Name Solution

Computer Architecture
EE 4720
Midterm Examination Two

Wednesday, 19 November 2003, 10:40–11:30 CST

Problem 1		(50 pts)
Problem 2		(50 pts)
Exam Total		(100 pts)

Alias Ouch!!

Good Luck!

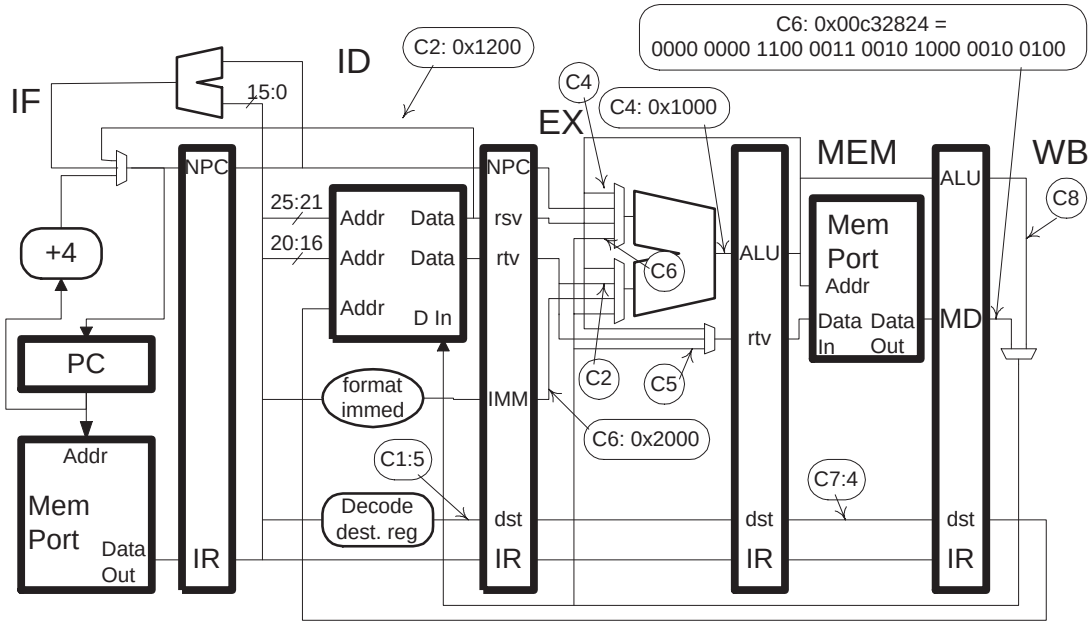
Problem 1: In the diagram below some wires are labeled with cycle numbers and values that will then be present. For example, C6:9 indicates that at cycle 6 the pointed-to wire will hold a 9. Other wires are labeled just with cycle numbers, indicating that the wire is used at that cycle. There are no stalls during the execution of the code.[50 pts]

☑ Write a program consistent with these labels.

☑ Show the address of every instruction.

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

☑ One immediate value and two register numbers are found in interesting ways.



Solution shown below. Upper case in an instruction name indicates something that is known for certain, lower case is for something guessed but consistent with labels. For example, `Sw` means it's definitely a store but the size could be word, or something else (half word, byte).

The interesting part was the load instruction. First of all, it's definitely a load word since the loaded value spans more than 16 bits. The effective address computed by the load happens to be the address of the first instruction, so 0x00c32824 is not any-old data, but the first instruction. To determine the first instruction one has to hand-disassemble 0x00c32824 (which is why it's shown in binary). The zero opcode indicates that it is type R (as does its use of the rt register in the EX stage). The register numbers are easy to parse from its register field. No one was expected to remember the function field value, full credit was given for getting the registers right.

The offset of the load instruction is determined by subtracting `r31`'s contents from the effective address, `0x1000`. Since `r31` is the return address it must be `0x100c`, so the difference is -12.

```
Cycle                   0  1  2  3  4  5  6  7  8
0x1000 ADD R5, R6, R3   IF ID EX ME WB
0x1004 JALR r31, r10       IF ID EX ME WB
0x1008 LW r11, -12(r31)       IF ID EX ME WB
0x1200 Sw r31, 0(r12)            IF ID EX ME WB
0x1204 andI R4, r11, 0x2000         IF ID EX ME WB
```

Problem 2: On the next page is a MIPS pipeline that implements the post-increment load instruction from Homework 4. In the code below the post-increment loads load the data at the address in `r2` into destination register `r1`. They then store the address +4 (the address of the next word) in `r2`.

The implementation on the next page uses a register file with only one write port, just like in Homework 4. The loaded value gets written back in the post-increment load's writeback cycle, the incremented address normally gets written back in the writeback cycle of the next instruction that does not itself perform write-back. That may be the very next instruction, or several instructions later. Or never (see the cases below). [50 pts]

(*a*) Design the control hardware for the five **bold** multiplexors (one in EX, two in WB, and two for the new pipeline latch) so that the implementation correctly executes as many cases below as possible with as few stall cycles as possible. Do not worry about exceptions.

(*b*) Design the logic to generate the stall signal needed for at least one case below and for similar cases. Stall in as few cases as possible.

```
# Pipeline segments provided for convenience.  In at least
# one case stalls will be necessary, AND THOSE STALLS ARE NOT SHOWN.
# In examples focus on r1-r9.

# Case 1:  Ordinary case, use WB of branch.
# Cycle:                0  1  2  3  4  5  6  7
lw r1, (r2)+            IF ID EX ME WB
add r12, r11, r12          IF ID EX ME WB
bne r15,r16 SOMEWHERE         IF ID EX ME WB
add r12, r11, r12                IF ID EX ME WB

# Case 2:  Register r2 not needed.
# Cycle:                0  1  2  3  4  5  6  7
lw r1, (r2)+            IF ID EX ME WB
add r12, r11, r12          IF ID EX ME WB
add r2, r14, r15              IF ID EX ME WB
add r18, r2, r19                 IF ID EX ME WB

# Case 3:  Bypass needed.
# Cycle:                0  1  2  3  4  5  6  7  8  9
lw r1, (r2)+            IF ID EX ME WB
add r12, r11, r12          IF ID EX ME WB
add r12, r11, r12             IF ID EX ME WB
add r12, r2,  r11                IF ID EX ME WB
bne r15,r16 SOMEWHERE               IF ID EX ME WB
add r12, r11, r12                      IF ID EX ME WB

# Case 4:  Notice that base address registers different. (r2, r4)
# Cycle:                0  1  2  3  4  5  6  7  8  9
lw r1, (r2)+            IF ID EX ME WB
lw r3, (r4)+               IF ID EX ME WB
add r12, r11, r12             IF ID EX ME WB
add r12, r11, r12                IF ID EX ME WB
bne r15,r16 SOMEWHERE               IF ID EX ME WB
add r12, r11, r12                      IF ID EX ME WB

# Case 5:  Make sure this works correctly.
# Cycle:                0  1  2  3  4  5  6  7  8
lw r1, (r2)+            IF ID EX ME WB
add r12, r11, r12          IF ID EX ME WB
add r12, r11, r12             IF ID EX ME WB
add r2, r13, r14                 IF ID EX ME WB
add r18, r2, r19                    IF ID EX ME WB
```

Problem 2, continued:

- The control logic can be in any stage, not just ID.
- The logic is relatively simple, it should easily fit on the page.

```
# Case used in solution diagram.
# Case S:
# Cycle:                0  1  2  3  4  5  6  7  8
lw r1, (r2)+            IF ID EX ME WB
add r12, r2, r12           IF ID EX ME WB
bne r15,r16 SOMEWHERE         IF ID EX ME WB
lw r4, (r5)+                     IF ID EX ME WB
```

Solution shown below. Logic for bypass (EX) multiplexor shown in blue, logic for the writeback muxen shown in red, logic controlling the new deferred writeback (DW) multiplexors shown in green, and stall logic is in purple.

A label such as *Case 5, Cycle 6* indicates that the pointed-at wire is used in cycle 6 by the code in case 5 (from the previous page). (That's an example of one time the wire is used, it may be used in other cases and at other times.)