ECE 697CE: Foundations of Computer Engineering

Project 3

MIPS Simulator

This assignment will give you experience in programming in C++ and the operation of a MIPS pipelined processor. Further, you will gain insight into how multiple events that occur in parallel can be simulated using a sequential machine.

1. Problem Statement

This assignment requires a simple 5 stage pipelined machine to be simulated. The simulator should be capable of implementing the MIPS architecture on a cycle by cycle basis. The simulator must be cycle accurate with respect to contents of the registers, but need not be faithful to other hardware details such as control signals. The output of the simulator, in addition to the register contents and latch values should include the utilization factor of each functional unit and the total time in cycles to execute a set of instructions. Implement the simulator according to the specifications described below in C++. Submit the code, simulation results and a project description write-up.

1.1 Instructions to be implemented

The simulator should implement the following instructions: *add, sub, addi, mul, lw, sw, beq, lui, and, andi, or, ori, sll, srl, slti, and sltiu*. Note that these instructions operate integer instructions only. The MIPS instruction format can be used for all instructions except mul. Assume the syntax for mul is *mul $a,$b,$c*, meaning that we multiply the contents of $b and $c, the least significant 32 bits of results are placed in register $a and the most significant 32-bits of the result will be stored in register $(a+1). For example, **mul $t0, $t8, $t9** will store lower 32-bits of the product of $t8 * $t9 in register $t0 and the upper 32-bits of the product in register $t1 (Hint: See MIPS green sheet instructions summary for registers numbering). This is different from the *mult* instruction in MIPS. Assume the opcode and function code for *mul* to be same as that of *mult.*

1.2 Inputs to the simulator

1) MIPS machine code as a text file: Convert the assembly level instructions to machine level by using https://www.eg.bucknell.edu/~csci320/mips_web/ or http://www.kurtm.net/mipsasm/

2) A query to the user to select between instruction or cycle mode

- Instruction mode: To observe execution of the program instruction by instruction
- Cycle mode: To observe execution of the program cycle by cycle

3) A query to the user to select the number of instructions or cycles (depending on the choice made in the previous query) to be executed.

4) After executing the number of instructions or cycles entered initially by the user, a third query to the user to choose to continue execution or not.

- If yes, Repeat from step 3
- If no, exit the execution and display the results

1.3 Memory, Registers and PC

The memory is one word wide and 2K bytes in size. There are physically separate instruction and data memories for the instruction and data. Data memory is initialized to 0 at the beginning of each simulation run. There is no cache in this machine.

There are 32 registers; register 0 is hardwired to 0. In addition, there is a Program Counter (PC). PC should start execution by fetching the instruction stored in the location to which it is initialized.

1.4 CPU

The pipelined MIPS processor has 5 stages: IF, ID, EX, MEM, WB. There are pipeline registers between the stages: IF/ID, ID/EX, EX/MEM, MEM/WB. Assume the pipeline registers to contain following latches:

- IF/ID : IR, NPC
- ID/EX : IR, NPC, A, B , Imm
- EX/MEM : IR, B, ALUOutput , cond
- MEM/WB : IR, ALUOutput, LMD

1.5 Output of the simulator

In addition to displaying the register contents and latch values after the execution of each cycle/instruction, it should output the following statistics

- Utilization of each stage. Utilization is the fraction of cycles for which the stage is doing useful work. Just waiting for a structural, control, or data hazard to clear in front of it does not constitute useful work.
- Total time (in CPU cycles) taken to execute the MIPS program on the simulated machine. (This is NOT the time taken to execute the simulation; it is the time taken by the machine being simulated.)

1.6 Dealing with branches

The processor does not implement branch prediction. When the ID stage detects a branch, it asks the IF stage to stop fetching and flushes the IF_ID latch (inserts NOP). When the EX stage resolves the branch, IF is allowed to resume instruction fetch depending on the branch outcome.

1.7 Other remarks

- No interrupts.
- Does not support out of order execution
- Does not support data forwarding
- Assume register writes are completed in the first half of clock cycle and register reads are carried out in the second half.
- All data, structural and control hazards must be taken into account.
- Branches are resolved in the EX stage.

2. Way to approach (Suggestion)

Start by figuring out how to implement the Pipeline registers (use of *class is* recommended), the 5 stages of the pipeline, instruction and data memory, 32 registers and PC. Think of how the execution of 5 stages of the pipeline (which is a parallel operation) could be done in C++ (where instructions are executed sequentially). Figure out a way to account for data, structural and control hazards. Finally think of how the utilization and total time to execute the program can be measured.