All students need to have access to (a) sec. 5.4 from the 1st edition as well as either (b.1) sec. 5.5 of the 3rd edition or (b.2) sec. 5.4 of the 2nd edition. We cover multicycle implementation from the first edition as well as the third edition. The second and the third editions are essentially the same. Lab #4 and questions at the end of lab #4 are based on the 1st edition. However this homework (HW#5b) is based on the 2nd/3rd edition. In the 2nd/3rd edition, the authors have introduced new temporary registers A, B, ALUout, MDR (Memory Data Register). They have removed the TARGET temporary register of the 1st edition as the new ALUout register served the purpose of the TARGET register. Because of the presence of the new temporary registers, unlike in 1st edition design, we do NOT have to KEEP PERFORMING the same operation over several clocks. For example, the ALU computes the memory address in state 2 and drops it off in the ALUout register. In states 3 and 5 (during memory access), the ALU does NOT need to keep computing the memory address as it was "safely" held in the ALUout register. You notice that the control signals of state 2 are NOT repeated in states 3 and 5 of fig. 5.38 in 3rd edition (of fig. 5.42 in 2nd edition).

Exercises for this homework are provided from both 2nd edition and 3rd edition of textbook. The questions are reproduced below.

Please do exercise 5.15 from the 2nd edition and exercises 5.41, 5.43, 5.44, 5.45, 5.17 and 5.18 from 3rd edition.

Please use photocopies of the textbook figures to answer design questions by modifying the figures as necessary. You can either photocopy the necessary figures or download the figures of second edition from the web-site http://www-classes.usc.edu/engr/ee-s/457/COD/ . The two figures needed from the 2nd edition (figures 5.33 and 5.42) are posted at the above location as (Multi_cycle_dp.pdf and Multi_cycle_sd.pdf ). They are password protected and the password is given on the BB). The figures are also given at the end of this homework. The corresponding figures in the 3rd edition are figures 5.28 and 5.37 (COD_3rd_Figure_5.28.jpg and COD_3rd_Figure_5.37.jpg posted above).

**PART I: Adding Instructions to the Datapath**

(2nd Edition):

**5.15 [15] <85.4>** We wish to add the instruction **addi** (add immediate) to the multicycle datapath described in this chapter. Add any necessary datapaths and control signals to the multicycle datapath of Figure 5.33 on page 383 in 2nd edition (fig. 5.28 in 3rd edition) and show the necessary modifications to the finite state machine of Figure 5.42 on page 396 in 2nd edition (fig. 5.38 on page 339 in 3rd edition). You may find it helpful to examine the execution steps shown on pages 385 through 388 in 2nd edition (pages 325 through 329 in 3rd edition) and consider the steps that
will need to be performed to execute the new instruction. You can photocopy existing figures to make it easier to show your modifications. Try to find a solution that minimizes the number of clock cycles required for the new instruction. Please explicitly state how many cycles it takes to execute the new instruction on your modified datapath and finite state machine.

(3rd Edition):

5.41 [15] <ß5.5> This question is similar to previous exercise (Exercise 5.15 of the 2nd edition) except that we wish to add the instruction jal (jump and link), which is described in Chapter 2.

5.43 [15] <ß5.5> This question is similar to Exercise 5.41 except that we wish to add a new instruction, wai (where am I), which puts the instruction’s location (the value of the PC when the instruction was fetched) into a register specified by the rt field of the machine language instruction. Assume that the datapath has not changed and that, as usual, the clock cycle is too short to allow an ALU operation and a register file access in a single clock cycle if one of them is dependent on the results of the other.

5.44 [15] <ß5.5> This question is similar to Exercise 5.41 except that we wish to add a new instruction, jm (jump memory). Its instruction format is similar to that of load word except that the rt field is not used because the data loaded from memory is put in the PC instead of the target register.

5.45 [20] <ß5.5> This question is similar to Exercise 5.41 except that we wish to add support for four-operand arithmetic instructions such as add3, which adds three numbers together instead of two:

\[
add3 \; t5, \; t6, \; t7, \; t8 \quad \# \; t5 = t6 + t7 + t8
\]

Assume that the instruction set is modified by introducing a new instruction format similar to the R-format except that bits [0-4] are used to specify the additional register (we still use rs, rt, and rd) and of course a new opcode is used. Your solution should not rely on adding additional read ports to the register file, nor should a new ALU be used.

PART II:

PART II CANCELLED

(3rd Edition):

5.17 [5] <ß5.5> Describe the effect that a single stuck-at-0 fault (i.e., regardless of what it should be, the signal is always 0) would have on the multiplexors in the multiple-cycle datapath in Figure 5.28 on page 323. Which instructions, if any, would still work?
Consider each of the following faults separately:
RegDst = 0, MemtoReg = 0, IorD = 0, ALUSrcA = 0.

5.18 [5] <ß5.5> This exercise is similar to Exercise 5.17, but this time consider stuck-at-1 faults (the signal is always 1).
Q 5.15 2nd Ed. addi instruction
Q5.41 3rd Ed. jal instruction

- Start
  - Instruction fetch
    - MemRead
      - ALUSrcA = 0
      - ALUSrcB = 0
      - IRWrite
      - ALUOp = 00
      - PCWrite
      - PCSource = 00
    - PCSource = 01
      - ALUSrcA = 1
      - ALUSrcB = 00
      - ALUOp = 10
      - RegDst = 1
      - RegWrite
      - MemtoReg = 0
      - MemRead
        - MemWrite
          - IRWrite
            - ALUSrcA = 1
            - ALUSrcB = 10
            - ALUOp = 00
            - MemWrite
              - ALUSrcA = 0
              - ALUSrcB = 11
              - ALUOp = 00
            - PCWrite
              - PCSource = 10
    - MemRead
      - ALUSrcA = 1
      - ALUSrcB = 10
      - ALUOp = 00

- Memory address computation
  - (Op = 'LW') or (Op = 'SW')

- Execution
  - Branch completion
    - (Op = 'R-type')
  - (Op = 'BEQ')

- R-type completion
  - Jump completion

- Write-back step
  - (Op = 'J')

- (Op = 'SW')
  - (Op = 'LW')
Instruction fetch

Instruction decode/register fetch

PCWrite
PCSource = 10

ALUSrcA = 1
ALUSrcB = 00
ALUOp = 01

PCWriteCond
PCSource = 01
ALUSrcA = 1
ALUSrcB = 00
ALUOp = 10

RegDst = 1
RegWrite
MemtoReg = 0

MemRead
IorD = 1

MemWrite
IorD = 1

RegDst = 0
RegWrite
MemtoReg = 1

Write-back step

Memory address computation

(Op = ‘LW’) or (Op = ‘SW’)

Execution

Branch completion

(Rop = R-type)

R-type completion

Jump completion

(Pop = ‘J’)

MemRead
IorD = 1

MemWrite
IorD = 1

RegDst = 1
RegWrite
MemtoReg = 0

Memory access

Memory access

4

Start

2

3

5

6

7

8

9

1
Shift left 2

PC

Mu

x

0

1

 Registers

Write

register

Write data

Read data 1

Read data 2

Read register 1

Read register 2

Instruction [15–11]

Mu

x

0

1

 Mu

x

0

1

4

Instruction [15–0]

Sign extend 32

16

Instruction [25–21]

Instruction [20–16]

Instruction [15–0]

Instruction register

ALU

control

ALU

result

ALU

Zero

Memory

data

register

A

B

IorD

MemRead

MemWrite

MemtoReg

PCWriteCond

PCWrite

IRWrite

PCSource

RegDst

PC

ALUOp

ALUSrcB

ALUSrcA

RegWrite

Control

Outputs

Op [5–0]
Q5.45 3rd Ed. add3 instruction

Start

0. Instruction fetch
   - MemRead
   - ALUSrcA = 0
   - lorD = 0
   - IRWrite
   - ALUSrcB = 01
   - ALUOp = 00
   - PCWrite
   - PCSource = 00

1. Instruction decode/ register fetch
   - ALUSrcA = 0
   - ALUSrcB = 11
   - ALUOp = 00

2. Memory address computation
   - (Op = 'LW') or (Op = 'SW')
   - ALUSrcA = 1
   - ALUSrcB = 10
   - ALUOp = 00

3. Memory access
   - MemRead
   - lorD = 1

4. Write-back step
   - RegDst = 0
   - RegWrite
   - MemtoReg = 1

5. Memory access
   - MemWrite
   - lorD = 1

6. Execution
   - ALUSrcA = 1
   - ALUSrcB = 01
   - ALUOp = 00

7. R-type completion
   - RegDst = 1
   - RegWrite
   - MemtoReg = 0

8. Branch completion
   - (Op = R-type)
   - ALUSrcA = 1
   - ALUSrcB = 00
   - ALUOp = 01
   - PCWriteCond
   - PCSource = 01

9. Jump completion
   - (Op = 'J')
   - PCSource = 10

10. Start of next instruction cycle

Instructions:
- Instruction fetch
- Instruction decode/register fetch
- Memory address computation
- Memory access
- Write-back step
- Memory access
- Execution
- R-type completion
- Branch completion
- Jump completion