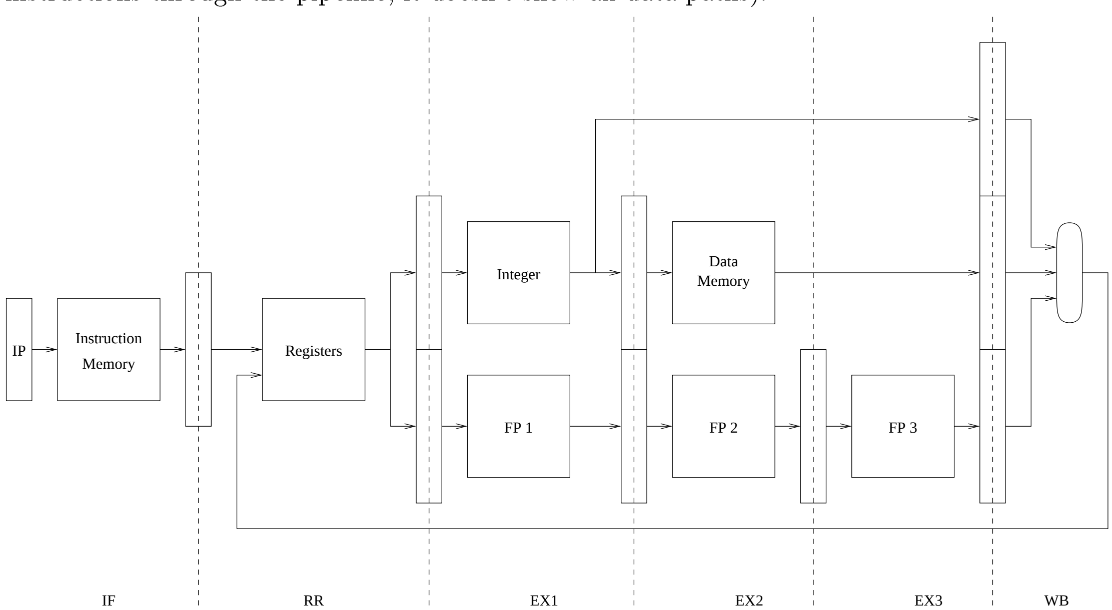
Department of Computer Science
Computer Architecture Qualifying Exam
August, 2002

The following exam is open book and open notes. You may feel free to use whatever additional reference material you wish, but **no calculators** are allowed. Please note the following instructions:

- show your work. The reasoning you use to get your answers is far more important than the answers themselves; an answer with no justification is **wrong**.
- be succinct. I will take off points for facts that, while true, are not relevant to the question at hand

1. (30 points) For this question, assume a pipelined processor with the following characteristics:

   - It is a three-operand processor somewhat similar to a MIPS, except that it is capable of performing either integer or floating point operations, and the pipeline length varies according to the operation to be performed. Here's a highly schematic picture of the pipeline (it only shows the flow of instructions through the pipeline; it doesn't show all data paths):



   - There are three possible paths through the pipeline:
     – An integer instruction will use the IF, RR, EX1, and WB stages (it follows the top path through the pipeline).

- A memory read or write will use the IF, RR, EX1, EX2, and WB stages (it follows the middle path).
- A floating point instruction will use the IF, RR, EX1, EX2, EX3, and WB stages (it follows the bottom path).

An instruction that does not write to a register needn't use the WB stage.

- Instructions can stall in any of the pipeline registers. The segments of the pipeline registers are independent, so (for instance) an arithmetic instruction can be stalled after the Integer unit while a floating point instruction is proceeding after the FP3 unit.
- The assembly language is likewise similar to the MIPS; instructions take the following form:
  - Integer and floating point instructions:
    aluop result operand1 operand2
    where *result* and *operand1* are the result register and a register containing the first operand, and *operand2* is either a register or a constant. Integer arithmetic instructions all begin with the letter `i` (as in `iadd` for integer addition or `isub` for integer subtraction). Floating point arithmetic instructions all begin with the letter `f` (as in `fadd` for floating point addition or `fsub` for floating point subtraction).
  - Memory instructions:
    memop register1 index(register2)
    where *register1* is either the destination register for a load or the source register for a store, and the memory address is formed by adding the *index* to the contents of *register2*. The memory instructions are `ld` (load) and `st` (store).

(a) Draw arrows to show the dependencies and antidependencies between all the instructions in the following code. Use solid arrows for dependencies, and dashed lines for antidependencies.

```
ld  r1, 100(r2)
ld  r3, 200(r4)
fadd r5, r1, r3
st r5, 300(r2)
iadd r5, r2, r4
st r5, 200(r4)
```

(b) Draw a timing chart showing how this code would be executed on this processor, assuming all necessary forwarding and stalling takes place to honor the dependencies and antidependencies. Use arrows to show any forwarding necessary. *Assume the processor is as aggressive as possible in issuing instructions and moving them down the pipeline, subject to the requirements of dependencies and hazards.*

(c) Write a code fragment that would exhibit a WAW hazard. How would the pipeline have to stall to avoid the hazard?

2. (25 points) Suppose you could improve the hit rate of a cache from 98% to 99%, at the cost of a 10% increase in the time required to perform a cache lookup. How much slower than cache would memory have to be in order for this to be a good idea? Assume a single level of cache.

3. (25 points) What is the strongest memory consistency model for which the following is a valid sequence of memory operations? Assume all variables have an initial value of 0.

```
P1: W(x)1 W(y)2
---------------------------
P2:             R(x)1 R(y)4
---------------------------
P3:             R(x)3 R(y)2
---------------------------
P4: W(x)3 W(y)4
```

Draw a picture of a network topology which would be likely to yield these results for this sequence of memory transactions.

4. (20 points) In the Linux kernel, two mechanisms are used for low-level synchronization depending on whether the kernel has been compiled for a single CPU or for a shared-memory multiprocessor. In the first case, locks are implemented by disabling (to acquire) and enabling (to release) the lock; in the second, a spinlock is used to wait for the lock to become available.

   Why is each of these appropriate for the situation in which it is used?