CSC/CPE 315
Retz
Lab # 4 (Retz.4b)

Objectives: to build a MIPS simulator and performance monitor

Description

For this lab, you will implement a program that simulates the operation of a MIPS 32-bit processor using the instruction opcodes generated by your last project. This simulator will be able to evaluate the time, in clock cycles, used by each instruction – assuming a multi-cycle implementation of the processor, and produce running statistics for the resulting program (something a hardware implementation of any processor doesn't do).

We will use a standard "benchmark" program, called countbits, as a means of testing the simulator and for evaluating the performance of your non-pipelined, multi-cycle processor. This will be written in either C or Java, and requires the following:

1. You will add functionality to your assembler to support the following instructions: **ori**, **addi**, **addui**, **lui**, **lw**, **sw**. Additionally, you will implement the ability to define memory-resident constant values using .**word** and .**byte** with either decimal or hexadecimal (0x…) operands.

2. You will load the machine-coded binary (unsigned 32-bit integer) values into a simulated memory array. It is your choice to establish a standard program load address anywhere from 0 and above in memory, and will have to implement an offset from a simulated "base address" and the base of your simulated memory array. For example, SPIM uses 0x400000 as the base of user memory, so you might use that number (but subtract that to get your array offset). Note that MIPS addresses are always *byte* addresses, but on word boundaries (multiples of 4).

3. In "simulation mode" you will support a machine that is just like a hardware MIPS processor, including a memory array and a set of registers (0..31), and the Program Counter (PC) value.

4. In "simulation mode", once assembled and machine code loaded into memory, you will perform a simulated operation of a MIPS machine in either "single step" or "run" mode. A command line will be issued to allow you to either single step, run, or exit.

In single step mode you will execute a single instruction, showing the resulting values of the registers. In run mode you will allow the program to run and display the registers upon completion. In both modes, you will keep a count of the following:

- Number of instructions simulated (executed).
- Number of memory references (reads and writes) made (ld or st only).
- Number of clock cycles expended. (calculated per instruction).

5. The result of these statistics, as well as the contents of the registers, will be displayed either after a single-step or after a run.

6. Termination of the program will be flagged with a special opcode (syscall) with the contents of $v0 equal to 10. (This is the standard MIPS exit instruction.) You may also optionally include the additional syscalls in your simulator using the associated C or Java console commands such as printf.

Note: you do not have to worry about separation of code (.text) and data (.data) segments: you can just have one segment for code and data for your simulator, i.e., they can share the address space.