Computer Architecture

Fall, 2025
Week 11
2025.11.10

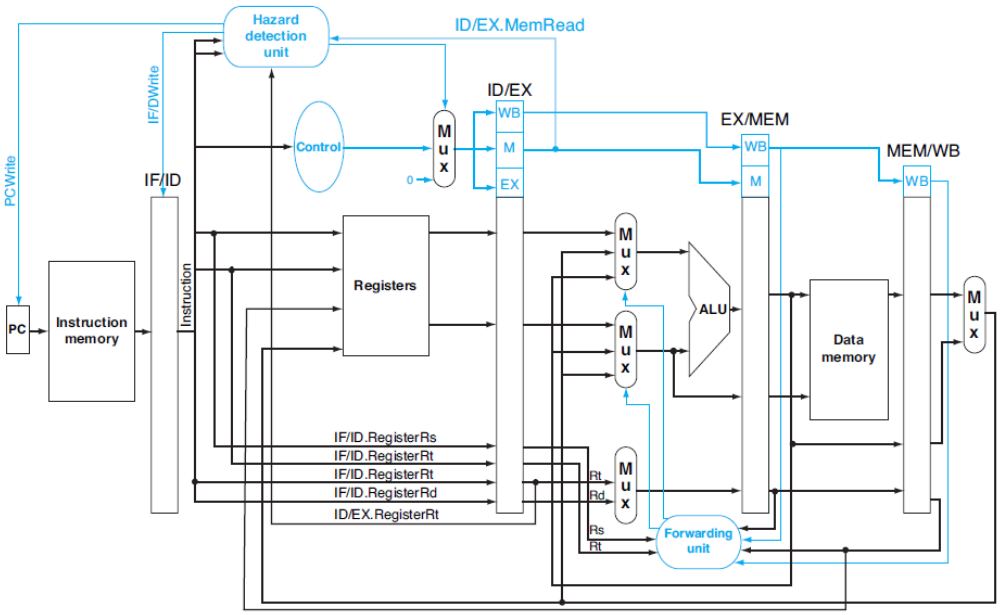
組別：_____ **簽名：**_____

[group 2]

1. Qustion:

Assume that the following sequences of instructions is executed on a 5-stage pipelined datapath:

LW R2, 100(R1)
SUB R3, R2, R5
AND R4, R2, R1
ADD R5, R3, R4
SW R5, 0(R2)



(a) If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.
(b) If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?
(c) If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units

Ans:

(a) LW R2, 100(R1)
NOP
NOP
SUB R3, R2, R5
AND R4, R2, R1
NOP
NOP
ADD R5, R3, R4
NOP
NOP
SW R5, 0(R2)

(b) Forwarding solves ALU-to-ALU hazards, but it cannot solve the load-use hazard between LW R2 and SUB R3, R2.

The SUB instruction needs the value of R2 at the start of its EX stage (Cycle 4). However, the LW instruction only gets the data from memory at the end of its MEM stage (Cycle 4). Forwarding is not possible, and without hazard detection, the SUB instruction will proceed with the old value of R2, leading to a wrong result.

(c) C1: LW (IF) **| Signals: None**
C2: LW (ID), SUB (IF) **| None**
C3: LW (EX), SUB (ID), AND (IF) | **PCWrite = 0** , **IF/IDWrite = 0**
C4: LW (MEM), SUB (ID), NOP (EX) **| None**
C5: LW (WB), SUB (EX), AND (ID) | **ForwardA = 01**

[group 1]

2. Question:

Consider the following two dependent MIPS R-type instructions executing in a 5-stage pipeline **without** forwarding:

1. ADD R1, R2, R3
2. SUB R4, R1, R5

How many **NOP (No-Operation) cycles** must be inserted between these two instructions to ensure the correct data for **R1**is available for the SUB instruction's **EX** stage? (Assume the ADD writes back its result in the **WB** stage).

Ans:

3 NOP cycles.

(Rationale: ADD writes at the end of Cycle 5. SUB needs R1 at its EX stage, which would be Cycle 3 if it followed immediately. Therefore, SUB must be stalled until the start of Cycle 6 to proceed to EX, requiring a 3-cycle stall/3 NOPs.)

[group 4]

3. Question:

Consider the following sequence of two MIPS instructions running on a 5-stage pipeline that **includes** a full forwarding unit:
lw $t0, 0($s1) # Instruction 1
and $t2, $t0, $s2 # Instruction 2
(a) In which pipeline stage does the and instruction first need its operands (specifically, the value from $t0)?
(b) In which pipeline stage is the result of the lw instruction (the data from memory) first available?
(c) Is forwarding alone sufficient to resolve this "load-use" data hazard? Why or why not?

Ans:

(a) The and instruction needs the value of $t0 at the beginning of its **EX (Execute)** stage.
(b) The lw instruction only has the data available after the **MEM (Memory)** stage is complete.
(c) **No**, forwarding alone is not sufficient. **Reason:** The and instruction needs the data in its EX stage (let's say cycle 3). However, the lw instruction is still in its MEM stage during cycle 3 and won't have the data ready until the end of that cycle. The and instruction needs the data "sooner" than forwarding can provide it.

[group 8]

4. Question:

True or False

a) Structural Hazard means that different instructions try to use the same hardware resource at the same time.

b) Pipelining can improve instruction latency.

c) In MIPS, the RAW (Read After Write) type of data hazard never occurs.

d) In a pipeline, all control signals must be passed through every pipeline register to the last stage, regardless of whether they are needed.

e) In a pipeline, the destination register number must be passed to the last stage (WB) so that the result can be written back to the correct register

Ans:

a) True

b) False,pipelining improves throughput

c) False, RAW hazards are the common data hazards in MIPS.

d) False, Only the necessary control signals are passed to later stages. Unused signals are not propagated to avoid wasting resources.

e) True

[group 9]

5. Question:

In the MIPS pipeline, which instruction can still cause a Data Hazard that requires a Stall, even when Forwarding is implemented?

(A) add (R-type instruction)

(B) sw (store word instruction)

(C) lw (load word instruction)

(D) beq (branch equal instruction)

Ans:

(C) lw (load word instruction)

Conceptual Explanation: This is because the lw instruction must reach the **MEM (Memory access)** stage to read the data from Data Memory. If the immediately following instruction needs this loaded data for its **EX (Execute)** stage, Forwarding cannot supply the data in time. Therefore, a **Stall** (or pipeline bubble) is still required for one clock cycle. This specific situation is called a **"load-use hazard". (**P.80)

[group 11]

6. Question:

Here's a sequence of MIPS instruction:

1. sub $2, $1, $3
2. and $12, $2, $5
3. or $13, $6, $2
4. add $14, $2, $2
5. sw $15, 100($2)

How many clock cycles can we save if we use forwarding instead of inserting NOPs to solve data hazards?

Ans:

We can save two clock cycles by forwarding. These instructions need two nops to make sure the data is written before the following instructions read it, which means there will be two wasted clock cycles.

[group 12]

7. Question:

A load instruction is in **ID/EX** and the next instruction in **IF/ID** uses its destination register (load-use hazard). How should the pipeline stall be handled?

A. Keep IF fetching and send ID to EX without NOP, PC unchanged.
B. Keep PC and IF/ID unchanged, insert a NOP in EX by clearing ID/EX control signals for 1 cycle.

C. Clear IF/ID to NOP, let ID and later stages continue, increment PC.

D. Overwrite EX with 0, forcing 2-cycle wait.

Ans: B

[group 13]

8. Question:

Data Hazard and NOP Insertion

Instruction sequence:

```
sub   $2, $1, $3
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
```

(a) If the processor has **no forwarding hardware**, how many NOPs are required for correct execution?
(b) If the processor supports **full ALU-ALU forwarding**, are any NOPs still needed?

Ans:

(a) Without forwarding, the result from sub is written back only at the **WB stage**, meaning the following instruction (and) cannot get $2's value until then.

→ **Insert two NOPs** between sub and and:

```
 sub   $2, $1, $3
nop
nop
and   $12, $2, $5
or    $13, $6, $2
add   $14, $2, $2
```

(b) With ALU forwarding, the value of $2 can be sent directly from EX/MEM or MEM/WB to the next instruction's EX stage.

→ **No NOPs required.**

[group 14]

9. Question:

Identify all data hazards that require forwarding or stalling in the following code.

(1) add $2, $5, $4

(2) add $4, $2, $5

(3) sw $5, 20($2)

(4) add $3, $2, $4

Ans:

	Data hazard
$2	[(1), (2)], [(1), (3)]
$4	[(2), (4)]

[group 3]

10. Question:

True / False

Q1. Structural hazards happen when two instructions need the same hardware at the same time.

Q2. Data hazards occur when an instruction depends on the result of a previous one.

Q3. Control hazards are caused by branch or jump instructions.

Q4. In MIPS 5-stage pipeline, all three types of data hazards (RAW, WAR, WAW) can occur.

Q5. Inserting NOPs is a hardware method to solve data hazards.

Q6. Forwarding (bypassing) sends data from later pipeline stages back to the ALU inputs.

Ans:

1. True

They both try to use one resource (like memory) in the same cycle.

2. True

Example: add needs a value that lw is still loading.

3. True

The CPU doesn't know which instruction to fetch until the branch result is known.

4. False

Only RAW can happen in MIPS; WAR and WAW are impossible because execution is in order.

5. False

NOPs are added by the **compiler** — a **software** solution.

6. True

It lets the CPU use the result immediately without waiting for write-back.

[group 7]

11. Question:

Can WAR or WAW occur? Why or why not?

Ans:

WAR (Write After Read): Cannot occur. Every instruction reads its source regs in ID (stage 2), and later instructions write in WB (stage 5). Because the pipeline is in order and reads happen before any later writeback, a later instruction can't "overtake" and write before an earlier one reads.

WAW (Write After Write): Cannot occur. All writes happen in WB and in program order (single writeback point, in-order). A later instruction can't reach WB before an earlier one.