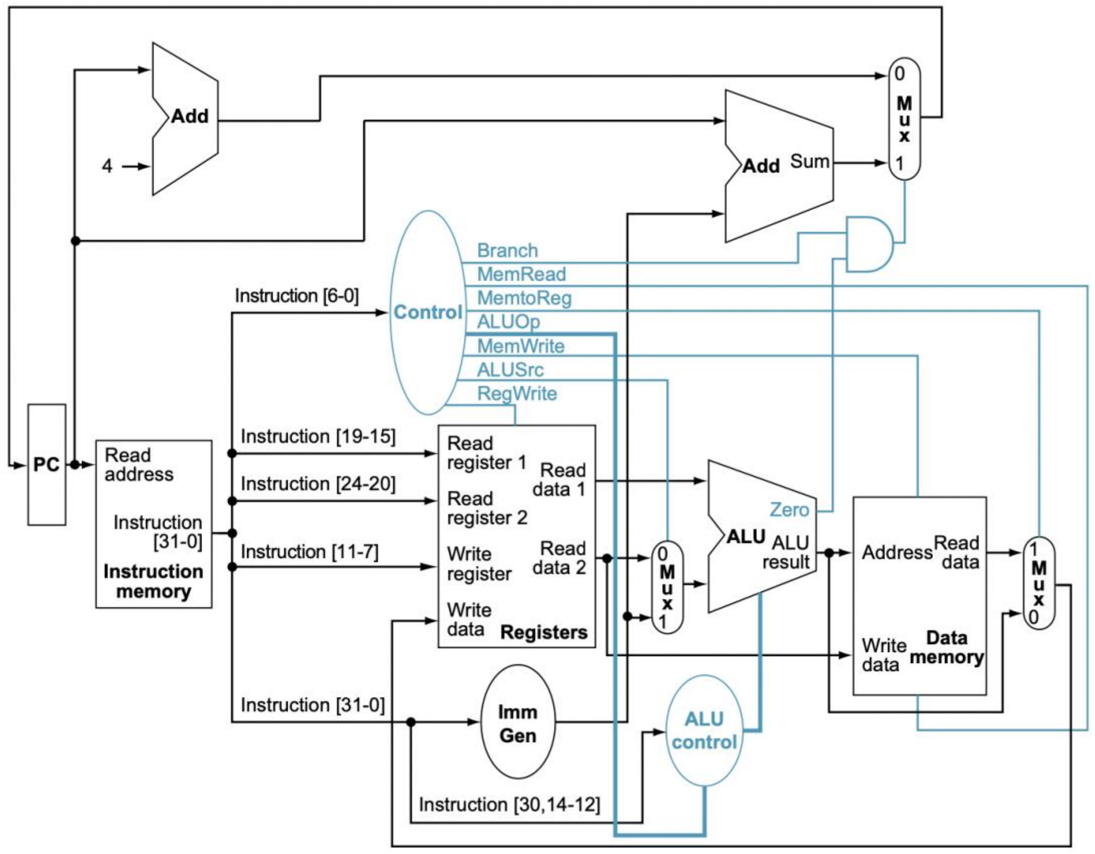
ECE-GY-6913 Computer Systems Architecture Project A, INET Section

Performance Modeling – RISC-V Processor Simulator

Xiaoyu Nie
Xn2014

Task1:
Single Stage Processor



In a single stage design, instructions do not run in parallel. Instructions are fetched from instruction memory and then decoded. If the instruction is a branch operation (JAL, or conditional branch), the control set the branch signal and compares the two read data if the current instruction is a conditional branch. The immediate number is 11-bit because the architecture is word aligned. This means the last digit of the offset should be a zero. The 11-bit immediate number is left shifted by 1 bit to add the default zero-bit. Then the immediate number value is added to the PC value. The new PC is set correctly after the branch operation.

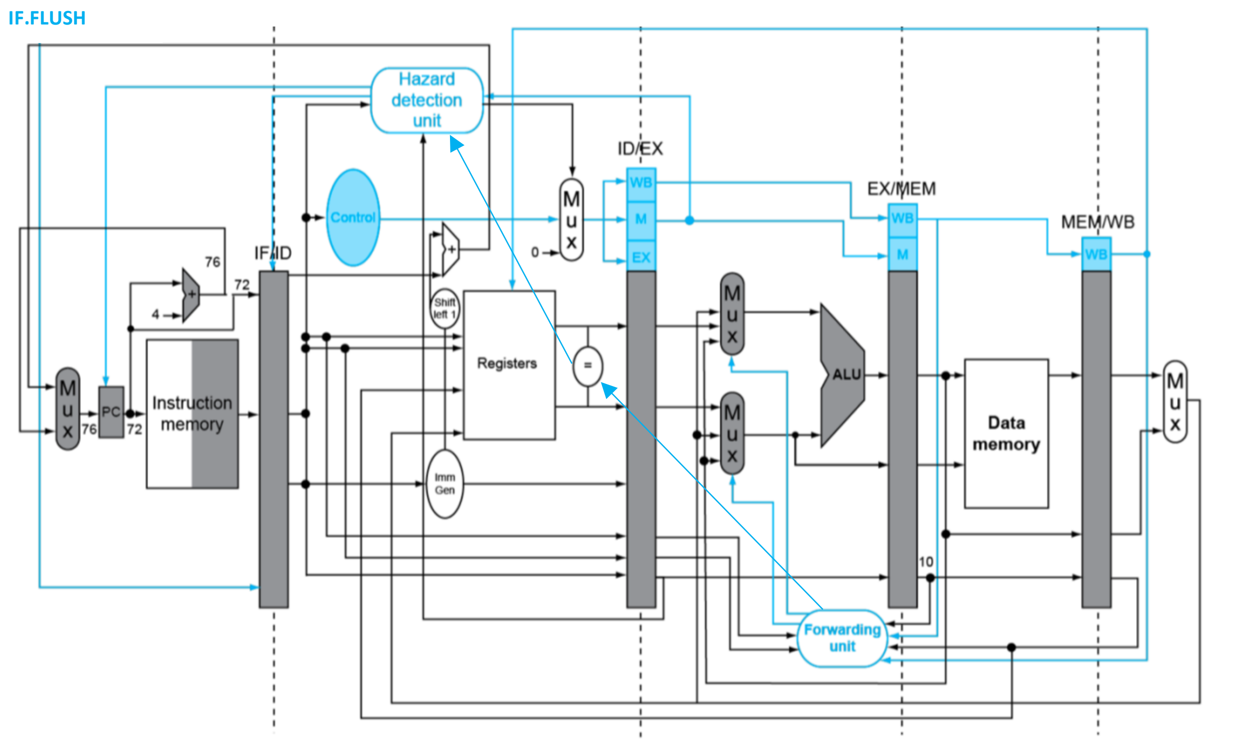
Otherwise, if the instruction is R-type, two correct data are read, calculated by the ALU and written back to the register file.

If the instruction is LW or SW, the immediate number is sign extended to 32 bit and computed with one of the two read data. (Imm Gen contains hardware that allow shift and sign extend to 32 bit function). The ALU computes the read or write memory address. Memory gets read and written back to register file for LW instructions. In a SW instruction, memory is written with the other data that was prepared(Not computed by ALU).

If the instruction is I-type, the immediate number is sign extended to 32-bit and computed with the other read data similarly from the method described above. The result is written back to the register file.

Task2:

Five-Stage Processor



In a five-stage design, the processor runs instruction overlappingly.
There are IF(Instruction Fetch), ID(Instruction Decode), EX(Execution), MEM(Memory), WB(Write-back).

Interestingly enough, there is an adder(IF stage) that by default increment the PC by four, which is the default prediction of a branch operation. There is also an extra adder(ALU) at ID stage that adds the Immediate Value to the PC to set the new PC value to handle branch instructions. The advantage of moving resolving branching condition to ID stage is that only one stall is sufficient because at the next cycle the correct PC will be set. It should not be forgotten that a comparator (=) is installed between two read data values before entering the EX stage to help determine conditional branching conditions.

R-type, I-type, LW and SW instructions are not changed by much. Control signals are introduced to determine the instruction type as well as the correct ALU operation. For example, the LW instruction set rd_mem to true and wrt_enable(write back to register) to true.

All data hazards except for load-use data hazard can be resolved by forwarding with no stall. Load-use hazards have to be resolved by introducing one stall. Non load-use hazards can be detected by comparing the [EX/MEM and MEM/WB stage rd] with [ID/EX stage rs or rt]. Load-use hazard can be detected when rd_mem is true and EX/MEM rd is the same as ID/EX stage rs or rt.

Note that hazard detection has to work for conditional branch instructions as well. Because the branch instructions have to be determined at ID stage and can't wait till EX stage. Forwarding for branch instructions have to be done before comparing the two read data values.

3. The instruction count, total execution cycles and CPI are written to a text file in the same directory. "PerformanceMetrics_Result.txt"

Although for some test cases, my answers are <u>different</u>.

Instruction Count: If there is a loop, all instructions that get executed should be counted. I don't think the standard answer counts that.

Total execution cycles: In a five-stage processor, since Halt is determined at ID stage, that means EX has also done its job for the last instruction. IF, ID, EX stages can be installed nop at next cycle. However, the MEM and WB signals should be checked before nop can be installed. MEM stage might still need to read or write to memory. WB might need to write back. I only set all nextState.XX["nop"] to true if all rd_mem, wrt_mem and wrt_enable signals are false. This is the reason for the discrepancies for cycle count for some test cases.

4. They are both good. It depends what the user/vendor values the most(production cost, performance etc.)

The single stage processor takes less cycles. But the length of these cycles are much longer than multicycle processors. Therefore, the single stage processor take longer time. But the advantage of a single stage processor is that the hardware is not complicated, and no data hazard will happen. (No forwarding hardware needed) Branch conditions are also correctly resolved each cycle.

The five-stage processor takes seemingly more cycles. But each cycle length is much shorter. Instructions run in parallel, which delivers an improved performance in general cases. However, in this design, we did not get rid of excess ALUs because we need the branch conditions to be resolved in ID stage. We cannot save cost in this way. On top of that, we also introduce hazard detection and data forwarding, which is also much more complicated for the hardware as well as the programming part.

If performance is the only thing that we are after, we might choose the pipelined processor. However, if performance is not prioritized, cost and complexity are considered. Single stage processor might not be such a bad idea.

5. I noticed that each non-default branch taken, a stall has to be introduced. The PC will have to wait to next cycle to have a correct value. If in some way, ID can happen slightly before IF stage, then the PC can be immediately updated and the correct instruction can be fetched within the same cycle (just a bit later). This way no stall is needed, and the performance will have a significant boost when the program has a lot of branch instructions.