CIS501 – Computer Architecture	**Midterm Exam**
Prof. Martin	**Thursday, Nov. 2nd, 2010**

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	9	
2	3	6	
3	4	6	
4	5	8	
5	6	8	
6	7	4	
7	8	12	
8	11	12	
Total		65	

1. **[9 Points] True or False**. If a statement is "false," briefly explain how so by describing how the statement may most simply be made true. Unjustified (or poorly justified) "false" answers will be marked wrong. Simply stating the negation of the false statement is *not* sufficient justification. *Please be specific!*

   (a) Integrated circuits can lead to such wonders as home computer, automatic controls for automobiles, and personal portable communications equipment.

   (b) Moore's 1965 paper predicts an exponential improvement in transistor switching speeds over time.

   (c) The cost to manufacture a chip is proportional to the area of the chip.

   (d) DRAM capacity is improving faster than its access latency.

   (e) The SPEC CPU benchmarks are a reasonable way to compare the performance of two processors with **different ISAs** and **different compilers**.

   (f) A compiler optimization can increase performance yet hurt CPI.

   (g) The ARM ISA is an example of a RISC ISA.

   (h) The principle of spatial locality states that recently referenced data is likely to be referenced again soon.

   (i) A loop stream detector mitigates the dependence cross-checking problem in a superscalar processors.

2. **[6 Points] Processor Performance.**

(a) Assume a typical program has the following instruction type breakdown:

- 30% loads
- 15% stores
- 50% adds
- 4% multiplies
- 1% divides

Assume the current-generation processor has the following instruction latencies:

- loads: 2 cycles
- stores: 5 cycles
- adds: 1 cycle
- multiplies: 14 cycles
- divides: 50 cycles

If for the next-generation design you could pick one type of instruction to make twice as fast (half the latency), which instruction type would you pick? Show your work.

(b) Consider a simple in-order five-stage pipeline with full bypassing, a two-cycle branch misprediction penalty, and a single-cycle load-use delay penalty. For a specific program, 40% of the instructions are loads, 20% are branches, the remaining 40% of instructions are simple single-cycle ALU operations. Half of the load instructions are followed immediately by a dependent instruction. 75% of branches are predicted correctly. What is the average CPI of this program on this processor?

3. **[6 Points] Virtual Memory**.

(a) What are two benefits of virtual memory?

1.

2.

(b) What decides upon the mapping of virtual pages to physical pages?

(c) How does the hardware avoid walking the page table at every memory access?

(d) What is the primary advantage of a multi-level page table? What is the primary disadvantage?

4. **[8 Points] Instruction Set Architectures**.

(a) ISAs have different numbers of registers, and there is no consensus on what is the "right" number of registers. Give two reasons not to have too many registers (e.g., 1000 registers) in an ISA:

1.

2.

Give one reason not to have too few registers (e.g., 4 registers):

(b) The IBM 801 project moved from a 16-register ISA to a 32-register ISA. What technological shift was the driving force behind this change?

(c) The IBM 801 project abandoned a variable-length instruction format in favor of a fixed-length instruction format. Give two advantages and one disadvantage of this change.

Advantage 1:

Advantage 2:

Disadvantage:

(d) The IBM RS/6000 and PowerPC ISAs are both direct decedents of the IBM 801 project. What was the most notable addition to these ISAs (and other modern ISAs) over and beyond what was present in the IBM 801?

5. **[8 Points] Branch Prediction Conflicts and Tagged Predictors**.

You're a microprocessor designer, and your trace-based simulations of an important workload indicate that branch instructions at the following two 32-bit addresses (in binary) are executed frequently:

- Address `A`: `1000 1101 1100 0011 0110 1011 0100 1001`
- Address `B`: `1000 1101 0110 1001 1110 1011 0100 1001`

(a) The above two instructions are likely to conflict (hash to the same entry) in a branch predictor. For a simple "bimodal" predictor of two-bit saturating counters, how many entries must the predictor have to prevent these two branches from interfering (conflicting) with each other? How many total bytes must the predictor be?

(b) You have a brilliant idea: "Why not create a set-associative branch direction predictor?" Your simulations indicate that a predictor with just 2048 entries (512 bytes) would be sufficient if it wasn't for these two trouble branches. Consider a two-way set-associative predictor that uses a straightforward tagging strategy (one full tag for each two-bit counter, instructions have 32-bit addresses). How large (in KBs) is a two-way set-associative 2048-entry tagged predictor?

(c) You have *another* brilliant idea: "Because this is just a predictor, it can be wrong, so it doesn't actually need the full tags, just enough of a tag to avoid this particular conflict". With this new insight, (1) how large should the tags be and (2) what is the total size in KBs of this predictor?

(d) How might a predictor capture both the conflict-mitigating benefits of a tagged set-associative predictor and most of the area efficiency of a tag-less predictor?

6. **[4 Points] Superscalar Simulation.** In the second homework assignment you simulated a single-issue pipelined processor with varying load-use latency using a simple "scoreboard" algorithm.

   Modify the pseudo-code below to simulate idealized four-wide superscalar execution of independent instructions (assume perfect fetch, perfect branch prediction, and a maximum of four instructions per cycle with any mixture of operations).

```
next_instruction = true

while (true) {

  if (next_instruction) {

      read next micro-op from trace

      next_instruction = false

  }

  if instruction is "ready" based on scoreboard entries for register inputs {

      if the instruction writes a register {

          "execute" the instruction by recording when its register will be ready

      }

      next_instruction = true

  }

  advance by one cycle by decrementing non-zero entries in the scoreboard

  increment total cycles counter

}
```

7. **[12 Points] Predication Performance.** Consider the code below and two versions of it in x86 assembly:

```
int sum_square_max(int x[], int y[], int size)
{
  int sum = 0;
  for(int i = 0; i < size; i++) {
    int val = 0;
    if (x[i] > y[i]) {
      val = (x[i] * x[i]);
    } else {
      val = (y[i] * y[i]);
    }
    sum = sum + val;
  }
  return sum;
}
```

Code A:

```
.L14:  movl  (%rdi,%rdx), %r8d
       movl  (%rsi,%rdx), %ecx
       cmpl  %ecx, %r8d
       jg .L16   // difficult to predict branch (50% taken, 50% not-taken)
.L15:  imull %ecx, %ecx
       addq  $4, %rdx
       addl  %ecx, %eax
       cmpq  %r9, %rdx
       jne .L14  // always predicted correctly
       // loop exit


.L16:  movl  %r8d, %ecx
       jmp .L15  // unconditional branch
```

Code B:

```
.L14:  movl  (%rdi,%rdx), %r9d
       movl  (%rsi,%rdx), %r8d
       movl  %r9d, %ecx
       movl  %r8d, %r11d
       imull %r9d, %ecx
       imull %r8d, %r11d
       cmpl  %r8d, %r9d
       cmovle   %r11d, %ecx
       addq  $4, %rdx
       addl  %ecx, %eax
       cmpq  %r9, %rdx
       jne .L14  // always predicted correctly
       // loop exit
```

In the first assembly translation, assume the branch labeled "difficult to predict branch" is taken/not-taken 50%/50%. The second assembly translation uses `cmov` to avoid the branch using predication.

Consider a simple single-issue pipelined processor with a **CPI of 1** when all branches are predicted correctly. The branch mis-prediction penalty is **4 cycles**.

(a) If the "difficult to predict branch" is predicted with 100% accuracy (always predicted correctly), which of the two codes is faster and how much faster is it (for example, X is 50% faster than Y)?

(b) If the "difficult to predict branch" is predicted with 0% accuracy (never predicted correctly), which of the two codes is faster and how much faster is it (for example, X is 50% faster than Y)?

(c) For what range of prediction accuracies for the "difficult to predict" branch will Code B (the predicated code) be faster than Code A (the non-predicate code)?

Now consider a superscalar pipeline. Assume for the moment that its **CPI is 0.5** (2 IPC) on both of these codes. The branch mis-prediction penalty remains **4 cycles**.

(d) For what range of prediction accuracies for the "difficult to predict" branch will Code B (the predicated code) be faster than Code A (the non-predicate code)?

(e) What does this say about the value of the predicated code as the processor's base CPI improves?

(f) In reality, there is at least one more additional aspect of wide-issue superscalar execution that favors Code B over Code A.

(g) What hardware feature included in recent Intel microprocessors (for example, the Core 2 and Core i7) would benefit Code B but not benefit Code A?

8. **[12 Points] Memory Hierarchy Performance**

Consider the following C code:

```
double sum_square_diff(double x[], double y[], int size)
{
  double sum = 0.0;
  for(int i = 0; i < size; i++) {
    double diff = x[i] - y[i];
    sum = sum + (diff * diff);
  }
  return sum;
}
```

The loop body can be translated into eight x86 assembly instructions, two of which are loads:

```
.L2: movsd (%rdi,%rax), %xmm1     // Load
     movsd (%rsi,%rax), %xmm2     // Load
     subsd %xmm2, %xmm1
     mulsd %xmm1, %xmm1
     addsd %xmm1, %xmm0
     addq  $8, %rax
     cmpq  %rdx, %rax
     jne   .L2
```

In this question, the loop executes an extremely large number of iterations (`size` is extremely large), but the function is called only once. Each array element is a 64-bit (8-byte) floating point value.

(a) Consider a simple non-pipelined processor core with a 1Ghz clock frequency (1ns clock period) and a blocking data cache. All non-memory instructions and loads that hit in the cache have a single-cycle latency. The data cache has 8-byte blocks, is two-way set associative, and has a miss penalty of 256 nanoseconds (256 cycles at 1Ghz). The cache is initially empty. What is the CPI?

(b) **Increasing block size.** What is the CPI if the cache block size is increased to be 64 bytes?

(c) **Adding stream buffers.** To further reduce the cache miss penalty, consider the addition of a pair of stream buffers to the system (one stream buffer for each of the two arrays). How many 64-byte entries must each stream buffer have to entirely hide the memory latency?

(d) **Bandwidth demands.** With deep enough stream buffers and enough memory bandwidth, the system can completely hide the memory latency in this program. What is the minimum amount of bandwidth that the main memory must sustain to keep up with the processor? Give your answer in gigabytes per second (GB/second). Hint: 1 GB/second is one byte per nanosecond.

(e) **Impact of limited bandwidth.** If the memory system provides only 1 GB/second of bandwidth, what is the new best-case CPI?

(f) **Impact of faster processor.** Consider replacing the simple processor core with a pipelined superscalar processor that improves the CPI by 2x (the new CPI is 0.5) and the core clock frequency by 2x (the new clock frequency is 2Ghz). Would this change the number of entries per stream buffer needed to hide memory latency? If so, approximately how much larger or smaller? (Assume sufficient bandwidth.)

— 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