CPSC 213, Winter 2009 - Term 2 — Final Review Questions

1 Execution in a Simple Machine

1.1 For each of the following, indicate whether the value is know statically or dynamically by placing the word **static** or **dynamic** next to each.

1.1a The address of a global variable in C:

1.1b The address of an element of a static array in Java:

1.1c The address of an instance variable in Java.

1.1d The offset of an instance variable from the beginning of the object that contains it in Java:

1.1e The address of the code that implements a return statement.

1.1f The address of the instruction that executes immediately after a return statement.

1.1g The value of `&a.i` where `a` is a global variable in the C program with the following declaration.

```
typedef struct { int i; } A;
A a;
```

1.1h The value of `&a->i` where `a` is a global variable in the C program with the following declaration.

```
typedef struct { int i; } A;
A* a;
```

1.1i The address of a local variable.

1.1j The position of a local variable in its activation frame.

1.1k The code address of a static method.

1.1l The code address of an instance method (i.e., a method that is not declared `static`).

1.2 What is the value of i after the following C code executes?

```
int a[10] = {0,1,2,3,4,5,6,7,8,9};
int i = *( &a[3] + 2 + *(a+1) );
```

1.3 Consider the following C snippet. All three variables are global variables.

```
int i;
int a[i];
int * b;
void foo () {
    a[i] = b[3];
}
```

Give the SM213 instructions that implement the statement `a[i] = b[3];`. Give all variables allocated by the compiler some arbitrary address and clearly comment your code.

1.4 Java and C deallocate dynamically created objects (i.e., heap-allocated objects) differently. Briefly describe the approach each uses. Give one significant benefit of each approach.

1.5 What is a *memory leak*? Give an example of a type of program where it could be a big problem. State whether Java solves this problem entirely, somewhat or not at all. Justify your answer.

1.6 Would it be possible to implement a compiler that used the heap, instead of the stack, to store activation frames? Give one important drawback of this approach and explain.

1.7 What is the difference between pc-relative and absolute jumps? Give one advantage of each. Explain.

1.8 Java and C do not allow the size of local variables or instance variables to be determined at run-time. How does this restriction simplify the machine instructions the compiler generates for accessing these variables?

2 Device I/O

2.1 Short Answer.

2.1a Give one advantage and one disadvantage of interrupts. What operating system feature is designed to compensate for this disadvantage. Explain very briefly.

2.1b What are *Programmed IO (PIO)* and *DMA* used for? In what way is DMA better? List one thing PIO can do that DMA can't.

2.2 Give Java pseudo code necessary to add interrupt handling to the Simple Machine. Assume the CPU has the following new registers:

- `isDeviceInterrupting` — 0 iff a device has interrupted
- `interruptType` — number (starting at 0) of the device that has interrupted
- `interruptVectorBase` — base address of the interrupt-transfer jump table

3 Virtual Processors

3.1 Short Answers.

3.1a What is it about device access that lead to the development of the virtual processor?

3.1b Give one key difference between virtual processors (i.e., threads and/or processes) and the physical CPU.

3.1c What is the difference between a process and a thread?

3.2 Use the memory-register-access syntax (e.g.,, `m[a] <= \verbr[n]`—) to list all of the steps necessary to perform a thread switch between the current thread and a thread who's thread-control block's base address is stored in register `r0`. Be sure to save what is necessary about the current thread so that it can be started sometime later. You may layout the thread control block any way you like. Document your assumptions and show your work.

4 Synchronization

4.1 Short Answers.

4.1a Is any special memory hardware needed to implement synchronization in a multi-processor system?

4.1b What is the difference between spin waiting and blocking waiting. Give one advantage of each relative to the other.

4.1c Why is it better to spin-wait reading a lock with a normal read instruction first before attempting to lock the lock with an atomic read-write instruction?

4.1d Do semaphores provide spin- or blocking-based synchronization?

4.2 Give pseudo code of a program that uses semaphores to implement mutual exclusion.

4.3 Give pseudo code of a program that uses semaphores to implement a shared queue with two operations: `enqueue` and `dequeue`. Store the queue in an array. Use semaphores to have `dequeue` block if the queue is empty and to have `enqueue` unblock a waiting `dequeue` when it adds an item to the queue.

5 Virtual Memory

5.1 Short Answers.

5.1a Describe when in the execution of a program virtual addresses are translated into physical addresses?

5.1b What is the difference between a segment and a page?

5.1c Is fragmentation a problem for segment-based systems? For page-based systems? Why or why not? Explain.

5.1d What is a context switch? How is it like or unlike a thread switch? Are they the same speed? If not, which is faster?

5.2 Describe the tradeoff (i.e., pros and cons) associated with the three alternative VM mapping data structures we discussed in class (i..e, single segment, multiple segments and pages).

5.3 Consider the execution of the following two SM213 instructions in a system with a 32-bit virtual address space, 4-KB pages and the type of linear page table we discussed in class.

```
ld $0x4008, r0
ld (r0), r1
```

The second of these two instructions accesses memory, **using a virtual address** (NOTE: this is different from the simple machine we covered in the first part of the course, which used physical addresses for memory).

The page table for this process looks like this:

index	pfn
0:	4
1:	2
2:	5
3:	3
4:	9
5:	a
6:	b

What is the physical address accessed by this instruction? Show your work.

6 Hardware-Enforced Encapsulation

6.1 Short Answer.

6.1a What is encapsulation? Is it provided by Java? How is it enforced in Java?

6.1b How do user programs set the CPU's `isModeSystem` register to 1. Does this mechanism relate in any way to `public` and `private` access modifiers in Java? If so, how?

6.1c Why do operating systems use hardware-enforced encapsulation instead of relying on language-based encapsulation?

6.1d What is a protected call?

6.1e What is the role of the interruptVectorBase in making protected calls? Is it stored in memory? If so, is it stored in memory accessible to a user's program? Why or why not? Explain.

6.2 Give Java or C pseudo code for a protected call. Assume that the procedure to be called is identified by `r0`. Assume that the CPU has these special-purpose registers.

1. `isModeSystem` — 1 iff processor is running in system mode
2. `interruptVectorBase` — base address of the protected-call-transfer jump table

State all other assumptions you make and show your work.