

**Department of Computer Science**  
**Computer Architecture Qualifying Exam**  
**Fall, 2003**

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 I see how answers were arrived
  - be succinct. You may lose points for facts that, while true, are not relevant to the question at hand
1. (30 points) Some computer system has a 63 (not 64!) bit virtual address space, an 8K byte page size, and an eight byte page table entry size (if the system has a multilevel paging system, the entries at all other levels will be eight bytes, too).

Also, assume it has a 64 entry, 2-way set-associative translation lookaside buffer (TLB) using LRU replacement.

- (a) Explain why it would be reasonable for this system to use a five level page table hierarchy. How would a virtual address be broken up into six fields to perform a virtual memory translation?
- (b) How would a virtual address be broken up into three fields to perform a lookup in the TLB?
- (c) Here's a short loop taken from a C program:

```
for (i = 0; i < 1024; i++)
    a[i] = b[i]*c[i];
```

The arrays begin at the following virtual addresses:

```
a: 0x000000001a37bc900000
b: 0x000000006954ad100000
c: 0x00000000aac983200000
```

Describe the TLB's behavior during one iteration of this loop (you may pick a specific iteration if you like; the behavior should be essentially the same for all of them).

2. (15 points) A four-processor, bus-based shared-memory computer uses an invalidate-based MSI snoopy cache coherence strategy to implement serial consistency. Variables X and Y are in a single cache block, while Z is in a separate block. Initially, all caches are empty. The four processors perform the following operations.

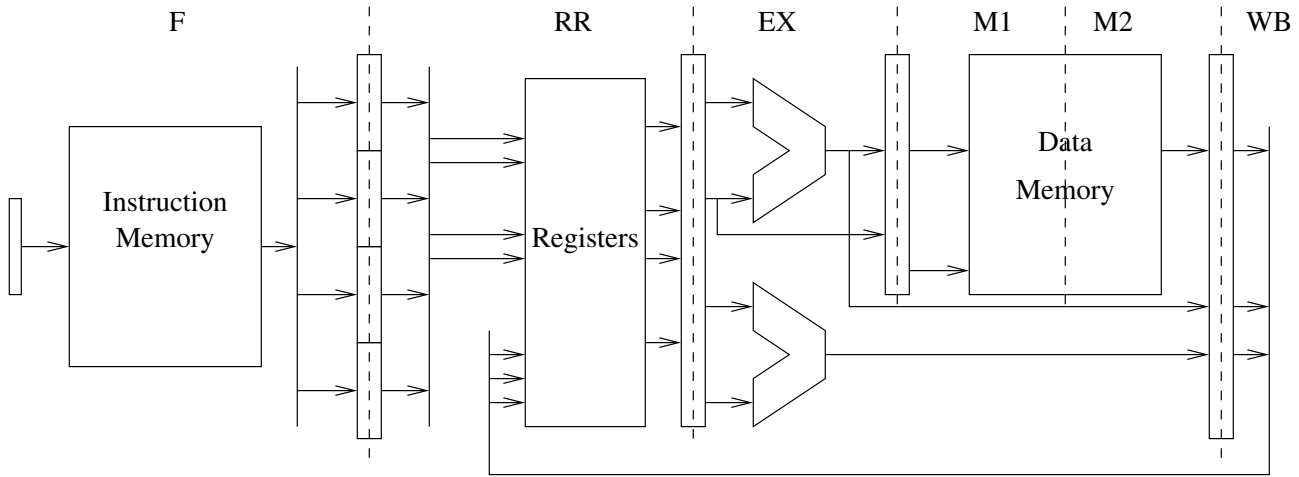
P1: W(X)1		R(Z)
P2:	W(Y)2	
P3:		R(Z)
P4:	W(Z)3	

At the conclusion of this sequence of operations, which processors have copies of the variables, what states are they in, and what are their values?

3. (25 points) The following 13 bit number consists of an eight bit value, four check bits, and a parity bit in the order M8 M7 M6 M5 C8 M4 M3 M2 C4 M1 C2 C1 P: 0010100101111.

Is it correct? If it's incorrect, does it have a one-bit or a two-bit error? If it's a one-bit error, correct it.

4. (30 points) A standard RISC-like (three-operand load-store; similar to RISC, MIPS, SPARC, or Alpha) processor is implemented using out-of-order, superscalar techniques. The following figure is a block diagram of its pipelines.



The stages are:

- F is the instruction fetch stage. It tries to keep an instruction buffer (the F/RR pipeline register) full; it can fetch up to four instructions at a time to accomplish this. Instructions are fetched from the instruction memory in-order.
- RR is the register read and instruction decode stage. Up to two instructions at a time can be read from the instruction buffer into this stage. Instructions are read out of this buffer out-of-order; an instruction can be read whenever its operands are ready and there are no structural hazards preventing it from proceeding through the pipeline. Instructions are removed from the instruction buffer as soon as they are transferred to the RR stage (we aren't going to worry about trying to maintain in-order retirement).
- EX is the execute stage. There are two ALUs; the top ALU can handle any instruction, while the bottom ALU can only handle arithmetic instructions. There is no requirement that the top data path has to handle instructions occurring earlier in the instruction stream than the bottom data path.
- M1 and M2 are the data memory stages. `load` and `store` instructions must pass through these stages; the data memory isn't actually pipelined, so only one instruction at a time can be in either of these stages. An otherwise-ready instruction can't be issued until it will be able to enter the M1 stage on time (this is the structural hazard alluded to earlier). An arithmetic instruction can catch up to and "pass" a `load` or `store`. An arithmetic instruction in the top pipeline does **not** need to go through these stages.
- The WB stage writes the results of instructions back to the registers. It is possible to write the results of any number of instructions back at once.

The figure doesn't attempt to show most of the forwarding that is available; you can assume any possible forwarding between instructions as convenient.

The assembly language below follows the convention that the first operand of arithmetic instructions is the destination register.

Draw a Gantt (timing) chart showing the execution of the following code on this processor. Draw arrows showing forwarding; be sure you show the forwarding as occurring between the correct stages.

```
load $1, 100($2)
add $3, $2, $1
sub $7, $8, $9
store $3, 200($7)
store $11, 100($0)
add $9, $10, $11
```