Department of Computer Science
Computer Architecture Qualifying Exam
Fall, 2007

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no electronic aids** are allowed.

Please note the following:

- show your work whenever appropriate. There can be no partial credit unless you show how you derived your answers
- be succinct. You may lose points for facts that, while true, are not relevant to the question at hand
- The questions are equally weighted.

1. A computer system uses a 32 bit virtual address. Its virtual memory system uses a 4K page, and two levels of page table entries. A PTE is four bytes, and a table contains 1024 entries (this is the standard 32-bit Intel virtual memory structure).

   Some program has the following memory map:

Addresses	Contents
`00000000-0000ffff`	Program text
`00010000-00ffffff`	Heap
`0f000000-7fffffff`	Stack

   What is the overhead for the virtual memory tables? *I.e.*, how many pages are used for maintaining translations, and how many are used for the actual program code and data? The question deliberately doesn't consider memory usage by the operating system, as that's shared among all processes.

2. Some computer system uses an MESI snoopy cache protocol, as follows:

   Initially, all memory locations are in state I (for Invalid) in all caches.

   When a processor writes to a location, it enters state M (Modified) in the processor's cache and is invalidated in all other caches.

   When a processor reads a location, it enters state E (Exclusive) in the processor's cache if no other processor had a copy, and state S (Shared) if any other processor had it in state E or S. If it was in state E in another cache, it is changed to state S in that cache.

   (a) If a processor has a location in state M, and another processor reads it, there is more than one possible state transition for the two processors. Identify two different states we could decide to put the location in in the processor that previously held it, and the corresponding state changes for the new processor. Argue that one of these is likely to result in fewer overall memory bus transactions; use the one you believe to be better in part (b) below.

   (b) Suppose variables `x` and `y` are in two different memory lines, but which map to the same cache index (so their indexes are the same, but their tags are different). Assume the cache is direct-mapped.

   Now suppose processors P1, P2, and P3 execute the following code:

   ```
   P1: x = 1;
   P2: y = 2;
   P3: x = x + y;
   ```

   Assuming this occurs in the order listed (so P1 executes its line without interference, then P2 executes its line, and then P3 executes its line), write the memory transactions that take place using the standard *`op(var)val`* notation (this notation is reviewed following the last question on this exam). What are the final states of the variables in the three processors' caches?

3. You need to execute the line of C code

```
awry[i] = awry[i++] + 17;
```

(a) First, compile this code for a computer system with an instruction set that is very similar to Intel's. More specifically, instructions take the form

instr op_1 op_2

where:

`instr` is the instruction's op code

op_1 and op_2 are the operands. They can both be registers, or at most one of them can be a memory location or immediate operand. The first operand is the destination. Assume the registers are named `ax`, `bx`, `cx`, `dx`, etc. Assume you have as many registers as you need.

One of the available addessing modes allows the programmer to specify two registers r_1 and r_2, an offset o, and a multiplication factor f in resolving an address. The effective address calculated by this mode is

$[r_1 + f*r_2 + o]$

I.e., the contents of register r_2 are multiplied by f, added to the contents of r_1, and then added to o; the resulting value is the address of the operand. So, the instruction

```
sub ax, [ax+4*bx+6]
```

will add the contents of `ax` to four times the contents of `bx`, and add six to that. Then it'll look up that address in memory, and subtract its contents from `ax`.

For this exam, you can use the same syntax for this addressing mode as the exam does.

Assume the frame pointer register is named `fx`, and that `awry` is an array of 4-byte integers located at an offset of 36 bytes from the base of the activation record. You can use any register you want to hold `i`.

Use the addressing mode just described for reading and writing elements of `awry`.

(b) Now, suppose your microarchitecture supports three types of micro-instructions:

i. arithmetic instructions of the form

$uinst\ r_1\ r_2\ r_3$

where r_1 is the destination register

ii. immediate arithmetic instructions of the form

$uinst\ r_1\ r_2\ const$

where r_1 is the destination register

iii. load and store instructions of the form

$uinst\ r_1\ r_2$

where either r_1 is written to the address contained in register r_2 by a store instruction, or read from the address contained in register r_2 by a load instruction.

You may assume register `r0` always contains 0.

Translate your code for part (a) into microinstructions. You may assume you have as many hardware registers as you want.

A Memory Operation Notation

This appendix provides a brief review of the memory operation notation being used in this exam.

A.1 Operations

In this notation, a memory operation is represented with the characters *op*(*var*)*val* where

- *op* is the operation: R is used for a read operation, and W is used for a write operation.
- *var* is the variable to be read or written.
- *val* is the value that is read or written.

So, using this notation, W(x)1 means "write the value 1 to the variable x", and R(y)2 means "read the value of y from memory. The value memory returned is 2.

A.2 Sequences of Operations

A sequence of operations performed by a single processor is written horizontally on a line:

```
R(x)0  W(x)1
```

states that a processor first reads the value of x, and obtains a 0. It then writes x to memory again; the value it writes is a 1.

An example of a line of C code which might result in this sequence of memory operations is

```
x = x + 1;
```

As we can see, the addition performed by the processor has been abstracted away: all that is shown in the memory operation notation is the actual memory operations performed.

A.3 Multiprocessor Memory Operations

Finally, if we have several processors performing memory operations, we represent each processor's operations on a line, like this:

P1	R(x)0	W(x)1	
P2	R(x)0		W(x)2

In this example, P1 and P2 simultaneously read x, and both obtain a value of 0. In the second time unit P1 writes a new value to x (a 1), and on the third time unit P2 writes a new value to x (a 2). The final value of x in this example is 2.

It's important to keep in mind that there is nothing saying that this is the only possible sequence of operations – from the information provided, P2 could equally well have written its result first, or it might not have even read x until after P1 had written it. The notation merely specifies what did happen, it *doesn't* say there's any reason it needed to happen that way.

A.4 Extensions

There are a variety of extensions to this notation, to either insert synchronization or to explicitly represent communication. Those are beyond the scope of what we need for this!