CMPE 110 Homework #2

John Allard
February 2nd, 2015
jhallard@ucsc.edu

1. **Question #1 - Pipeline Stages**

   (a) **Question 1.A Baseline**
   Fill out the table below for the given pipelined processor. Cycle time is the clock period at which this machine can run. Max clock frequency is the macimum frequency that can be run given the cycle time (lower frequencies are possible). Latency of instruction is the time it takes to get the first output after it starts. Throughput is the rate at which output is produced.

   Explanation - The cycle time for the baseline architecture is 400ps, which is the time associated with the memory access stage. This is the longest stage length, if we shortened the cycle length below 400ps we wouldn't be able to reliably access memory under our current architecture. Max frequency is the reciprocal of the cycle time, and instruction latency is $8 *$ cycle time because there are 8 stages for an instruction to cycle through.

   (b) **Question 1.B Faster Memory**
   Suppose there is an optimization that reduces the memory stage latency by 100 ps. However, the optimization in a memory stage results in a 100ps increase in the write back stage latency. How much would this improve the overall performance of the 8-stage pipeline? Fillout the second row in the table below for this pipeline.

   Explanation - This new configuration would improve performance by 25%, or a 1.25x speed up. I got this number by dividing the latency of the baseline (3200 ps) by the latency f the improved pipeline (2560). The cycle time for this new configuration is 320ps, which is the time associated with the decoding stage. After reducing the memory access stage by 100ps to 300ps the new longest stage was decoding so the cycle time became associated with this stage. The max clock frequency is simply the reciprocal of the cycle time. The instruction latency is $8 *$ cycle time which is 2560ps (2.56ns).

   (c) **Question 1.C Proposed Scheme**
   If you had the option to rearrange the pipeline, how would you arrange it to achieve maximum frequency?

   Explanation - As given originally the total time to get through the pipeline by adding up the different times of each stage was 2240 ps. Dividing this by 8 (for 8 stages in the pipeline) gives us a round number, 280. So if we adjust the boundaries between the stages so as to give us a constant stage latency of 280ps, this would be optimal. Beyond this, decreasing the time for any one stage will necessarily increase the time it takes another stage to run. The max clock frequency is $\frac{1}{\text{Cycle Time}}$. The instructon latency in $8 *$ (cycle time), and since the cycle time is constant, this is just $8 * 280 = 2240$.

Table 1: Q1 Calculations

Processor	Cycle time	Max Clock Freq.	Instr. Latency	Throughput
a.) baseline	400	2.50 Ghz	3200 ps	1 per cycle
b.) faster mem	320	3.125 Ghz	2560 ps	1 per cycle
c.) new scheme	280	3.5714 Ghz	2240 ps	1 per cycle

2. **Question #2 - Extended Tiny ISA**
   Assume we are about to use an extended version of our Tiny ISA in an embedded system (say, a washing machine). Our implementation has 5 pipeline stages, and the ISA, as explained in lecture 2, does not have a multiply instruction. Analyzing the applications required to run on the processor, we learn that the applications needs to perform integer multiply about 1% of time. Now you as the

architect have to decide whether we should add a multiply instruction to the ISA and support it in the hardware, or just use the addition instruction to perform the multiplication through multiple additions.

(a) **Question 2.A** Compare options 1, 2, and 3 given above (in the handout, not rewritten here). Which option do you recommend? Show some sort of justification involving the performance of each option.c

Answer : I will start with a comparison of the three options. The table below summarizes the comparison. To begin, I define a baseline processor that has no multiply instruction or any software support for a integer multiply instruction. Such a processor has CPI = 1, frequency = f, latency = l, and instruction count = IC. All of the options will be compared by their performance relative to this baseline. The table shows the values for how the different options compare in different areas. The formula for performance is $\frac{1}{\text{Time(A, P)}}$ where A is a given machine and P is a given program. Time is calulcated by the following formula : Time $= IC * CPI * Period_{clk}$.
Option #1 is the worst of the 3 options when measuring performance. This method manages to keep the CPI the same as the baseline processor, which is good, but each of those cycles is taking twice as long to complete as the baseline. This effectively reduces the frequency by half, which in turn doubles the latency of the pipeline. The time for this option was calculated as : $1 * IC * 2P_{clk} = \frac{2*IC}{f}$.
Option #2 is the second best by my calculations. This method also manages to keep the CPI at 1, which is good. The frequency and hence the latency is unchanged, which also is a good thing. The increase in instruction count isn't very significant, but it's not enough to edge out option #2 over option #3. This makes sense to me intuitively, forcing integer multiplication to be implemented in the software shouldn't be as performance effective as an intelligent implementation in the hardware. It is interesting that this option is better than the first, it shows that some naive hardware implementations will be slower than basic software implementations. The time calculated for this option was $1.17IC * P_{clk} = \frac{1.17IC}{f}$.
Option #3 is the best option by my calculations. This approach does increase CPI by 20%, which is not insignificant. This unwanted increase in CPI is more than beaten out by the halving of the cycle time that increasing the number of stages from 5 to 8 affords us. This halving of the cycle time means the frequency doubles, so even if we are using 20% more cycles per instruction, each cycle is happening twice as fast and thus we end up better than where we started. Interestingly, the latency of the pipeline also decreases, but this time it decreases to 80% of the value that the baseline processor had. This is because although the frequency (and thus the delay at each stage) was halved, the number of stages was increased from 5 to 8. The time for this option was calculated as : $1.2 * IC * \frac{1P_{clk}}{2} = \frac{1.2IC}{2f} = \frac{0.6IC}{f}$
Since Performance $= \frac{1}{\text{Time}}$, and option 3 had the lowest time, option 3 will have the highest performance, and thus that is the option that I recommend.

Table 2: Q 2.A : Performance Calculations

Processor	CPI	Freq.	Latency	# Instructions
Baseline	1	f	l	IC
Option #1	1	$\frac{1}{2}f$	$2l$	IC
Option #2	1	f	l	$1.17IC$
Option #3	1.2	$2f$	$\frac{4}{5}l$	IC

(b) **Question 2.B** Lets assume we decided to go with option #3. The software team comes to you and propose a compiler optimization which helps limit the increase in CPI, so it only increases by 5% instead of 20%. However, the software team points out that this optimization could possibly increase the instruction count. What is the maximum increase in the instruction count that you can accept?

Answer : The maximum percentage increase for the inctruction count that would still be acceptable is 14.29%. At this percentage, all of the gains in performance that have been made by reducing the CPI from 1.2 to 1.05 will be erased by the increase in instruction count. Any higher of a percentage and we actually lose performance, even with the decrease in CPI. I determined this by setting $\text{Time}_{old} = 1.2 * CPI * IC * P_{clk}$ and $\text{Time}_{new} = 1.05 * CPI * (x * IC) * P_{clk}$, where x is the factor by which the instruction count would increase. Setting these two equations equal to eachother and solving for x yield $x = 1.14285$, thus the maximum increase is roughly 14.3%.

3. **Question #3 : CPI Changes** The table below (in the handout) shows the distribution of each instruction type, on a processor running at a 1 GHz clock frequency.

   (a) **Question 3.A : Average CPI** What is the average CPI of the processor?
   Answer : The average CPI is simply the weighted average of the CPIs of the individual instruction, with the weights being the % of times that instruction shows up in a representative code sample. Taking a weighted average over the first two columns (the dot product of C_1 and C_2^T) gives a final CPI value of 2.08.

   (b) **Question 3.B : Optimization** With some optimization, and the support of some extra logic, the microarchitecture can decrease the load and store CPI to 2 and branch to 1 (option #1). Alternatively, the extra logic that can be spent can be used to optimize the mult/div logic to support CPI = 1 (option #2). Which optimization makes better sense to improve performance?

   Answer : Given the data in the table describing the two different optimizations, optimization #1 comes out as the clean winner, at least as far as CPI is concerned. This has to do with the principle of simplifying the operations which occur most frequently. Optimization #1 makes modest decreases in the CPI of very common operations, which ends up reducing its CPI from 2.08 to 1.53, a 26.5% decrease in CPI. For optimization #2, they got a very good optimization on a single instruction that rarely occurs, and thus there CPI only decreased from 2.08 to 1.9, a respectable but non-optimal 8.7% decrease in CPI. Given these facts, optimization #1 is the clear winner.

4. **Question #4 : CPI Computation for Different Pipelines**

   Assume the instruction breakdown as defined in the table (in the handout). Branch and Jump instructions are resolved in the end of Execute stage and there is no branch delay slot. Assume no data or structural hazards. Branches are taken 80% of the time. Answer each of the following questions. Each sub-question is independent.

   (a) **Qestion 4.A : CPI** Calculate the CPI of the system given in the handout. Assume no data dependencies.
   Answer : The CPI of the system is calculated almost as normal, where we take the weighted average of the individual CPIs for each instruction multiplied by their individual probabilities of occuring. The CPI for the instructions is normally 1, because there are no data hazards, but when a branch occurs the CPI will increase from 1 to 3 for that instruction. The CPI is 4 when a branch occurs because all of the instructions below it are already into their 4th stage by the end of the branch instructions execute cycle, at which time the branch instruction is resolved. This means that those 4 cycles are wasted, increasing the CPI to 4 for taken branch instructions. Jump instructions occur infrequenctly (only 5% of the time), but when they do occur the CPI is 3, because any instructions below are already in their 3rd stage by the time the jump instruction is resolved at the end of its execute stage. Thus the total CPI is

   $$.15 * .8 * 4 + .15 * .2 * 1 + .05 * 3 + 0.3 * 1 + 0.3 * 1 + 0.2 * 1 = 1.46$$

   The first two terms account for the CPI of the branch instruction (2 terms for when it taken 80% of the time and not taken 20% of the time). The 3rd term accounts for the jump instruction's CPI of 3, the next two terms account for load and store, and the final term accounts for all other instructions.

(b) **Qestion 4.B : Taken Branches** What is the CPI if all branches are assumed to be taken? Assume no data dependencies
Answer : If the branch is taken 100% of the time, then the CPI increases slightly. Adjusting the last equation slightly to account for this change yields ...

$$.15 * 1 * 4 + .05 * 3 + 0.3 * 1 + 0.3 * 1 + 0.2 * 1 = 1.55$$

This result is predictable, if we are always taking the branch then we are always taking an instruction with a higher CPI. This is because branch instruction has a CPI of 4 when a branch is taken and 1 when a branch is not taken.

(c) **Qestion 4.C : Branch Delay Slot** Assume the CPU has one branch delay slot and it is taken 50% of the time. What is the CPI, ignoring start-up?
Answer :

(d) **Qestion 4.D : CPI** Assume we have data hazards. Fifty percent of the instructions following Load and Other use the result from that previous instruction. Example: the following sequence has an ADD instruction following a LW that uses the LW result. This occurs in 50% of the instructions. Notice that the instruction following the ADD also has a 50% chance of using the result of the ADD. **Answer :**

(e) **Qestion 4.E : Data Hazards II** Consider that 25% of the time the second instruction after the Load and Other instruction can use the data from the instruction. Example: the following sequence has an ADD instruction following the LW with no data hazard (50% chance from the previous part), but the following SUB instruction has a 25% chance of using the data generated by the LW. This 25% is independent of the 50% chance from the first dependenceCalculate the CPI of the system given in the handout. Assume no data dependencies.

```
lw r1, 0(r2)
add r5, r2, r3
sub r4, r1, r3
```

Answer :

5. **Question #5 : Multipliers** In this problem, we consider several different implementations of an unsigned two-inpuit integer multiplier capable of multiplying a 32-bit operand by a 4-bit operand to produce atruncated 32-bit result. Any of these implementations may be used to allow for the Tiny ISA to support the multiply instruction mentioned in the previous problem The diagram below illustrates the datapaths for five multiplier microarchitectures: (a) a single-cycle microarchitecture, (b) a four-cycle iterative microarchitecture, (c) a two-cycle pipelined microarchitecture, (d) a four-cycle pipelined microarchitecture, and (e) a variablecycle pipelined microarchitecture. For all parts in this problem we will assume that our multiplier is processing the following sequence of four independent instructions:

```
mul r0, r0, 0xf
mul r1, r1, 0x7
mul r2, r2, 0x3
mul r3, r3, 0x1
```

In each part, we will study one of these microarchitectures, and our goal is to gradually fill in the table below. The table already includes the results for the single-cycle multiplier microarchitecture.

Table 3: Q 5 : Performance Calculations

	Microarchitecture	Num Instr. #	Cycle Time τ	B2B Instr. Latency cyc τ	Throughput CPI	Total Exec. Time τ
a	1-Cycle	4	63	1 63	1	252
b	Iterative					
c	2-Cycle					
d	4-Cycle					
e	Var-Cycled					