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	5/8
b.) faster mem	320	3.125 Ghz	2560 ps	
c.) new scheme	280	3.5714 Ghz	2240 ps	

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%.