Name Solution

Computer Architecture
EE 4720
Midterm Examination

Friday, 31 October 2008, 10:40–11:30 CDT

Problem 1		(50 pts)
Problem 2		(10 pts)
Problem 3		(20 pts)
Problem 4		(20 pts)
Exam Total		(100 pts)

Alias Well ARMed

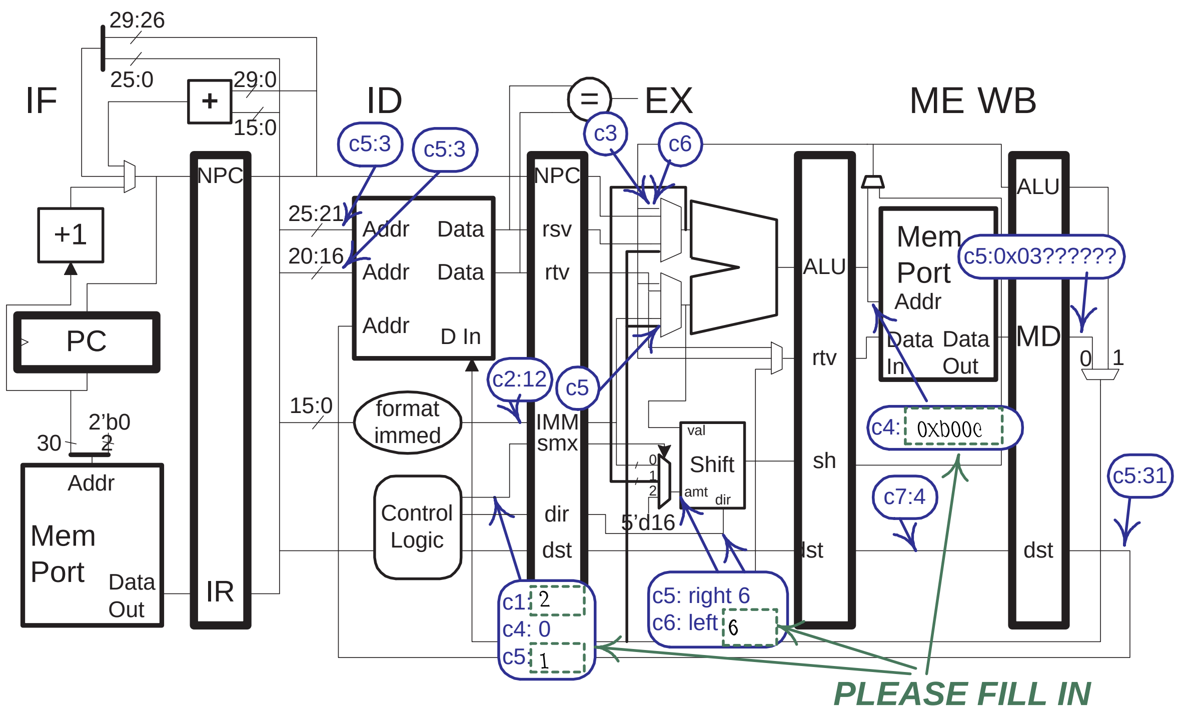
Good Luck!

Problem 1: In the MIPS implementation below some wires are labeled with cycle numbers and values that will then be present. For example, $\boxed{\text{c5:3}}$ indicates that at cycle 5 the wire will hold a 3. Other wires are labeled just with cycle numbers, indicating that the wire is used at that cycle. If a value on any labeled wire is changed the code would execute incorrectly. Instruction addresses and the first instruction have been provided. [50 pts]

(a) Finish a program consistent with these labels.

✓ All register numbers and immediate values can be determined.

✓ Be sure to fill the **three** blocks marked *PLEASE FILL IN*.



```
# SOLUTION
#  Cycle                    0  1  2  3  4  5  6  7  8
0xb0000 lui r2, 0xb         IF ID EX ME WB
0xb0004 lw r31, 12(r2)         IF ID EX ME WB
0xb0008 j TARG                    IF ID EX ME WB
0xb000c srl r3, r31, 6               IF ID EX ME WB
0xe0000 sllv r4, r3, r3                 IF ID EX ME WB
#  Cycle                    0  1  2  3  4  5  6  7  8
```

Solution discussion on next page.

0xb0004: The $\boxed{\text{c4}}$ item in ME indicates that this is a load or store, the $\boxed{\text{c5:0x03??????}}$ in WB reveals that it must be a load and because of the size of the value, it must be a load word. The destination register is directly given (the $\boxed{\text{c5:31}}$ in WB), the bypass with **lui** reveals the **rs** register. The address can be calculated using the known immediate values (12 from **lw** itself and 0xb from **lui**). Note that the load address matches the address of the fourth instruction.

0xb0008: See the next part for a discussion of this instruction.

0xb000c: The identity of the instruction is revealed by the shifter control signals, the dependence with the fifth instruction reveals the destination register.

0xedd900: The control signals reveal the type of instruction. The interesting part is determining the shift amount for the $\boxed{\text{C6:left...}}$: The **lw** instruction loads the fourth instruction into **r31**. The fourth instruction shifts the value (itself) right by six bits, putting the **sa** field into the least significant bits of **r3**. We know from $\boxed{\text{c5:right 6}}$ that the **sa** field of the fourth instruction is 6, so the least significant bits of **r3** must be a 6, and that is what is used for the shift amount in this last instruction: $\boxed{\text{c6:left 6}}$.

Problem 1, continued: Continue referring to the implementation on the previous page.

(*b*) Explain whether each instruction below could be the instruction at address 0xb0008 (on the previous page). If it could be the instruction write "Possible" otherwise write "Impossible because ..." (The grade will be based on the reason.)

☑ add

Could not be. Since the address of the fifth (and last) instruction, 0xedd900, is not the address of the fourth instruction, 0x0b000c, plus 4 the third instruction must be some kind of a control transfer. Therefore it can't be an add.

☑ j

It could be. As explained above, the third instruction has to be some kind of control transfer instruction, and there's nothing to rule out a j.

☑ jal

Could not be. The jal instruction always writes the return address to r31. A bypass path is used to send the value of r31 written by the second instruction to the fourth instruction. If the third instruction also wrote r31 that bypass path would not be used, and so the third instruction can't be a jal.

☑ beq

Could not be. Yes, a beq is a control transfer, but its range is limited to $\pm 2^{15}$ instructions, so it could not reach address 0xedd900 from 0x0b0008.

Problem 2: [10 pts] Based on the experience of preparing SPECcpu benchmark runs manufacturers A and B each release a new compiler that improves their respective SPECcpu scores.

(a) In the course of preparing a SPECcpu benchmark run manufacturer A discovers a new optimization technique that improves the performance of most of the SPECcpu programs, and other programs. This optimization technique is added to the compiler for use at the `-O1` and higher optimization levels. The manufacturer sells the compiler, proudly boasting about the performance benefits.

☑ Is it in the spirit of the SPECcpu rules to use this new optimization technique for the base results? Explain.

Yes. The only difference between it and other optimizations is that it was invented in the process of running the SPECcpu benchmarks.

(b) To prepare a SPECcpu benchmark run manufacturer B has its best programmers prepare hand-written assembly code for the most time consuming portion of each benchmark. The compiler will recognize each benchmark based on the source code and substitute the hand-written routines where needed; the hand-written code will only work for these benchmarks.

These optimizations are included in manufacturer B's compiler and used at `-O1` and higher levels. Manufacturer B sells this new compiler.

☑ Is it in the spirit of the SPECcpu rules to use this new optimization technique for the peak results? Explain.

It is not in the spirit of peak rules because the score does not reflect the performance that users can obtain on similar programs. Re-writing benchmark code is against the SPECcpu rules and that is what this testing procedure effectively does. It is likely that this technique might violate the letter of the rules too.

☑ Explain why it is not in the spirit of SPECcpu rules to use such optimizations for base results.

Since it's not in the spirit of the peak rules it can't be in the spirit of the stricter base rules either. One reason that applies to base but not peak: The amount of effort needed to produce the hand-optimized code is well beyond base intent, and so it's not in the spirit of base rules.

☑ Assuming that the compiler can not be reverse engineered (there is no way to inspect the compiler itself) and assuming that manufacturer B can keep secrets, how might this cheating be discovered?

The performance on similar programs would be disappointing in comparison to the SPECcpu benchmarks under peak rules. In fact, one might change a few lines of the SPECcpu benchmarks and see a large performance drop.

Problem 3: Answer the following ISA questions.

(*a*) [10 pts] A RISC advocate claims that by having fixed-length instructions and alignment restrictions a branch can reach twice as far for a given displacement field size than would be possible in CISC ISAs.

☑ Explain why.

RISC instruction addresses (in most RISC ISAs) are multiples of 4 (since the size is four bytes and the alignment feature imposes the multiple of 4 restriction). Therefore the displacement can be the number of instructions to skip. In variable-instruction-length ISAs the displacement would have to be the number of bytes to skip and because most instructions are more than one byte a given displacement size can not go as far.

A CISC advocate responds that branches in CISC programs would take less space anyway.

A primary feature of CISC ISAs is variable instruction size. Among other things this allows instructions' immediate size to match their needs. So, for a small displacement control-transfer there might be a branch instruction with a one-byte immediate field and for a large displacement control transfer there might be a branch instruction with a four-byte immediate field.

☑ Provide a reason for small-displacement branches.

A small displacement branch instruction could be encoded in two bytes, half the size of any MIPS branch.

☑ Provide a reason for large-displacement branches.

Consider MIPS. A control transfer to a target more than 2^{15} instructions away would have to be realized using three instructions: two to put the target address in a register, and a `jr` to perform the jump. A fourth instruction would be needed for a conditional control transfer. A CISC ISA might just use a five-byte instruction.

(*b*) [10 pts] Indicate whether each item below is usually an ISA feature or an implementation feature.

Grading Note: Easy. Maybe too easy.

☑ Number of bits in immediate.

ISA.

☑ Clock frequency.

Implementation.

☑ Number of branch delay slots (if any).

ISA, though chosen with certain implementations in mind.

☑ Floating-point format.

ISA.

☑ Minimum distance between load instruction and a dependent instruction to avoid a stall.

Implementation.

Problem 4: Equivalent MIPS and SPARC code fragments appear below.

(*a*) [10 pts] Taking advantage of SPARC's condition code features, modify the SPARC code to use one fewer instruction (without changing what it does).

```
# MIPS Branch Example
 addi $t1, $t1, -1
 bne  $t1, 0  LOOP
 add  $t2, $t2, $t3
...

! Equivalent SPARC Branch Example
 add l1, -1, l1      ! l1 = l1 - 1
 subcc l1, 0, g0     ! g0 = l1 - 0
 bne LOOP
 add l2, l3, l2
...
```

There is no need to use `subcc` to set the condition code bits since the `add` writes register `l1` and it could also set the condition code bits. Do that using an `addcc` instead of an `add`:

```
! SOLUTION: Modified SPARC code.
 addcc l1, -1, l1      ! l1 = l1 - 1
 bne LOOP
 add l2, l3, l2
...
```

(*b*) [10 pts] For branch-in-ID implementations, why might it be possible to attain higher clock frequencies for condition-code ISAs, like SPARC, than for register-test branche ISAs, like MIPS.

☑ Condition code performance advantage.

The branch logic in a SPARC implementation only needs to examine four bits (the condition code), in MIPS-32 implementations two 32-bit values need to be compared.