5008: Computer Architecture
HW#2

1. We will now support for register-memory ALU operations to the classic five-stage RISC pipeline. To offset this increase in complexity, all memory addressing will be restricted to register indirect (i.e., all addresses are simply a value held in a register: no offset or displacement may be added to the register value.) For example, the register-memory instruction `ADD R4, R5, (R1)` means add the contents of register R5 to the contents of the memory location with address equal to the value in register R1 and put the sum in register R4. Register-register ALU operations are unchanged. Answer the following for the integer RISC pipeline.

- **a.** List a rearranged order of the five traditional stages of the RISC pipeline that will support register-memory operations implemented exclusively by register indirect addressing.
- **b.** Describe what new forwarding paths are needed for the rearranged pipeline by stating the source, destination, and information transferred on each needed new path.
- **c.** For the reordered stages of the RISC pipeline, what new data hazards are created by this addressing mode? Give an instruction sequence illustrating each new hazard.
- **d.** List two ways that the RISC pipeline with register-memory ALU operations can have a different (one for more, the other for less) instruction count for a given program than the original RISC pipeline (there are several ways). Give a pair of specific instruction sequences, one for the original pipeline and one for the rearranged pipeline, to illustrate each way.
- **e.** Assume all instructions take 1 clock cycle per stage. List two ways that the register-memory RISC can have a different (one for more and the other for less) CPI for a given program as compared to the original RISC pipeline.

2. Suppose the branch frequencies (as percentages of all instructions) are as follows:

Conditional branches	15% (60% are taken)
Jumps and calls	1%

We are examining a four-deep pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be done independent of whether the branch goes and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?

3. A reduced hardware implementation of the classic five-stage RISC pipeline might use the EX stage hardware to perform a branch instruction comparison and then not actually deliver the branch target PC to the IF stage until the clock cycle in which the branch instruction reaches the MEM stage. Control hazard stalls can be reduced by resolving branch instructions in ID, but improving performance in one respect may reduce performance in other circumstances. How does determining branch outcome in the ID stage have the potential to increase data hazard stall cycles?

4. Scheduling branch delay slots (see the three ways in Figure A.14) can improve performance. Assume a single branch delay slot and an instruction execution pipeline that determines branch outcome in the second stage.

- **a.** For a delayed branch instruction, what is the penalty for each branch delay slot scheduling scheme if the branch is taken and if it is not taken, and what condition, if any, must be satisfied to ensure correct execution?
- **b.** A cancel-if-not-taken branch instruction (also called branch likely and implemented in MIPS) does not execute the instruction in the delay slot if the branch is not taken. Thus, a compiler need not be as conservative when filling the delay slot. For each branch delay slot scheduling scheme, what is the penalty if the branch is taken and if it is not taken, and what condition, if any, must be satisfied to ensure correct execution?
- **c.** Assume that an instruction set has a delayed branch and a cancel-if-not-taken branch. When should a compiler use each branch instruction and from where should be the slot be filled?