CPSC 313, 04w Term 2— Final Exam

Date: April 21, 2005; Instructor: Mike Feeley

This is a closed book exam; no notes; you may use calculators to perform simple arithmetic calculations. Answer in the space provided; use the backs of pages if needed. There are **14** questions on **11** pages, totaling **100** marks. You have **2 hours and 50 minutes** to complete the exam.

NAME: _____ **SCORE:** _____ / 100

STUDENT NUMBER: _____

1. **(20 marks)** Short answers.

1a. Explain the following parts of a UNIX process' address space, by giving an example of one thing that is stored in each of these parts:

.text:

.data:

.bss:

heap:

stack:

1b. What does a `call` instruction do that a `jmp` instruction does not?

1c. What is a *jump table* and what C-language control structure is typically implemented by one?

1d. A pipelined CPU can deal with hazards by introducing pipeline bubbles. List one technique that eliminates bubbles for some *data hazards* and one for some *control hazards*.

1e. Why are the sizes of cache blocks and sets powers of two?

1f. Briefly, how do caches seek to exploit *spatial locality* to improve performance?

1g. What is the main difference between *interrupts* and other types of exceptions such as traps and faults?

1h. In terms of their implementation, what is the main difference between a procedure call and a switch between two user-mode threads (i.e., implemented by `setjmp` and `longjmp`).

1i. Virtual memory replacement is based on LRU, but set-associative cache replacement is random. List two differences between these two problems that justify the different solutions.

1j. Give one benefit and one drawback of inverted page tables.

2. **(3 marks)** The following assembly language was generated by the x86 C compiler from a simple control structure and a few additional statements.

```
            testl   %edx, %edx
            je      .L7
    .L5:    addl    %eax, %eax
            decl    %edx
            jne     .L5
    .L7:
```

Give the simplest C program that could have produced this assembly language.

3. **(7 marks)** Compile this C procedure into x86 (or y86) assembly language. Give both the `.text` and `.data` sections. You can use "`.skip <number-of-bytes>, 0`" to allocate space for variables. Be sure to include the prologue and epilogue. **Comment your code.**

```
int g[4,3];

int foo (int i, int j)
{
  int a;

  g[i,j] = &a;
  return g[i,j];
}
```

4. (8 marks) Consider the following C-language procedure.

```
void foo (int a1, int *a2)
{
    *a2=bar (a1);
}
```

These three statements could be implemented by the following assembly language code:

```
    # prologue omitted
    movl    8(%ebp), %ebx
    pushl   %ebx
    call    bar
    movl    %12(%ebp), %esi
    movl    %eax, (%esi)
    # epilogue omitted
    ret
```

Now, someone has decided to modify the x86 stack discipline to eliminate the frame pointer, freeing `%ebp` for general-purpose use (i.e., `%ebp` is not the frame pointer anymore).

4a. Explain how this decision complicates the compiler's code generation for procedures.

4b. Illustrate this complexity by modifying the two instructions above that use the frame pointer; make no changes to this code other than removing the two instructions and adding other instructions in the same place. Carefully comment your code and state any assumptions you make. (You can make this change by modifying the original code above or by writing the code below.)

4c. Does this change complicate the restoration of the calling stack frame in the epilogue? If so, carefully explain the problem and a possible solution.

5. (6 marks) This problem is a bit more challenging. Consider the following assembly-language code. Start by commenting every line of code. If you aren't sure what this code does, big partial credit for clearly commenting the individual lines of code. (Remember that `%dl` is the low-order byte of `%edx`.)

```
foo:    # prologue omitted
        movl    8(%ebp), %eax
        xorl    %edx, %edx
        subb    12(%ebp), %dl
        jne     .L1
        movl    $1, %eax
        jmp     .L3
.L1:    leal    .L2(%edx, %edx, 2), %edx
        jmp     *%edx
.L2:    imull   %eax, %eax      # this instruction is 3 bytes long
        imull   %eax, %eax
        ... repeat for a total of 255 imull lines
.L3:    # epilogue omitted
        ret
```

What function does foo(a,b) implement?

What is foo(2,3)?

6. (6 marks) Consider a three-stage pipeline where the gate delay of the stages are 13ns, 18ns and 8ns. The delay of the registers between stages is 2 ns. A ns is 10^{-9} seconds. Answer the following questions; you can skip the addition and multiplication by just giving me a formula that I can plug into a calculator to get the answer. Show your work.

6a. What is the shortest possible clock period (in units of ns)?

6b. What is the maximum throughput of the processor (in units of instructions per second)?

6c. Suggest a single architectural change that might improve throughput (I can think of two). Say **why it might** and **might why it might not** improve throughput.

6d. A bit more challenging. If 10% of instructions introduce a single pipeline bubble, what is the throughput of the processor (in units of instructions per second or cycles per instruction; be sure to label your answer with the units you use)?

7. **(5 marks)** For each of the following y86 code snippets, indicate three things. **First**, say whether a data dependency exists and if so show where it is, indicate its type, and explain. **Second**, say whether there is a data or control hazard for the *y86 PIPE* implementation and if so, show where it is and explain. **Third**, say whether the *y86 PIPE* would insert any pipeline bubbles, indicate where, say how many bubbles are inserted, and explain. You can combine your answers if, for example, the same pair of instructions are dependent, cause a hazard and insert a pipeline bubble.

7a.
```
addl    %eax, %ebx
addl    %ebx, %exc
```

7b.
```
movl    (%eax), %ebx
addl    %ebx, %exc
```

7c.
```
addl    %ebx, %exc
addl    %eax, %ebx
```

7d.
```
rrmovl  %eax, %eax
subl    %eax, %eax
jne     foo
...
```

8. (7 marks) Consider a 512-byte, 4-way, set-associative cache with 16-byte blocks. Draw a picture and use pseudo code to carefully explain how the cache handles a long-word (i.e., 4-byte) read request. **First show how the cache determines whether the access is a hit or a miss and then show how it would satisfy the read if it is a hit.** Show how every bit of the data's physical address is used. Your pseudo code should represent the cache using a combination of arrays and structs (e.g., something like $C[i].foo[j].bar$).

```
phys_addr = [ ___ bits | ___ bits | ___ bits ]
```

9. (6 marks) You work at Intel again and are allowed to make one single change to an existing cache design. For each change listed below list one potential **advantage** of making that change. In each case, the total size of the cache (i.e., the number of data bytes is stores) remains the same.

9a. Increase the block size?

9b. Decrease the block size?

9c. Increase the set associativity (e.g., from 4-way to 8-way)?

9d. Decrease the set associativity?

9e. Change from write through to write back?

10. **(6 marks)** Shared Libraries

10a. Give one benefit of dynamically-linked shared libraries, compared to static linking.

10b. Why can't shared libraries be statically linked? Explain briefly.

10c. Why must dynamically-linked shared libraries use *position independent code*?

10d. Outline the key idea that allows an instruction in shared library X to read a global variable allocated in shared library X in a position-independent fashion. (No need for assembly code; just describe the idea.)

11. **(6 marks)** Answer these questions about a system call trap from an application (i.e., user-mode) process into the operating system. Consider only the interval that starts with the trap instruction (i.e., `int $80`) and ends with the execution of the first instruction of the particular system call specified by the application (i.e. in `%eax`).

11a. Is this trap a *protected call*, a *context switch*, *both*, or *either*? Explain briefly.

11b. What hardware registers are involved? Carefully explain how the hardware uses them.

11c. What additional steps are handled by the operating system? Give a very brief outline (no need for details).

12. **(6 marks)** For each of the following virtual-memory mangement operations indicate whether, for the architectures discussed in class, it is handled by *hardware*, *software*, *both* or *it depends on the architecture*. **Give a very brief explanation for each answer.**

12a. Translation of a virtual address that maps to a physical-memory-resident page who's page-table entry (i.e., PTE) is stored in the translation look-aside buffer (i.e., TLB)?

12b. Handling of a TLB miss for a virtual address that maps to a physical-memory resident page who's PTE is stored in the current process's page table?

12c. Handling of a TLB miss for a virtual address that maps to a page that is not resident in physical memory?

12d. Handling of a TLB miss for a virtual address that maps to a page who's PTE is not stored in the current process's page table?

13. **(8 marks)** You are responsible the page-table implementation for a new, hypothetical, Pentium 4 processor that uses 42-bit virtual addresses instead of the current 32-bit addresses. Answer the following questions.

13a. Do you work at Intel or Microsoft? Explain briefly.

13b. If you stick with the current two-level page-table design, where each chunk of the level-two page table is exactly one 4096-byte page in size (and page table entries are 4 bytes), how many bytes are required to store the level-one page table? Explain briefly.

13c. You decide to change to a three-level page-table design. What improvement are you expecting and why?

13d. Your three-level page table is comprised of chucks each of which is a single page in size; that is, no two page-table pages need be contiguous (i.e., next to each other) in physical memory. **Draw a picture of this design showing how an virtual address is translated into a physical address** (ignore the TLB and data caches; just show the page-table lookup). **Indicate how each bit of the virtual address is used. Give a pseudo-code expression for this computation.** The pseudo-code should use $M[i]$ to indicate reading the value of memory at physical address i.

```
virt_addr = [ ___ bits | ___ bits | ___ bits | ___ bits ]
```

```
phys_addr =
```

_____ / 0

14. **(6 marks)** The Fifo-with-Second-Chance page replacement algorithm organizes pages on several fifos, one of which is called the *inactive list*. For the following two architectural variations briefly explain how the inactive list gives pages a second chance to be accessed before they are replaced.

14a. Architectures that have an *accessed* bit in the PTE.

14b. Architectures that do not have an *accessed* bit.

You do not need to write below here