CMPE 110 Computer Architecture, Fall 2014
Homework #2

Computer Engineering
UC Santa Cruz

October 23, 2014

Name: ______________________________

Email: ______________________________

Submission Guidelines:

- This homework is due on Thursday 10/23/14.
- Bring the homework to class before 8pm
  - Anything later is a late submission
- **Please write your name and your UCSC email address**
- **The homework should be "readable" without too much effort**
  - If your handwriting is like mine, type it or risk not being graded
- Provide details on how to reach a solution. An answer without explanation has no credit. Clearly state all assumptions.
- Points: $40 = 7 + 12 + 15 + 6$

Question 1. Pipeline Stages (7 points)

Let us consider five stage pipelined (fetch, decode, execute, memory, write back) processor.

Pipline Stage	Latency of each stage (ps)
Fetch	240
Decode	320
Execute	280
Memory	400
Write Back	200

Question 1.A Baseline (2 points)

Fill out the first row in 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 largest frequency clock 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.

Question 1.B Faster Decode (2 points)

Suppose there is an optimization that reduces the Decode stage latency by 100 ps. So the new Decode latency would be 220ps. How much would this improve the overall performance of the 5-stage pipeline? Fill out the second row in the table below for this new pipeline.

Question 1.C Split Stage (3 points)

Suppose we split one of the stages of the pipeline into two stages of equal latency. Assume no extra latency due to the additional flops for the new stage. Which stage will you split to obtain optimal performance? How will it affect the performance of the processor? Fill out the last row in the table below for this new pipeline.

Processor	Cycle Time	Max Clock Frequency	Latency of Instr	Throughput
a) baseline				
b) faster decode				
c) split stage				

Question 2. Extended Tiny ISA (12 points)

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 the architect has 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.

We can add multiply logic to ALU in EXE stage, and assume each multiply instruction (mul $r3, $r2, $r1) takes 1 cycle to finish each of the stages (CPI = 1). However, addition of the multiply logic doubles the latency of EXE state, which was already the slowest stage in the pipeline along with IF and MEM.

Or, we can compute the multiplication in software instead of hardware (e.g. 5 * 3 = 5 + 5 + 5). This option, however, increases the average number of instructions by 17%.

Question 2.A Comparison (4 points)

Compare the impact of each of the solution on the average performance.

Question 2.B Hardware (4 points)

The architect decides to support the multiplication in hardware by adding a multiplier in the ALU, and introducing a `mul` instruction to be able to access the multiplier by software. However, they decide to break the EXE, IF, and MEM stages into multiple cycles. As a result, the new microarchitecture has 8 stages instead of 5. With this change, the processor can operate at twice the frequency compared as before. However, the CPI increases by a factor of 1.2. what is the relative speedup compared to the option A) with multiplier?

Question 2.C Software (4 points)

The software team has come up with some compiler optimization which helps limit the increase in CPI by 1.1 times. However, it is going to increase the instruction count. What is the maximum increase in the instruction count that can still allow this compiler optimization to result in a faster processor compared to option B?

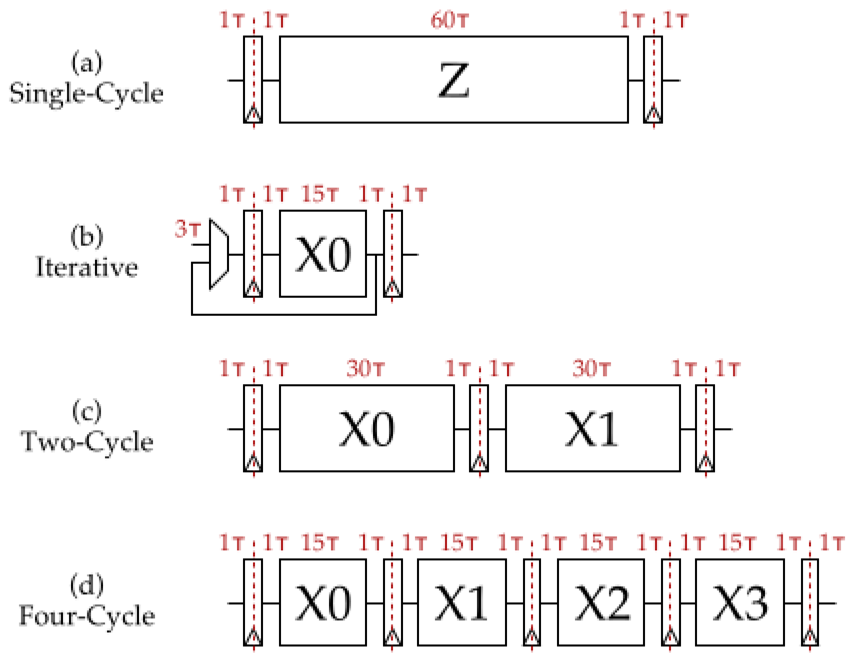
Question 3. Multipliers (15 points)

In this problem, we consider several different implementations of an unsigned two-input integer multiplier capable of multiplying a 32-bit operand by a 4-bit operand to produce a truncated 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 diagrams below illustrates the datapaths for four microarchitectures: (a) a single-cycle microarchitecture, (b) a four-cycle iterative microarchitecture, (c) a two-cycle pipelined microarchitecture, and (d) a four-cycle pipelined microarchitecture.

For all parts in this problem we will assume that our multiplier is processing the following sequence of four independent instructions:

```
1  mul $r0, $r0, #15
2  mul $r1, $r1, #7
3  mul $r2, $r2, #3
4  mul $r3, $r3, #1
```



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

	Microarchitecture	Num Instr (#)	Cycle Time (τ)	B2B Instr Latency (cyc)	(τ)	Throughput CPI	Total Execution Time (τ)
(a)	1-Cycle	4	62	1	62	1	248
(b)	Iterative	4					
(c)	2-Cycle Pipelined	4					
(d)	4-Cycle Pipelined	4					

The *back-to-back instruction latency* is the number of cycles between when an instruction reads its inputs and when it produces its output.

The *total execution time* is the total time (in units of τ). to execute all four instructions. The table includes the total execution time with and without the pipeline startup overhead.

An instruction vs. time diagram shows the stages of instruction in every cycle. There should be one column per cycle and one row per instruction. An example instruction vs time diagram for the 1-Cycle multiplier is shown below.

	C0	C1	C2	C3
I0	Z			
I1		Z		
I2			Z	
I3				Z

Question 3.A Iterative Microarchitecture (4 points)

Consider the iterative multiplier microarchitecture shown in (b). This microarchitecture computes a multiplication iteratively, across four cycles. It does this by sending back intermediate values building up to the result which is ready by the fourth cycle. In other words, we can complete each instruction in exactly four cycles, and we are ready to start the next instruction after exactly four cycles.

Fill in the instruction vs. time diagram below, illustrating the execution of the four multiplication instructions on this microarchitecture. Use the symbol X0 to indicate on which cycle each instruction is using the iterative multiplier. Use your instruction vs. time diagram to fill in the row in the first table for the iterative multiplier.

	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15
I0																
I1																
I2																
I3																

Question 3.B 2-Cycle Pipelined Microarchitecture (4 points)

Consider the two-cycle pipelined multiplier microarchitecture shown in diagram (c). In this microarchitecture, we use a similar approach as in (a) but we break it up the operation into two cycles. In this microarchitecture, two different instructions can use the multiplier at the same time (i.e., one in the X0 stage and one in the X1 stage).

Fill in the instruction vs. time diagram below, illustrating the execution of the four multiplication instructions on this microarchitecture. Use the symbols X0 and X1 to indicate on which cycle each instruction is using that part of the multiplier. Use your instruction vs. time diagram to fill in the row in the first table for the 2-Cycle pipelined multiplier.

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
I0																
I1																
I2																
I3																

Question 3.C 4-Cycle Pipelined Microarchitecture (4 points)

Consider the four-cycle pipelined multiplier microarchitecture shown in diagram (d). In this microarchitecture, we use the same basic approach as the two-cycle multiplier, but we go even further and break up the operation into four stages, where there can be four different instructions using the multiplier at the same time (i.e., different instructions in X0, X1, X2, and X3).

Fill in the instruction vs. time diagram below, illustrating the execution of the four multiplication instructions on this microarchitecture. Use the symbols X0, X1, X2, and X3 to indicate on which cycle each instruction is using that part of the multiplier. Use your instruction vs. time diagram to fill in the row in the top table for the 4-Cycle pipelined multiplier.

	C0	**C1**	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**	**C15**
I0																
I1																
I2																
I3																

Question 3.D Comparison of Designs (3 points)

Which microarchitecture has the highest performance? Explain some of the trade-offs between these microarchitectures. Would we ever want to consider a multiplier with many more stages (e.g., a 20-cycle pipelined multiplier)?

Question 4. CPI Changes (6 points)

The table below shows the distribution of each instruction type, on a processor running at a 1 GHz clock frequency.

Instr	**Proportion**	**CPI (current)**	**CPI (opt #1)**	**CPI (opt #2)**
Load/Store	35%	3	2	3
Branch	20%	2	2	2
Mul/Div	3%	5	5	3
Other	42%	1	1	1
Total	100%			

Question 4.A Average CPI (2 points)

What is the average CPI of the processor?

Question 4.B Optimization (4 points)

With some optimization, and the support of some extra logic, the microarchitecture can decrease the load/store CPI to 2 (option #1). Alternatively, the extra logic that can be spent can be used to optimize the mul/div logic (option #2). Which optimization makes better sense to improve performance?