# COSC 211 Lab 7 – Exceptions

## 0.Introduction

In this lab, you will have the opportunity to implement an exception handler in MIPS assembly. Additionally, there are a series of questions from the text book on pipelining. There is a total of 36 marks available for this assignment.

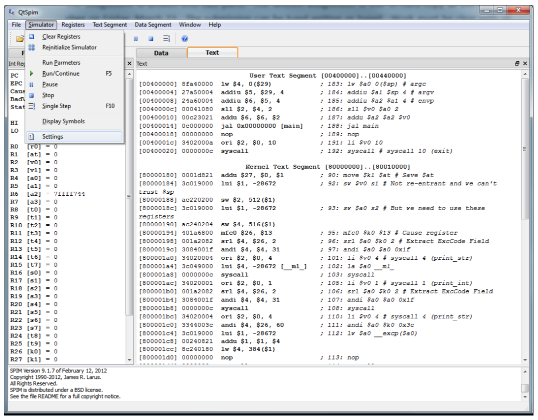
The assignment is broken into two parts. The assignment is due **at the end of the day** on Sunday, November 26th, 2023. The written submission can be hand written and scanned or typed. Work must be clear with all work shown. The coding part of the assignment requires two submissions. Both parts are to be submitted through Canvas before **the end of the day on Sunday, November 26th, 2023**. Submissions will be accepted earlier. Late submissions will not be accepted. Even if you do not complete all questions, please try them all and submit what you have for partial marks.

In this assignment, we implement an exception handler that handles

1. exceptions generated by program errors
2. exceptions generated by the instruction `break n`

The task is divided into two parts so that you can complete them incrementally. Please read the following before you start to work on the questions. Your goal is to be able to interpret the cause and status registers and have your exception handler act accordingly. Please consult your class notes. **It is imperative that you review Appendix A.7 and section 4.9 before starting this lab as it will provide details on interrupts and exception handling.**

When we start SPIM, a default exception is loaded and included in the user program. Implemented are some exceptions handling routines and a start-up routine that calls the user program's main procedure. We want to create our own exception handler and not use the default one provided. To tell SPIM that you want to use your own exception handler, you need to explicitly tell the simulator where your exception is to be found.

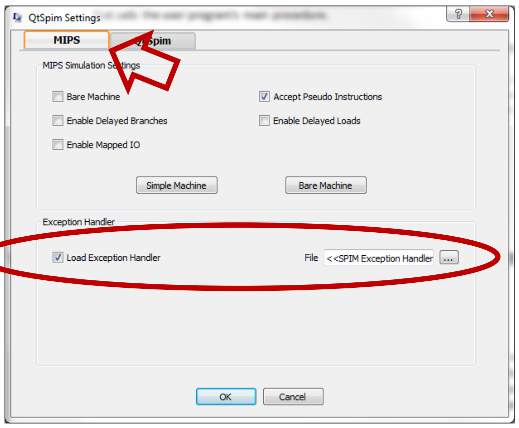


```
PC
EPC
Caus
BadV
Stat

HI
LO

R0  [r0] = 0
R1  [at] = 0
R2  [v0] = 0
R3  [v1] = 0
R4  [a0] = 0
R5  [a1] = 0
R6  [a2] = 7ffff744
R7  [a3] = 0
R8  [t0] = 0
R9  [t1] = 0
R10 [t2] = 0
R11 [t3] = 0
R12 [t4] = 0
R13 [t5] = 0
R14 [t6] = 0
R15 [t7] = 0
R16 [s0] = 0
R17 [s1] = 0
R18 [s2] = 0
R19 [s3] = 0
R20 [s4] = 0
R21 [s5] = 0
R22 [s6] = 0
R23 [s7] = 0
R24 [t8] = 0
R25 [t9] = 0
R26 [k0] = 0
R27 [k1] = 0
```

```
                         User Text Segment [00400000]..[00440000]
[00400000] 8fa40000  lw $4, 0($29)            ; 183: lw $a0 0($sp) # argc
[00400004] 27a50004  addiu $5, $29, 4         ; 184: addiu $a1 $sp 4 # argv
[00400008] 24a60004  addiu $6, $5, 4          ; 185: addiu $a2 $a1 4 # envp
[0040000c] 00041080  sll $2, $4, 2            ; 186: sll $v0 $a0 2
[00400010] 00c23021  addu $6, $6, $2          ; 187: addu $a2 $a2 $v0
[00400014] 0c000000  jal 0x00000000 [main]    ; 188: jal main
[00400018] 00000000  nop                      ; 189: nop
[0040001c] 3402000a  ori $2, $0, 10           ; 191: li $v0 10
[00400020] 0000000c  syscall                  ; 192: syscall # syscall 10 (exit)

                        Kernel Text Segment [80000000]..[80010000]
[80000180] 0001d821  addu $27, $0, $1         ; 90: move $k1 $at # Save $at
[80000184] 3c019000  lui $1, -28672           ; 92: sw $v0 s1 # Not re-entrant and we can't
trust $sp
[80000188] ac220200  sw $2, 512($1)
[8000018c] 3c019000  lui $1, -28672           ; 93: sw $a0 s2 # But we need to use these
registers
[80000190] ac240204  sw $4, 516($1)
[80000194] 401a6800  mfc0 $26, $13            ; 95: mfc0 $k0 $13 # Cause register
[80000198] 001a2082  srl $4, $26, 2           ; 96: srl $a0 $k0 2 # Extract ExcCode Field
[8000019c] 3084001f  andi $4, $4, 31          ; 97: andi $a0 $a0 0x1f
[800001a0] 34020004  ori $2, $0, 4            ; 101: li $v0 4 # syscall 4 (print_str)
[800001a4] 3c049000  lui $4, -28672 [__m1_]   ; 102: la $a0 __m1_
[800001a8] 0000000c  syscall                  ; 103: syscall
[800001ac] 34020001  ori $2, $0, 1            ; 105: li $v0 1 # syscall 1 (print_int)
[800001b0] 001a2082  srl $4, $26, 2           ; 106: srl $a0 $k0 2 # Extract ExcCode Field
[800001b4] 3084001f  andi $4, $4, 31          ; 107: andi $a0 $a0 0x1f
[800001b8] 0000000c  syscall                  ; 108: syscall
[800001bc] 34020004  ori $2, $0, 4            ; 110: li $v0 4 # syscall 4 (print_str)
[800001c0] 3344003c  andi $4, $26, 60         ; 111: andi $a0 $k0 0x3c
[800001c4] 3c019000  lui $1, -28672           ; 112: lw $a0 __excp($a0)
[800001c8] 00240821  addu $1, $1, $4
[800001cc] 8c240180  lw $4, 384($1)
[800001d0] 00000000  nop                      ; 113: nop
```

```
SPIM Version 9.1.7 of February 12, 2012
Copyright 1990-2012, James R. Larus.
All Rights Reserved.
SPIM is distributed under a BSD license.
See the file README for a full copyright notice.
```

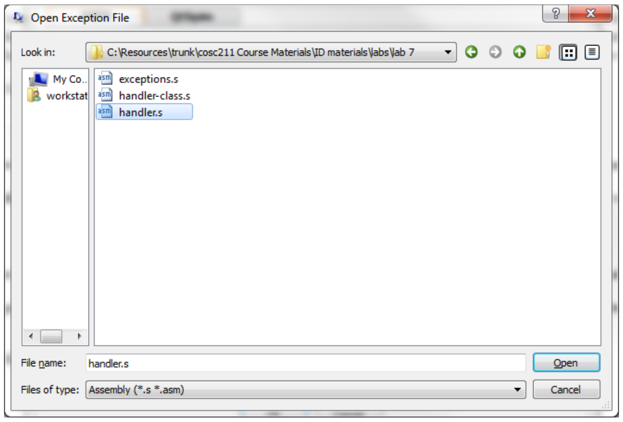
Figure 1



☐ Bare Machine
✓ Accept Pseudo Instructions
☐ Enable Delayed Branches
☐ Enable Delayed Loads
☐ Enable Mapped IO
✓ Load Exception Handler
File <<SPIM Exception Handler

Figure 2

This is set by Simulator->Settings from the main menu bar (Figure 1). To load your exception handler, select the MIPS tab from the settings window and make sure Load Exception Handler is checked. From the file selection box next to the check box, navigate to and select your exception handler (Figure 2 & 3).



Look in: C:\Resources\trunk\cosc211 Course Materials\ID materials\labs\lab 7

File name: handler.s

Files of type: Assembly (*.s *.asm)

Figure 3

When you return to the main window, any code you load into QtSpim will now use your exception handler that you have created meaning that when your code encounters an exception, it will use your provided exception handler file. When you run your program now, you will need to change the start address. The program start address can be found under Settings->Set Run Parameters. The new address is `0x00400000`.

## 1. Programming Assignment

1. (/20) Write an exception handler that replaces every instruction that causes an exception with a **NOP** instruction (this is an instruction that does 'no operation', but delays once cycle). A framework has been provided for you named **`lab7handler.s`**, setting up your initial `.kdata` and `.ktext` sections (these sections are part of the kernel segment which will handle your exception). You are required to setup the rest of the file to make it functional. Please consult your lecture notes.

   While under normal conditions it is not permissible to write to the .text section of your code during runtime; when an exception occurs, control is transferred away from your program to the kernel which executes your exception handler. As a result, you can modify memory in your code space when control is transferred to the exception handler. For this first exercise, the exception handler will replace each offending instruction with a no-operation instruction. A `nop` instruction is encoded as the word: `0x00000000`. It essentially allows one instruction cycle to pass without anything happening. To overwrite the instruction with a `nop`, you need to determine the address of the offending instruction which can be stored in a `k` register. Using a second `k` register, you can load the machine instruction value for `nop` and then store it at the address of the offending instruction. This can be done as:

   ```
   li          $k1, 0x0          #this is the nop instruction value
   sw          $k1, ($k0)        #now write is to the address in $k0
   ```

   You will need to determine the address of the offending instruction. In this case, make sure you return to the address of the offending instruction and re-execute the instruction. Your code will need to handle an **arithmetic exception** and an **unaligned address exception**. You will need to examine the **Status** and **Cause** registers to determine the source of the exception (see section 4.9, A-34 & A-35). The **`lab7handler.s`** also contains a start-up procedure which calls the main procedure of the user program. Test your exception handler with this assembly program **`testexceptions.s`**.

The output should look like:

```
---exception loop---
Exception occurred at PC=0x00400038
Arithmetic overflow
Exception occurred at PC=0x0040003c
Exception occurred at PC=0x00400040
Unaligned address in store: 0x7fffffff
---exception loop---
---exception loop---
---exception loop---
---exception loop---
```

QtSpim will also generate errors that will be displayed in popup windows which can be ignored for this exercise. Your code will display a text message indicating the type of exception encountered. Note that the output "Exception occurred at PC=0x0040003c" is from SPIM itself.

Include a program comment header that has your name, student number and program name. **Name your file `handler1.s` and submit through the submission link in Canvas.**

2. (/10) In the program **`testexceptions.s`**, there is an instruction **`break 20`**. This is a special instruction that will generate an exception and is handled by QtSpim. It generates a breakpoint exception. Modify your exception handler so that it will handle the exception generated by the instruction **`break n`** in the following way: for every exception generated by **`break n`**, output the value `n` but leave the instruction unchanged.

   The number **`n`** in the instruction **`break n`** is encoded as part of the instruction. You have to get the instruction from the memory and extract the number (There is an issue with the break n encoding format with SPIM, it won't be difficult for you to figure out what it is). The output, when testing with **`testexceptions.s`**, should now look like

```
---exception loop---
Exception occurred at PC=0x00400054
Arithmetic overflow
Exception occurred at PC=0x00400058
20
Exception occurred at PC=0x0040005c
Unaligned address in store: 0x7fffffff
---exception loop---
Exception occurred at PC=0x00400058
20
---exception loop---
Exception occurred at PC=0x00400058
20
---exception loop---
Exception occurred at PC=0x00400058
20
---exception loop---
Exception occurred at PC=0x00400058
20
```

   Unlike the exercise in part 1, when you encounter and handle a **`break`** exception, **you will need to continue execution at the next instruction so you will have to increment the EPC accordingly (so that you can return**

**to the instruction following the instruction which generated the exception)**. Include a program comment header that has your name, student number and program name. **Name your file `handler2.s` and submit through the submission link in Canvas.**

## 2. Written Assignment

1. (/6) Exercise 4.9 (from 5th edition)
    a. 4.9.1 (4 marks)
    b. 4.9.2 (2 marks)

**Written Submission**

Submit your written assignment via Canvas on the above indicated due date. Please take care and effort to make your assignment answers as clear as possible.