Name Solution

Computer Architecture
EE 4720
Midterm Examination

Friday, 21 March 2003, 13:40–14:30 CST

Problem 1		(30 pts)
Problem 2		(30 pts)
Problem 3		(40 pts)
Exam Total		(100 pts)

Alias Freedom Quarter?

Good Luck!

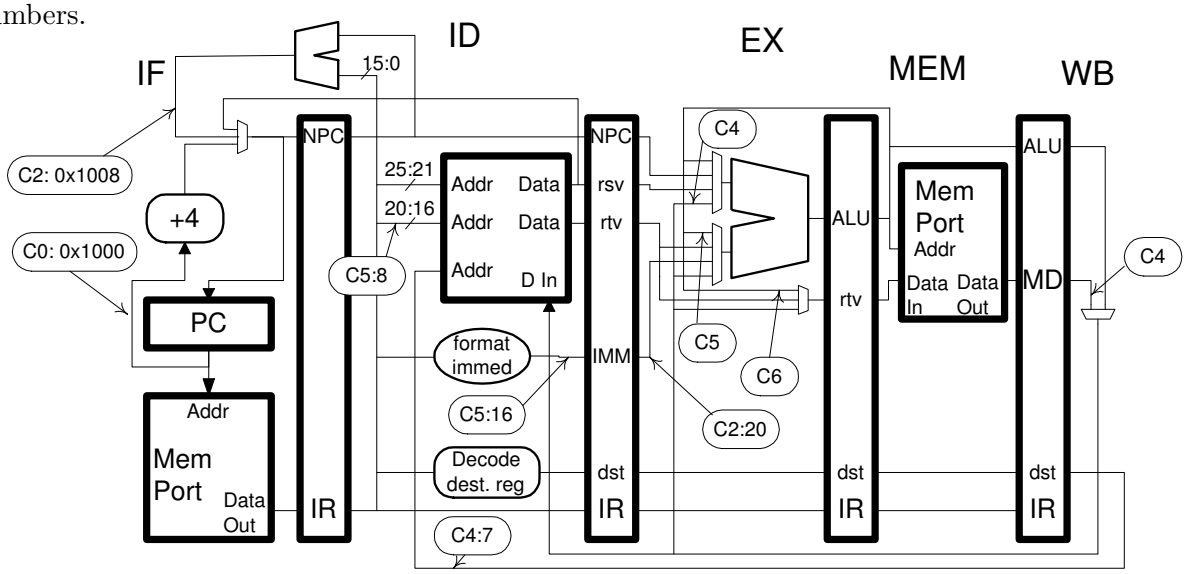
Problem 1: In the diagram below certain wires are labeled with cycle numbers and values that will then be present. For example, C5:8 indicates that at cycle 5 the pointed-to wire will hold an 8. Other wires are labeled just with cycle numbers, indicating that the wire is used at that cycle. There are no stalls during the execution of the code.[30 pts]

☑ Write a program consistent with these labels.

☑ The branch is taken. Use labels for branch targets and label the target line.

☑ Show the address of every instruction.

☑ Fill in every register number that can be determined and use `r10`, `r11`, etc. for other register numbers.



The solution appears below. One unusual feature is the branch. That the instruction in ID at cycle 2 is a branch can be determined by signal on the PC mux, C2:0x1008. The signal indicates that the branch target is `0x1008` which happens to be the delay slot instruction, and so the `add` executes twice, once as the delay slot instruction and once as the target. (The `C0:0x1000` indicates that the first instruction's address is `0x1000`.)

Grading Notes: Most people got this substantially correct. One relatively common error was omitting the branch.

The problem as given did not specify that there was a branch. Also, as given there was nothing to indicate that the branch was taken and so a solution with a not-taken branch was correct.

```
Solution:  Note that the add is executed twice.
       Cycle              0  1  2  3  4  5  6  7  8
0x1000 lw $7, 20($10)     IF ID EX ME WB
0x1004 bgt $11, TARG         IF ID EX ME WB
TARG:
0x1008 add $8, $7, $8           IF ID EX ME WB
0x1008 add $8, $7, $8              IF ID EX ME WB      [sic]
0x100c sw $8, 16($12)                 IF ID EX ME WB
       Cycle              0  1  2  3  4  5  6  7  8
```

Problem 2: The CISC-A ISA is making its world premier in this exam!

- It has 32 integer registers (the MIPS names can be used) and the address size is 32 bits.
- Each operand in each instruction can use any appropriate addressing mode, including register, immediate, and the full range of memory addressing modes described in class.
- Instructions are variable size, the entire first byte is the opcode.

(*a*) Re-write the code below in CISC-A, making up appropriate instructions as needed. [10 pts]

☑ Take advantage of CISC-A's characteristics to reduce code size within the loop (primary goal) and outside the loop (secondary goal). (See the next problem.)

☑ Take advantage of: the available *addressing modes* (memory, register, and immediate), the *variable instruction size*, and *operand flexibility*.

☑ A correct solution includes at least three big-improvement-over-MIPS instructions.

☑ Identify non-MIPS addressing modes used.

```
 lui $t0, 0x1234
 ori $t0, $t0, 0x5678
 lw $t2, 0($s0)
 lw $t2, 0($t2)
 addi $t3, $t0, 0x1000
LOOP:
 lw $t1, 0($t0)
 addi $t0, $t0, 4
 bne $t0, $t3  LOOP
 add $t2, $t2, $t1
```

```
 mov $t0, 0x12344578   # Uses 32-bit immediate.
 mov $t2, @($s0)       # Uses memory-indirect addressing.
 add $t3, $t0, 0x1000
LOOP:
 add $t2, $t2, ($t0)+  # Uses autoincrement addressing in an arithmetic insn.
 bneq $t0,$t3 LOOP     # No delayed branches.
```

(*b*) Describe a possible coding for three of the CISC-A instruction used above. Don't choose three similar instructions. [10 pts]

☑ It should be obvious (to a computer engineer) that the full range of operands is available.

☑ The coding must follow the • bulleted • points above.

☑ Instruction size should be small, though not the absolute minimum size.

Coding:

Every instruction consists of an opcode byte followed by zero or more sets of operand bytes. Each set of operand bytes specifies information about an operand (destination or one of the sources).

```
opcode OPINFO1 [OPEXT1] [IMMED1] OPINFO2 [OPEXT2] [IMMED2] ...
```

The first operand byte is OPINFO which is split into a 3-bit MODE field and a 5-bit NUM field. The MODE field specifies the addressing mode (see table) and the NUM might specify a register number, an immediate size, or something else. The OPEXT field is only used by some addressing modes as an extension of the MODE field (since the MODE field is only 3 bits it cannot specify all the addressing modes needed). The IMMED field, if present, holds an immediate, with the size specified in the size specified in OPINFO and possibly OPEXT.

```
MODE field values.
  0:  Register, NUM field holds register number.
  1:  Immediate, NUM field holds size and padding of immediate.
      The immediate itself is in the next NUM bytes.
  2:  Register indirect, NUM holds register number.
  3:  Register indirect autoincrement, NUM holds register number.
  4:  Register indirect autodecrement, NUM holds register number.
  5:  Memory indirect, NUM holds register number.
  6:  Direct, address in next four bytes.
  7:  Additional modes, NUM has mode details.
```

Problem 2, continued: Consider a second ISA, CISC-B, which differs from CISC-A in the following RISCy ways:

- Only load and store instructions can access memory.
- Each instruction has a fixed set of operand types, for example, an `addi` might be limited to one register destination, one register source and one immediate source.
- CISC-B still has variable-size instructions.

(*c*) Show how one such instruction might be coded. Try to choose a good example from the problem above. [5 pts]

☑ The coding should reflect and exploit CISC-B's operand restrictions.

Since the operand types are fixed there is no need for a MODE field. With the field omitted it is possible to specify three register operands in two bytes, rather than three bytes if each register number had to be accompanied by a MODE field.

(*d*) Show two program fragments: [5 pts]

☑ Show a program fragment that would be smaller in CISC-B than CISC-A. (No more than four instructions.)

```
add $t0, $t1, $t2
# CISC-A Coding: opcode opinfo1 opinfo2 opinfo3 (Four bytes)
# CISC-B Coding: opcode rd-rs1-rs2 (Three bytes, rd-rs1-rs2 is two bytes.)
```

☑ Show a program fragment that would be larger in CISC-B than CISC-A. (No more than four instructions.)

```
CISC-A:
 add $t0, $t1, ($t2)   # Four bytes.
CISC-B:
 load $t3, ($t2)       # Three bytes.
 add $t0, $t1, $t3     # Three bytes.  (Total six bytes.)
```

Problem 3: Answer each question below.

(*a*) Answer the following optimization questions. [10 pts]

☑ Explain the difference between front-end and back-end optimizations.

Back-end optimizations are performed at the machine-language level or close to it. Front-end optimizations are performed closer to the high-level language level.

☑ Can typical front-end optimizations be performed by the back end? Explain using an example.

Yes. For example, dead-code elimination. The optimizer would remove instructions that write registers that are never read (before being written again).

☑ Can typical back-end optimizations be performed by the front end? Explain using an example.

No. Many of these optimizations are based on machine instructions, or something close to it. For example, instruction scheduling. Since machine instructions are chosen after front-end optimization is done, there is no way for the front end to schedule (re-arrange) instructions.

(*b*) It is important to computer manufacturers to have high SPEC benchmark ratings. For each technique of improving SPEC ratings, describe whether it will work, and if it won't describe why not. [10 pts]

☑ Try to have a favorable benchmark added to the suite by sending the name of the benchmark and a little something for their trouble (a bribe) to morally weak SPEC members. *In the exam as given the sentence started "Have a favorable . . .".*

Some SPEC members work for the competition so they would not be easy to bribe. Could Compaq bribe Dell into making poor computers? If bribery did work the system would be corrupt and of little value because SPEC results could not be trusted. Without an accepted and trusted way of measuring performance, manufacturers would have little incentive to improve it. Societies that cannot control bribery will be doomed to non-competitive industries.

☑ Modify the source code to the benchmarks, honestly improving performance on your system.

It will not work because SPEC rules prohibit modifying source code.

☑ Modify your compiler so that it honestly produces faster benchmark executables.

This can be done, as long as the modified compiler is made available to the public (for free or as an ordinary product). If a company honestly improves its compiler it has every reason to make it available to their customers.

Grading Note: A common mistake was confusing compiler optimization switches with modifying the compiler itself.

☑ Report results with certain benchmarks omitted.

Not allowed. All benchmarks must be used.

(*c*) The typical ISA has one-, two-, four-, and eight-byte integers. [10 pts]

☑ Explain a possible benefit of having five-byte integers.

Smaller data size.

Grading Note: Many people gave the answer as smaller program size. This would be true to a small extent because of constant data, which could be five bytes, but the much larger benefit for most programs using the data type would be the reduction in space taken up by the data written to memory while the program is running.

☑ Explain a difficulty with adding five-byte integers, taking in to account a certain feature (some would call it a restriction) of many RISC ISAs. *Hint: The name of the feature begins with an "a".*

The a-word is alignment. If the memory system can only provide power-of-two aligned accesses then there would be no reasonable way of storing arrays of five-byte integers. If the first element of an array of five-byte items was stored in an aligned address the second one would not be.

(*d*) Provide the requested code examples and answer the questions. Two instructions suffice. [10 pts]

☑ Show code having a data dependency; circle the registers carrying the dependency.

```
add $t0, $t1, $t2  # Register t0 carries the dependency.
sub $t4, $t0, $t5
```

☑ What is the name of the corresponding hazard?

RAW hazard.

☑ Show code having an anti dependency; circle the registers carrying the dependency.

```
sub $t4, $t0, $t5  # Register t0 carries the dependency.
add $t0, $t1, $t2
```

☑ What is the name of the corresponding hazard?

WAR hazard.