# CPSC 313, 05w Term 1— Final Exam

Date: December 12, 2005; Instructor: Mike Feeley

This is a closed book exam; no notes; you may use calculators to perform simple arithmetic calculations. Pages of potentially useful notes appear at the end of the exam. Answer in the space provided; use the backs of pages if needed. There are **10** questions on **10** pages, totaling **100** marks. You have **3 hours** to complete the exam.

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

**STUDENT NUMBER:** ____________________

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

**1a.** List one difference between an object file produced by the compiler or assembler and an executable file produced by the static linker.

**1b.** Carefully describe the single difference between an x86 `call` and `jmp` instruction.

**1c.** Why does the x86 use two registers to point to positions on the runtime stack? Why not just one?

**1d.** List one advantage of separating reading from the register file and writing to the register file into different pipeline stages as we have done, with the read coming in the second stage and the write coming in the last stage.

**1e.** List one disadvantage of separating register-file access this way in the pipeline.

_____ / 20

**1f.** If a pipelined processor has stage delays of 10, 100 and 50 ns (including pipeline-memory delays), what is its maximum clock frequency (just give the formula, don't do the math)?

**1g.** Explain, by referencing parts of a C program, what is stored in each of the following three sections of an object file.

.text

.data

.bss

**1h.** What object-file section does the static linker use to locate positions in the code of an object file where it should write virtual addresses?

**1i.** Why does it make sense to use random replacement for hardware data caches (e.g., L1 and L2), but use an LRU approximation for virtual-memory replacement.

**1j.** What, specifically, about caches is designed to improve the performance of programs the exhibit spatial locality?

---

_____ / 0

**2.** **(12 marks)** Give an assembly-language translation for each of the following pieces of C code. Do not assume that the variable values are stored in registers or that the assignments need only store to a register. Give assembly language only for the C statements that are listed. Other stuff has been omitted from the C; you should also omit this stuff from the assembly. Be sure to include code and data declarations.

**2a.**
```
int a,b;        // two global variables; show their declaration
...
a=b;            // show code for this statement
```

**2b.**
```
void foo(int a,b) {     // you don’t have to give any code for this
    ...
    a=b;                // just give the code for this statement
    ...
}
```

**2c.**
```
int *a, *b;             // two global variables; show declaration
...
*a = *b;                // show code for this statement
```

_____ / 12

**2d.**
```
// you can assume that "b" and "i" are in registers
// implement switch using a jump table
switch (i) {
    case 10:
        b=0;
        break;
    case 11:
        b=1;
        // notice there is no "break" here!
    case 13:
        b=3;
        break;
    default:
        b=4;
}
```

**3.** **(8 marks)** Consider the following IA32 (x86) gas assembly-language file.

```
.text
    foo:    pushl   %ebp
            movl    %esp, %ebp
            pushl   j
            pushl   i
            call    bar
            movl    %ebp, %esp
            pop     %ebp
            ret
```

*(continued on next page)*

_____ / 8

```
    bar:    pushl   %ebp
            movl    %esp, %ebp
            movl    $1, -8(%ebp)
            movl    8(%ebp), %eax
            movl    %eax, -4(%ebp)
    .L1:    movl    -4(%ebp), %eax
            cmpl    12(%ebp), %eax
            jge     .L2
            movl    -8(%ebp), %eax
            addl    %eax, -8(%ebp)
            incl    -4(%ebp)
            jmp     .L1
    .L2:    movl    -8(%ebp), %eax
            movl    %ebp, %esp
            pop     %ebp
            ret
.data
    i:      .long 10
    j:      .long 20
```

Give a complete C-language source file that could have produced this assembly language. Be sure to show your work (e.g., by placing comments by each assembly language instruction, by writing down intermediate observations and by clearly stating any assumptions you make).

**4.** **(10 marks)** Pipelines and instruction-level parallelism.

**4a.** In the following piece of C-language code, consider all pairs of statements to identify *every* data dependency that exists among them. For each dependency name the type of dependency and list the variable(s) involved. (Note that some statements may be involved in more than one dependency.)

```
i = j;      // (1)
j = 0;      // (2)
k = 0;      // (3)
k = i;      // (4)
```

**4b.** The y-86 PIPE architecture discussed in class handles data hazards using a technique called *data forwarding*.

**i.** Of the three types of data dependencies, which one(s) result in pipeline hazards for y86-PIPE? Carefully explain why the hazard exists.

**ii.** Under what circumstances, if any, do these hazards result in pipeline stalls (and/or bubbles)? Carefully explain.

_____ / 10

**5.** **(10 marks)** Control hazard for *jump* instructions in the y86-PIPE implementation discussed in class.

**5a.** Consider the following piece of y86 code:

```
            jle .L0
    .L1:    ...
```

When the `jle` instruction is in the *decode* phase, there are three possible things that could be executing in the *fetch* phase: (1) a bubble, (2) the instruction at .L0 or (3) the instruction at .L1. Which is it? Explain.

**5b.** **[harder]** Lets modify the y86 branch prediction logic to predict in the fetch stage that a branch is taken if and only if the last branch to have completed the execute stage was actually taken. You will need to add a one-bit memory somewhere to remember what happened to the last jump, call it `last-jump-taken`. Give two HCL statements, one that sets the value of `last-jump-taken` and the other that computes `predict PC`. `Predict PC` is the program counter for the next instruction to enter the fetch phase; be sure to handle every instruction, not just jumps. Refer to the diagram at the back of the exam; it will help a lot. Be sure to use the standard naming convention (e.g., e_blah or E_blah).

_____ / 10

**6.** **(6 marks)** Consider the following set of potential cache-design changes. List one potential **disadvantage** of making each change. In each case, the total size of the cache (i.e., the number of data bytes is stores) remains the same.

**6a.** Increase block size?

**6b.** Decrease block size?

**6c.** Increase the set associativity (e.g., from 4-way to 8-way)?

**6d.** Decrease the set associativity?

**6e.** Change from write through to write back?

**6f.** Use the highest-order address bits for the set index, instead of the tag?

**7.** **(10 marks)** Consider a 64-byte, 4-way set associative cache with 8 byte blocks.

**7a.** List the number of address bits for each of these:

```
tag:           ______
set index:     ______
block offset:  ______
```

**7b.** Use pseudo code to explain how the cache checks for a cache hit.

**7c.** Consider the following C code.

```
int A[2][3];

for (i=0; i<2; i++)
    for (j=0; j<3; j++)
        k = A[i][j];
```

For each of the following accesses, indicate whether it is a cache hit, cache miss or whether you can't tell. For ”miss” indicate the reason for the miss. For ”can't tell” indicate the conditions necessary for the access to be a hit.

```
A[0][0]:
```

```
A[0][1]:

A[0][2]:

A[1][0]:

A[1][1]:

A[1][2]:
```

**8.** **(10 marks)** Consider the code in two libraries, A and B, listed below. The two listed instructions load values of global variables a and b into registers. You are the compiler. Replace these two statements with position-indepdent alternatives suitable for making the two libraries into dynamically-linked shared libraries. You may introduce symbols to represent any useful values that the compiler would know. Document your solution and symbol definitions; state all assumptions clearly.

```
LIBRARY A:
    .text
                ...
                movl    a, %eax
                movl    b, %ebx
    .data
        a:      .long   0

LIBRARY B:
    .data
        b:      .long   0
```

Position-independent equivalent of `movl a, %eax`:

Position-independent equivalent of `movl b, %ebx`:

**9.** **(6 marks)** Processes.

**9a.** Carefully explain what changes occur, if any, to the following processor registers when an application makes a system call:

protection mode:

program counter:

page-table base register:

stack pointer:

**9b.** Is a context-switch different from a system call? If so, explain how and indicate whether any of the four registers listed above are handled differently for a context switch.

**10.** **(8 marks)** Consider a two-level page table similar to the one discussed in class, but where pages are 2-KBytes (and where each level-two page table page is 2-KBytes).

**10a.** Labeling the bits of a virtual and physical addresses from 0 to 31 (0, being the lowest order bit), list which bits store each of the following:

virtual page number (VPN):

page offset:

level-one page-table index:

level-two page-table index:

**10b.** How big is the level-1 page table?

**10c.** Use pseudo code to show how the virtual-memory hardware translates a virtual address into a physical address (be sure to account for the entire process and for every bit of the physical address). Indicate what happens if the virtual address is mapped but the data at that address is not stored in physical memory. Recall that the register `PTBR` stores the physical address of the base of the page table.

**10d.** **[harder]** If the PTE stored at physical address 0x22484 stores page-frame number (PFN) 0x12, the PDE stored at physical address 0x88040 stores PFN 0x44 and the PTBR stores physical address 0x88000, can you determine which virtual address maps to physical address 0x092a8. If so, what is that virtual address. Explain briefly.