Prelim 2

Computer Science 3410, Cornell University
Prof. Weatherspoon 30 April 2015

Read all of the following information before starting the exam:

Please write your name and NetId/email on the exam **now**.

This is a **closed book and notes** examination: *NO BOOK, NOTES, CALCULATORS, OR CELL PHONES*. You have **120 minutes** to answer as many questions as possible.

There are 6 problems on this exam. It is 23 pages long; make sure you have the whole exam. You will likely find some problems easier than others; read **all** problems before beginning to work, and use your time wisely. The exam is worth 100 points total.

Before starting the exam, write your name and netid on all pages of the exam.

Do all written work on the exam itself. If you are running low on space, write on the back of the exam sheets and be sure to write (OVER) on the front side. It is to your advantage to show your work — we will award partial credit for incorrect solutions that are headed in the right direction. If you feel rushed, try to write a brief statement that captures key ideas relevant to the solution of the problem.

Make your answers as concise as possible. If a question is unclear, please simply answer the question and state your assumptions clearly. If you believe a question is open to interpretation, then please ask us about it!

Problem	Points	Score
1	1	
2	9	
3	23	
4	22	
5	20	
6	25	
Total	100	

1. Write your name and NetID [1 pt]

[This page intentionally left empty]

2. Short Answers [9 pts] (parts a–c)

   Be brief. Answer should be one or two sentences at most. Verbose answers will lose points.

   (a) [3 pts] TRUE/FALSE. A `Privileged Mode` is necessary to enforce protection between processes. Justify answer.

   (b) [3 pts] TRUE/FALSE. Increasing the cacheline size always increases performance (i.e. increases hit rate). Justify answer.

   (c) [3 pts] List/describe three advantages of virtual memory.

3. Caches and Memory Hierarchy [23 pts] (parts a–f)

   Assume that we have a 32-bit processor (with 32-bit words) and that this processor is byte-addressed (i.e. addresses specify bytes). Suppose that it has a 512-byte cache that is two-way set-associative, has 4-word cachelines, and uses LRU replacement. Split the 32-bit address into "tag", "index", and "cacheline offset" pieces.

   (a) [3 pts] Which address bits comprise each piece?

   tag:
   index:
   cacheline offset: bits 3-0 (Given)

   (b) [3 pts] How many sets does this cache have? Explain.

(c) [4 pts] Draw a block diagram for this cache. Show a 32-bit address coming into the diagram and a 32-bit data result and Hit signal coming out.
Include, all of the comparators in the system and any muxes as well. Include the the tag matching logic, and any muxes, etc.

(d) [6 pts] Below is a series of memory read references set to the cache from part (a). Assume that the cache is initially empty and classify each memory references as a hit or a miss. Identify each miss as either **compulsory**, **conflict**, or **capacity**.

`Byte Address`	`Hit/Miss`	`Miss Type?`
0x300		
0x1BC		
0x206		
0x109		
0x308		
0x1A1		
0x1B1		
0x2AE		
0x3B2		
0x10C		
0x205		
0x301		
0x3AE		
0x1A8		
0x3A1		
0x1BA		

(e) [3 pts] Calculate the miss rate and hit rate?

(f) [4 pts] You have a 1 GHz processor (i.e. 1 clock cycle per ns) with 2-levels of cache, 1 level of DRAM (memory), and a DISK for virtual memory. Assume that it has a Modified Harvard architecture (separate instruction and data cache at level 1). Assume that the memory system has the following parameters:

`Component`	`Hit Time`	`Miss Rate`	`Block Size`
First-level Cache	1 cycle	4% Data 1% Instruction	64 bytes
First-level Cache	20 cycles + 1 cycle/64 bits	2%	128 bytes
DRAM (Memory)	100ns + 25ns/8 bytes	0.1%	16K bytes
DISK	50ms + 20ns/byte	0%	16K bytes

Finally, assume that there is a TLB that misses 0.1% of the time on data (it does not miss on instructions) and which has a fill penalty of 40 cycles. What is the average memory access time (AMAT) for Instructions? For Data (assume all reads)?
Write down the general formula for AMAT.

[This page intentionally left empty]

4. MMU and Virtual Memory [22 pts] (parts a–i)

   64-bit computers are *very* common. We want to understand what 64-bits really means for the virtual memory system. As a result, in this problem, we consider a byte addressable virtual memory system with **64-bit virtual addresses**, **48-bit physical addresses** and **16 kB pages**.

   Write answers using *both* **binary** and **decimal** notation.
   (For example, for 2^{14} *bytes, we could write both 16 kB and 16 thousand-bytes)*

   For *binary notation*, you may write the amount of bytes using k for kilo-, M for mega-, G for giga-, T for tera-, P for peta-, E for exa-, Z for zetta-, and Y for yotta-byte: kB, MB, GB, TB, PB, EB, ZB, YB, respectively.

   For *decimal notation*, you may approximate and just write thousand-, million-, billion-, trillion-, quadrillion-, quintillion-, sextillion-, septillion-, octillion-bytes (. . . googol-bytes); or 10^3, 10^6, 10^9, 10^{12}, 10^{15}, 10^{18}, 10^{21}, 10^{24}, 10^{27} (. . . 10^{100}), respectively.

   (a) [2 pts] What is the maximum amount of physical memory that the system could support?

   (b) [2 pts] What is the maximum size of a single process; i.e. what is the size of the virtual memory space for a single process?

(c) [3 pts] If this system used a *single-level* page table, how many entries would it need to have? Why might this be a problem?

(d) [3 pts] If all entries in the *single-level* page table took up 8 bytes each, what is the minimum amount of physical memory that a 2 MB process would occupy (if the entire process and page table was in memory)?

(e) [3 pts] How many entries would the **first level** of a *two-level* page table have, assuming that the same number of bits are used for both levels?

(f) [3 pts] If all entries in the *two-level* page table (both first and second level) took up 8 bytes, what is the minimum amount of physical memory that a 2 MB process would occupy (if the entire process and page tables were in memory)?

(g) [2 pts] Why is a page table entry 8 bytes, instead of 4. Briefly, explain in *one sentence*. More than one sentence or run-on sentences will lose points.

(h) [2 pts] What is the problem of 64-bit virtual memory address spaces (based on your answers above)?

(i) [2 pts] If instead of two-level page tables, we had *five*, what are some other issues with 64-bit virtual memory address spaces?

5. Multicore, Parallelism, and Synchronization [20 pts] (parts a–d)

A *producer/consumer ring buffer* is a very common data structure used in parallel programs. It is used to pass information between threads. There can be many producer threads and many consumer threads that all share the same producer/consumer ring buffer. The invariant is that a producer can only produce if the buffer is not full, and an item is never overwritten; and a consumer can only consume if the buffer is not empty, and an particular item is consumed only once.

In this problem, we will consider the following code snippets executed in parallel by potentially many `Producer` and `Consumer` threads:

```
Consumer:
   empty = (tail==head);
   if(!empty)
     head++;

Producer:
   full = (tail-head)==n;
   if(!full)
     tail++;
```

These might be compiled into the following assembly code. Assume that variables `head` is at location `0($a0)`, `tail` is at location `0($a1)`, and an extra shared variable at `0($a2)`.

```
Consumer:
    LW $t0, 0($a0)     // A0 : read head
    LW $t1, 0($a1)     // A1 : read tail
    BEQ $t0, $t1, Consumer
    NOP
    ADDIU $t0, $t0, 1
    SW $t0, 0($a0)     // A2 : store head+1

Producer:
    LI $t3, n
    LW $t0, 0($a0)     // B0 : read head
    LW $t1, 0($a1)     // B1 : read tail
    SUB $t2, $t0, $t1,
    BEQ $t2, $t3, Producer
    NOP
    ADDIU $t1, $t1, 1
    SW $t1, 0($a1)     // B2 : store tail+1
```

The load and store instructions are marked `A0`, `A1`, `A2`, `B0`, `B1`, `B2`. Assume that the architecture ensures that memory accesses from one thread are never done out of order from the viewpoint of all threads. For example, this means `A0` always comes before `A1`, and `B0` always comes before `B1`.

(a) [6 pts] Let us assume there are *two* `Consumer` threads, i and j and one item in the ring buffer. Show two possible interleavings if each thread executes the `Consumer` code one time with the starting values: `head` $= 1$, `tail` $= 2$. For example, the interleaving (A_i0, A_i1, A_i2, A_j0, A_j1) results in outcome (`head = 2`, `tail = 2`), one item being consumed, and A_j2 not being executed because the buffer would be empty.
Show two more possible outcomes for (`head`, `tail`), and for each, state how many items were consumed, and show the corresponding interleaving of memory operations.

(b) [10 pts] Briefly explain how to use a standard synchronization mechanism to enforce the invariant: Producers can only produce if the buffer is not full and a consumers can only consume if the buffer is not empty, and no single item is consumed more than once, and no item is overwritten by different producers.

Write pseudo-code using the pair of atomic instructions `LL` and `SC` to achieve synchronization for the code.

[This page intentionally left empty]

(c) [2 pts] Is there any way to optimize the code if there is only one `Consumer` (i.e. there is only one `Consumer` thread)? Explain, briefly.

(d) [2 pts] Is there any way to optimize the code if there is only `Consumer` and `Producer` (i.e. there is only one `Consumer` thread and only one `Producer` thread)? Explain, briefly.

6. Calling Conventions and Assembly Code [25 pts] (parts a–g)

   For all parts of this question attempt to follow the calling and register convention described in class and in Chapter 2 and Appendix A of P&H.

   (a) [2 pts] Describe the advantages of having both caller- and callee-saved registers.

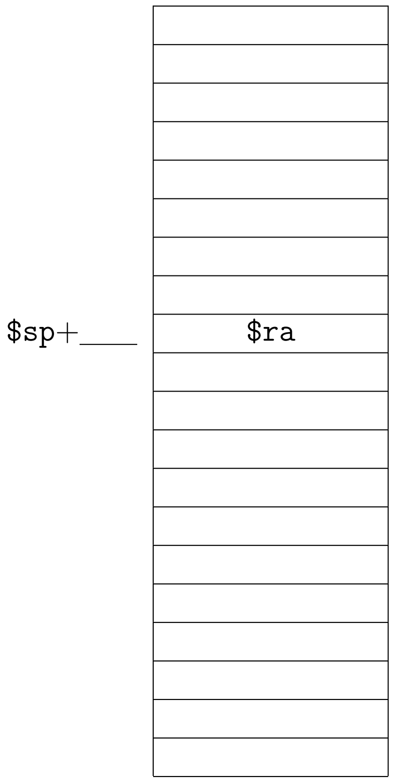
   (b) [3 pts] Consider the following C program. How many caller- and callee-saved registers would you use and for which variables?

```
int foo() {
    int a = 0;
    int b = 12;
    int c = 1;

    while(b + c > 0) {
        int e = b + bar(c);
        c = b + e;

        int d = c + baz(b);
        a = d - e;
    }

    return a;
}
```

(c) [3 pts] Consider the following C program. How large is the stack frame?

Fill in the stack frame for the function. Show and mark the location where the space for each incoming and outgoing argument, callee-/caller-save, etc is allocated. Also, write the location of each item on the stack with respect to the `$sp`, and identify the current and parent stack frame.

```
int recursive_foo(int a, int b, int c, int d, int e) {
   int g;
   if (a > 0)
      g = recursive_foo(a - 1, b, c, d, e)

   return b + g;
}
```



(d) [3 pts] Starting with the arguments and then local variables for function `recursive_foo()`, state whether the assembly implementation will require a callee- or caller-save register. Further, specify which register to use for each arg and local variable, and if it needs to be spilled to the stack (i.e say "stack" if it needs to be saved to the stack).

arg/variable	**caller/callee**	**register (and stack)**
a		
b		
c		
d		
e		
g		

(e) [4 pts] Write the prologue for `recursive_foo()`. Also, use your register assignment from part 6(d).

```
RECURSIVE_FOO:
```

(f) [2 pts] Using your answers from parts a and b, write the epilogue for `recursive_foo()`.

```
EPILOGUE:
```

(g) [8 pts] Now, write the implementation for the body of the `recursive_foo()` function. Use the same registers in the body as your answer in part 6(d).

Write your soln below. Your branches should only have to use labels `RET`, `EPILOGUE`, if necessary. Also, you may use any instruction in your implementation on the MIPS reference sheet including pseudoinstructions.

```
BODY:



IF:  # if (a > 0), call recursive_foo



RET:
```

[recursive_foo() function is repeated for reference]

```
int recursive_foo(int a, int b, int c, int d, int e) {
   int g;
   if (a > 0)
      g = recursive_foo(a - 1, b, c, d, e)

   return b + g;
}
```