ITSC 3181 Introduction to Computer Architecture, Spring 2023

Homework #4, Due 04/04/2023

Covered topics: CPU execution and pipeline

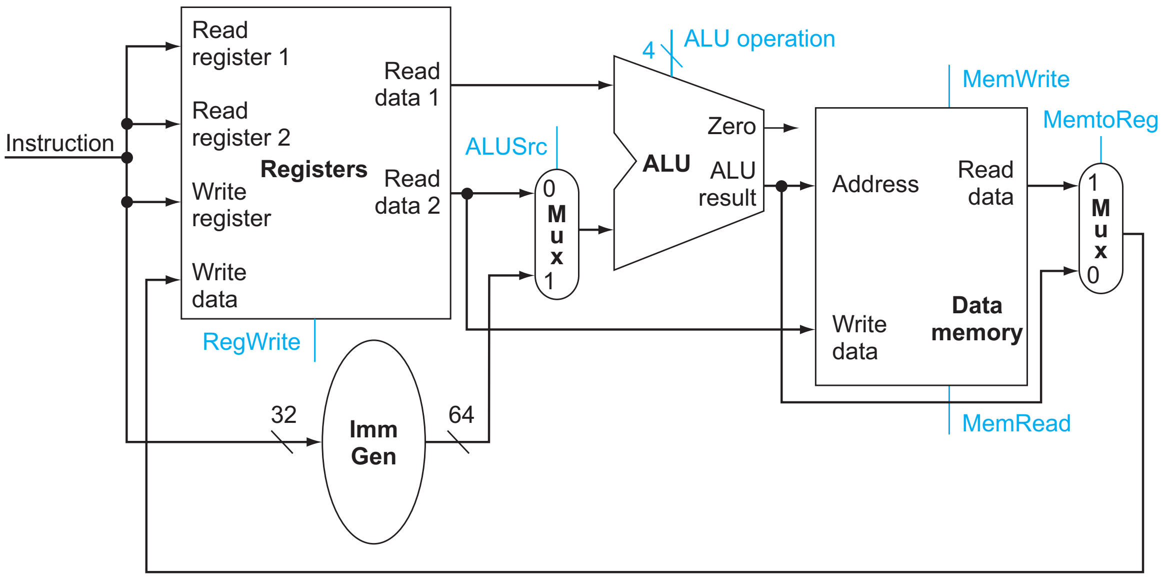
Question	1	2	3	Total points
Total Points	10	45	46	101
Grade				

5% of the grade of this homework will be added to your final grade.

Submission:

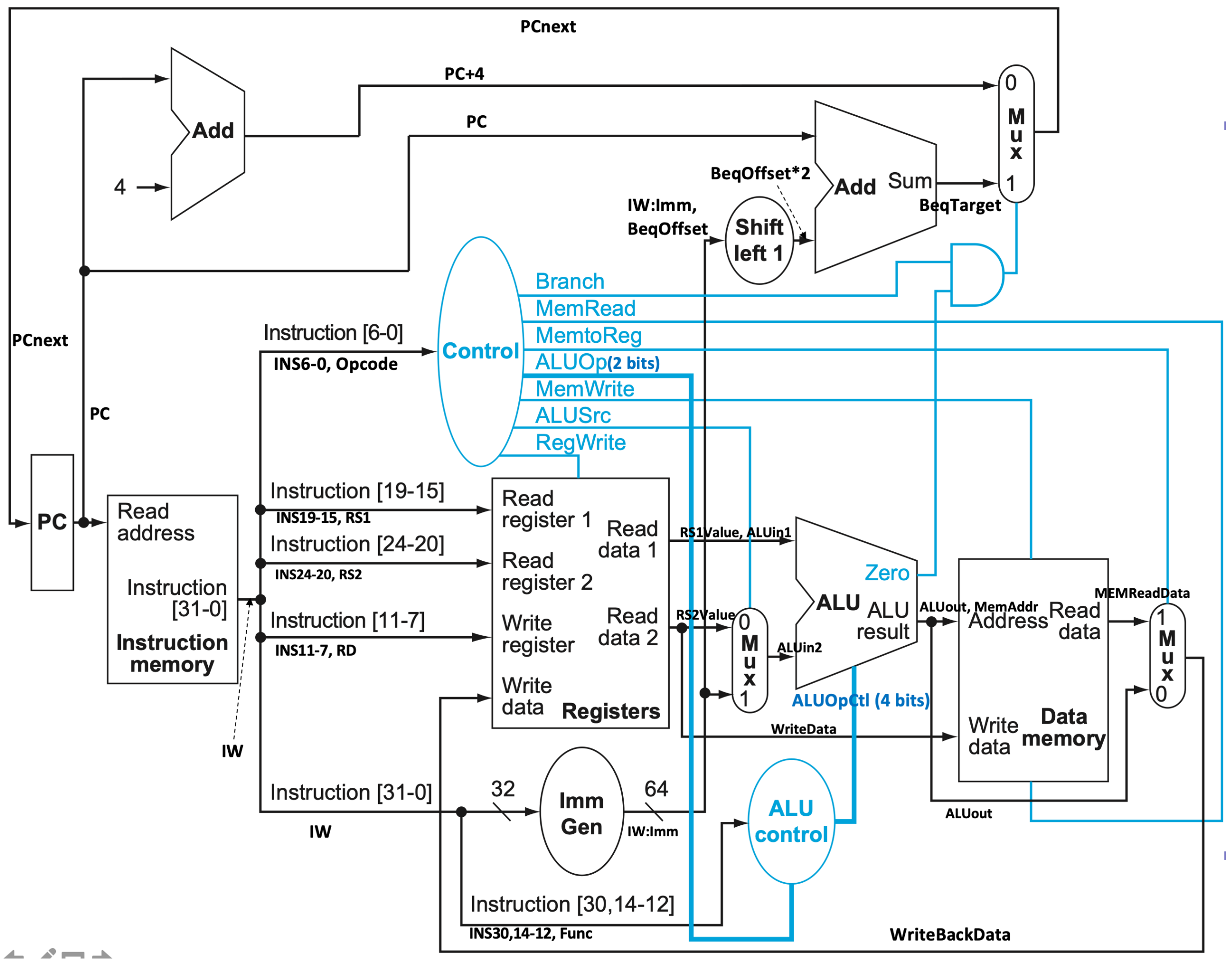
1. **Only electronic submissions on Canvas are accepted.**
2. **You should submit your solutions in two files, one PDF file for question 1 and 3.a, and the Excel sheet for the rest.**
3. **Number your solutions in the same way and in the same order as the questions are numbered in this document.**
4. **You must show the necessary steps when you solve each question. 0 point will be given to the question if the answer only includes the final answer with no necessary steps.**
5. **Scanned copy of handwritten work will be accepted and graded only if it is written clearly and readable.**

1. In this question, you are adding addi I-type instruction, which is in the format of "addi Rd, Rs1, imm". The operation is to perform Reg[Rd] = Reg[Rs1] + Imm. Highlight the datapath used for this instruction. The instruction is very similar to LW, but needs to write the ALU output to the register, and you can use LW datapath as reference for this question. Check slide 38-42 of the lecture about highlighting data path for instructions.



Answer sheet for Question 2 and 3 can be downloaded from https://passlab.github.io/ITSC3181/HW4/Homework_4_AnswerSheet.xlsx . The answer sheets provide answered examples for both question 2 and 3. **Please submit your solution to the first question in PDF file, and submit your solution to the second and third question using the sheet, do NOT modify the sheet structure, e.g. do not add or removing any row or column, do not merge or unmerge cells.**

2. In this question, you will exercise how CPU execute an instruction and how data are moved along with the datapath and how control signals are set. Your work is to fill in the provided Excel sheet the values of the datapath that are relevant to the instruction CPU is executing, and the setting of control signals of each type of instruction. Datapaths are labeled as in the following diagram. If you need references, book chapter 4.4 and 4.5 provide detailed description about how each type of instruction is executed, what datapath each uses and what control signals each instruction set or reset. Some of the pictures of the textbook are already copied to the answer sheet to help you look up. We went through examples during the class. Check the comments of some cell (if they have comments) for details how the value is calculated. To simplify answering with the sheet, we assume that the execution of the instruction does not change the actual value in the register files and memory. Your answers should be in the yellow-colored area of the sheet. Those cell that are marked with Red color in the answered examples are the critical cells that you need make sure they are right for the instruction you work on.



3. Pipeline execution and RAW (Read-After-Write) data hazard and control hazards. The following high-level C code is translated to RISC-V assembly (a Midterm 2 question). Instructions are executed on the CPU using the standard 5-stage pipeline (IF, ID, EXE, MEM, and WB). Register files can be read/write in the same cycle, and instruction memory and data memory are separated. Branch outcome is determined at the end of EXE stage, which means that it cause two cycle delay for pipeline.

```
for (i=0; i!=n-2; i++) a[i] += a[i+1];
```

Variable **n** is stored in register **x6** and **i** is in register **x10**. Array **a** is an array of **integers (a word)**.

```
        # Initialize registers for variable i and n-2
        li x10, 0               # i=0
        add x11, x6, -2          # x11 now has n-2

        # branch check
loop:   beq x10, x11, exit      # if loop condition is NOT true, exit loop

        # load a[i] to register
        lw  x7, a(x10)          # load a[i] to x7

        # load a[i+1] to register
        addi x12, x10, 1        # x12 now has i+1
        lw  x8, a(x12)          # load a[i+1] to x8

        # Do the addition of a[i] + a[i+1] and store the result to a[i]
        add x9, x7, x8          # a[i] + a[i+1], and in x9;
        sw x9, a(x10)           # store a[i] + a[i+1] into a[i]

        # Increment loop index i and jump to the beginning of the loop
        addi x10, x10, 1        # i=i+1
        beq x0, x0, loop        # back to condition check

        # loop exit
exit:
```

a) Fill in the rest of the following table about the RAW data dependency between two instructions that have at most one instruction in between and the register that introduces the dependency. Highlight it if it is Load-Use.

Instruction that writes the register	Instruction that reads the register	The register
li x10, 0	**beq x10, x11, exit**	**x10**

Answers to the following three questions need to be done in provided Excel sheet. Your answers should be in the yellow-colored area of the sheet. Fill in cell with red color for the stall cycles caused by data and control hazards. The sheet “Examples discussed at the class” shows how this is done using a more complicated examples and detailed comments are given in the sheet for you to work on the problem 3.

b) Draw the 5-stage pipeline execution of the first TWO iterations using stage labels (IF, ID, EXE, MEM, and WB) with no any data forwarding in the CPU. All RAW dependencies between consecutive two instructions (including AL-Use and Load-use) cause 2 cycle delay. BEQ cause two cycle delay for issuing next instruction. Based on the execution diagram, count how many cycles does it need to execute the whole loop if n is 1000. The number of cycles for each iteration should be counted from the cycle when the first instruction of the iteration should be fetched if no dependency on previous instruction to the cycle when the last instruction of the iteration is fetched. Again, it is counted from the cycle when the first instruction of the iteration should be fetched (not the cycle when the instruction is actually fetched since there could be stall cycles for issuing the first instruction).

c) Draw the 5-stage pipeline execution of the first TWO iterations using stage labels on the CPU but with fully data forwarding. With forwarding, the AL-Use RAW dependencies between consecutive two instructions cause 0 cycle delay. The Load-use dependency between consecutive two instructions cause 1 cycle delay. BEQ still has two cycle delay for issuing next instruction. Based on the execution diagram, count how many cycles does it need to execute the whole loop if n is 1000. The number of cycles for each iteration should be counted from the cycle when the first instruction of the iteration should be fetched if no dependency on previous instruction to the cycle when the last instruction of the iteration is fetched. Again, it is counted from the cycle when the first instruction of the iteration should be fetched (not the cycle when the instruction is actually fetched since there could be stall cycles for issuing the first instruction).

d) For the CPU with fully data forwarding, rearrange instructions using the techniques we discussed in the Chapter 4 lecture to eliminate the stall(s) from load-use hazard. All the load-use stall cycles should be completely eliminated, and then draw the 5-stage pipeline execution of the first TWO iterations using stage labels. When you do reschedule, you are allowed to change instruction to make sure the code is executed correctly. Hints, you are allowed to have “SW, a+/-ConstantOffset (base)” format, e.g. “sw x11 a-4(x10)” meaning to store x11 to address “a-4+[x10]”. Based on the execution diagram, count how many cycles does it need to execute the whole loop if n is 1000.