CIS501 – Computer Architecture **Midterm Exam**
Prof. Martin **Thursday, Oct. 27th, 2011**

This exam is an individual-work exam. Write your answers on these pages. Additional pages may be attached (with staple) if necessary. Please ensure that your answers are concise and legible. Read and follow the directions of each question carefully. Please attempt to answer all the questions (don't allow yourself to get stuck on a single question). You have 80 minutes to complete the exam (approximately one point per minute).

Write only as much as necessary. Be brief!

Name: ____

Problem	Page	Possible	Score
1	2	13	
2	3	11	
3	4	7	
4	5	13	
5	6	15	
6	7	7	
Total		66	

1. **[13 Points] Short Answer**.

(a) Caches are effective because real-world programs have two key properties. Name and describe these two properties:

1.

2.

(b) In the past, processors had smaller caches than today. Give two reasons:

1.

2.

(c) Energy consumption is an important design constraint for portable devices. Why?

(d) Energy consumption is an important design constraint for servers. Why?

(e) What are the two types of energy (or power) consumption of a processor? What is the source of each type?

1.

2.

(f) How can a compiler optimization increase performance yet hurt CPI?

2. **[11 Points] Performance Calculations**.

   (a) Consider a pipeline with a **five-cycle branch misprediction penalty** and a **single-cycle load-use delay penalty**. For a specific program, 20% of the instructions are loads, half of loads are followed immediately by a dependent instruction, 10% are store instructions, 15% are branches, the remaining 55% of instructions are simple single-cycle ALU operations. 80% of branches are predicted correctly. What is the average CPI of this program on this processor?

As a chip designer, you have been asked to determine the cache configuration of your company's next-generation chip. Two of your options are: (1) a direct-mapped cache with a **two-cycle hit latency** or (2) a set-associative cache with a **three-cycle hit latency**. In both cases, the miss latency is (an additional) 10 cycles.

   (b) If the miss rate of the direct-mapped cache is 15%, what is its average latency of a memory operation?

   (c) What miss rate must the set-associative cache obtain to have a lower average latency than the direct-mapped cache with a 15% miss rate?

In a different project, you must choose between two different pipelines: (i) a relatively short pipeline with a 3-cycle mis-prediction penalty running at 1Ghz, or (ii) a longer pipeline with a 16-cycle mis-prediction penalty running at 2Ghz. In the target workload, 40% of the instructions are branches.

   (d) Calculate under what conditions is pipeline *(ii)* faster than pipeline *(i)*.

   (e) Based on what you have learned about branches, does pipeline *(ii)* seem like a reasonable design point? Why or why not?

3. **[7 Points]** **Pipelining**.

   (a) We discussed a “fast branch” optimization in which certain simple branches (such as compare to zero) can resolve during the “D” stage rather than the “X” stage. What was the advantage of this approach for a simple 5-stage pipeline?

   (b) What are two disadvantages and/or limitations of this fast branch optimization?

   1.

   2.

The standard pipeline discussed in class has five stages: fetch (F), decode (D), execute (X), memory (M), and writeback (W). Consider the following **four-stage** pipelines, formed by combining two stages of the classic five-stage pipeline.

   (c) What specific **positive** impact would combining the “D” & “X” stages have on CPI?

   (d) What specific **positive** impact would combining the “X” & “M” stages have on CPI?

   (e) What specific **positive** impact might combining the “M” and “W” stages have on the clock frequency of the pipeline?

   (f) What specific impact would combining the “F” and “D” stages have on the implementation of branch prediction?

4. **[13 Points] Branch Prediction**

(a) What are the two purposes of a branch target buffer (BTB)?

1.

2.

(b) What is the purpose of a *return address stack*? Assuming a processor with a BTB, why might it also have a *return address stack* in addition to the BTB?

(c) What is a *gshare* branch direction predictor and why does it generally increase prediction accuracy?

(d) For fixed branch predictor size, sometimes a *bimodal* outperforms *gshare*. Give two distinct reasons this can occur.

1.

2.

(e) What technique is used to mitigate these tensions between bimodal and gshare?

(f) Consider a predictor with a single saturating counter which is either 1-bit or 2-bits. The 1-bit counter encodes either "Taken (T)" or "Not-taken (N)", and it is initialized to "Not-taken". The 2-bit counter encodes "Strongly Taken (T)", "Weakly Taken (t)", "Weakly Not-taken (n)" and "Strongly Not-taken (N)", and it is initialized to "Weakly Not-taken (n)".

i. Give a short sequence (4 or fewer) of branch directions (T or N) in which a 2-bit saturating counter is better than a 1-bit saturating counter. What is the prediction accuracy for *both* predictors?

ii. Give a short sequence (4 or fewer) of branch directions (T or N) in which a 1-bit saturating counter is better than a 2-bit saturating counter. What is the prediction accuracy for *both* predictors?

5. **[15 Points] Caches.**

(a) Consider system with 48-bit addresses and a 128KB direct-mapped data cache with 128-byte blocks. How many bits are in the offset, index, and tag for this cache?

offset bits: index bits: tag bits:

(b) What is the tag overhead percentage for the cache?

(c) In an important program, two program variables continually conflict in this cache. For example, the variables might be at the following two addresses:

```
0000 0000 0010 0100 0100 1000 1001 0000 0100 1000 1101 1010
0000 0000 1000 0010 1000 0011 0001 0100 0100 1000 1010 0100
```

What are three general options for how the cache geometry might be changed so that these two addresses no longer conflict. For each option, also give the primary (or most likely) disadvantage and an approach than can help mitigate (or reduce) that disadvantage.

Option #1:

Disadvantage:

Mitigation:

Option #2:

Disadvantage:

Mitigation:

Option #3:

Disadvantage:

Mitigation:

(d) Beyond just changing the cache geometry, what else could be done that might help prevent these two variables from conflicting in the cache?

6. **[7 Points] Pipeline Simulation.**

   Consider using the simple “scoreboard” algorithm you used in the second homework assignment to simulate a pipelined processor with full bypassing, two read ports, a single write port, and a configurable load-use latency. When applied to a standard pipeline, there is an implicit assumption in the model that all instructions flow through all stages before entering the writeback stage to avoid a specific type of hazard.

   What type of hazard is being avoided? What is the specific cause of the hazard?

   With longer load latencies (say, three or more cycles) having all instructions travel through all stages has a significant disadvantage. What is that disadvantage?

   In lecture, we discussed a pipeline with a multi-cycle multiplier in which multiply and non-multiply instructions traveled through a different number of stages in the pipeline. Inspired by such a design, consider a modified pipeline in which non-memory instructions enter the writeback stage (W) directly after the execute stage (X). Memory instructions travel through additional stages based on the specific load latency being simulated.

— question continued on next page —

How would you model a pipeline that accounts for the hazard introduced by the change described above? Briefly describe your approach and modify the pseudo-code below to implement it. *Hint:* You may add additional tracking structures to the code, but doing so is not actually necessary to model the desired effect.

```
int scoreboard[MAX_REGS]
next_instruction = true
cycle_count = 0


while (true) {
  cycle_count = cycle_count + 1


  decrement all non-zero entries in the scoreboard


  if (next_instruction) {
      read next instruction from trace
      next_instruction = false
  }


  if (source1_register is valid and scoreboard[source1_register] > 0) or
     (source2_register is valid and scoreboard[source2_register] > 0) {

      continue        // Stall detected; go to next loop iteration

  }


  next_instruction = true


  if (instruction.destination_register is valid) {

      scoreboard[instruction.destination_register] = instruction.latency

  }

}
```

— End of exam —

For reference:

Hex	Binary	Decimal
0x0	0000	0
0x1	0001	1
0x2	0010	2
0x3	0011	3
0x4	0100	4
0x5	0101	5
0x6	0110	6
0x7	0111	7
0x8	1000	8
0x9	1001	9
0xA	1010	10
0xB	1011	11
0xC	1100	12
0xD	1101	13
0xE	1110	14
0xF	1111	15

Power	Bytes	Kilobytes	Megabytes	Gigabytes
2^{0}	1			
2^{1}	2			
2^{2}	4			
2^{3}	8			
2^{4}	16			
2^{5}	32			
2^{6}	64			
2^{7}	128			
2^{8}	256	0.25 KB		
2^{9}	512	0.5 KB		
2^{10}	1024	1 KB		
2^{11}	2048	2 KB		
2^{12}	4096	4 KB		
2^{13}	8192	8 KB		
2^{14}	16,384	16 KB		
2^{15}	32,768	32 KB		
2^{16}	65,536	64 KB		
2^{17}		128 KB		
2^{18}		256 KB	0.25 MB	
2^{19}		512 KB	0.5 MB	
2^{20}		1024 KB	1 MB	
2^{21}		2048 KB	2 MB	
2^{22}		4096 KB	4 MB	
2^{23}		8192 KB	8 MB	
2^{24}		16,384 KB	16 MB	
2^{25}		32,768 KB	32 MB	
2^{26}		65,536 KB	64 MB	
2^{27}			128 MB	
2^{28}			256 MB	0.25 GB
2^{29}			512 MB	0.5 GB
2^{30}			1024 MB	1 GB
2^{31}			2048 MB	2 GB
2^{32}			4096 MB	4 GB
2^{33}			8192 MB	8 GB
2^{34}			16,384 MB	16 GB
2^{35}			32,768 MB	32 GB
2^{36}			65,536 MB	64 GB