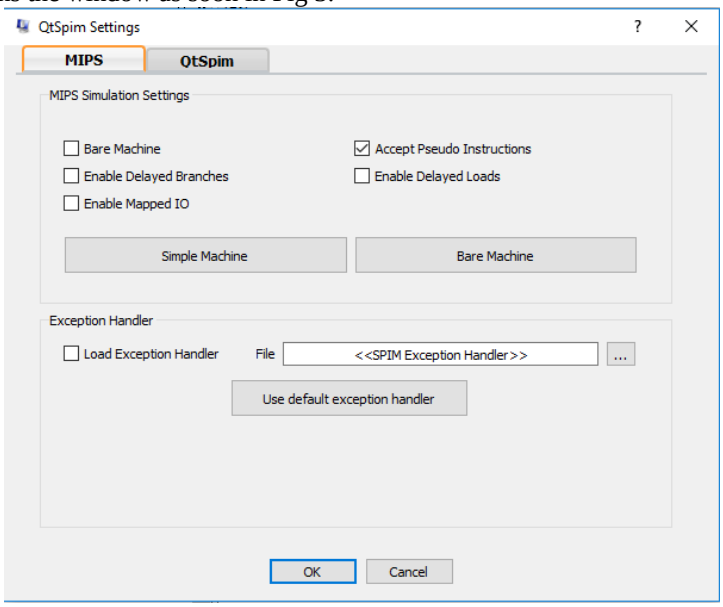

Fig 3

As per Fig 3, you can see the region of Exception Handler.
On selecting the Load Exception Handler checkbox, the default SPIM Exception handler will be loaded in the kernel text section (from address 80000180 to 8000024c).

Following are some assembly codes that will help you in understanding few exception types. You can run these codes with and without the default Exception Handler, and check the value of the exception handling registers getting updated in each case. (You may use the single stepping method to understand the update in register values after each instruction execution).

a.
```
# Arithmetic Overflow Exception
li $t0, 0x7fffffff          # $t0 = MAX_INT
li $t1, 1                   # $t1 = 1
addu $t2, $t0, $t1          # Ignores Overflow
add $t3, $t0, $t1           # Detects Overflow
```

b.
```
# Store Address Exception # Cannot store at address 4
li $t0, 4
li $a0, 5
sw $a0, ($t0)
```

c. # Misaligned Load Address Exception
```
.data arr: .word 12, 17
.text
la $t0, arr
lw $t0, 1($t0)
```

You will see the exception errors and also the exception register values being updated accordingly.

2. Extended Assembly Code:

An extended assembler creates a view that is a level higher than the assembly level. Many of the statements accepted by the extended assembler correspond to several machine instructions. To use such instructions, the respective settings have to be done in the QtSpim simulator. From Fig 3, we can see that the checkbox to Accept Pseudo Instructions have been checked. This allows the assembler to accept the pseudo instructions and translate them into basic assembly instructions. A pseudo instruction is an instruction that the extended assembler replaces with one or more basic assembly instructions.
For example:

- mov d,s
  This instruction moves the contents of source register s to destination register d
  This instruction is translated into the following assembly instruction
  Addu d, s, $zero

- li d, value
  load register $d with the positive or negative integer "value". Value may be a 16 or a 32-bit integer.
  This instruction is translated into the following assembly instruction
  Ori d, $Zero, value

There are many other pseudo instructions that you can use for the PART B exercises. Use Single Step method to understand the program flow.

PART B:

In this part, you will write MIPS assembly language programs to do the following exercises.

1. Reverse a string entered by user. (Hint: Ask user to enter a string. After reading the string in a buffer, copy it in reversed order to a second buffer. Write out the reversed string.)

2. Compute the dot product of two vectors each of length 5. Ask the user to enter the value of each element of the two vectors. Display the dot product.
   (Hint: The dot product of two vectors is sum of product of the corresponding elements. For example, (1,2,3) dot (4,5,6) is 1*4+2*5+3*6 = 32)

3. Use lb $t1, 5($zero) to cause an exception when attempting to load a byte from address 5. What is the address of the lb instruction in your program? What is the value of the cause register, the exception code, the vaddr, and the epc when the exception occurs?