Name Solution

Computer Architecture
EE 4720
Midterm Examination

Friday, 20 March 2015, 9:30–10:20 CDT

Problem 1		(25 pts)
Problem 2		(25 pts)
Problem 3		(15 pts)
Problem 4		(10 pts)
Problem 5		(10 pts)
Problem 6		(15 pts)
Exam Total		(100 pts)

Alias The Implementation Game

Good Luck!

Problem 1: [25 pts] Appearing on the next page is the *HW 3 Implementation*, one of the solutions to Homework 3, Problem 2, in which the `bgt` instruction resolves in `EX`. In the HW 3 Implementation there is a 1-cycle branch penalty when `bgt` is taken. (The branch penalty is the number of squashed instructions.) Suppose that based on benchmark analyses we find that `bgt` is mostly taken. For that we would like a New Implementation [tm] in which there is no penalty when `bgt` is taken, and a 1-cycle penalty when it's not taken. The PEDs below show execution examples for the HW3 and New implementations.

```
 # Cycle            0  1  2  3  4  5  6  7   HW 3 Impl, bgt taken, 1-cyc penalty
 bgt r1, r2  TARG   IF ID EX ME WB
 xor r3, r4, r5        IF ID EX ME WB
 or r6, r7, r8            IF IDx
TARG:
 and r9, r10, r11            IF ID EX ME WB
```

```
 # Cycle            0  1  2  3  4  5  6  7   HW 3 Impl, bgt not taken, no penalty
 bgt r1, r2  TARG   IF ID EX ME WB
 xor r3, r4, r5        IF ID EX ME WB
 or r6, r7, r8            IF ID EX ME WB
TARG:
 and r9, r10, r11
```

```
 # Cycle            0  1  2  3  4  5  6  7   New Impl, bgt taken, no penalty
 bgt r1, r2  TARG   IF ID EX ME WB
 xor r3, r4, r5        IF ID EX ME WB
 or r6, r7, r8
TARG:
 and r9, r10, r11         IF ID EX ME WB
```

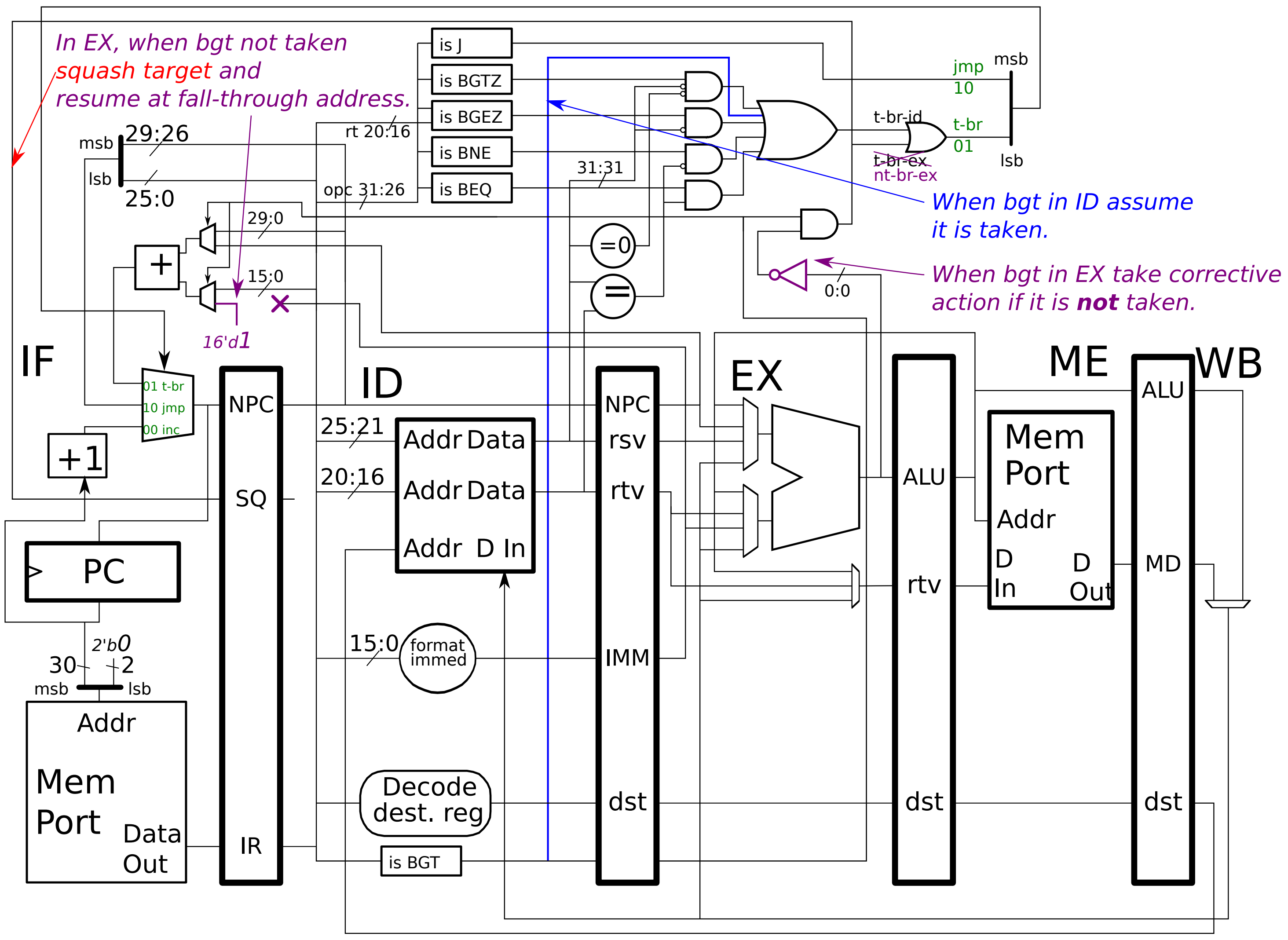
```
 # Cycle            0  1  2  3  4  5  6  7   New Impl, bgt not taken, 1-cyc penalty.
 bgt r1, r2  TARG   IF ID EX ME WB
 xor r3, r4, r5        IF ID EX ME WB
 or r6, r7, r8               IF ID EX ME WB
TARG:
 and r9, r10, r11         IF IDx
 # Cycle            0  1  2  3  4  5  6  7
```

Problem 1, continued: Convert the HW 3 Implementation below into the New Implementation. *Hint: Calm down! A correct solution only requires three minor changes, one of those changes is substituting a mux input with a constant.*

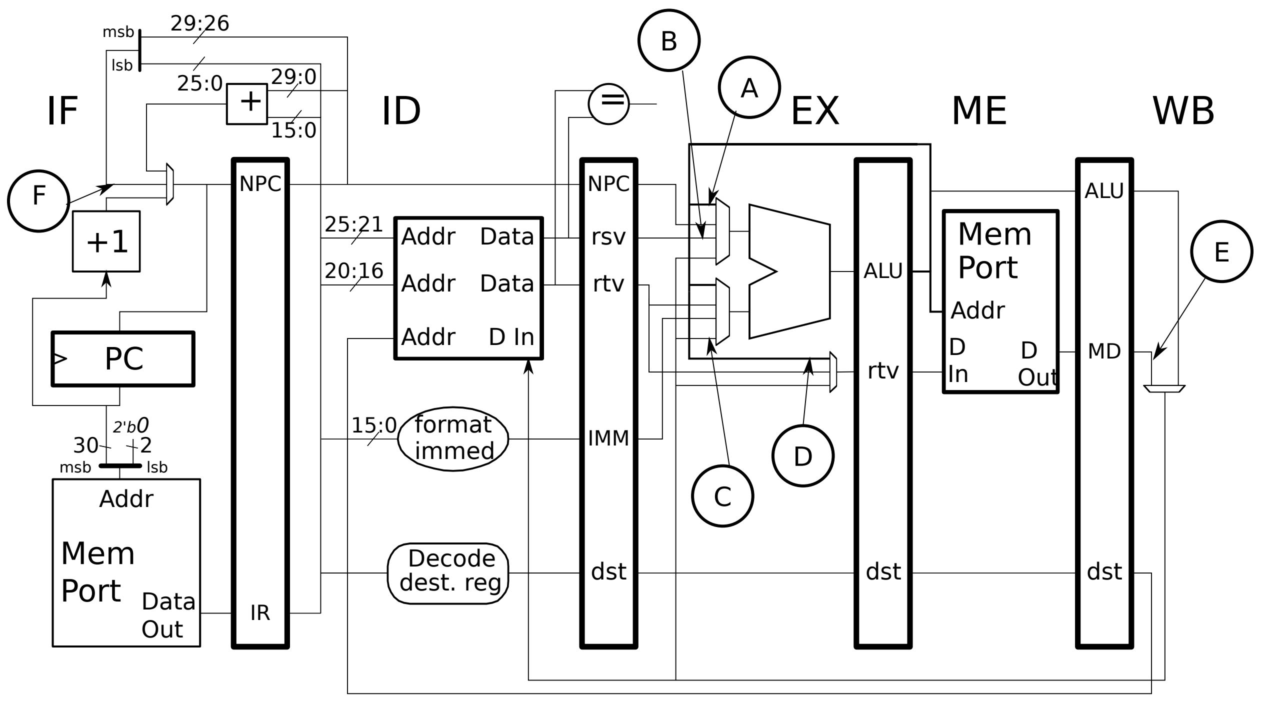
☑ `ID` changes: `bgt` acts like it's always taken.

☑ `EX` changes: if `bgt` resolves not taken, fetch insn after delay slot insn.

Solution appears below.



Problem 2: [25 pts] In the implementation below several multiplexor inputs are labeled. For each labeled input write a program that uses it. A sample solution is provided for **A**.



[✓] Write a code fragment for mux input **A**. [✓] Mux input used in cycle 3, [✓] for register r1.

```
# SAMPLE SOLUTION -- Mux input A.
# Cycle          0  1  2  3  4  5
add r1, r2, r3   IF ID EX ME WB
sub r4, r1, r5      IF ID EX ME WB
```

[✓] Write a code fragment for mux input **B**. [✓] Mux input used in cycle 2, [✓] for register r2.

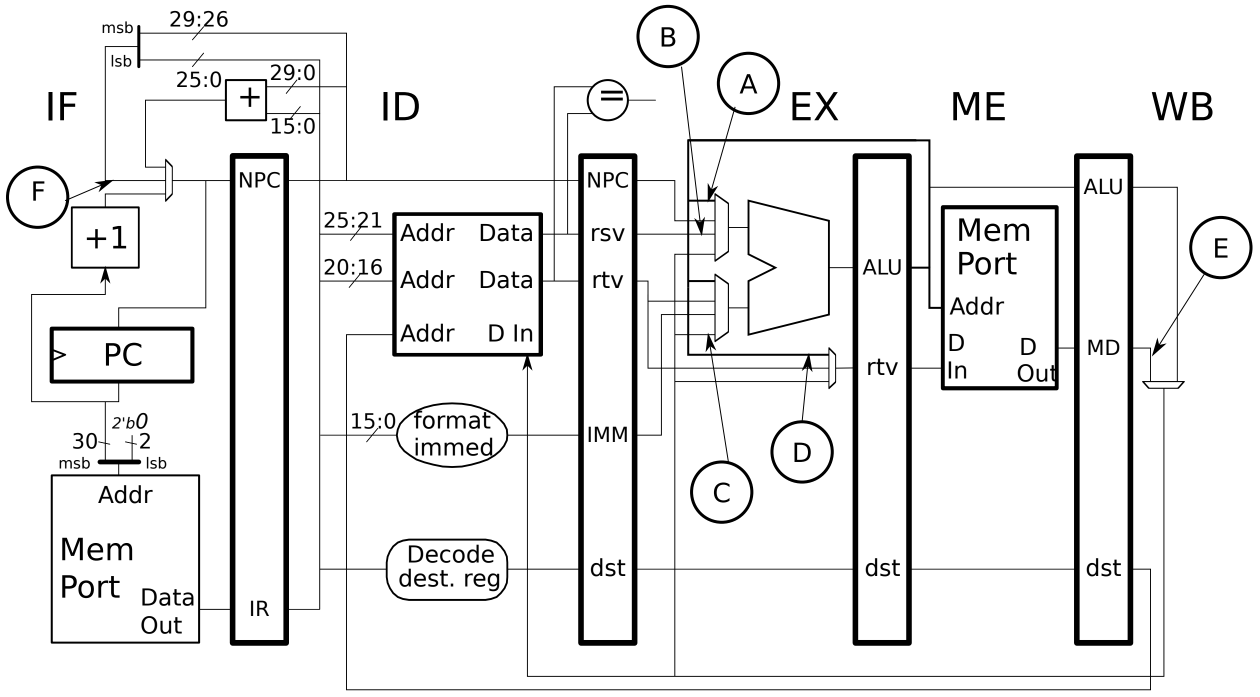
```
# SOLUTION
# Cycle          0  1  2  3  4
add r1, r2, r3   IF ID EX ME WB
```

[✓] Write a code fragment for mux input **C**. [✓] Mux input used in cycle 4, [✓] for register r1.

```
# SOLUTION
# Cycle          0  1  2  3  4  5  6
add r1, r2, r3   IF ID EX ME WB
sub r4, r5, r6      IF ID EX ME WB
or  r7, r8, r1         IF ID EX ME WB
```

Note: To be correct, the `r1` register must be the second source operand of the `or` instruction.

Problem 2, continued:



☑ Write a code fragment for mux input **D**. ☑ Mux input used in cycle 3 , ☑ for register r1 .

```
# SOLUTION
# Cycle          0  1  2  3  4  5
add r1, r2, r3   IF ID EX ME WB
sw r1, 0(r4)        IF ID EX ME WB
```

☑ Write a code fragment for mux input **E**. ☑ Mux input used in cycle 4 , ☑ for register r1 .

```
# SOLUTION
# Cycle          0  1  2  3  4
lw r1, 0(r2)     IF ID EX ME WB
```

☑ Write a code fragment for mux input **F**. ☑ Mux input used in cycle 1 .

```
# SOLUTION
# Cycle          0  1  2  3  4
j FORJOY         IF ID EX ME WB
```

Problem 3: [15 pts] Answer each question below.

(*a*) Show the encoding of the MIPS instructions below. The opcode for `beq` is `0x4` and the opcode for `lw` is `0x23`. *Hint: For the fields' bit positions see the implementation diagrams in previous problems.*

```
TARG:
 lw r1, 2(r3)
 beq r4, r5, TARG
```

☑ Encoding of `lw r1, 2(r3)`:

Solution:

opc	rs	rt	immed
0x23	3	1	2
31 26	25 21	20 16	15 0

☑ Encoding of the `beq` above ☑ with the correct immediate field value.

The solution appears below. The immediate field contains the number of instructions to skip, starting at the delay slot instruction. To jump up to the `lw` we need to skip -2 instructions.

opc	rs	rt	immed
0x4	4	5	-2
31 26	25 21	20 16	15 0

(*b*) Many MIPS Format-R instructions, such as `add` and `sub`, have an opcode value of 0. Explain why they don't have their own opcode field values, such as `0x11` for `add` and `0x12` for `sub`.

☑ R-format instructions have opcode 0 because . . .

. . . they have enough space for an extension of the opcode field, called the `func` field. Assigning opcode 0 to type R instructions leaves lots of opcodes for format-I and -J instructions.

☑ If R-format instructions each had their own opcodes that would be a problem because . . .

. . . there would not be enough opcodes available for format-I and -J instructions. These instructions have immediate fields, the larger the immediate the better, and so it's better to omit any kind of opcode extension field.

Problem 4: [10 pts] Answer each question below.

(*a*) Describe what the dead-code elimination optimization is and provide an example.

☑ Description of dead-code elimination.

In dead code elimination statements (or instructions) are eliminated if they write variables (or registers) that are never used.

☑ Example.

In the example below line L1 is eliminated because the value of x that it writes is never used, we know that because it is overwritten by L2.

```
// SOLUTION EXAMPLE
L1: x = a + b;  // This line is eliminated.
L2: x = c + d;
L3: my_procedure(x);
```

Grading Note: Many examples were much longer than they had to be, for example, showing the π program with π not actually begin printed.

(*b*) For instruction scheduling optimizations is it necessary or just helpful for the the compiler to know the implementation?

☑ Implementation: Necessary, [✓ important], not needed. ☑ Circle one ☑ and explain.

Instruction scheduling is rearranging instructions in order to avoid stalls. The amount by which to separate two true-dependent instructions depends on the bypass paths that are available and (which was not quite covered before the test) the latency of the functional unit. A compiler can make assumptions about which instructions have longer latency and which bypass paths are available. If the assumptions are reasonable then scheduling based on them will be better than not scheduling at all.

Problem 5: [10 pts] Answer each question below.

(*a*) A CISC ISA might have an instruction like `add (r1), r2, 8(r3)`, in which the source and destination come from memory.

✓ Explain why such an instruction is not suitable for a RISC ISA, ✓ refer to RISC goals in your answer.

A goal of a RISC ISA is to enable simple, low-cost pipelined implementations. The instruction above accesses memory twice, once to fetch an operand and once to write the result. It also does arithmetic twice, once to compute the second source operand address, and to add the two operands together. A pipelined implementation would need to adders and worse two data memory ports to execute that instruction, which is too costly for RISC goals. Alternatively an implementation can use a single data memory twice, but such an implementation would not be pipelined. Either way it's not RISC.

(*b*) Describe a feature and goal of VLIW ISAs.

✓ VLIW feature:

Instructions managed in groups called *bundles*. Dependency info placed in *template* fields.

✓ VLIW goal:

Low-cost implementations that can execute more than one instruction per cycle.

Problem 6: [15 pts] Answer each question below.

(*a*) SPECcpu benchmark results have two levels of tuning, *peak* and *base*. In one of these tuning levels the same optimization flags must be used for all benchmarks of the same language.

☑ Which tuning level is this for?

Base.

☑ What is the purpose for requiring the same flags?

The base tuning level is supposed to reflect the amount of tuning performed by a conscientious and skilled programmer for whom performance is one of many things to do. In other words, the programmer also needs to add features and fix bugs, and so cannot waste time getting an extra 0.001% performance. It is assumed that such a programmer will come up with a good set of flags and then use that same set consistently on the different programs he or she compiles.

Note: In class we discussed what might be a better base rule: just have one optimization flag, such as `-O3` or `-fast`.

(*b*) Explain how the following corruptions of SPECcpu would be (we hope) prevented.

☑ *The suite excludes benchmarks that perform poorly on Evil Company's products.* This won't happen because . . .

. . . Evil Company's competitors are also members of SPEC, and they won't sit idly while Evil Company stacks the deck against them. Either they will veto Evil Company's efforts, or else they will quit SPEC and make sure that their names are removed from the list of SPEC members.

Grading Note: Many students misunderstood the question. The question is asking about the design of the SPECcpu suite, it is not asking about a tester running the suite.

☑ *The SPECcpu results for Evil Company's System X are just made-up numbers.* This won't happen because . . .

. . . the SPEC results disclosure must include a config file that would allow anyone to duplicate the test. If the numbers are made up they will quickly be caught and few will believe anything they say in the future.