Lab# 7 EXCEPTION HANDLER

Instructor: I Putu Danu Raharja.

Objectives:
To understand the way MIPS processor handles exceptions.

Method:

Using MARS to simulate an arithmetic overflow.

Preparation:

Download the file named *exception.s.*

7.1 INTRODUCTION

In addition to its normal computational functions, any CPU needs units to handle interrupts, configuration options, and some way of observing or controlling on-chip functions like caches and timers. But it is difficult to do this in the neat implementation-independent way that the ISA does for the computational instruction set.

The method implemented by the MIPS designers to the above issues is to include some additional hardware referred to as **Coprocessor 0** that contains a number of specialized additional registers that can be accessed at the assembly language level for exception handling.

Some of those registers are:

- EPC, register 14
- Cause, register 13
- BadVaddress, register 8
- Status, register 12

The following are examples of the only two instructions available to access the coprocessor registers:

mfc0	**$k0, $13**	# CPU register $k0 is loaded with the contents of the cause register.
mtc0	**$zero, $12**	# CPU register 0 is stored in the status register.

Notice that we reference the coprocessor registers only by their numbers. They are not referenced by a name, so in-line comments are essential.

Coprocessor 0 is designed to send a signal to the CPU control unit when an exception occurs. Wires from external devices provide input interrupt signals to Coprocessor 0. Bits within the status register can be manipulated at the assembly language level to selectively enable or disable certain interrupts.

7.2 EXERCISE:

1. Write a simple program in MIPS to perform some addition operations that contains an instruction that will cause an arithmetic overflow happens. Before you assemble the program, Go to Setting, select Exception Handler, and set the file name: ***exception.s***. Assemble and execute the program. Observe the output.

2. Write a function, *Adduovf(x, y, s)*, that will find the 32-bit sum of two unsigned arguments x and y. An exception should be generated if the unsigned representation of the sum results in overflow. Perform all communication on the stack.