CSO-Recitation 14

CSCI-UA 0201-007

R14: Assessment 13 & ALU & RegFile & Pipeline

Today’s Topics

- Assessment 13
  - Review pipelined CPU
- Lab6
  - Review ALU & RegFile

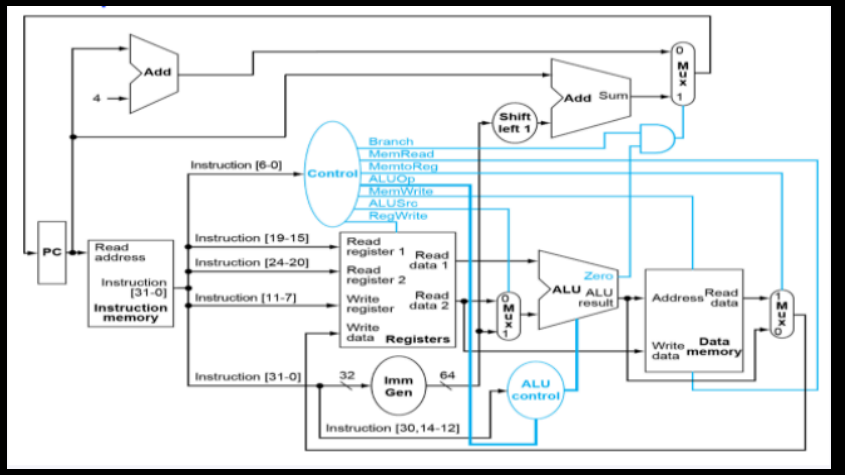
Something you may care about..

- The overall grade decomposition published on campuswire #206
- 1 lab required:
  - Lab5: due Dec 14 11pm
- 2 assessments to do:
  - Assessment-14: release this Friday 9am and due Sat 9am
  - Final assessment: release this Sunday 9pm and due next Monday 9pm
- Optional lab6:
  - postpone to due Dec 17 11pm

Assessment 13

Q1 Single-cycle CPU

Q1 single-cycle CPU



- **Q1.1** Data path
- Suppose the RISC-V instruction being executed is add x6, x7, x8, where x6 is the destination register, and x7/x8 are the 1st/2nd source register operand, respectively.
- What are the values corresponding to Instruction[11-7] that are fed into the "write register" pins of the RegisterFile? To facilitate autograding, please write your answer in exactly 5 binary digits, with higher order bits on the left.
- 00110

- Write register → destination register
- x6
- use 5-bit pattern to registers, since we have 32 registers
- 6: 00110

Q1.2 Control path

- There are 3 Mux in the Figure whose selectors are to be set by the control logic, located at the top-right, bottom-middle, and bottom-right.
- Which of the following instructions' execution cause the top-right Mux's selector to be set to **1**?

A. add x6, x7, x8 //x6 = x7+x8

B. beq x6, x7, 100, if x6 and x7 have the same value.

C. beq x6, x7, 100, regardless of whether x6 and x7 have the same value.

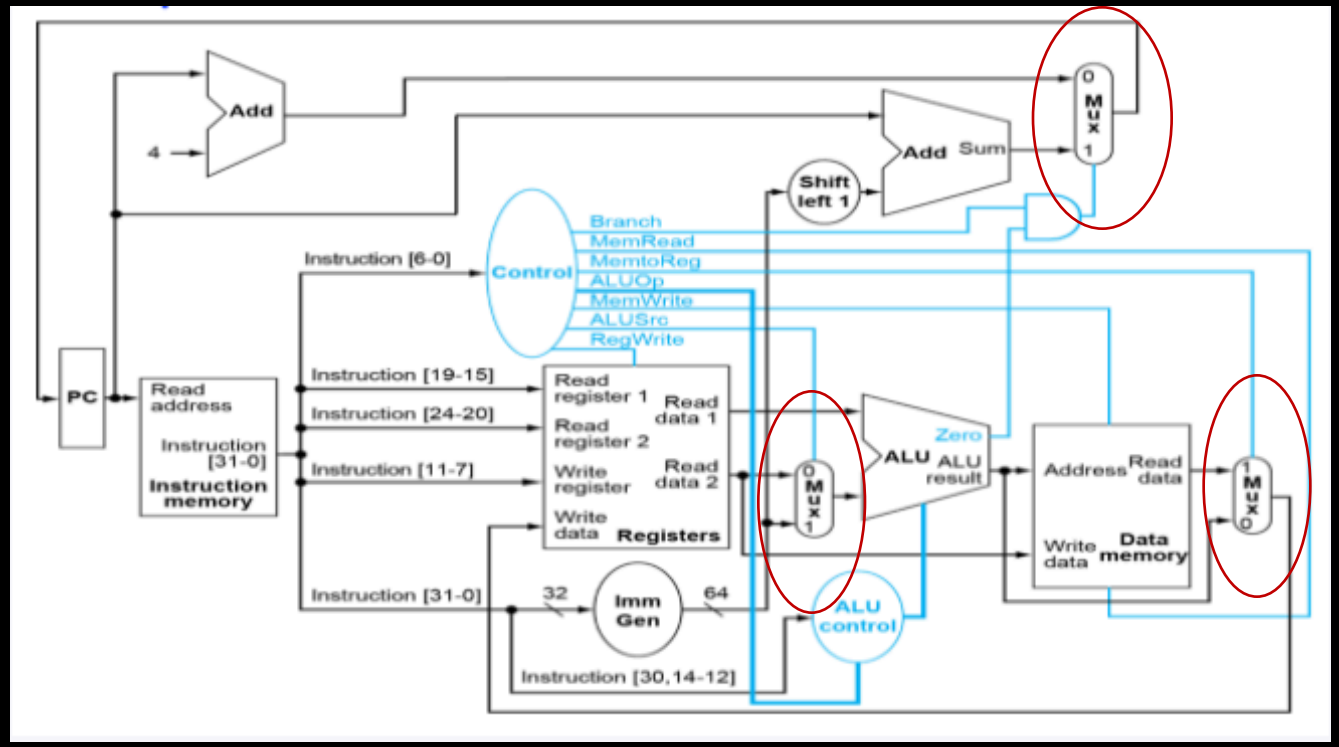
D. ld x5, 40(x6) //load a doubleword (8-byte) from Memory[x6+40]*Memory*[*x*6+40] to register x5

E. addi x6, x7, 200 //x6 = x7+200

F. sd x5, 40(x6) //store a doubleword (8-byte) from register x5 to Memory[x6+40]*Memory*[*x*6+40]

Q1.2 Control path

Control conditional branch for SB-type instruction



Control whether write what read from memory back to register

Control the input for ALU

Q1.2 Control path

- There are 3 Mux in the Figure whose selectors are to be set by the control logic, located at the top-right, bottom-middle, and bottom-right.
- Which of the following instructions' execution cause the top-right Mux's selector to be set to **1**?

for branch

A. add x6, x7, x8 //x6 = x7+x8

B. beq x6, x7, 100, if x6 and x7 have the same value. beq: equal → set to 1

C. beq x6, x7, 100, regardless of whether x6 and x7 have the same value.

D. ld x5, 40(x6) //load a doubleword (8-byte) from Memory[x6+40]*Memory*[*x*6+40] to register x5

E. addi x6, x7, 200 //x6 = x7+200

F. sd x5, 40(x6) //store a doubleword (8-byte) from register x5 to Memory[x6+40]*Memory*[*x*6+40]

Q1.3 Control path

- Continuing from Q1.2, which of the following instructions cause the value for the bottom-middle Mux's selector (aka ALUSrc) to be set to **1**?

A. add x6, x7, x8 add dest, rs1, rs2 -> set to 0

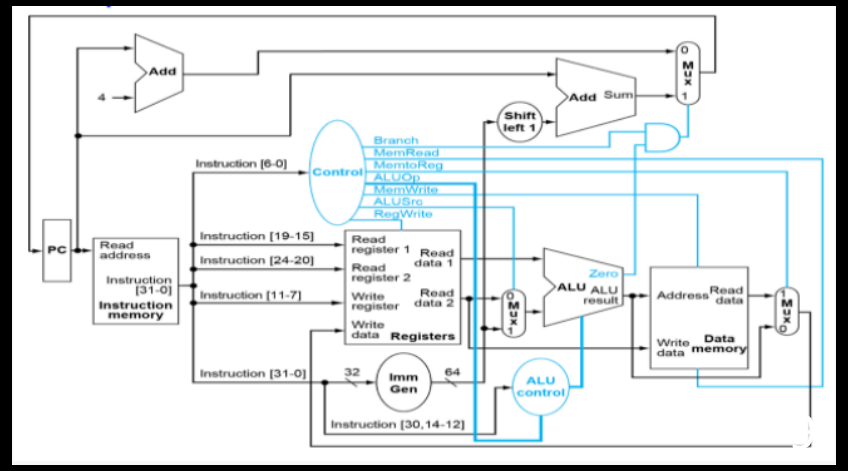
B. beq x6, x7, 100, if x6 and x7 have the same value.

C. beq x6, x7, 100, regardless of whether x6 and x7 have the same value.

D. ld x5, 40(x6)

E. addi x6, x7, 200

F. sd x5, 40(x6)



Q1.4 Control path

- Continuing from Q1.3, which of the following instructions cause the value for the bottom-right Mux's selector (aka MemToReg) to be set to **1**?

A. add x6, x7, x8

B. beq x6, x7, 100, if x6 and x7 have the same value.

C. beq x6, x7, 100, regardless of whether x6 and x7 have the same value.

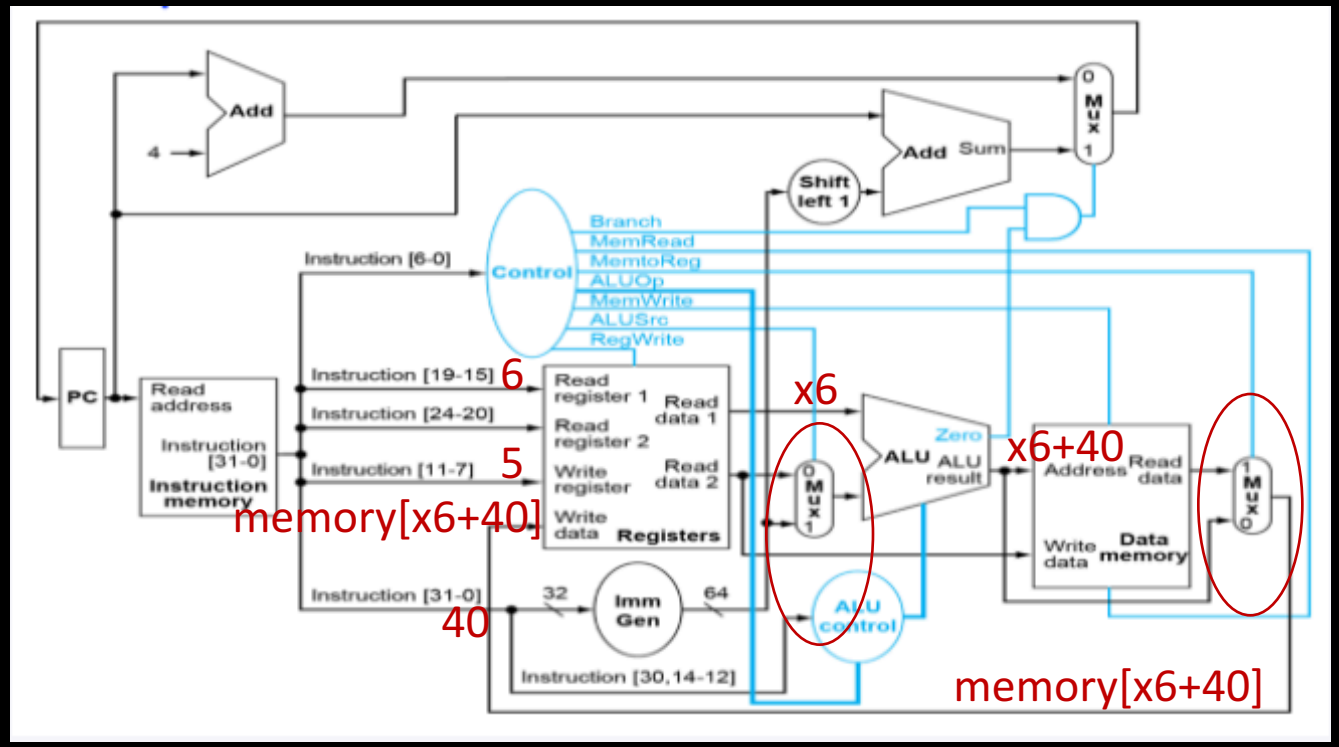
D. ld x5, 40(x6)

need to do mem_to_reg for Load instruction.

E. addi x6, x7, 200

F. sd x5, 40(x6)

Q1.2 Control path



x5= memory[x6+40]

Q1.5 Control path

- RegWrite:
- as long as there has write to register
- no matter the MemToReg is set or not:
  - set to 1: mem to reg
  - 0: write what calculated from ALU to reg
- as long as the instruction has destination register

- Which of the following instructions cause the value of the RegWrite input to the RegisterFile to be set?

A. add x6, x7, x8

B. beq x6, x7, 100, if x6 and x7 have the same value.

C. beq x6, x7, 100, regardless of whether x6 and x7 have the same value.

D. ld x5, 40(x6)

x5= memory[x6+40]

E. addi x6, x7, 200

F. sd x5, 40(x6)

MemWrite will be set. write in memory:
memory[x6+40]=x5

Pipeline

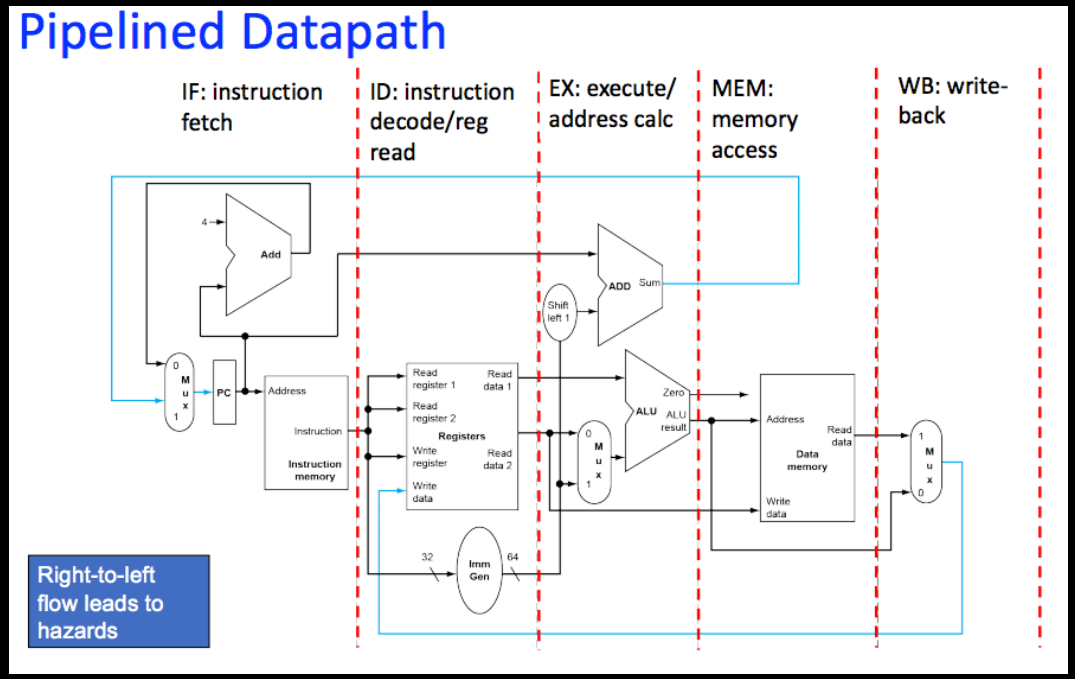
Design & Hazards

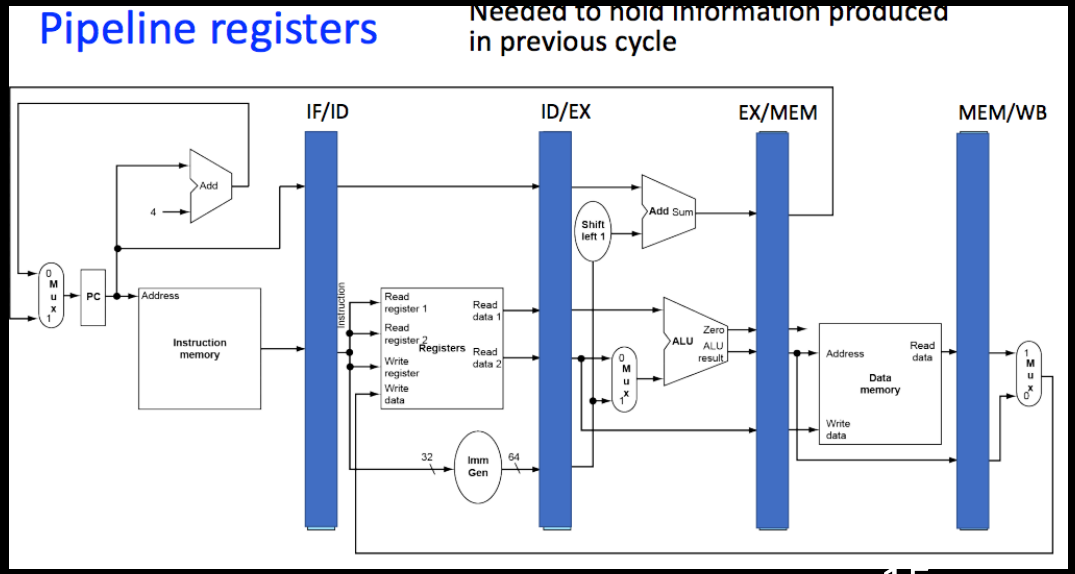
RISC-V Pipeline

- Pipeline increases throughput by overlapping execution of multiple instructions
- We split the instruction memory from the data memory
  - Otherwise reading data would delay reading an instruction
- There are 5 stages in the RISC-V pipeline
  - Instruction Fetch – IF
    - It takes a cycle to read an instruction from instruction memory
  - Instruction Decode – ID
    - Decode the instruction, read registers, and predict branching
  - Execute – EX
    - Performs computations using the ALU or performs shift operations
  - Memory Access - MEM
    - Read or write to memory
  - Write Back – WB
    - Write results to registers

Pipeline Registers

- In between each stage of the pipeline, we have registers store information
  - Each stage reads from the previous register to figure out what to do
  - Each stage writes to the next register to pass the instruction along





Pipeline hazard

- Sometimes instructions depend on each other
- In fact, that is very often the case, that some instruction uses the result of a previous instruction
- This becomes an issue, however, when an instruction depends on a value that has been computed but not written back yet

Hazard Motivation

- Consider the following C Program:

```
int calc(int a, int b, int c) {
    a = a + b;
    a = a + c;
    return a;
}
```

- Which becomes :

```
add x10, x10, x11
add x10, x10, x12
```

- What might cause sadness here?

Pipeline hazard

- Structure Hazard
  - Solution: add resources
- Data Hazard
  - Solution: forwarding
    - works for EX-hazard & MEM-hazard
  - Load-use Hazard:
    - Solution: stall and insert bubble
- Control Hazard
  - Solution: insert bubble
  - Reduce branch delay: by predicting branch in ID stage

Hazard Fix

- There are two approaches to avoiding hazards
- Forwarding
  - Be able to send data to an earlier location in the pipeline before write back
  - We do this by adding some multiplexers in the EX stage to choose between more recent values and the value from the register file
- Stalling
  - Add a “bubble” to the pipeline to give us time to resolve the hazard
  - We do this by setting some of the controls to 0 and by preventing the PC from changing

Assessment 13

Q2&3 Pipelined CPU

Q2 Pipelining performance

- Suppose the 5 stage pipeline has the following latency for each pipeline stage, 200ps (Instruction fetch aka IF), 100ps (Register read aka ID), 200ps (ALU operation aka EX), 200ps (Data access aka MEM), 100ps (Register write aka WB).
- Suppose we build a new CPU by adding the multiplication function to ALU, which causes the ALU latency to increase from 200ps to 400ps. Which of the following statements are true?

throughtput=1/time between instructions(clock cycle)

A. The new CPU has twice the instruction throughput of the original one.

B. The old CPU has twice the instruction throughput as fast as the new one.

C. The latency of a load ld instruction under the new CPU is twice as long as that under the original CPU.

latency: time for each instruction = clock cycle * #stages

D. The latency of a load ld instruction under the old CPU is twice as long as that under the new CPU.

E. The ALU latency increase would cause the new CPU to run at a slower clock rate than the old CPU.

F. The ALU latency increase would cause the new CPU to run at a faster clock rate than the old CPU.

Q3 Pipelining performance

- Suppose we change the RISC-V ISA to restrict load/store instructions to use a base register only (without an immediate offset/displacement). Thus, load/store instructions no longer need to use the ALU to compute addresses. As a result, we can change the CPU to overlap the data access (aka MEM) and ALU operation (aka EX) into one stage, resulting in a 4-stage pipeline. Note that in the merged MEM-EX stage, an instruction either performs data access or ALU operation, but not both. Hence the merged stage still takes 200ps, same as the latency of either MEM or EX in Q2.

Q3.1 Clock speed

- How does the new 4-stage design affect the clock speed?

A. Clock for 4-stage pipelined CPU must run faster than that in the 5-stage pipelined CPU.

B. Clock for 4-stage pipelined CPU must run slower than that in the 5-stage pipelined CPU.

C. Clock for 4-stage pipelined CPU can run at the same speed as that of the 5-stage pipelined CPU.

clock speed = time of clock cycle = 200ps

Q3.2 Instruction latency

- How does the new 4-stage design affect the instruction latency?

A. instruction latency (e.g. for load) for 4-stage pipelined CPU is lower than that in the 5-stage pipelined CPU.

B. instruction latency (e.g. for load) for 4-stage pipelined CPU is higher than that in the 5-stage pipelined CPU.

C. instruction latency (e.g. for load) for 4-stage pipelined CPU is the same as that in the 5-stage pipelined CPU.

- Instruction latency:
- previous: 200 instruction fetch + 100 register read + 200 ALU + 200 data access + 100 register write
- combine ALU & data access:
  - 200 instruction fetch + 100 register read + 200 ALU/data access + 100 register write

Q3.3 Instruction throughput

- How does the new 4-stage design affect the instruction throughput?

A. instruction throughput for 4-stage pipelined CPU is lower than that in the 5-stage pipelined CPU, under ideal (no hazard) scenarios.

B. instruction throughput for 4-stage pipelined CPU is higher than that in the 5-stage pipelined CPU, under ideal (no hazard) scenarios.

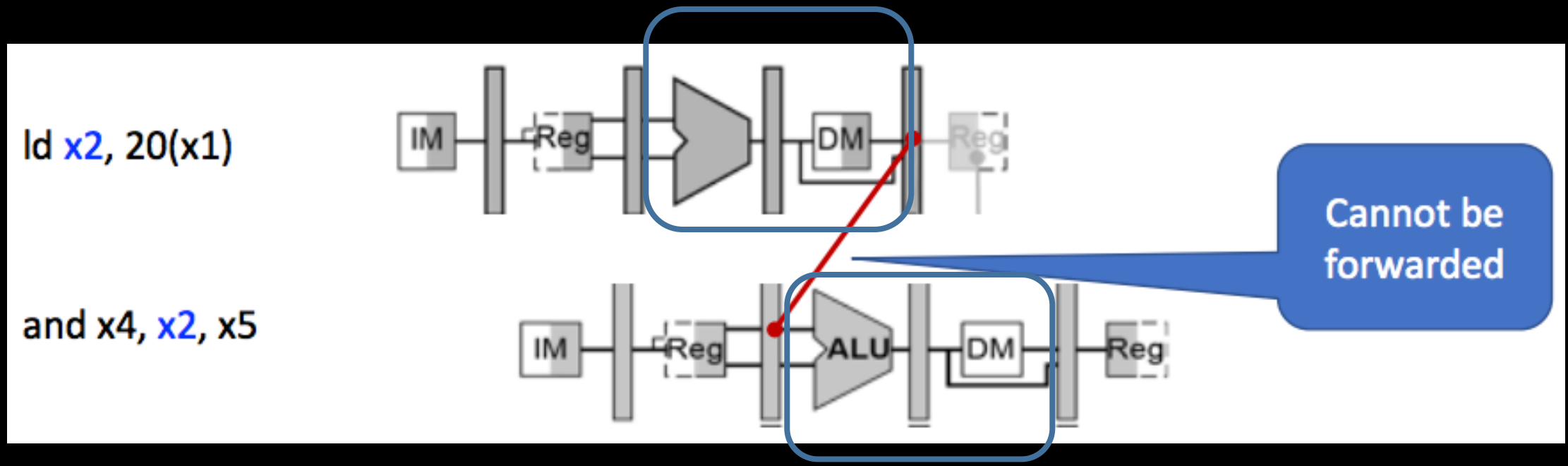
C. instruction throughput for 4-stage pipelined CPU is the same as that in the 5-stage pipelined CPU, under ideal (no hazard) scenarios. throughput is still 1/200ps

D. 4-stage pipelined CPU tends to have fewer hazards than 5-stage pipelined CPU.

E. 4-stage pipelined CPU tends to have more hazards than 5-stage pipelined CPU.

F. 4-stage pipelined CPU has the same amount of hazards as 5-stage pipelined CPU.

Q3.3 Instruction throughput



Then it can be fixed by **forwarding**!
It's like a MEM-hazard as before

Q3.4 others

displacement kind of things:
- handled inside CPU -> ALU
- handled outside CPU -> use more instructions

- How does the new 4-stage design affect the overall program performance?

A. Assuming no hazards, programs always execute faster under the 4-stage pipelined CPU than the original 5-stage pipeline.

B. Assuming no hazards, programs always execute slower under the 4-stage pipelined CPU than the original 5-stage pipeline.

C. Assuming no hazards, programs always execute at the same speed under the 4-stage pipelined CPU as the original 5-stage pipeline.

D. Assuming no hazards, programs can execute slower under the 4-stage pipelined CPU than the original 5-stage pipeline, because more instructions are needed (to calculate memory addresses for load/store).

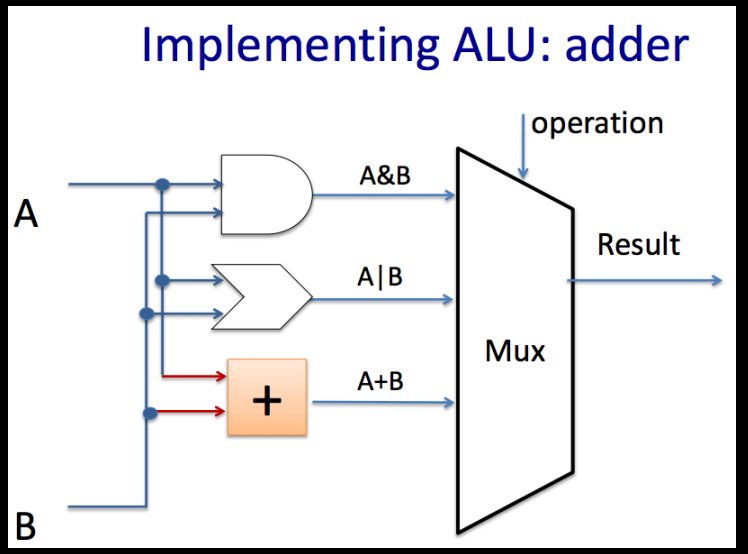
Common question for lab5

- Incomplete problem: Some circuits will be tested together
  - try to finish all circuits in one exercise, and check again
- When implementing
  - Don't change the contents above the dash line, and
  - use Tunnels (not pins) as inputs and outputs
- Building sub-circuits/sub-components and use Tunnels will help a lot
  - especially for complex implementation, i.e. lab6

Lab6

ALU & RegFile

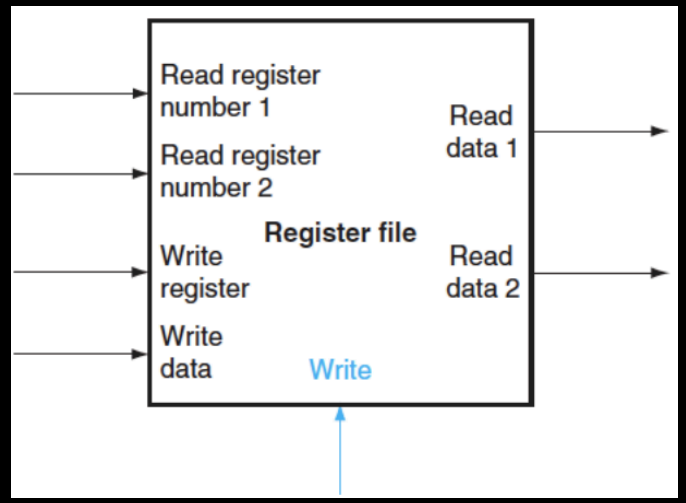
ALU



- 2 inputs, 1 ALU selector, 1 output(result)
- You need to implement 14 instructions
  - add, sub, and, or, xor, srl, divu,…
    - allowed and encouraged to use built-in Logisim blocks to implement the arithmetic operations
  - combinatorial circuits
    - how to build any combinatorial circuits?
    - think about srl(unsigned right shift), sra(signed right shift), sll(left shift)
- How to select the result?
  - “ALUSel”, MUX
- Building sub-circuits and using Tunnels is helpful.

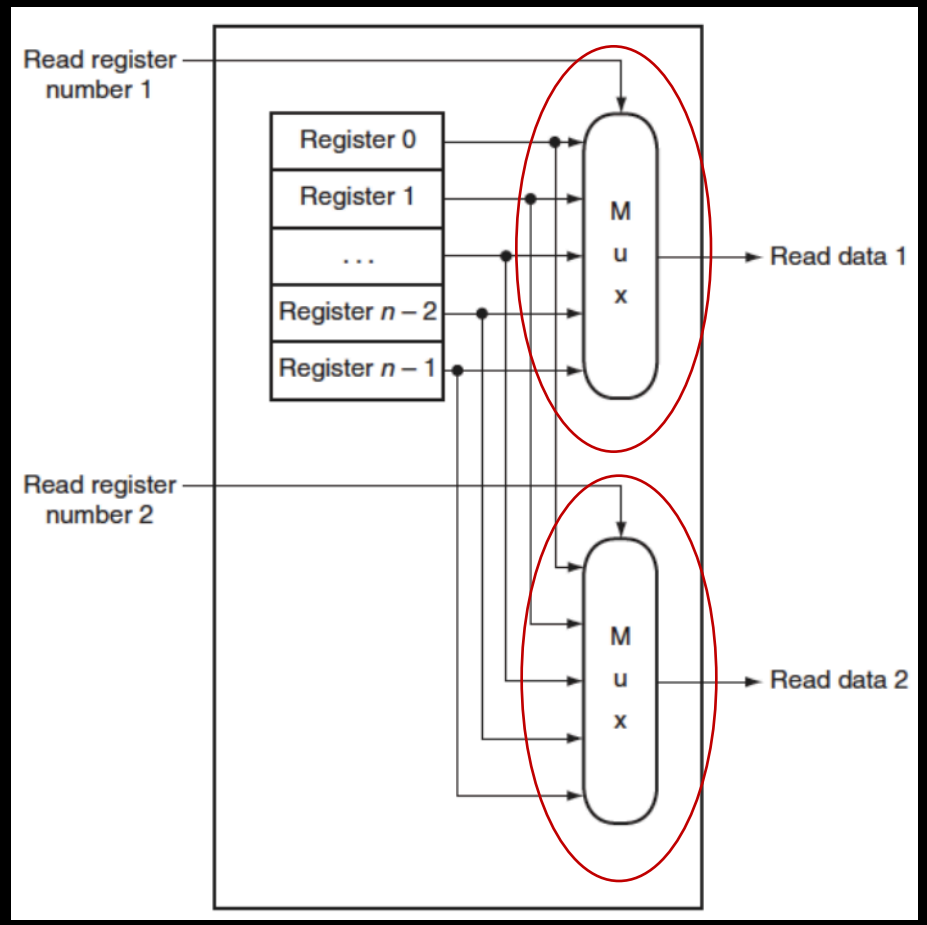
Register File

- Register file: a set of registers that can be read and written
  - It's a state element
- Six inputs:
  - clock, rs1, rs2, rd, write data, RegWEn
- Two outputs:
  - Read data 1, Read data 2
- Step:
  - build register
    - Lab asks to implement 9 registers instead of 32
    - Provide some outputs for testing and debugging purposes
  - build register file



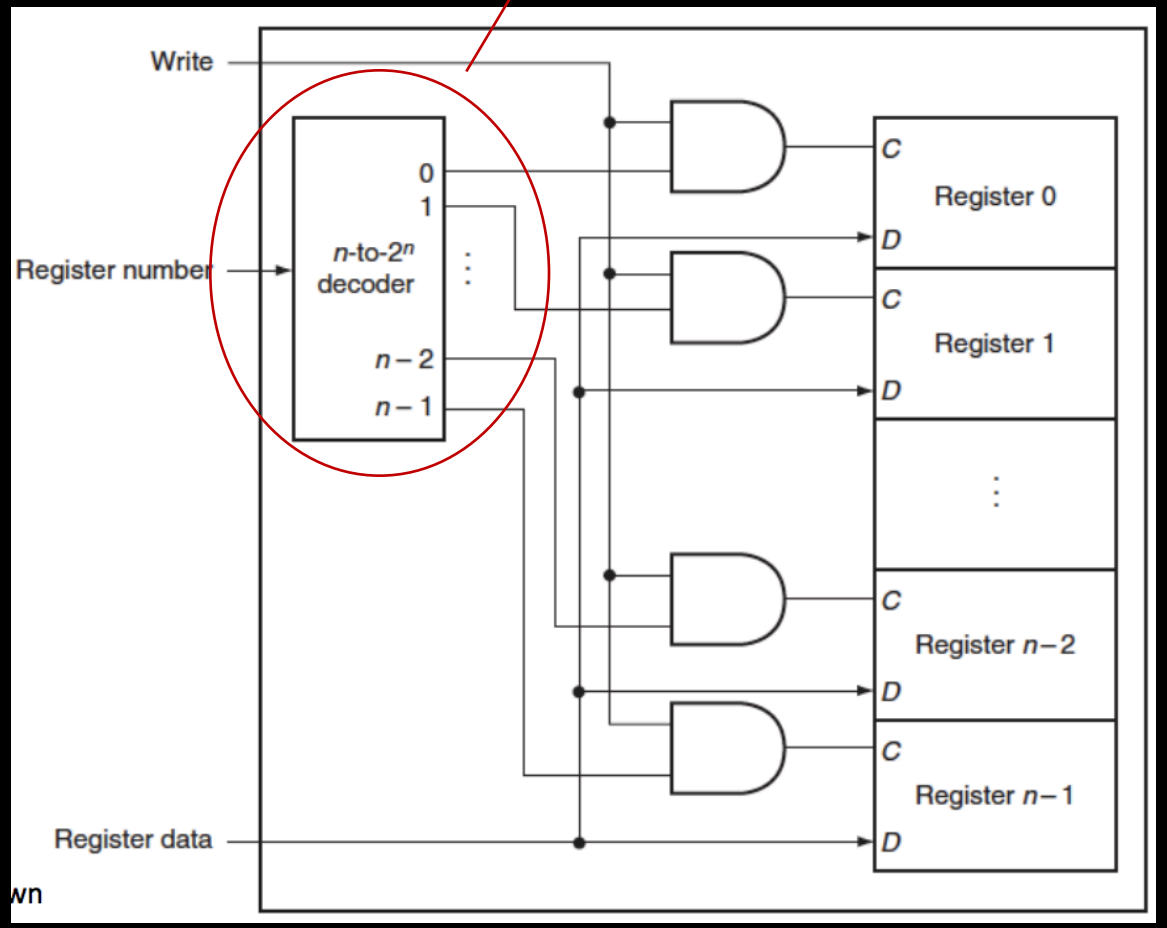
Register File

Read:



Think about DEMUX

Write:



Congrats to all

- Great job!
- Thanks for attending and supporting!
- Good luck to all your final works~