Discussion 9

Pipeling & Superscalar

Chenyu Wang
wangchy4@shanghaitech.edu.cn

Outline

- Midterm I number representation
- Pipeling
- Superscalar
- Q & A

Number representation

(b) A **quarter** is a single byte split into the following fields (1 sign, 3 exponent, 4 mantissa): `SEEEMMMM`. It has all the properties of **IEEE 754** (including denormal numbers, NaNs and $\pm\infty$) just with different ranges, precision and representations.

For a **quarter**, the bias of the exponent is 3, and the implicit exponent for denormal numbers are -2.

What is the largest number smaller than ∞?

In binary _____

In decimal _____

Which negative denormal number is closest to 0?

In binary _____

In decimal _____

Solution: 01101111; 15.5; 10000001; $-\frac{1}{64}$.

1) Largest number smaller than infinity...
In quarter (floating point), infinity's representation is 0b01110000, where the exponent field is maxed out and the fraction is 0.
The largest number smaller than infinity? Subtract 1! The result: 0b01101111. Now translate to decimal for the second column.
1.1111 * 2 ^ 3 = 1111.1 = 15.5

2) Negative denorm closest to 0 (but not -0)...
In a quarter, -0's representation is 0b10000000.
The smallest number of higher magnitude than -0? Add 1!
The result: 0b10000001.

(c) What is the value of `q1`, `q2`, `c`, `d`?

Hint Rounding mode: round toward even/0.

```
1 quarter q1, q2, q3, c, d;
2 q1 = -0.25;
3 q2 = -4.0;
4 q3 = 0.125;
5 c = q1 + (q2 + q3);
6 d = (q1 + q2) + q3;
```

`q1` in binary _____

`q2` in binary _____

`c` in decimal _____

`d` in decimal _____

Solution: 10010000; 11010000; [redacted] -4.0 -4.0

-4.0's representation is 0b11010000. Its least significant bit's value:
2^2 * 2^-4 = 2^-2 = 0.25.
The problem in the LSB of the quarter occurs when we try to add -0.125 that we have to deal with rounding.
Adding -0.125 will put you exactly half-way between two numbers with exponent 2.

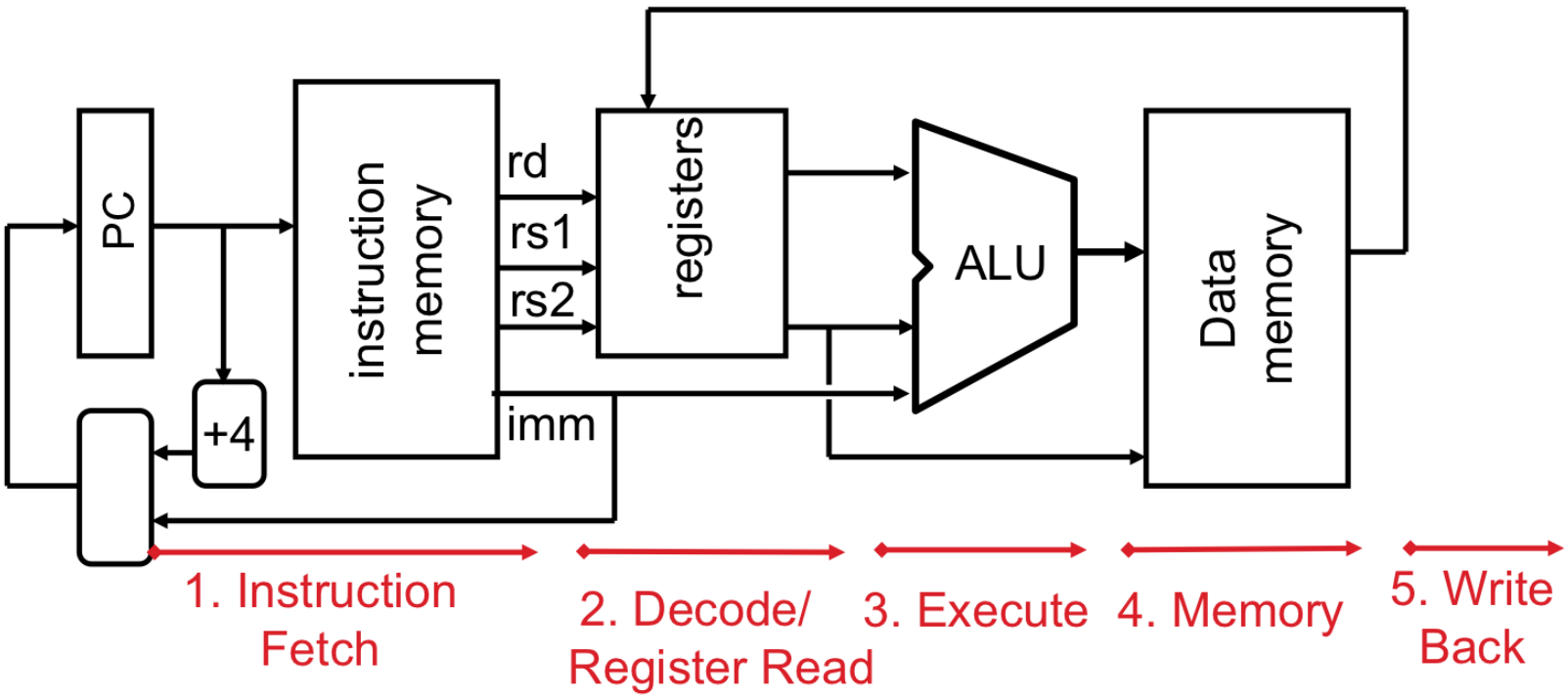
c) q2 + q3 happens first. -3.875
-3.875 -0.25 = -4.125puts me exactly half-way between -4.0 and -4.25.
-4.0 = 0b11010000
-4.125 falls right in between
-4.25 = 0b11010001
Round toward even tells us that we should round toward the quarter with a 0 in the LSB, so we round to -4.0.
Answer: -4.0

d) q1 + q2 = -4.25. No problems there
add on q3 (0.125) puts us exactly half-way between -4.0 and -4.25.

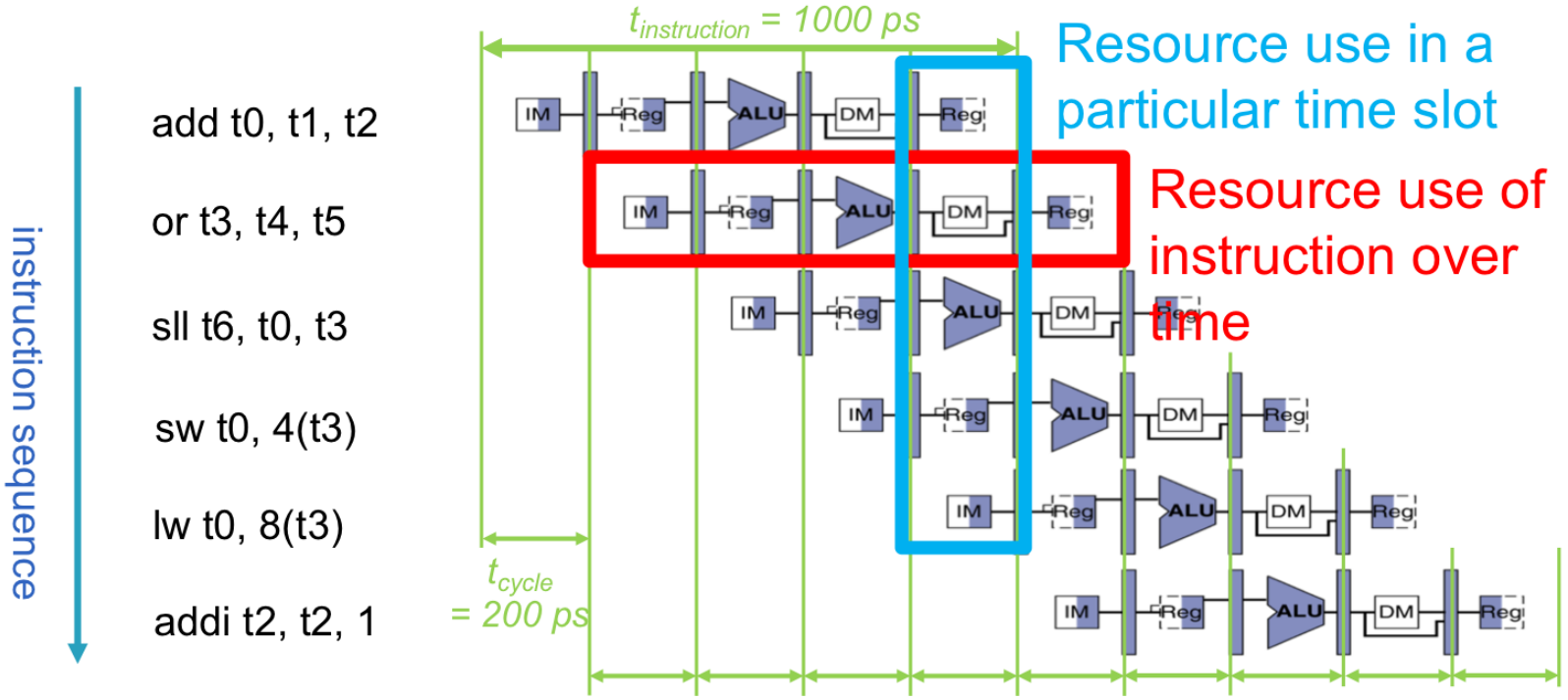
Answer: -4.0

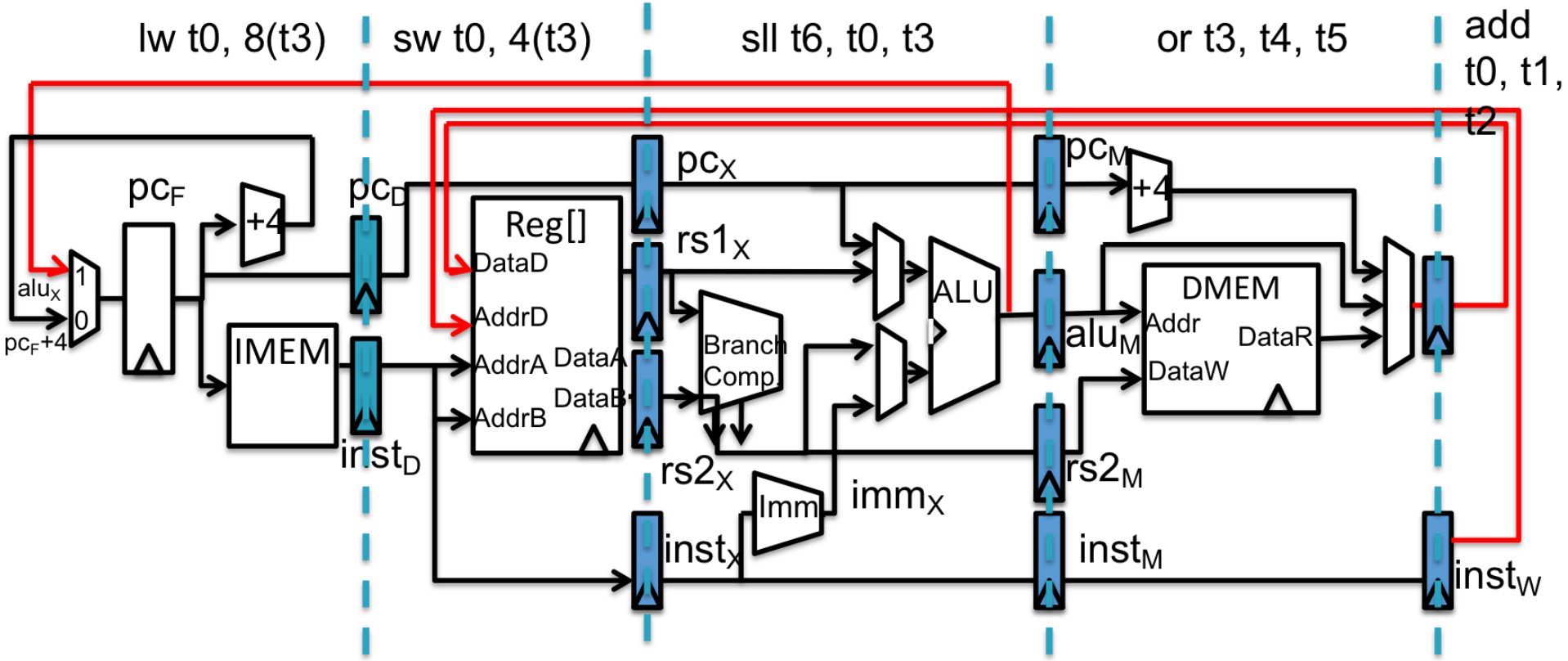
Pipeling

Single Cycle Datapath



RISC-V Pipeline





Pipeline registers separate stages, hold data for each instruction in flight

Pipelining Hazards

A *hazard* is a situation that prevents starting the next instruction in the next clock cycle

1) Structural hazard

- A required resource is busy (e.g. needed in multiple stages)

2) Data hazard

- Data dependency between instructions
- Need to wait for previous instruction to complete its data read/write

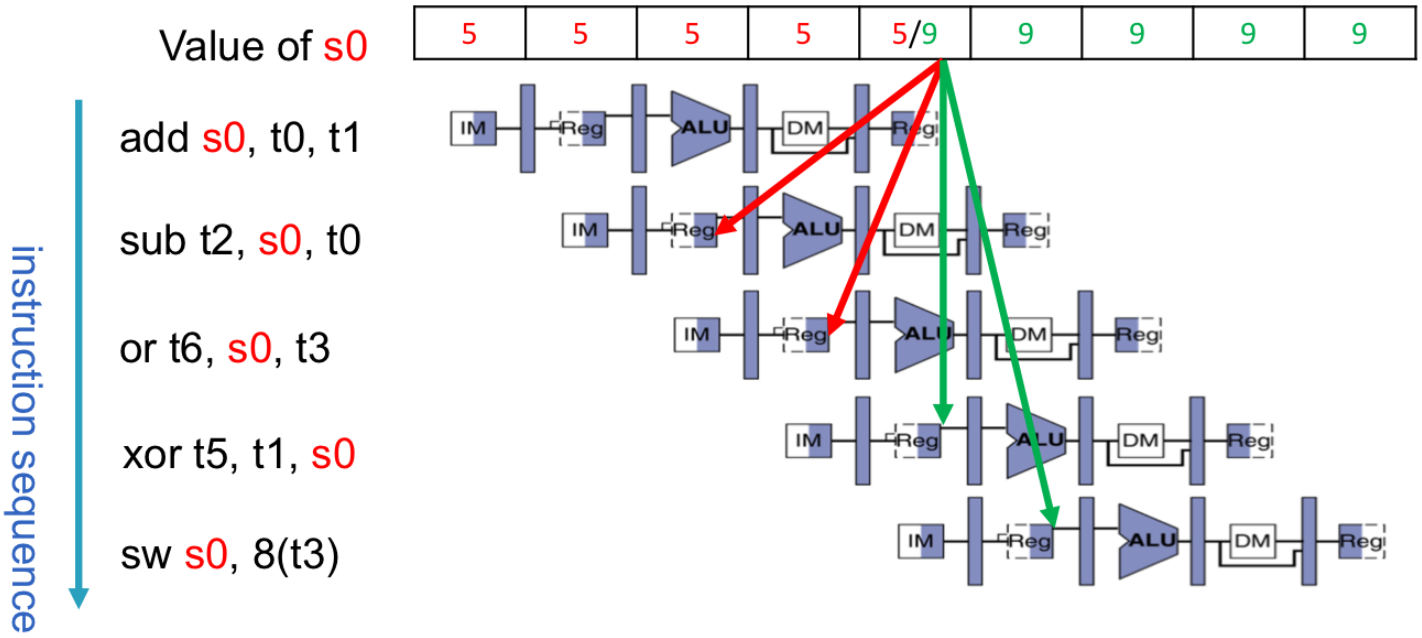
3) Control hazard

- Flow of execution depends on previous instruction

Structural Hazard

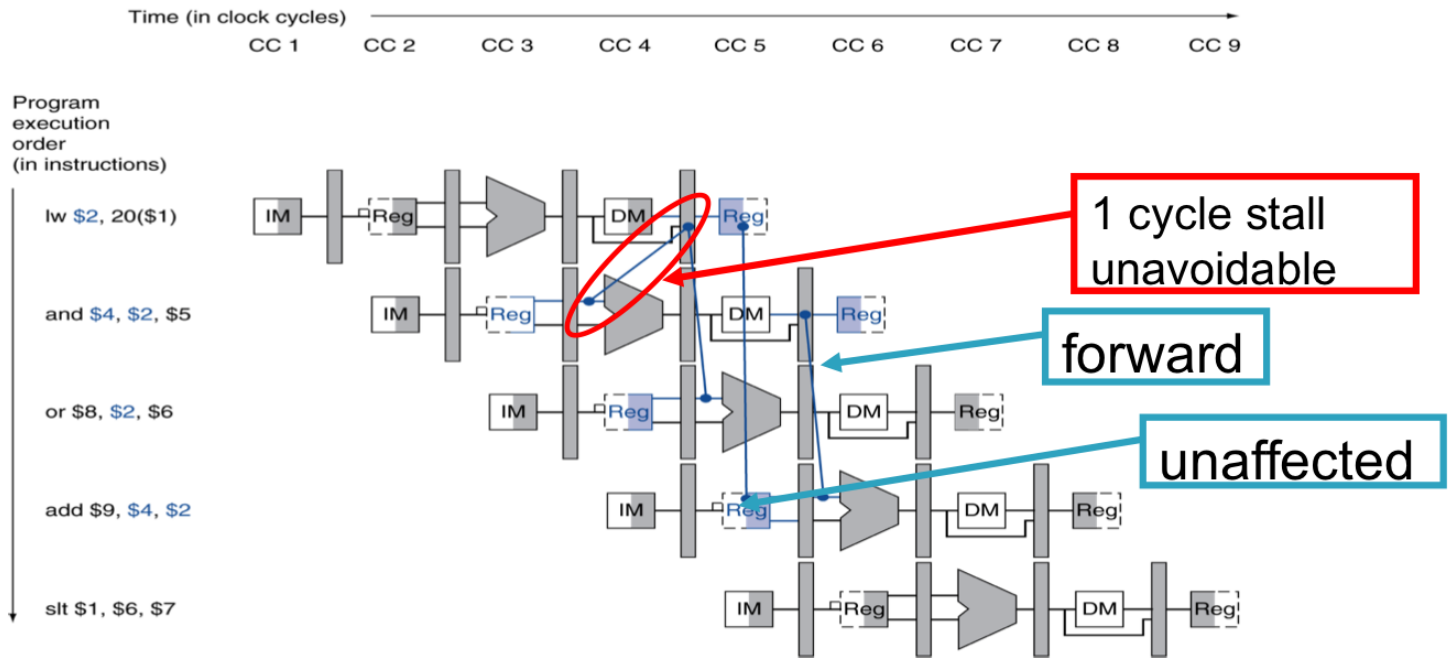
- **Problem:** Two or more instructions in the pipeline compete for access to a single physical resource
- **Solution 1:** Instructions take it in turns to use resource, some instructions have to stall
- **Solution 2:** Add more hardware to machine
- Can always solve a structural hazard by adding more hardware

Data Hazard: ALU Result



Without some fix, **`sub`** and **`or`** will calculate wrong result!

Load Data Hazard



Control Hazards

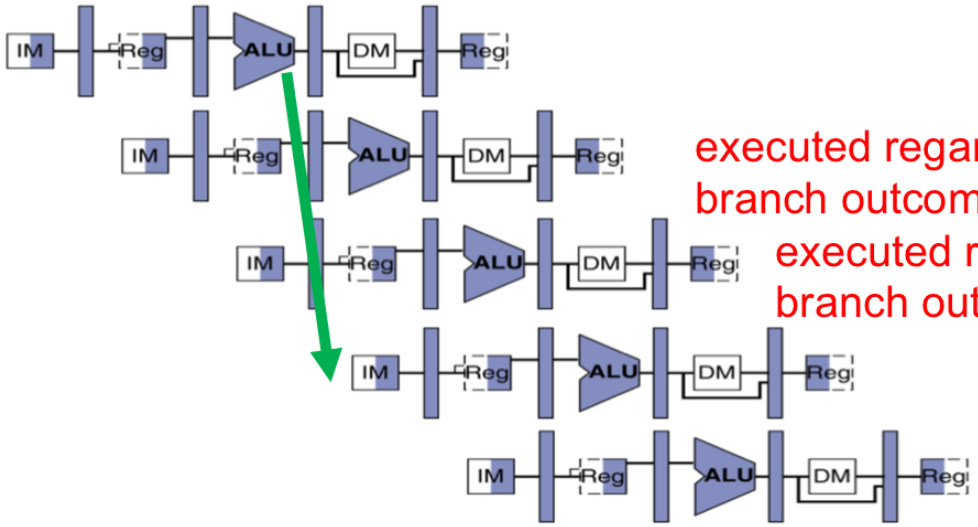
beq t0, t1, label

sub t2, s0, t5

or t6, s0, t3

xor t5, t1, s0

sw s0, 8(t3)



executed regardless of branch outcome!

executed regardless of branch outcome!!!

PC updated reflecting branch outcome

Case 1: Assume the address of an array with all different values is stored in `s0`.

```
addi  t0 x0 1
slli  t1 t0 2
add   t1 s0 t1
lw    t2 0(t1)
```

Each time you run this test, there is the same incorrect output for `t2`. All the commands work individually on the single-stage pipeline.

Pro tip: you shouldn't even need to understand what the code does to answer this.

a) What caused the failure? (select ONE)	**b)** How could you fix it? (select all that apply)
○ Control Hazard	■ Insert a nop 3 times if you detect this specific error condition
○ Structural Hazard	☐ Forward execute to write back if you detect this specific error condition
● Data Hazard	☐ Forward execute to memory if you detect this specific error condition
○ None of the above	■ Forward execute to execute if you detect this specific error condition
	☐ Flush the pipeline if you detect this specific error condition

The issue with the above code is the use of a register (aka we get the value during the decode phase) before we have written back the value of the previous instruction.

This is a data hazard as the data which we want is not restored to the regfile. This means that we would have the current instruction in the execute phase while we have the previous in decode.

This means that the next cycle, we would have to forward the execute output to the execute input to make sure the value is the correct, updated one. Inserting a nop when you realize this error happens will allow the system to do the write back. The other forwards in this problem are necessary for the given code above. Flushing the pipeline does not work as it means that we will no longer execute the instructions which were flushed. This means we would just drop instructions which would not get the correct value instead of just waiting till they can get the correct value.

Case 2: *After fixing that hazard,* the following case fails:

```
      addi  s0 x0 4
      slli  t1 s0 2
      bge   s0 x0 greater
      xori  t1 t1 -1
      addi  t1 t1 1
greater:
      mul t0 t1 s0
```

When this test case is run, `t0` contains `0xFFFFFFC0`, which is not what it should have been.
Pro tip: you shouldn't even need to understand what the code does to answer this.

c) What caused the failure? (select ONE)	**d)** How could you fix it? (select all that apply)
● Control Hazard	■ Insert a nop 3 times if you detect this specific error condition
○ Structural Hazard	☐ Forward execute to write back if you detect this specific error condition
○ Data Hazard	☐ Forward execute to memory if you detect this specific error condition
○ None of the above	☐ Forward execute to execute if you detect this specific error condition
	■ Flush the pipeline if you detect this specific error condition

- The issue with the code above is we do not clear/flush the instructions if the branch determines it is taken. Remember that we are running on a five stage pipeline CPU which just assumes PC + 4 unless an instruction says otherwise.
- This means that we will not determine if the branch is taken until the branch is in the execute phase. This means that we will have the next two instructions already in the pipeline (one in instruction fetch, the other in instruction decode). So we have a Control Hazard as we are not executing the correct instructions.
- Some ways how to fix it: insert nops if you detect a branch instruction in the instruction fetch stage OR flush the pipeline if the branch is in the opposite direction of what was predicted. Forwarding data in this case will not help at all.

Extra for Experience

Given the RISC-V code above and a pipelined CPU with no forwarding, how many hazards would there be? What types are each hazard? Consider all possible hazards from all pairs of instructions.

How many stalls would there need to be in order to fix the data hazard(s)? What about the control hazard(s)?

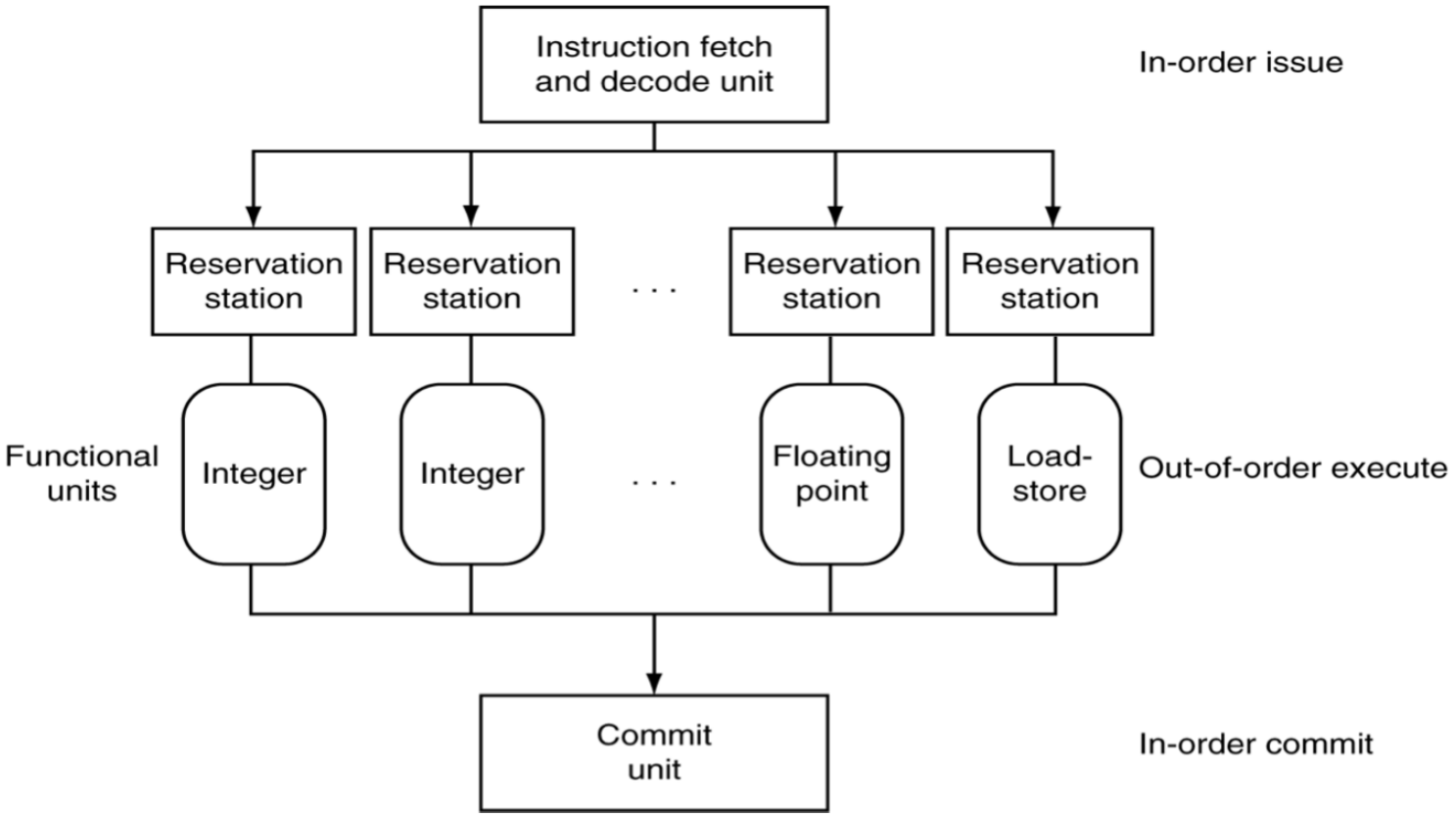
Instruction	C1	C2	C3	C4	C5	C6	C7	C8	C9
1. `sub t1, s0, s1`	IF	ID	EX	MEM	WB				
2. `or s0, t0, t1`		IF	ID	EX	MEM	WB			
3. `sw s1, 100(s0)`			IF	ID	EX	MEM	WB		
4. `bgeu s0, s2, l`				IF	ID	EX	MEM	WB	
5. `add t2, x0, x0`					IF	ID	EX	MEM	WB

There are four hazards: between instructions 1 and 2 (data hazard from t1), instruction 2 and 3 (data hazard from s0), instructions 2 and 4 (from s0), and instructions 4 and 5 (a control hazard).

Assuming that we can read and write to the RegFile on the same cycle, two stalls are needed between instructions 1 and 2, and two stalls are needed between instructions 2 and 3. No stalls are needed for the control hazard, because it can be handled with branch prediction/flushing the pipeline.

Superscalar

Superscalar Processor



Superscalar = Multicore?

https://en.wikipedia.org/wiki/Superscalar_processor

- NO!
- **Superscalar**: More than one Instruction per clock cycle!
  - Computing not a different thread!
  - Computing instructions from the same program!

  => Higher throughput

- In Flynn's taxonomy (later in course):
  - **a single-core superscalar processor is classified as an SISD** processor (Single Instruction stream, Single Data stream)
  - **But**: most superscalar processors support short vector operations => those are then SIMD (Single Instruction stream, Multiple Data streams).
  - And: nowadays most superscalar processors are multicore, too.

"Iron Law" of Processor Performance

CPI = Cycles Per Instruction

Can time **Can count** **Can look up**

$$\frac{\text{Time}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

$$\textbf{CPI} = \frac{\text{Cycles}}{\text{Instruction}} = \frac{\text{Time}}{\text{Program}} \div \left(\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Time}}{\text{Cycle}} \right)$$