CSC/CPE 315
Retz
Lab # 5 (Retz.5b)

Objectives: to build a MIPS simulator and performance monitor

Description

In the last lab, you implemented a program that simulates the operation of a MIPS 32-bit processor using the instruction opcodes generated by your last project. This used 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 lab takes a different approach to the simulation of the processor – this time simulating the operation of a five-stage pipeline. In this case your simulator will implement these stages which take action on each clock cycle:

- IF – Instruction Fetch
- ID – Instruction Decode
- EX – Execute
- MEM – Memory Access
- WB – Writeback

You will use the same functionality of the last two labs in terms of MIPS instructions supported. Each of the stages will have its own "inbasket" and "outbasket", corresponding to those shown in the lecture notes (MIPS Pipeline diagram). You will define the fields that are passed between stages of the pipeline.

Note that in a "real" processor, the interface data (latching registers) would all be loaded simultaneously at clock time, but in your case you have to simulate the stages sequentially. Consequently, it is advisable to implement these in reverse order at time of the clock: WB, MEM, EX, ID, IF.

Each interface between stages (inbasket/outbasket) will have a status flag indicating "busy" or "empty". In a given cycle, a stage has nothing to do if its status is "empty". However, if it does not finish an operation by the end of a clock cycle (because it is blocked for some reason) it will remain "busy".

As in the last lab, your simulator will keep statistics counters on the number of operations performed for each state (fetched, decoded, executed, memory reference, and write-back) and will also keep a running count of the number of cycles.

Your simulator will be able to display the number of instructions completed and the number of total cycles consumed.

As in the previous lab, we will use the standard benchmark program countbits_benchmark.asm.

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.

As in the previous lab, 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.