McMaster University
Department of Computing and Software
Dr. W. Kahl

COMP SCI 2GA3, SFWR ENG 3GA3
Sheet 4
2013-11-03

Computer Architecture

Individual solutions to the assignment questions here are due on paper into the dropbox for this course around the corner from ITB-101 on Monday, November 11, before 2:30 p.m.

Exercise 4.1 — Textbook Chapter 4 Exercises

(a) **Read Textbook Chapter 4 up to Section 4.4!**

(b) **For the tutorial, prepare** in particular the following Textbook Exercises:

- Datapath exploration: 4.1, 4.2, 4.6, 4.7, 4.8

(c) **Read Textbook Sections 4.5–4.8!**

(d) **For the tutorial, prepare** in particular the following Textbook Exercises:

- Pipelined datapath exploration: **4.13** (see also AQ 4.6a), 4.14, **4.15**, (4.16), 4.17, 4.19

Exercise 4.2: Floating-Point Representation — Extended from Midterm

Assume that `$s3` contains the base address of array `a`. Consider the following assembly fragment:

```
lui   $t0, 0x64CE
srl   $t1, $t0, 24
addu  $t2, $t1, $s3
sw    $t0, 0($t2)
sll   $t0, $t0, 5
srl   $t1, $t1, 3
addu  $t2, $t2, $t1
sw    $t0, 8($t2)
```

This can be understood as implementing the following pseudocode, with an `int` constant `i` and `j`, and a `float` constant `f` and `g`:

```
a[i] := f;
a[j] := g;
```

Determine the decimal values of the indices `i` and (possibly using decimal fractions $\frac{d_1}{d_2}$, so no calculator is necessary) of the floating-point numbers `f` and `g`.

Document the intermediate states and the bit patterns of the floating point representations of `f` and `g`.

Exercise 4.3

A friend is proposing that the control signal `MemtoReg` be eliminated. The multiplexor that has `MemtoReg` as an input will instead use the control signal `MemRead`. Will your friend's modification work? Consider both the single-cycle and the pipelined datapath.

Name Student Number

Assignment Question 4.4 — Register-Offset Addressing Mode

We wish to add the addressing mode to MIPS that allows offsets to come from registers; for this puspose, we introduce variants of the `lw` and `sw` instructions. Namely,

```
swr  $s1, $s2($s3)
```

stores the contents of register `$s1` into memory at the address obtained from adding the register contents of `$s2` and `$s3`.

```
lwr $s1, $s2($s3)
```

analogously loads the content of memory at the address obtained from adding the register contents of `$s2` and `$s3` into register `$s1`.

Since the `swr` instruction reads from **three** registers, instead of from at most two like all conventional MIPS instructions, it needs significantly more wiring and logic circuitry in and around the register file.

(a) Add both `lwr` and `swr` instructions to the single-cycle datapath described in this textbook chapter 4. Add any necessary datapaths and control signals to the single-cycle datapath of Figure 4.17 on page 322 and show the necessary additions to Figure 4.18 on page 323, reproduced here for convenience.

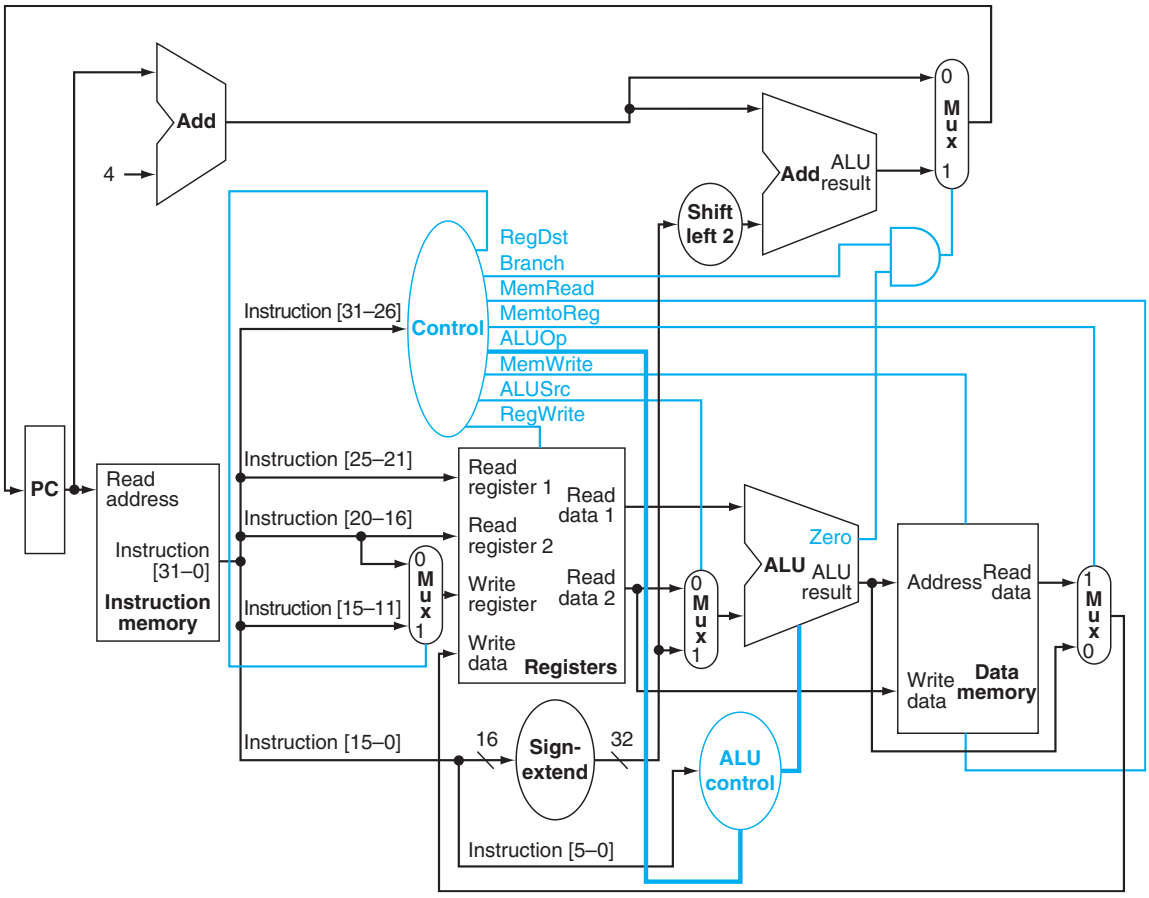

FIGURE 4.17 The simple datapath with the control unit. The input to the control unit is the 6-bit opcode field from the instruction. The outputs of the control unit consist of three 1-bit signals that are used to control multiplexors (RegDst, ALUSrc, and MemtoReg), three signals for controlling reads and writes in the register file and data memory (RegWrite, MemRead, and MemWrite), a 1-bit signal used in determining whether to possibly branch (Branch), and a 2-bit control signal for the ALU (ALUOp). An AND gate is used to combine the branch control signal and the Zero output from the ALU; the AND gate output controls the selection of the next PC. Notice that PCSrc is now a derived signal, rather than one coming directly from the control unit. Thus, we drop the signal name in subsequent figures. Copyright © 2009 Elsevier, Inc. All rights reserved.

Name Student Number

(b) Add both `lwr` and `swr` instructions to the pipelined datapath described in this textbook chapter 4. Add any necessary datapaths and control signals to the single-cycle datapath of Figure 4.51 on page 362, and explain the differences.

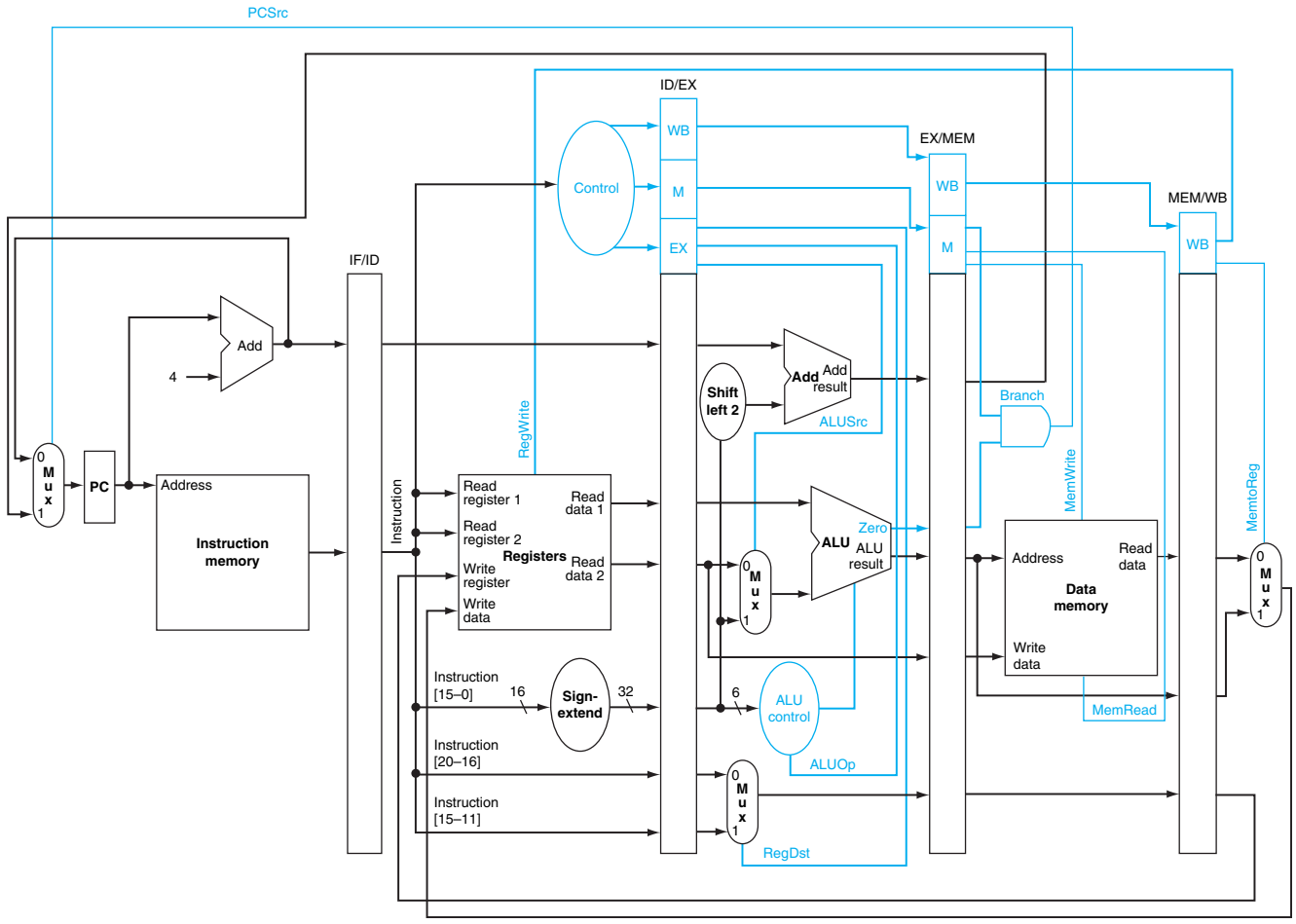

FIGURE 4.51 The pipelined datapath of Figure 4.46, with the control signals connected to the control portions of the pipeline registers. The control values for the last three stages are created during the instruction decode stage and then placed in the ID/EX pipeline register. The control lines for each pipe stage are used, and remaining control lines are then passed to the next pipeline stage. Copyright © 2009 Elsevier, Inc. All rights reserved.

(c) For the pipelined implementation, which new hazards are introduced by the addition of `lwr` and `swr`?

Assignment Question 4.5

This exercise is similar to textbook Exercise 4.8 (p. 414). This time consider the effect of cross-talk, in the following four scenarios:

(a) Cross-talk between RegDest and MemWrite in the single-cycle datapath of Figure 4.17 on page 322, assuming that both lines convey to their outputs the logical AND of their inputs.

(That is, each of RegDest and MemWrite behaves as asserted iff both of them are asserted by the control logic.)

(b) Cross-talk between RegDest and MemWrite in the single-cycle datapath of Figure 4.17 on page 322, assuming that both lines convey to their outputs the logical OR of their inputs.

(c) Cross-talk between last segment and the EX segment of the RegWrite line in the multiple-cycle datapath in Figure 4.51 on page 362, assuming that both segments convey to their outputs the logical AND of their inputs.

(d) Cross-talk between last segment and the EX segment of the RegWrite line in the multiple-cycle datapath in Figure 4.51 on page 362, assuming that both segments convey to their outputs the logical OR of their inputs.

For each scenario, describe the problems arising from this cross-talk fault, and write a MIPS assembly language fragment that runs to completion on a fault-free CPU, and branches to the label `Fault` on a CPU that has only the fault described.

Document if there could be circumstances that under which execution of the faulty machine cannot reliably be detected, or cannot reliably be made to branch to `Fault`.

Assignment Question 4.6

Consider the following sequence of MIPS instructions:

```
I1:    lw $t0, 4($s3)
I2:    addi $t0, $t0, 4
I3:    sw $t1, 0($t0)
I4:    add $t1, $t0, $t1
I5:    lw $t2, 12($s3)
I6:    addi $t2, $t2, -4
I7:    lw $t3, 8($t2)
I8:    sw $s3, 0($t1)
I9:    sw $t1, 0($t3)
```

(a) Indicate dependencies and their type.

(Dependency types are RAR "read after write", WAR "write after read", and WAW "write after write"; each dependency involves one register and two instructions, with the second instruction executed *after* the first.)

(b) Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

(c) Assume there is only ALU-ALU forwarding. Indicate hazards and add `nop` instructions to eliminate them.

(d) Assume there is full forwarding. Indicate hazards and add `nop` instructions to eliminate them.

(e) Assume the following cycle times:

- 250ps without forwarding
- 280ps with only ALU-ALU forwarding
- 300ps with full forwarding

What is the total execution time of this instruction sequence in the three setting of (b), (c), and (d)?

What is the speedup achieved adding full forwarding to a pipeline that had no forwarding?

What is the speedup achieved adding full forwarding to a pipeline that had only ALU-ALU forwarding?

(f) Reorder the instructions in a way that preserves the semantics of the whole instruction sequence, but minimises hazards.

(g) Add `nop` instructions to eliminate all remaining hazards, assuming no forwarding.

(h) What is the total execution time of the instruction sequence of (g) on the machine without forwarding?