CSO-Recitation 14

CSCI-UA 0201-007

R14: Assessment 13 & ALU & RegFile & Pipeline

Today’s Topics

- Assessment 13
  - Review pipelined CPU

Assessment 13

Q1 Single-cycle CPU

Q1 FSM

In the lecture example on "electronic eyes" (see slide 28 of https://nyu-cso.github.io/notes/arch-seq.pdf). The desired pattern of lights to be lit up is: left, middle, right, middle, left, middle, right ...

What is the **minimum** number of distinct state values needed for a FSM to implement this electronic eyes device?

4

Q2 single-cycle CPU

- **Q2.1** 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?
- 00110

- Write register → register code
- The code of register xi is i.
- For example, the code of x5 is $(00101)_2$,
- x6: $(00110)_2$
- Note: use 5 bits since we have 32 registers

Q2.2 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]

Q2.2 Control path

```
beq x5, x6, 100
```

If (x5==x6) goto PC+2*100

Control which how to compute next PC (SB-type instruction)

PC=PC+4

PC=target address (e.g. PC+2*100)

Q2.2 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]

Q2.3 Control path

- Continuing from Q2.2, 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

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)

Q2.3 Control path

- Control ALU input
- 1 -> select Imm
- 0 -> select read data 2 (i.e. register)

Q2.3 Control path

- Continuing from Q2.2, 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

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

C. beq x 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)

x6 + 40 (imm)

x7 + 200 (imm)

x6 + 40 (imm)

Q2.3 Control path

sd x5, 40(x6)

Q2.5 Control path

- 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)

E. addi x6, x7, 200

F. sd x5, 40(x6)

Q2.5 Control path

- Set when we want to store a value into a register

Q2.5 Control path

- 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)

E. addi x6, x7, 200

F. sd x5, 40(x6)

x5= mer

write in memory: memory[x6+40]=x5

Q2.5 Control path

- Set when we want to store a value into a register
- With what data?

Q2.4 Control path

- Set when we want to store a value into a register
- With what data?

Q2.4 Control path

- Control what to write back to the register
- 1 -> select read data
  - ld x5, 40(x6)
  - x5=Mem[x6+40]
- 0 -> select ALU result
  - add x6, x7, x8
  - x6=x7+x8

Q2.4 Control path

- Continuing from Q2.3, 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)

E. addi x6, x7, 200

F. sd x5, 40(x6)

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

Pipeline latency and throughput

- Latency=max(stage time) * (#stages-1) + last_stage_time
  - e.g., stage times is
    - IF: 200ps; 1st Reg: 100ps; ALU: 200ps; Data access: 200ps; 2nd Reg: 100ps;
  - max(stage time)=200ps=clock cycle
  - latency=200ps*4+100=900ps
- Throughput=1/max(stage time)=clock rate
  - e.g., 1/200ps

Assessment 13

Q2&3 Pipelined CPU

Q3 Pipelining performance

- clock rate=throughtput=1/max stage time
  - old: 1/200
  - new: 1/400
  - old 2x faster

	IF	ID	EX	MEM	WB	max
old	200	100	200	200	100	200
new			400			400

Q3 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?

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

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

Q4 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.

Q4

Why we can merge them into single stage?
Because there is no dependency.
No dependency => parallelism

Q4.1 Clock speed

	IF	ID	EX	MEM	WB	max
old	200	100	200	200	100	200
new			200			200

clock cycle (=max) remains unchanged

- 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.

Q4.2 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:
- 5-stage: 200ps*4+100ps
- 4-stage:
  - 200ps*3+100ps

Pipeline hazard

No dependency => parallelis

Pipeline: parallelly execute th

However, not always true => hazard.

- For a specific instruction, stage depends on the previous stage.
- But we can utilize the independency across instructions
  - We can execute the next instruction without waiting for the finish of the previous instruction => pipeline

Pipeline hazard

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Pipeline hazard

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Pipeline hazard

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Pipeline hazard

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Pipeline hazard

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Pipeline hazard

- Structure Hazard
  - Caused by limited hardware resources.
  - Solution: add resources (e.g., separating inst. and data mem)
- Data Hazard
- Control Hazard

Caused by the dependency between instructions:
The execution of i2 depends on some output of i1.
=> wouldn't happen in sequential model, because i2 is executed after finishing i1 and thus i2 can always see the output of i1.

Pipeline hazard

- Structure Hazard
  - Caused by limited hardware resources.
  - Solution: add resources (e.g., separating inst
- Data Hazard
- Control Hazard

Caused by the dependency betw
The execution of i2 depends on some output of i1.
=> wouldn't happen in sequential model, because i2 is executed after finishing i1 and thus i2 can always see the output of i1.

i2 must wait for the finish of i1?
Can we get the output sooner and thus doesn't affect the execution of i2?

Forwarding

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Forwarding

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Forwarding

x5 = 0, x6 = 1, x7 = 1
add x5, x6, x7 (x5 = x6 + x7)
add x4, x5, x6 (x4 = x5 + x6)
x4 should be 3

Forwarding

- The input of some stage s2 in i2 depends on the output of some stage s1 in i1:
  - E.g. i2's input of EX stage depends on the i1's output of EX stage
  - Trying to forward i1's output of s1 to i2's s2.
  - But this doesn't work all the time.
    - Need to delay i2's s2 until i1 has the output.
    - How? By adding bubble (nop instruction)

Sometimes i1 cannot have the output at the time when i2 needs it (e.g., i2's input of EX depends on i1's output of MEM)

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

However, when i2 starts EX stage, i1 just starts its MEM stage!
=> We do not have the correct input yet!

We need to delay i2's EX stage, so that i1 can prepare the required output. How many bubbles?

input of i2's EX stage depends on the output of i1's MEM stage

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

How many bubbles?
How many cycles it needs for i1 to prepare the output before i2 uses it.

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

How many bubbles?
How many cycles it needs for i1 to prepare the output before i2 uses it.

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

How many bubbles?
How many cycles it needs for i1 to prepare the output before i2 uses it.

Bubble

x5 = 0, x6 = 1, mem[101] = 1
i1: ld x5, x6(100) (x5 = mem[x6 + 100])
i2: add x4, x5, x6 (x4 = x5 + x6)
x4 should be 2

How many bubbles?
How many cycles it needs for i1 to prepare the output before i2 uses it.

Q4.3 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.

Q4.3 Instruction throughput

- More stages, tend to have more hazards
  - E.g. sequential model: 1 stage, no hazards
- Why?
  - Suppose i2 depends on the output of i1
  - Intuitively, with more stages, there are more cases that i1 is still in process when i2 needs the input, and causes a hazard.

Q4.3 Instruction throughput

i1: add x5, x6, x7
Nop
i2: add x4, x5, x6

Q4.3 Instruction throughput

i1: add x5, x6, x7
Nop
i2: add x4, x5, x6

Control hazard

assume x5 = 0 x6 = 0
beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
add x7, x8, x9
How many bubbles?
Note: Suppose a jump instruction can decide the address of the next instruction after EX stage

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

Q1: what (the output of which stage of i1) does i2 need to correctly execute?

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

Q1: what (the output of which stage of i1) does i2 need to correctly execute?

i2 needs the instruction address
=> decided by the output of
EX stage in i1

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

i1 should finish EX before i2 starts IF
i1 starts EX at cycle k, i2 starts IF at cycle k+1

i2 needs the instruction address
=> decided by the output of
EX stage in i1

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

Currently, i1 starts EX at cycle k, i2 starts IF at cycle ?

i2 needs the instruction address
=> decided by the output of
EX stage in i1

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

i2 needs the instruction address
=> decided by the output of
EX stage in i1

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

i1 should finish EX before i2 starts IF
i1 starts EX at cycle k, i2 starts IF at cycle k+1

i2 needs the instruction address
=> decided by the output of
EX stage in i1

Delay i2 for 2 cycles!
=> Insert 2 bubbles

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

i1 has just finished EX

Control hazard

assume x5 = 0 x6 = 0
i1: beq x5, x6, 100
add x1, x2, x3
…

Should jump to here:
i2: add x7, x8, x9
How many bubbles?

Q2: How many cycles does i1 need to finish that stage before i2 needs it?

How many bubbles?
How many cycles it needs
for i1 to prepare the output
before next instruction uses it.

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. bonus exercise: Logical Shift Right

Congrats to all

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