

Problem #1 : A program's execution can be broken up into 2 parts, floating point instructions and integer instructions. Assume we execute a program on two processors A and B. The only difference between processors A and B is that integer instructions execute 5 times faster on processor B (compared to processor A). If the same program is run on processor A and B, and the program run on processor B is 1.5 times faster than on processor A, what percent of the program's execution on processor A is due to the integer instructions? (Write all calculations down to receive full credit)

Problem #2 : Assume the pipeline for this problem is a single issue in-order 8 stage pipeline (Fetch, Decode1, Decode2, Decode3, Execute1, Execute2, Memory, Writeback). Branch conditions are done executing at the end of Execute2. So, if the branch was mispredicted, the program counter will be corrected at the end of Execute2 for the next cycle fetch. When executing a program X, assume 20% of the instructions are branches, and 60% of the branches are taken.

Assume a processor that always predicts branches as not-taken during fetch. What is the branch misprediction penalty? What is CPI with branch stalls? (Show all equations and calculations to receive credit, and clearly label where the stalls are coming from in the equations.)

Problem #3 : Assume we have the following six stage pipeline (IF, ID, EXE, MEM1, MEM2, WB). The pipeline is in-order and can only execute one instruction at a time. In this design, memory operations take two pipeline stages, so load result values are not available until after the MEM2 stage is completed. Branches are resolved in the EXE stage, and the architecture uses branch delay slots to try to eliminate **all** the branch penalty. For this problem we will use the following “assembly language sequence”

loop:

```

SUB      r4, r3, r4
LW       r4, 0(r4)
SW       r1, 0(r4)
SUBI     r2, r2, #4
bnez     r2, loop
LW       r3, 0(r2)
ADD      r1, r1, r4
SUB      r2, r1, r2
...

```

Part (a), Draw arrows and *label* all RAW, WAW, and WAR dependencies for the loop in the above code example.

Part (b), Draw the pipeline cycle timing diagram for the code sequence above for the described architecture. Start with the first instruction of the loop and draw the timing diagram through one full loop iteration including the first SUB of the next loop iteration. Therefore, the branch is a taken branch. Draw ALL stalls and all forwarding needed to help eliminate stalls. How many cycles does it take to start executing the 2nd SUB instruction in the loop (the cycle the SUB in the 2nd iteration completes the EXE stage)?

Problem #4 : We are considering adding a new addressing mode to MIPS. In this new addressing mode, loads and stores have the following format:

LD R1, 0(R2,R3)

This is essentially a variant of the displacement addressing mode but here the address is given by $R2 + R3 + \text{displacement}$, with the displacement being 11 bits. Using this instruction, we can replace the following sequence of instructions:

ADD R1, R1, R2

LD Rd, 0(R1)

with an instruction of the new format. Remember that this also works for store instruction. Assume that the instruction mix is 15% conditional branches, 20% loads, 5% stores and the rest being ALU instructions.

Part (a), Assume that the new addressing mode can be used for 10% of loads and 15% of stores. What is the ratio of instruction count on the MIPS with new addressing mode compared to the original MIPS?

Part (b), If the addressing mode lengthens the clock cycle by 5%, which machine will be faster and by how much?