Name: ____________________

CS451 Single Cycle Performance Solutions

1. Consider a CPU. Suppose we were given the option of decreasing the cycle time by 20% in exchange for a 20% increase in the number of instructions. Is this a good tradeoff?

   Original CPU: np
   new CPU: $(1.2n) * (.8p) = .96np$.

   Yes.

2. (From Patterson and Hennessey ARM edition) Consider the addition of a multiplier to the CPU discussed in class. This addition will add 300ps to the latency of the ALU, but will reduce the number of instructions by 5% (because there will no longer be a need to emulate the multiply instruction).

   (a) What is the clock cycle time with and without this improvement?

   Without Improvement: 925; With improvement: 1225

   (b) What is the speedup achieved by adding this improvement?

   The running time of a program on the original CPU is $925 * n$. The running time on the improved CPU is $1225 * (.95) * n = 1163.75$. Thus, the "speedup" is $.794$. (In other words, this "improved" CPU is actually slower than the original.)

   (c) What is the slowest the new ALU can be and still result in improved performance?

   Because adding a multiply instruction will remove 5% of the instructions, the cycle time can grow to as much as $925/(.95) = 973.7$. Thus, the time for the ALU can increase by up to 48 (from 200 to 248).

3. Consider the single-cycle CPU with a 925ps clock period. Suppose that doubling the size of the register file from 32 to 64 registers would reduce the number of `lw` and `sw` instructions, but would increase the delay of the register file from 150 to 175. What percent of these instructions must be removed? Assume that loads and stores account for 35% of the original workload.

Time for original CPU: $925n$
Time for new CPU: $(925 - 150 + 175)n'$

Divide the instructions into two groups: "memory instructions" (`lw` and `sw`) and "non-memory instructions" (everything else). Every "non-memory" instruction will appear in programs compiled for the new CPU. Only some of the memory instructions will appear; the additional registers will make some memory instructions unnecessary. Suppose the new CPU will retain x of the memory instructions. Then

$$n' = .65n + .35xn$$

For the new CPU to have better performance

$$950(.65n + .35xn) \leq 925n$$

This happens when $x \leq .9248$. This means that we can keep at most 92.48% of the loads and stores and must get rid of at least 7.5%.

Here is another apporach to the problem:

For the two CPUs to have equal performance, n' must equal $.974n$. Thus, 2.6% of all instructions must be removed. 2.6% of all instructions represents $.02631/.35 = .075$, or 7.5% of all loads and stores.

4. (From Patterson and Hennessy ARM edition) When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. Assume that 25% of all instructions are loads, and 10% are stores. Also assume the following costs for the MIPS CPU components:

I-Mem	Reg. File	Mux	ALU	Adder	D-Mem	Single Reg.	Sign ext.	Single gate	Control
1000	200	10	100	30	2000	5	100	1	500

Suppose doubling the number of general purpose registers from 32 to 64 would reduce the number of load and store instruction by 12%, but increase the latency of the register file from 150ps to 160ps and double the cost from 200 to 400.

(a) What is the speedup achieved by adding this improvement?

The additional registers will allow us to remove 12% of the loads and stores, or (.12)*(.25+.1) = 4.2% of all instructions. Thus, the time to run n instructions will decrease from $925 * n$ to $935 * .958 * n = 895.73 * n$. That corresponds to a speedup of $925/895.73 = 1.03$.

(b) Compare the change in performance to the change in cost.

The cost of the original CPU is 4007; the cost of the improved CPU is 4206.
PC: 5
I-Mem: 1000
Register file: 200
ALU: 100
D-Mem: 2000
Sign Extend: 100
Control: 500
2 adders: $30 * 2 = 60$
4 muxes: $4 * 10 = 40$
1 single gate: 1

Thus, for a 3% increase in performance, the cost of the CPU increases by about 5%.

(c) Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesnt make sense to add more registers.

From a strictly mathematical standpoint it does not make sense to add more registers because the new CPU costs more per unit of performance. However, that simple calculation does not account for the utility of the performance. For example, in a real-time system, a 3% performance may make the difference between meeting or missing deadlines. In which case, the improvement would be well worth the 5% additional cost.

5. Problem 2.43 from the Patterson and Hennessy textbook (3rd edition): As you know, MIPS is a "load-store" architecture. In contrast, the Intel architecture allows ALU operations to access memory. In this exercise, you will examine quantitatively the pros and cons of adding an "IA-32 style" addressing mode to MIPS.

Consider adding a new instruction:

```
addm $t2, 100($t3)   #  $t2 = $t2 + Memory[$t3 + 100]
```

Assume that the new instruction will cause the cycle time to increase by 10%. Use the instruction frequencies given below:

- Arithmetic / Logic: 42%.
- Loads: 24%.
- Stores: 12%
- Branch and Jump: 22%.

(these values are for for the SPEC2000int and are given in Figure 2.48). Assume that the new instruction affects only the clock speed, not the CPI. What percentage of loads must be eliminated for a single-cycle machine with the new instruction to have at least the same performance as the single-cycle machine?

Time for unmodified machine to execute n instructions: np, where p is the cycle time (i.e., the length of one clock period).

Time for modified machine to execute n instructions: $1.1np$.

In order for both machines to run the same program in the same amount of time, the modified machine must require fewer instructions. If the machines take the same amount of time, we have $np = 1.1(xn)p$, where x represents the fraction of instructions needed for the modified machine as compared to the unmodified machine.

Solving for x tells us that to take the same amount of time, a program with n instructions on the modified machine must require only $\frac{1}{1.1} = .909909n$ of instructions that the unomdified machine requires. In other words, we must eliminate 9% of the instructions (i.e., $.090909n$).

All of the eliminated instructions will be loads. Thus, $.090909n$ out of $.24n$ load instructions, or 37.88%, must be eliminated.

Here is another approach:

$$(1.1p)(n - .24nx) < np$$
$$1.1pn(1 - .24x) < np$$
$$1.1(1 - .24x) < 1$$
$$1.1 - .264x < 1$$
$$.1 < .264x$$
$$.379 < x$$

6. In class, we saw that re-arranging the single-cycle CPU so that an instruction could either use the ALU or access memory reduced the cycle time from 925ps to 725ps; however, some loads and stores now require an additional add instruction to compute the effective address (thereby increasing the total number of instructions).

   The number of additional add instructions can be reduced by placing a small 4-bit adder in front of data memory. This adder requires only 100ps (in contrast to 200ps for the general ALU); but, it can calculate the effective address for only some of the loads and stores. If loads and stores make up 45% of a typical workload, and, without the extra adder, 95% of them would require an extra add, what percentage of the loads and stores must have an offset that requires at most 4 bits (or no offset at all) in order for the addition of this "mini-adder" to generate a performance better than either the original CPU, or the CPU with the ALU and data memory in parallel.

As discussed earlier in the semester, placing the ALU and data memory in parallel reduces the clock period to 725ps. Using the numbers in the problem, this CPU would take $(1+.95*.45)n(725) = 1035n$ picoseconds to run a program that requires n instructions on the original CPU.

Placing a 100ps ALU in front of data memory increases the cycle time from 725ps to 825ps. The question at this point is: How many "extra" adds can we tolerate and have performance better than $925n$? Thus, we want to solve for x:

$$(1 + .45 * x)n(825) \leq 925n$$

$x \leq .27$. In other words, at most 27% of the loads and stores can require an extra add instruction. This means that 73% of the loads and stores must either (1) not require an add, or (2) have an offset requiring at most 4 bits.

7. Your company builds a single-cycle CPU that is very similar to that presented in the textbook, except its `lw` and `sw` instructions do not accept an offset parameter. This difference allows the ALU and the data memory to be organized in parallel. The lead architect is considering switching to a "Patterson and Hennessy" style CPU where the ALU and data memory are organized in series because (1) he estimates that the switch will reduce the number of instructions by 10% and (2) he has heard there has been a recent breakthrough that will significantly increase the speed of the ALU.

   Given that

   - the current CPU's clock period is 1000ps (i.e., the one with the ALU and data memory in parallel),
   - the ALU requires 200ps, and
   - the data memory requires 250ps.

   (a) Without any changes to the ALU, will the new design be faster or slower than the current design?

   Performance of current CPU: $1000n$
   Performance of new CPU: $(1000 + 200)(.9n) = 1080n$ (Need to add ALU to critical path.)
   New CPU is slower.

   (b) How fast must the new ALU run in order for the new design to be faster than the current design?

   We want the new CPU to be faster than the old. Our variable is the speed of the adder:

$$1000n < (1000 + x)(.9n)$$

   Solving for x gives that $x < 111.11$. In other words, the ALU must run in time 111.11 or less.

8. Consider the following proposed instruction: `incmp` (*inc*rement and *comp*are). This instruction increments the value of a register before comparing it. In other words, it performs an operation similar to this: `if (++R1 != R2)`. It would be most useful in combining the condition check and increment in a typical `for` loop. (i.e., combining `addi, R1, 1` and `bne R1, R2, TOP_OF_LOOP` into a single `incmp` instruction). Adding this instruction will remove some `addi` instructions at the cost of adding 30ps to the ALU time.

   Given that

   - The current CPU's clock period is 925ps.
   - The current CPU's ALU requires 200ps.
   - Adding `incmp` to the ALU will increase its latency by 30ps.
   - Branch statements comprise 11% of a typical workload.

   (a) What percent of branch instructions must be consolidated with an `addi` into a single `incmp` instruction in order for the new CPU to have better performance than the current CPU?

   $925n = 955n'$
   $n' = .9686n$.
   Thus, we must remove $1 - .9686 = .0314$ of the instructions. $.0314$ of all instructions represents $\frac{.03141}{.11} = 28.56\%$ of all branches.
   Another approach is to solve for x: $955n(1 - .11x) < 925n$
   Note: If you got 29.48%, then you did the first step backward.

   (b) Suppose the CPU had a clock period of 725 instead of 925. Would you need to consolidate more or fewer branch instructions to "break even"? Explain how you can answer this question without doing any additional calculations.

   You would need to consolidate more instructions. When the clock period is only 725, the 30 ps increase represents a larger percentage increase. Thus, a larger percentage of instructions must be removed to compensate.