Department of Computer Science
Computer Architecture Qualifying Examination
Fall, 2006

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 instructions. There will be a ten point deduction for failure to comply with them:

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

1. (30 points) Assume you have a shared-memory computer system using a MESI (cache states are Modified-exclusive, Exclusive-unmodified, Shared-unmodified, and Invalid) protocol, as described in the Papamarcos and Patel paper. Further, assume that when one processor reads a cache block that another currently has in the Modified-exclusive state, the new processor gets the block in the Exclusive-unmodified state.

   (a) What are the meanings of the four states?
   (b) What is the purpose of the Exclusive-unmodified state (*i.e.*, why does the cache have both the Shared-unmodified and the Exclusive-unmodified state)?

   Suppose variables `x`, `y`, and `z` are all in different memory blocks, but `x` and `y` map to the same cache block (*i.e.* their indexes will be the same, but their tags will be different) while `z` maps to a different cache block (so its index is different from that of `x` and `y`). Further, suppose the processors in the machine execute the following lines of C code, in the order shown (assume no reordering or optimization):

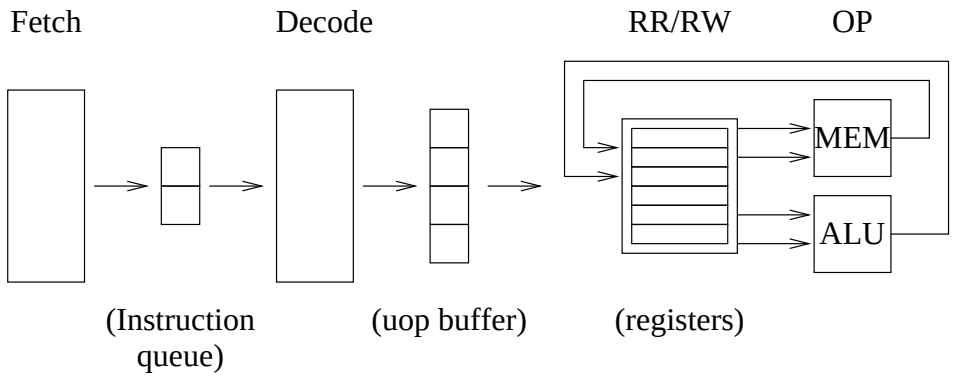
   ```
   P1: x = x + 1;
   P2: y = x;
   P3: z = x + y;
   ```

   (c) Using the *op*`(`*var*`)`*value* (*e.g.* `R(x)1`, `W(y)3`) notation for read and write operations, show the sequence of memory operations caused by this code.
   (d) What will the final cache states be?

2. (30 points) A new computer is being developed which has a 64 bit, byte-addressed, virtual address with a 64K page size. The computer will support a 64 gigabyte physical memory size, and will have a protection scheme with separate bits for writeable, executable, and user-accessible. It will also have an iospace bit in the page table entry, which will specify whether a page is "normal memory" or is being used for memory-mapped IO. This new computer will use an inverted page table.

   (a) Propose a layout for an inverted page table entry. Be sure to justify the sizes of the fields in the entry.
      i. Should a page with the iospace bit set be cached?

3. (40 points) Consider a computer CPU with the following five-stage pipeline:



The five stages are:

Instruction Fetch: the first stage of the pipeline can fetch up to two instructions per cycle into an instruction queue. The machine uses typical RISC-style instructions. For our purposes, the only instructions are

add Rx Ry Rz	Add register R*y* to register R*z*, placing the result in register R*x*.
lod Rx c,Ry	Read memory address *c*+R*y*, and put the contents in register R*x*.
sto Rx c,Ry	Store the contents of register R*x* to memory location *c*+R*y*.

In the above descriptions, R*x*, R*y*, and R*z* can each be any of the four architected registers RA, RB, RC, or RD. *c* is an integer.

An instruction can be fetched into memory if there is an empty slot in the instruction queue for it (there are only two slots in the instruction queue, as shown in the figure). If an instruction is being removed from the queue on a cycle, it is regarded as being available for a new instruction on that same cycle.

Decode and Register Rename: the second stage can take up to two instructions per cycle from the instruction queue, translate them into μops, and place them in the μop buffer. There are four μops:

adr Ri Rj Rk	Add register R*j* to register R*k*, and put the result in register R*i*
adc Ri Rj c	Add *c* to register R*j*, and put the result in register R*i*
ld Ri Rj	Load the contents of the memory location specified by R*j* into register R*i*
st Ri Rj	Store the contents of register R*i* into memory location R*j*

In the above descriptions, R*i*, R*j*, and R*k* are any of the μ-registers, R0 through R5.

When an instruction is decoded, two things happen: first, it is translated into one or two μops, as needed, to accomplish the task of the instruction. Also, the registers specified in the instruction (*i.e.* RA, RB, RC, or RD) are translated into μ-registers, numbered R0 thorugh R5. The destination register has to be newly allocated from μ-registers that aren't currently in use, while the source registers must be translated to previously-allocated μ-registers (as with register renaming in the MIPS R10000, Pentium II, or other similar processors). Note that the translation of some instructions may require allocation of extra μ-registers for storing intermediate results.

An instruction can only be decoded if (1) there is space for as many μops as needed in the μop buffer (there are only four slots in the μop buffer, as shown in the figure), and (2) it is possible to allocate any μ-registers needed. A μop buffer slot is regarded as available on a cycle on which the μop holding the slot is retired; a μ-register is regarded as available on the cycle when the last μop using the μ-register is retired.

The first two stages of the pipeline occur in-order. Once the translated μops have been placed in the μop buffer they can be processed out-of-order (though retirement will be in-order).

Register Read: up to two μops per cycle can be taken from the μop buffer, and up to four registers can be read. A μop can be executed whenever all of its operands are ready. A register can be read on the same cycle it is written, and the new result will be read.

Operate: up to two μops per cycle can be executed: one can be an ALU operation, and one can be a memory read or write.

Register Write: up to two μ-registers can be written per cycle. When a μop has executed, and all of its preceding μops have been retired, it can be retired (and removed from the μop buffer). It can be retired on the cycle on which these conditions are met (for instance, on the same cycle that its result is written).

Now, consider the following program:

```
lod RA 5,RB
lod RB 6,RC
add RD RA,RB
sto RD 4,RA
```

Initially, register RB is renamed to register R0, and register RC is renamed to register R1.

(a) Translate this code into μops.

(b) Draw a Gantt (timing) chart showing how this code is executed by the pipeline.