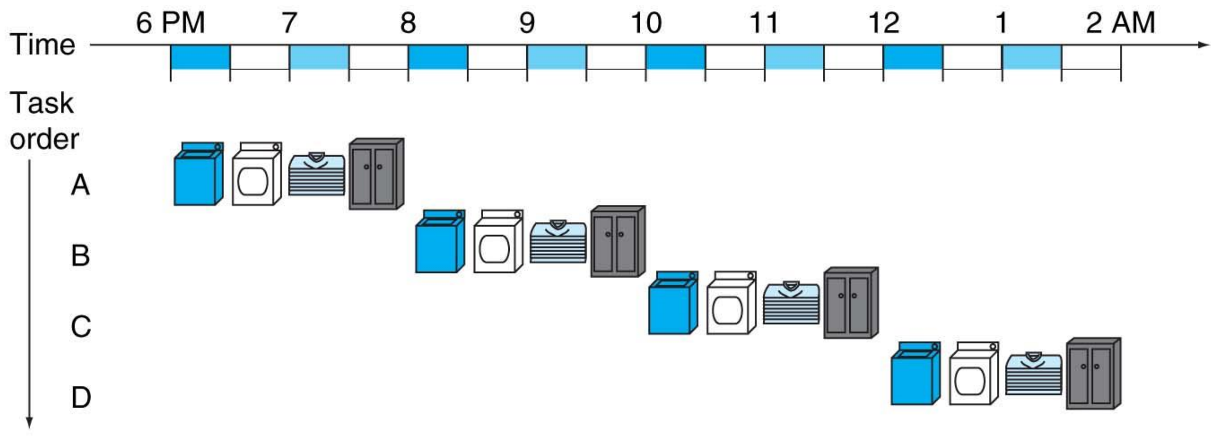
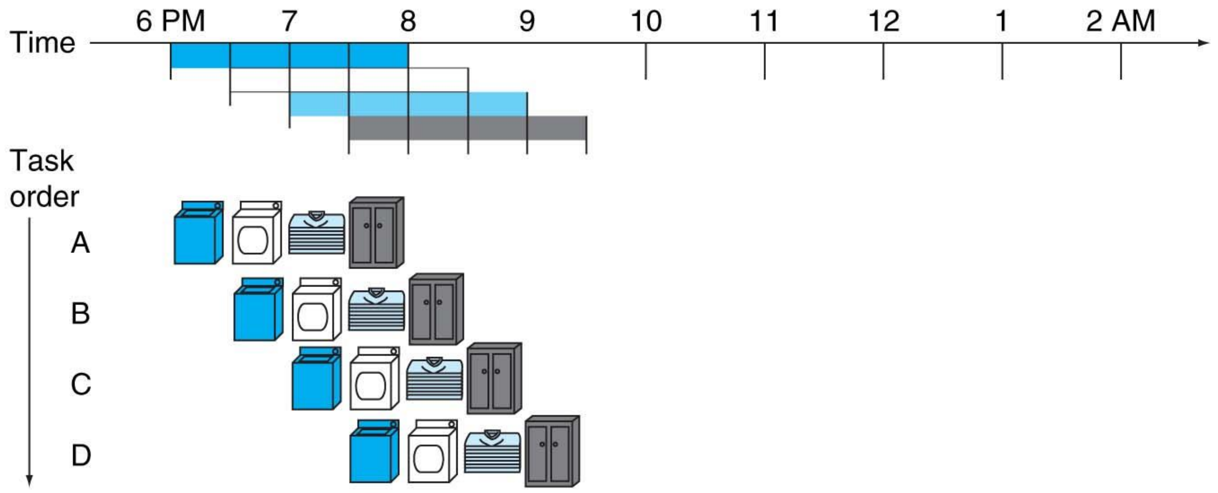
Chapter 2

Instruction-Level Parallelism (ILP)

Review





ZHEJIANG UNIVERSITY 1897

Data Dependences

- FLD F0, 0(R1)
- FADD.D F4, F0, F2

Name Dependences

```
FDIV.D      F2,  F6,  F4
FADD.D      F6,  F0,  F12
FSUB.D      F8,  F6,  F14
```

DIV&ADD: ***Anti-dependence***

Change F6 as S:

```
FDIV.D      F2,  F6,  F4
FADD.D      S,   F0,  F12
FSUB.D      F8,  S,   F14
```

```
FDIV.D      F2,  F6,  F4
FADD.D      F6,  F0,  F12
FSUB.D      F2,  F6,  F14
```

DIV&SUB: ***Output-dependence***

Change F2 as S:

```
FDIV.D      F2,  F0,  F4
FADD.D      F6,  F0,  F12
FSUB.D      S,   F6,  F14
```

Control Dependences

```
if p1 {
        Statement 1
    }
Statement
if p2 {
        Statement 2
    }
```

Hazards

- Situations that prevent starting the next instruction in the next cycle
- Structure hazards
  - A required resource is busy
- Data hazard
  - Need to wait for previous instruction to complete its data read/write
- Control hazard
  - Deciding on control action depends on previous instruction

Data Hazards

- Read after write: RAW

```
FADD.D     F6， F0， F12
FSUB.D     F8， F6， F14
```

- Write after read: WAR

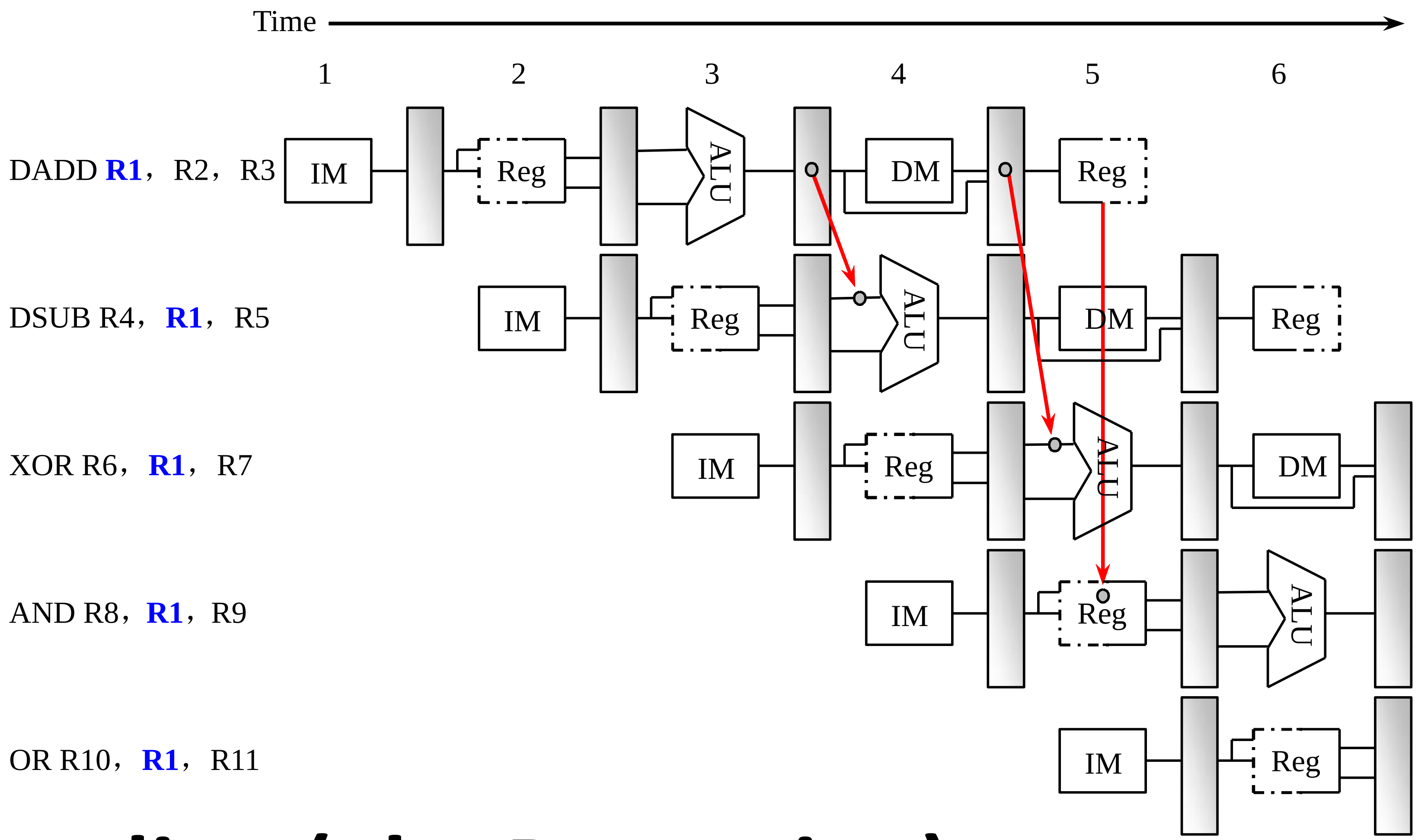
```
FDIV.D     F2， F6， F4
FADD.D     F6， F0， F12
```

- Write after write: WAW

```
FDIV.D     F2， F0， F4
FSUB.D     F2， F6， F14
```

Review

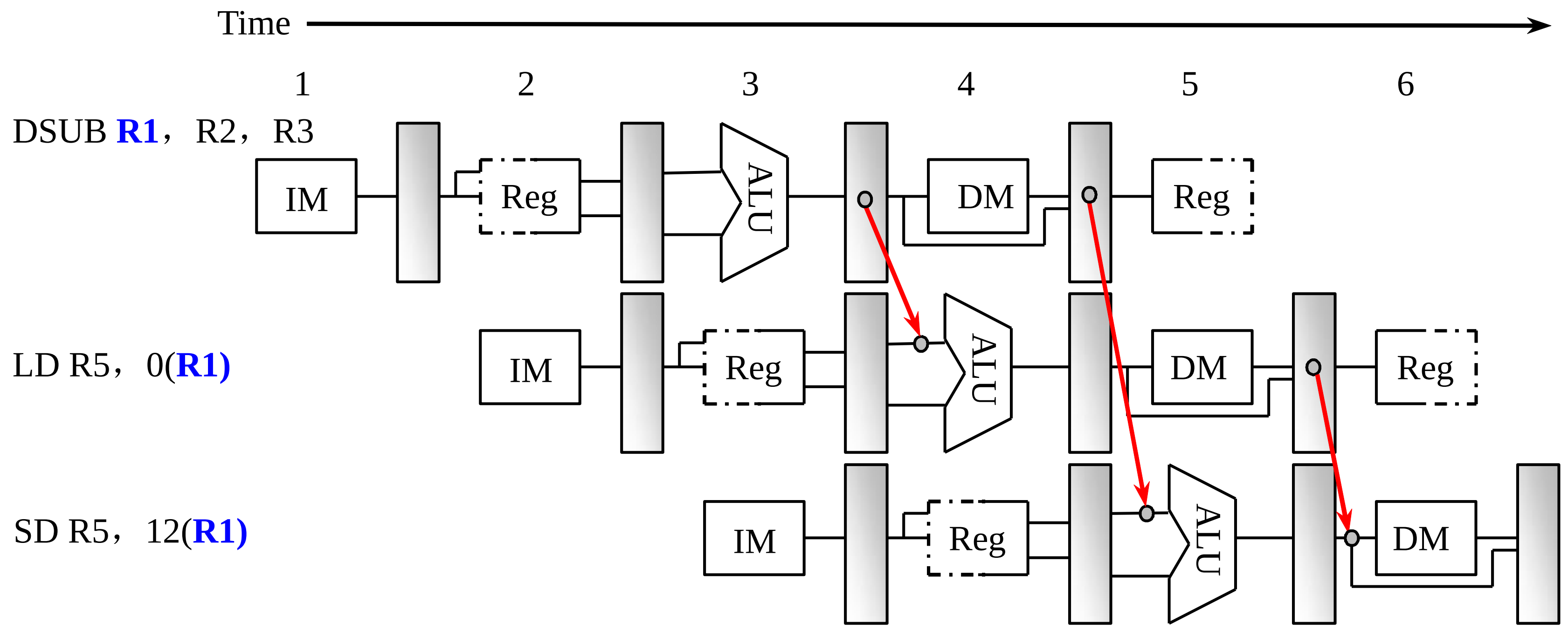
Data Hazards



Forwarding (aka Bypassing)

ZHEJIANG UNIVERSITY 1897

Data Hazards

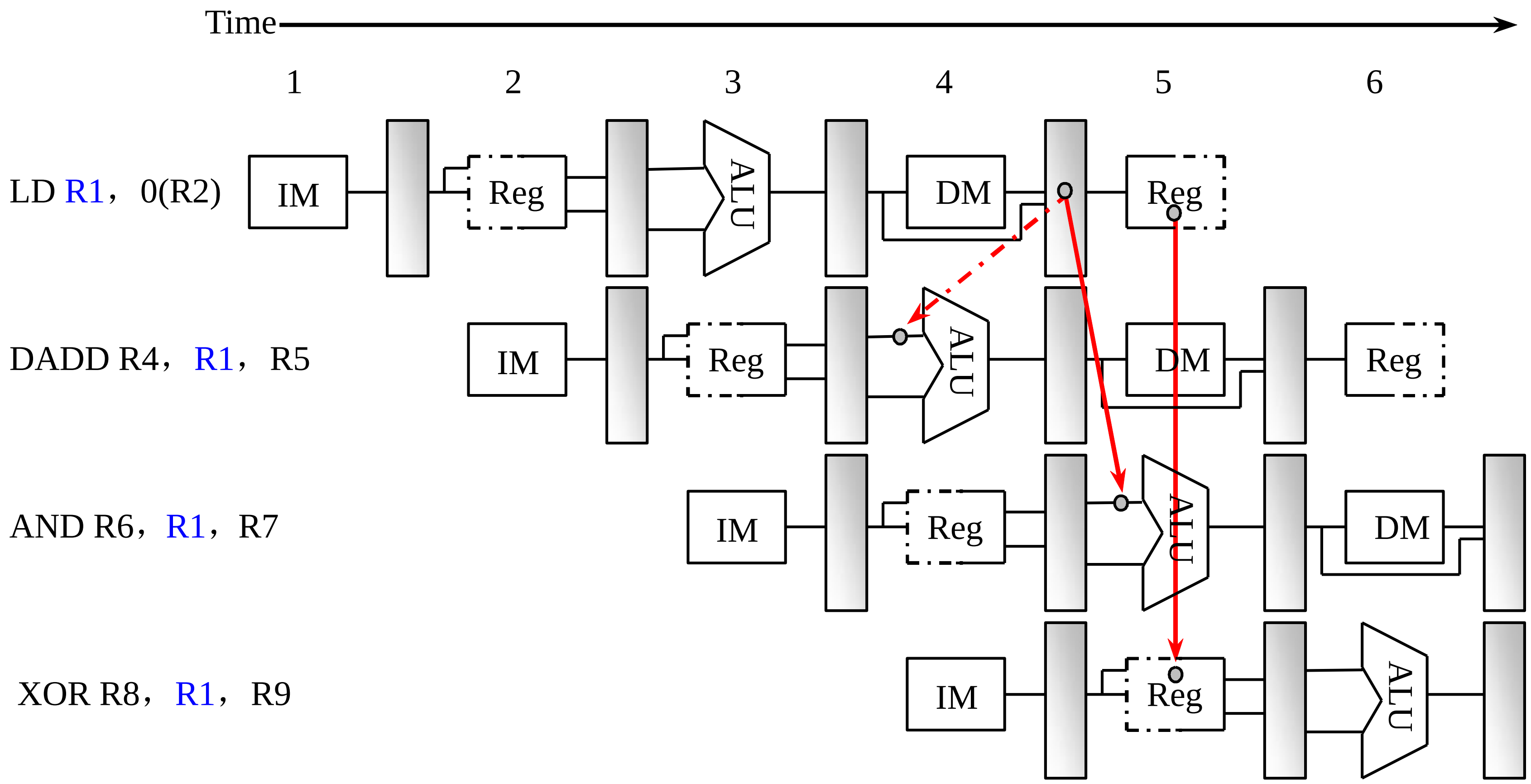


Forwarding (aka Bypassing)

ZHEJIANG UNIVERSITY 1897

Review

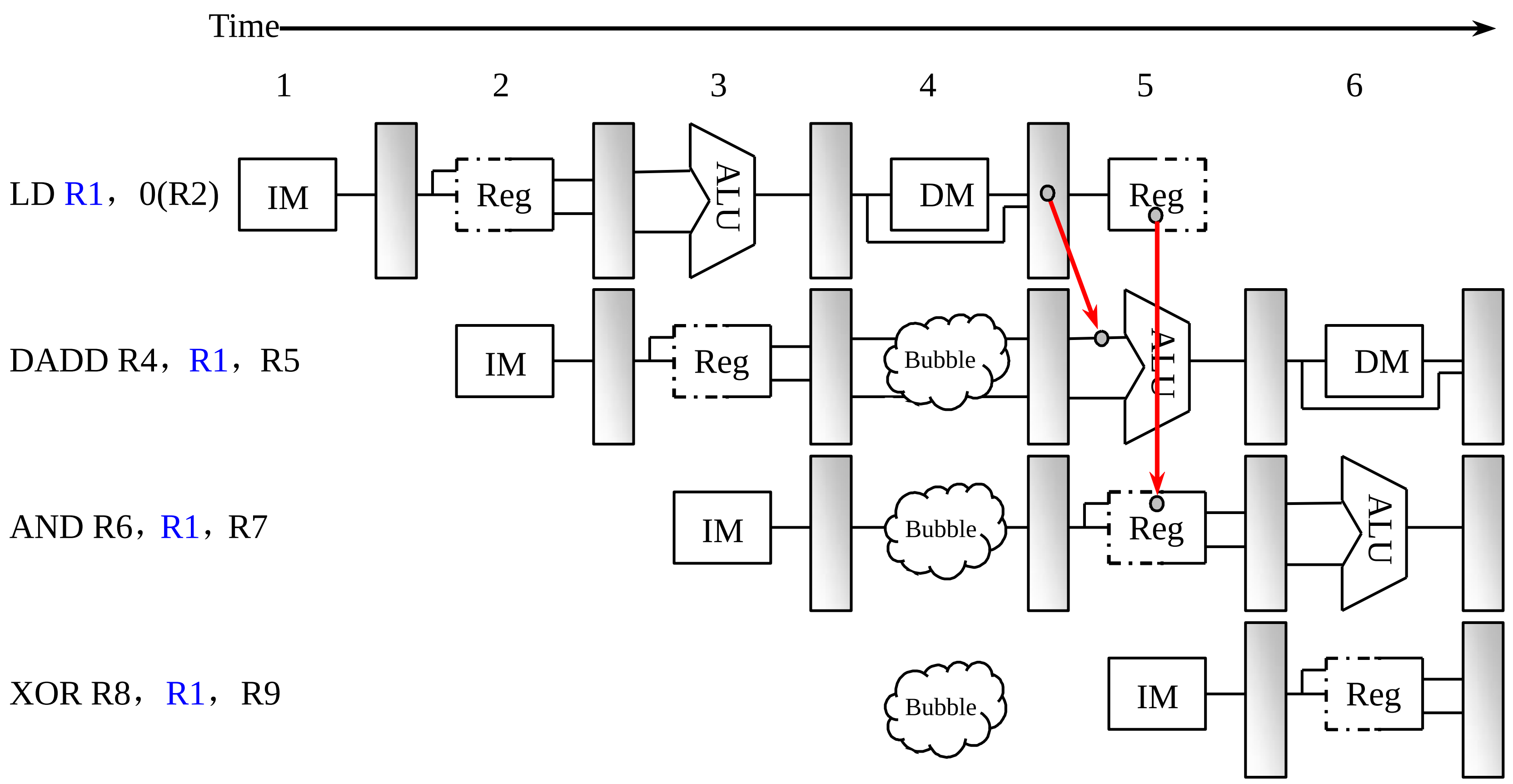
Data Hazards



Forwarding (aka Bypassing)

Review

Data Hazards



Forwarding with bubble

浙江大学 1897 ZHEJIANG UNIVERSITY

Review

Data Hazards

LD R1，0（R2）	IF	ID	EX	MEM	WB			
DADD R4，R1，R5		IF	ID	EX	MEM	WB		
AND R6，R1，R7			IF	ID	EX	MEM	WB	
XOR R8，R1，R9				IF	ID	EX	MEM	WB

LD R1，0（R2）	IF	ID	EX	MEM	WB			
DADD R4，R1，R5		IF	ID	stall	EX	MEM	WB	
AND R6，R1，R7			IF	stall	ID	EX	MEM	WB
XOR R8，R1，R9				stall	IF	ID	EX	MEM

Forwarding with bubble

Review

Data Hazards

A = B + C

LD Rb， B	IF	ID	EX	MEM	WB				
LD Rc， C		IF	ID	EX	EX	MEM	WB	WB	
DADD Ra， Rb， Rc			IF	ID	stall	EX	MEM	WB	
SD Ra ， A				IF	stall	ID	EX	MEM	WB

Code Scheduling to Avoid Stalls

ZHEJIANG UNIVERSITY 1897

Review

Data Hazards

A = B + C

D = E - F

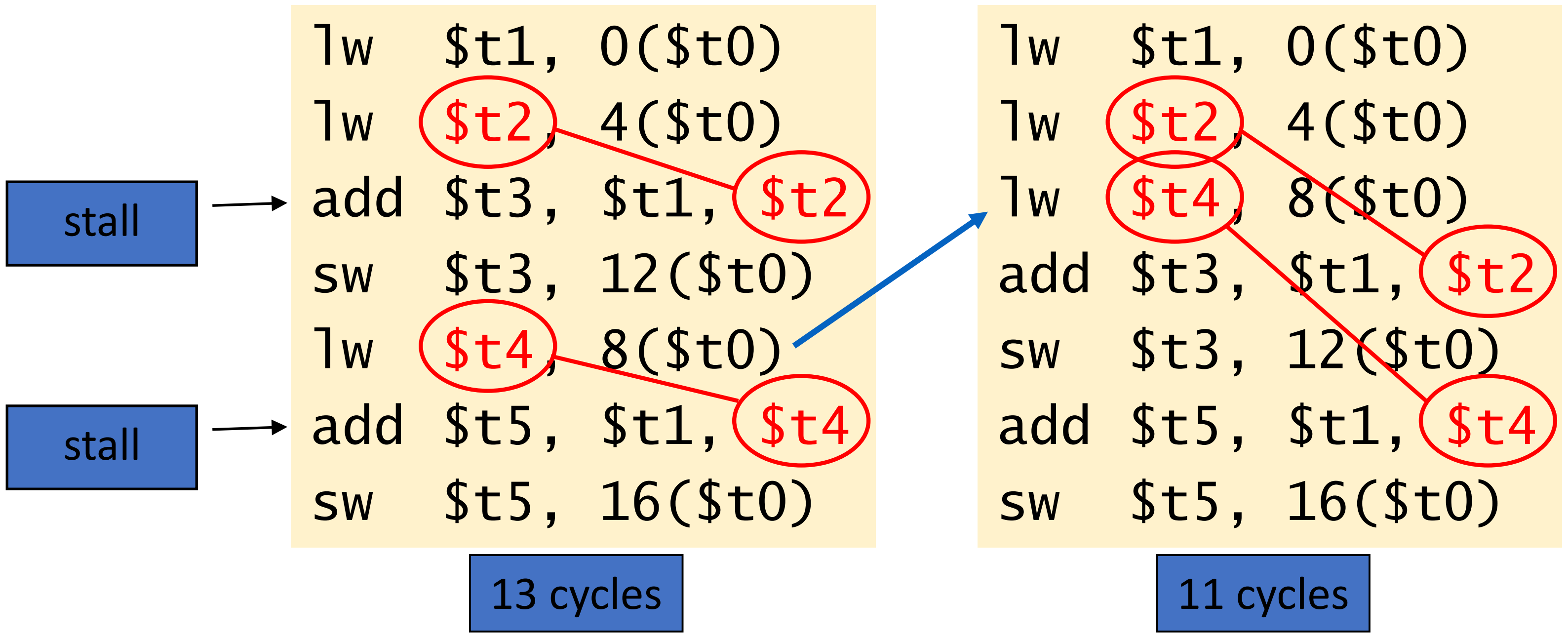
Before Scheduling	After Scheduling
LD Rb， B	
LD Rc， C	
DADD Ra， Rb， Rc	
SD Ra， A	
LD Re， E	
LD Rf， F	
DSUB Rd， Re， Rf	
SD Rd， D	

Code Scheduling to Avoid Stalls

浙江大学 1897 ZHEJIANG UNIVERSITY

Data Hazards

- Reorder code to avoid use of load result in the next instruction
- C code for A = B + E; C = B + F;



```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
lw   $t4, 8($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

```
lw   $t1, 0($t0)
lw   $t2, 4($t0)
lw   $t4, 8($t0)
add  $t3, $t1, $t2
sw   $t3, 12($t0)
add  $t5, $t1, $t4
sw   $t5, 16($t0)
```

Code Scheduling to Avoid Stalls

ZHEJIANG UNIVERSITY 1897

Control Hazards

- Branch determines flow of control
  - Fetching next instruction depends on branch outcome
  - Pipelining can’t always fetch correct instruction
    - Still working on ID stage of branch
- In RISC-V pipelining
  - Need to compare registers and compute target early in the pipelining
  - Add hardware to do it in ID stage

Unconditional Jump	Conditional Branch
Jal - Jump and Link	
Jalr - Jump and Link-Register	

ZHEJIANG UNIVERSITY 1897 浙江大学

Control Hazards: More-Realistic Branch Prediction

- Static branch prediction
  - Based on typical branch behavior
  - Example: loop and if-statement branches
    - Predict backward branches taken
    - Predict forward branches not taken
- Dynamic branch prediction
  - Hardware measures actual branch behavior
    - e.g., record recent history of each branch
  - Assume future behavior will continue the trend
    - When wrong, stall while re-fetching, and update history

Reducing Branch Delay

- Predict branch taken
- Predict branch not taken
- Delayed Branch

Predict Not Taken

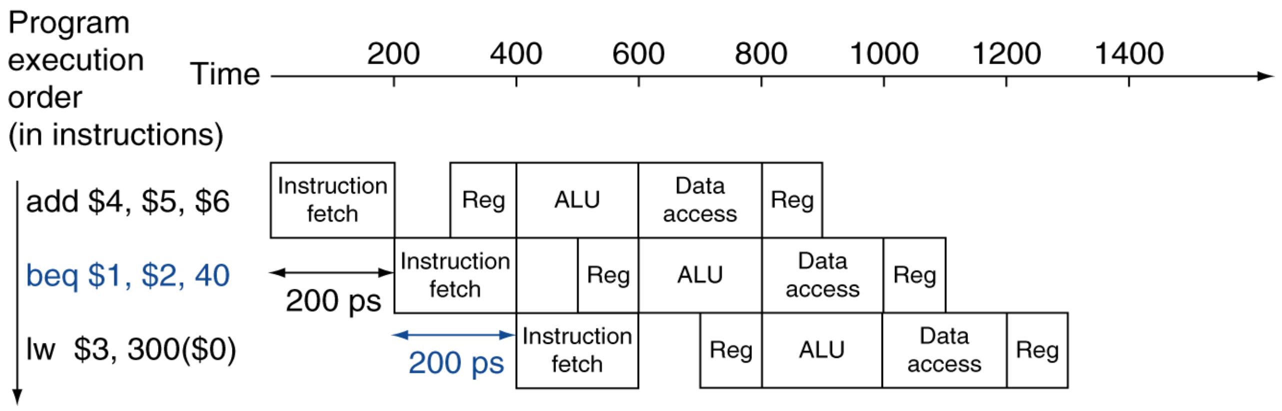
Branch i（correct）	IF	ID	EX	MEM	WB				
Instruction i+1		IF	ID	EX	MEM	WB			
Instruction i+2			IF	ID	EX	MEM	WB		
Instruction i+3				IF	ID	EX	MEM	WB	
Instruction i+4					IF	ID	EX	MEM	WB

Branch i（incorrect）	IF	ID	EX	MEM	WB				
Instruction i+1		IF	stall	stall	stall	stall			
Branch target j			IF	ID	EX	MEM	WB		
Branch target j+1				IF	ID	EX	MEM	WB	
Branch target j+2					IF	ID	EX	MEM	WB

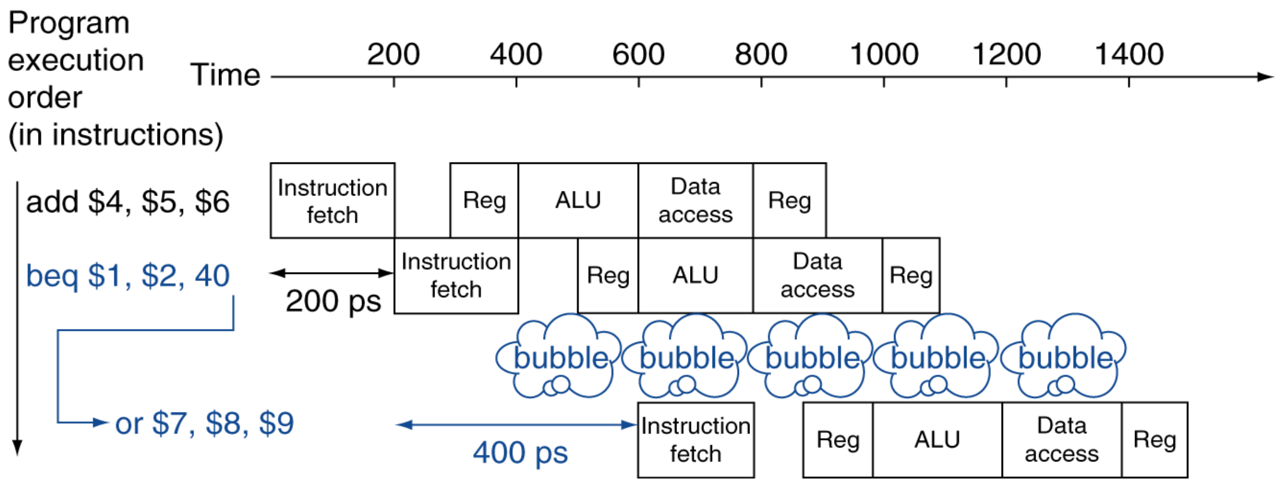
Review

Predict Not Taken

Prediction correct



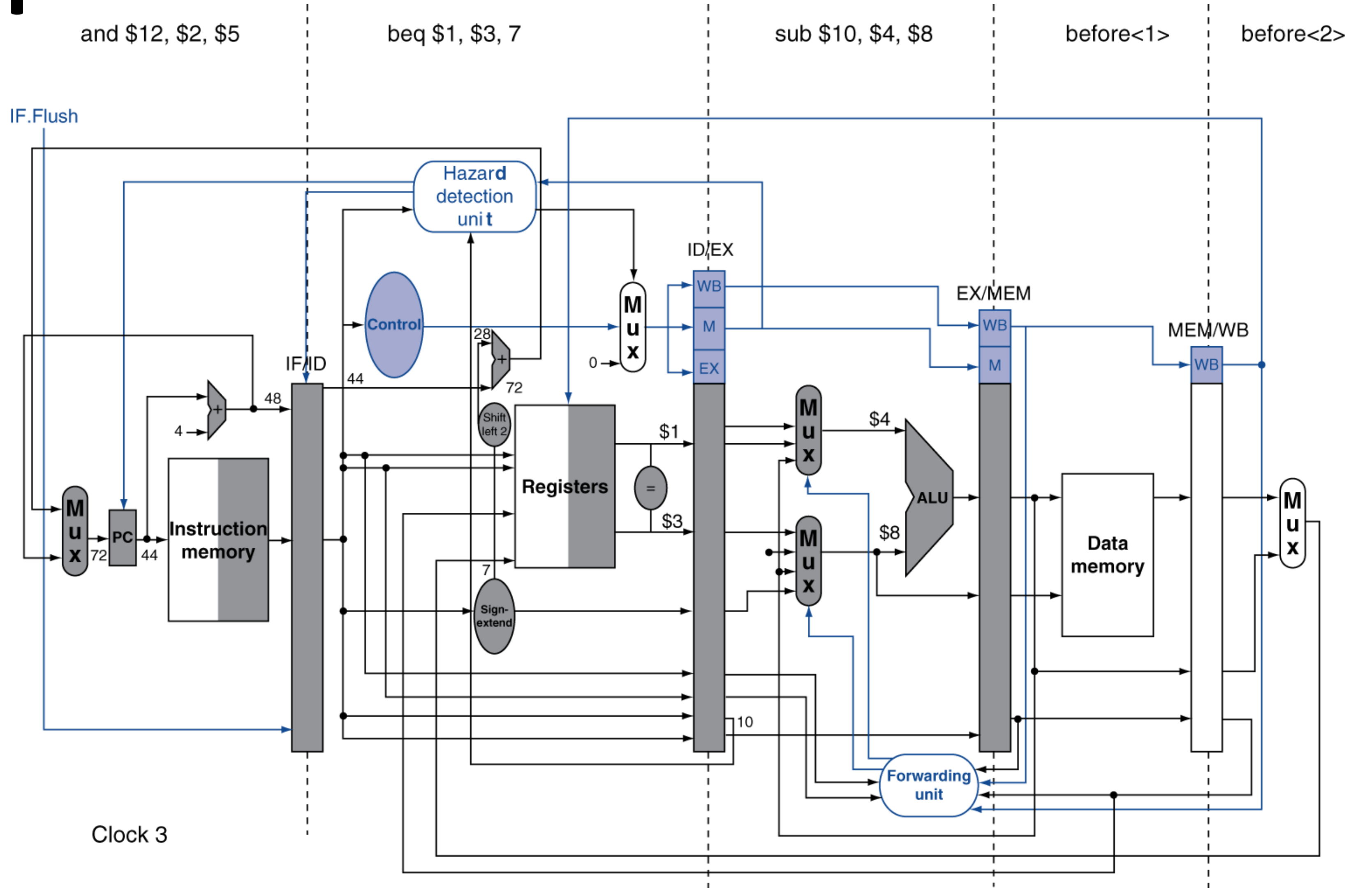
Prediction incorrect



ZHEJIANG UNIVERSITY 1897

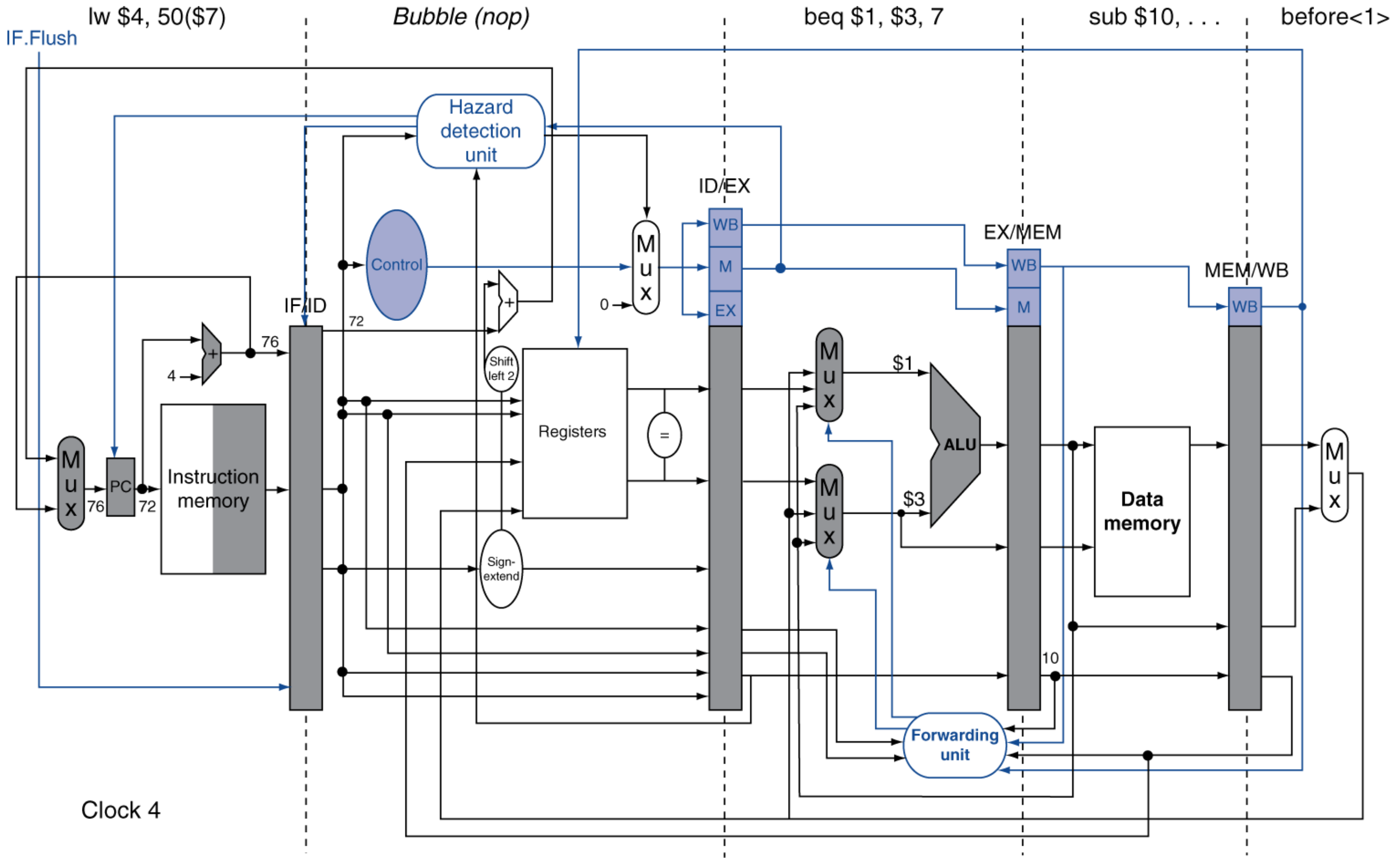
Review

Example: Branch Taken



Review

Example: Branch Taken



ZHEJIANG UNIVERSITY 1897

Review

Data Hazards for Branches

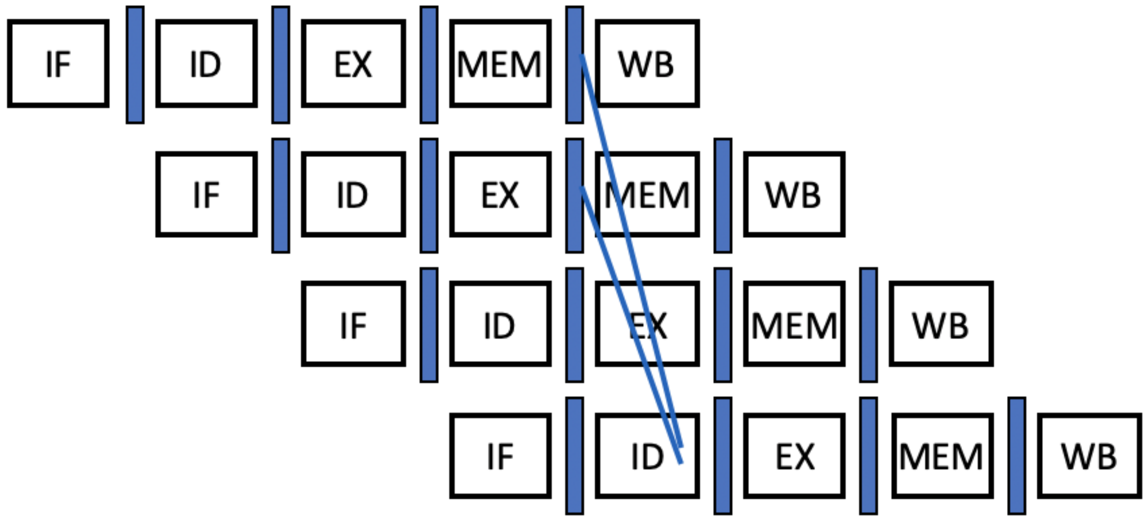
- If a comparison register is a destination of 2nd or 3rd preceding ALU instruction

```
add $1, $2, $3

add $4, $5, $6

…

beq $1, $4, target
```

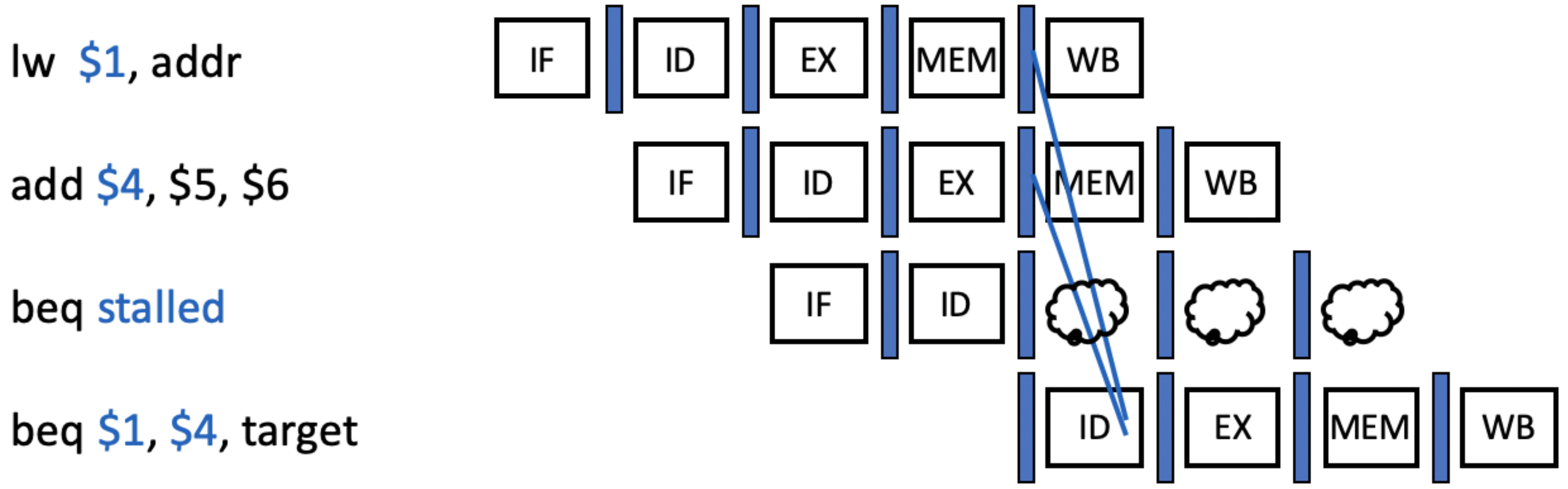


- forwarding

ZHEJIANG UNIVERSITY 1897

Data Hazards for Branches

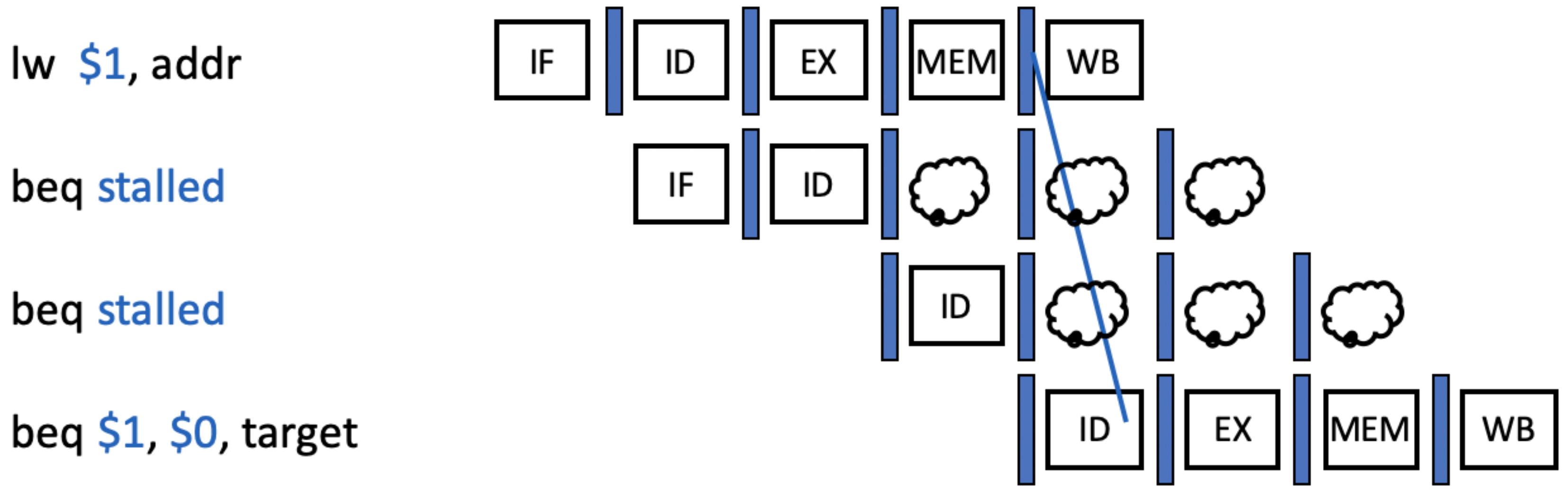
- If a comparison register is a destination of preceding ALU instruction or 2nd preceding load instruction
  - Need 1 stall cycle



ZHEJIANG UNIVERSITY 1897

Data Hazards for Branches

- If a comparison register is a destination of immediately preceding load instruction
  - Need 2 stall cycles



ZHEJIANG UNIVERSITY 1897

Review

Delay slot

Branch Not taken	Branch i	IF	ID	EX	MEM	WB				
	(Delay slot) instruction i+1		IF	ID	EX	MEM	WB			
	instruction i+2			IF	ID	EX	MEM	WB		
	instruction i+3				IF	ID	EX	MEM	WB	
	instruction i+4					IF	ID	EX	MEM	WB

Branch Taken	Branch i	IF	ID	EX	MEM	WB				
	(Delay slot) instruction i+1		IF	ID	EX	MEM	WB			
	Branch target j			IF	ID	EX	MEM	WB		
	Branch target j+1				IF	ID	EX	MEM	WB	
	Branch target j+2					IF	ID	EX	MEM	WB

Question: Is delay slot a really good design?

- "A ***RISC-V ISA*** is defined as a base integer ISA, which must be present in any implementation, plus optional extensions to the base ISA.

- The base integer ISAs are very similar to that of the early RISC processors except ***with no branch delay slots*** and with support for optional variable-length instruction encodings. "

——The RISC-V Instruction Set Manual Volume I

Dynamic Branch Prediction

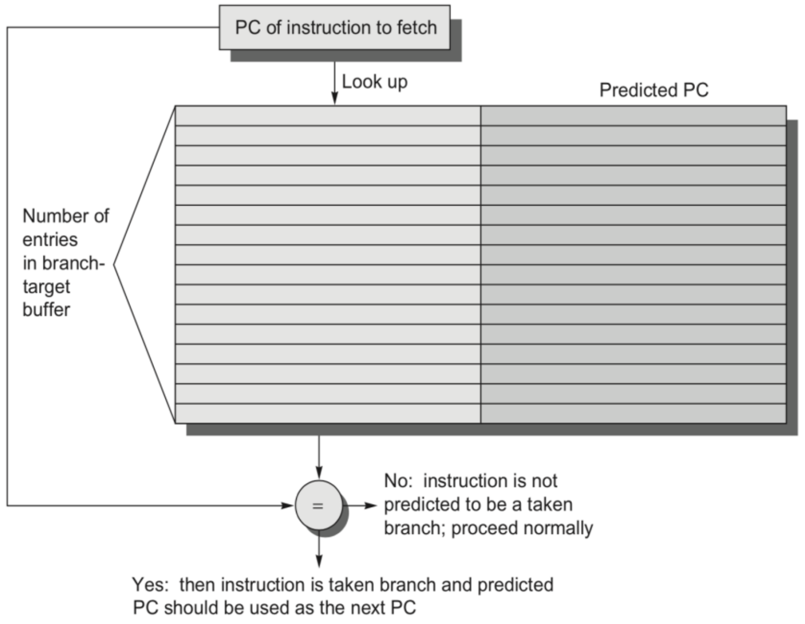
- In deeper and superscalar pipelines, branch penalty is more significant
- Use dynamic prediction
  - Branch prediction buffer (aka branch history table)
  - Indexed by recent branch instruction addresses
  - Stores outcome (taken/not taken)
  - To execute a branch
    - Check table, expect the same outcome
    - Start fetching from fall-through or target
    - If wrong, flush pipeline and flip prediction

Review

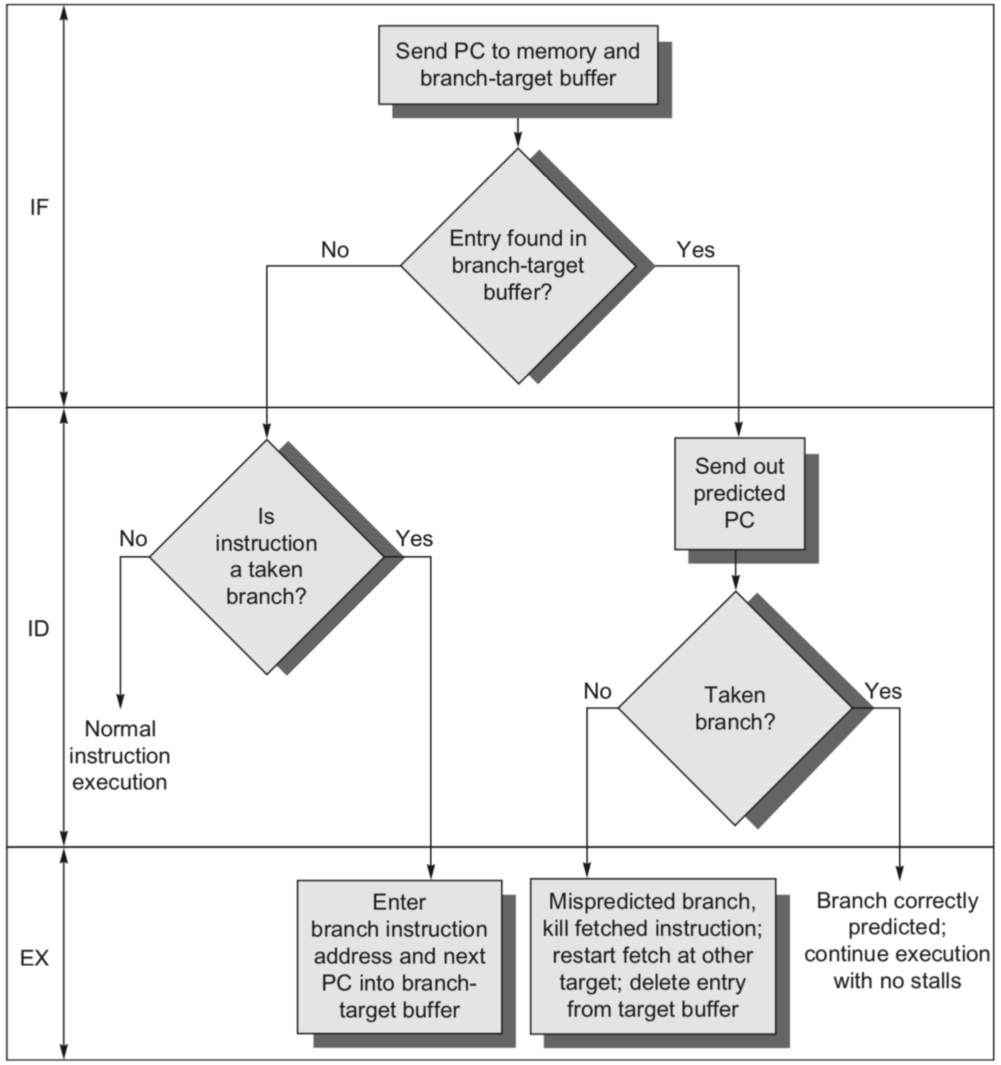
Branch History Table(BHT)

- ☐ 1-Bit Predictor
- ☐ 2-Bit Predictor

Branch-Target Buffers

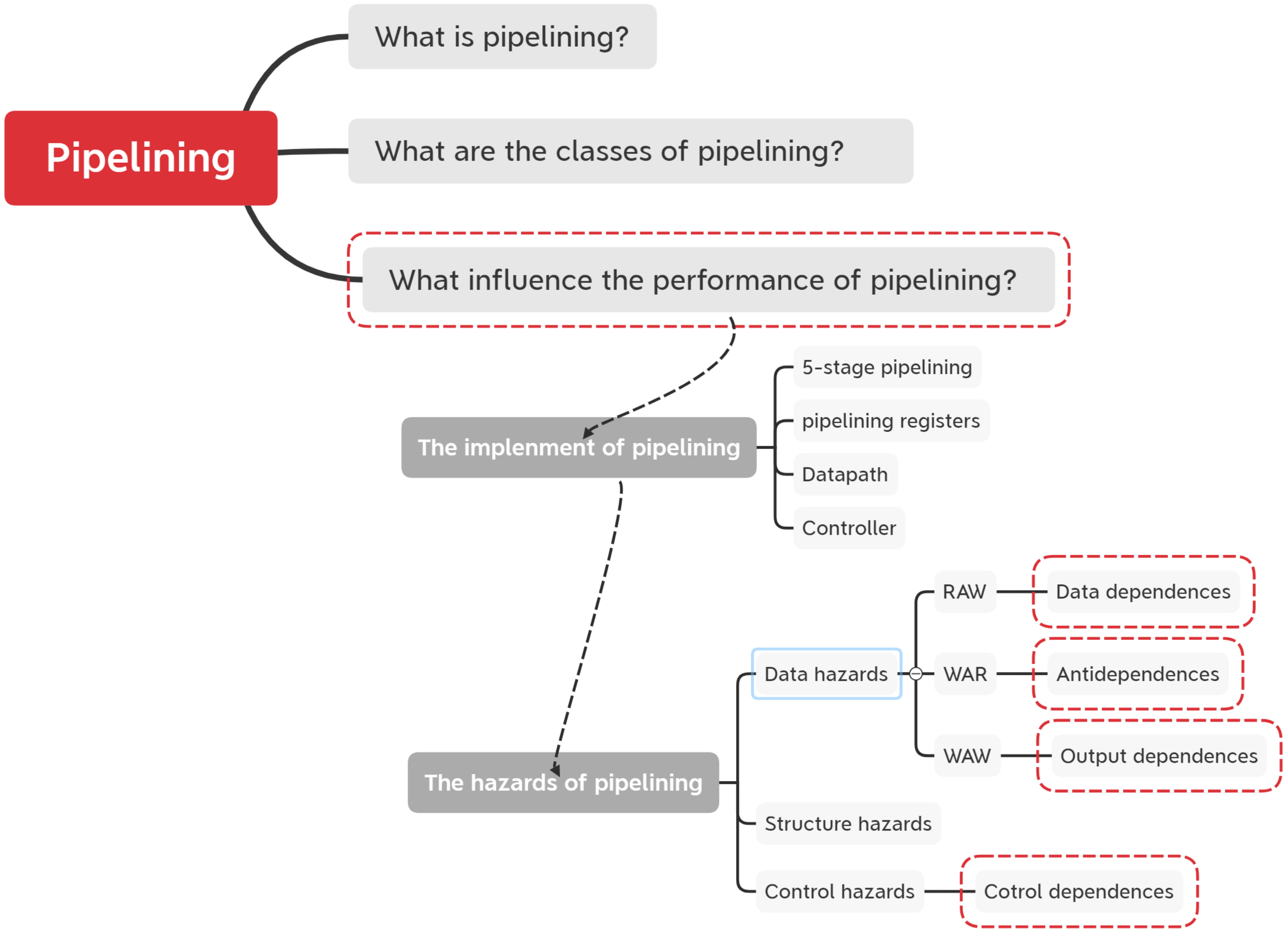


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be a taken branch; proceed normally
Yes: then instruction is taken branch and predicted PC should be used as the next PC



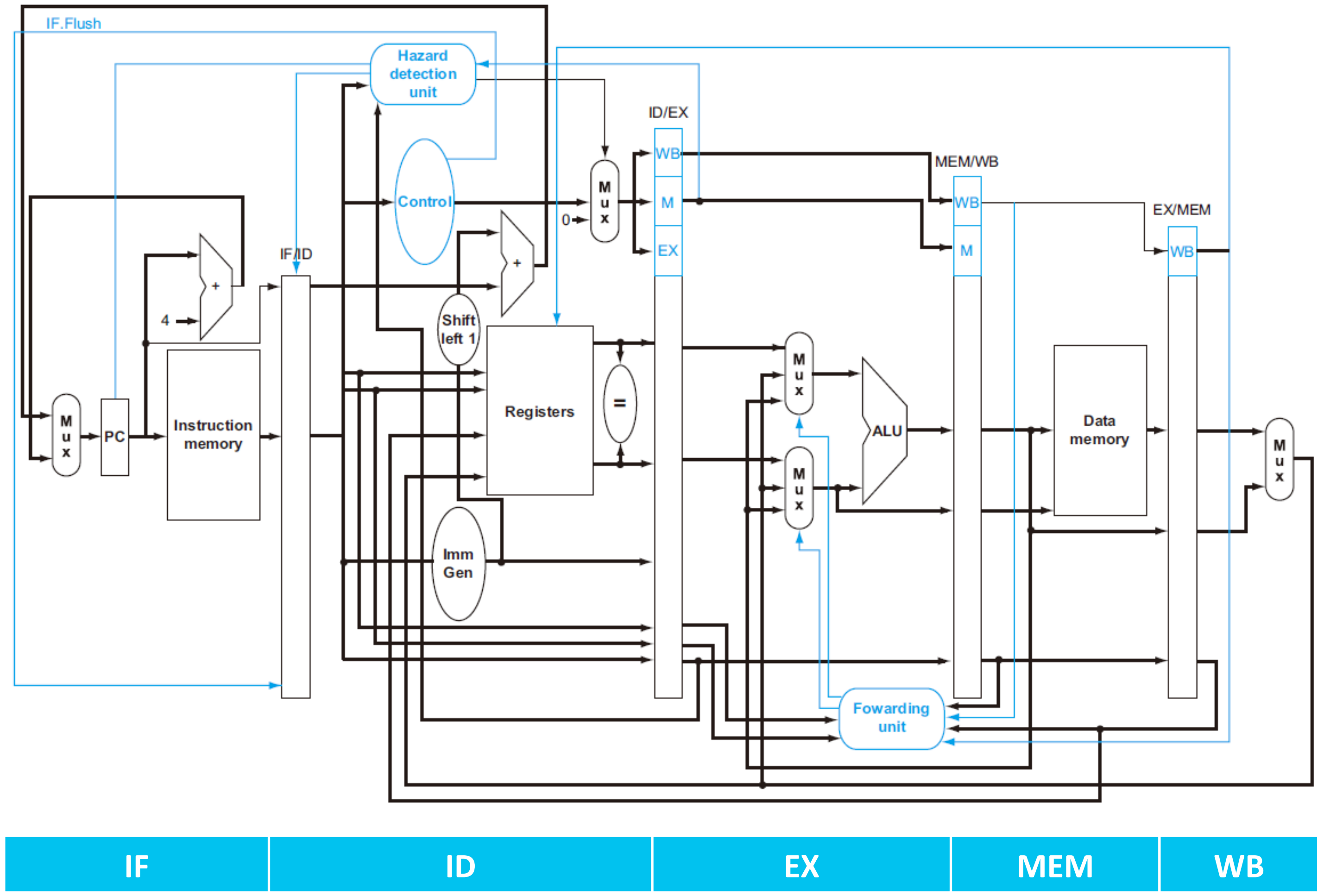
IF
ID
EX
Send PC to memory and branch-target buffer
Entry found in branch-target buffer?
No
Yes
Is instruction a taken branch?
No
Yes
Normal instruction execution
Send out predicted PC
Taken branch?
No
Yes
Enter branch instruction address and next PC into branch-target buffer
Mispredicted branch, kill fetched instruction; restart fetch at other target; delete entry from target buffer
Branch correctly predicted; continue execution with no stalls

Review


Pipelining
What is pipelining?
What are the classes of pipelining?
What influence the performance of pipelining?
The implenment of pipelining
5-stage pipelining
pipelining registers
Datapath
Controller
The hazards of pipelining
Data hazards
RAW
Data dependences
WAR
Antidependences
WAW
Output dependences
Structure hazards
Control hazards
Cotrol dependences

Review

The Classic Five-Stage Pipeline for a RISC Processor



ZHEJIANG UNIVERSITY 1897

Review

The Classic Five-Stage Pipeline for a RISC Processor

- A Simple Implementation of RISC-V

Dependences are a property of *programs*.

- Instructions Dependences
    - Data Dependences
    - Name Dependences
        - Anti-dependence
        - Output-dependence
    - Control Dependences

- Pipeline Hazards
    - Data Hazards
        - RAW
        - WAR
        - WAW
    - Branch Hazards
    - Structural Hazards

Hazard are properties of the *pipeline organization*.

The Classic Five-Stage Pipeline for a RISC Processor

Consider this code:

```
FADD.D  R1,  R2,  R4
FADD.D  R2,  R1,  1
FSUB.D  R1,  R4,  R5
```

(1) Point out all the pipeline Hazards (RAW, WAR, WAW).
(2) Analyze the hazards and give your solutions.

Dynamic Scheduling

A major limitation of simple pipelining techniques is that:

- they use in-order instruction issue and execution
- For example, consider this code:

```
FDIV.D      F4,  F0,  F2
FSUB.D      F10, F4,  F6
FADD.D      F12, F6,  F14
```

The FADD.D instruction cannot execute because the dependence of FSUB.D on FDIV.D causes the pipeline to stall; yet, FADD.D is not data dependent on anything in the pipeline.

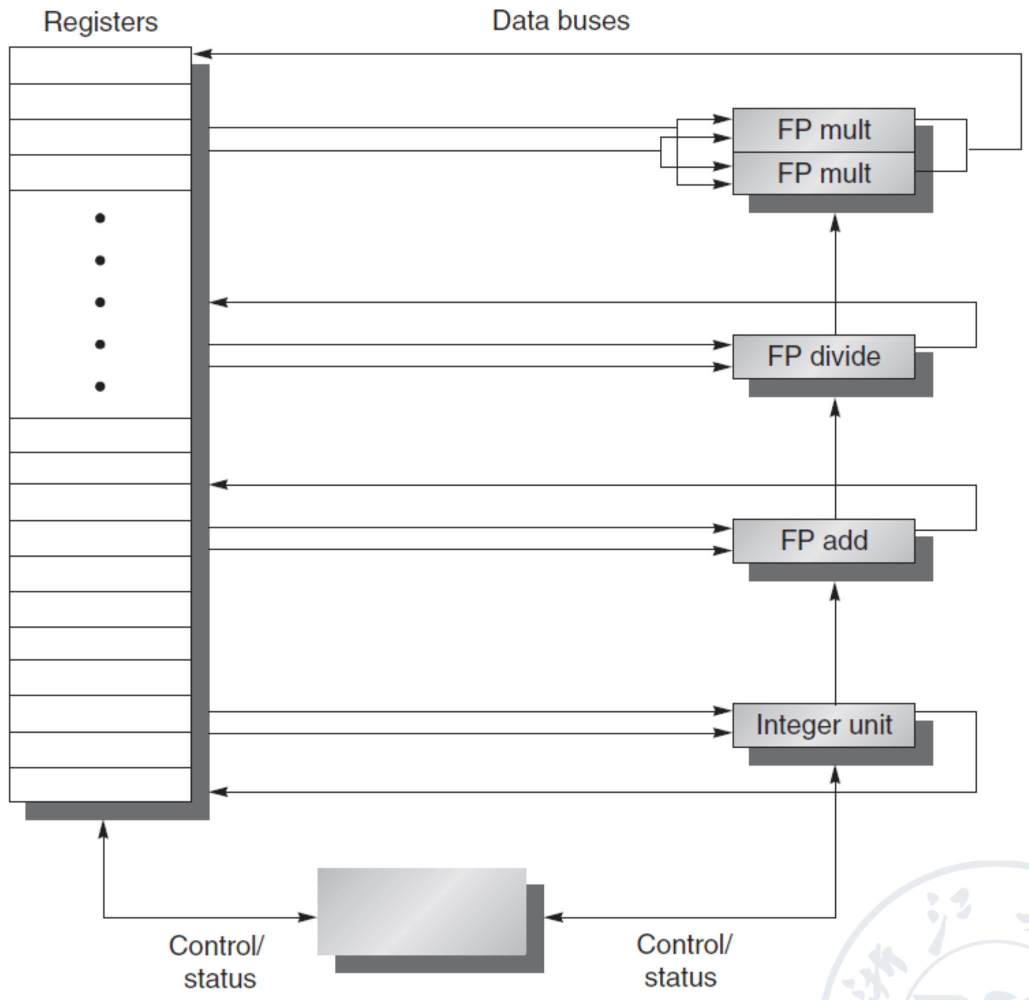
Instructions are issued in program order, and if an instruction is stalled in the pipeline no later instructions can proceed.

1897 ZHEJIANG UNIVERSITY

Dynamic Scheduling

Idea: Dynamic Scheduling

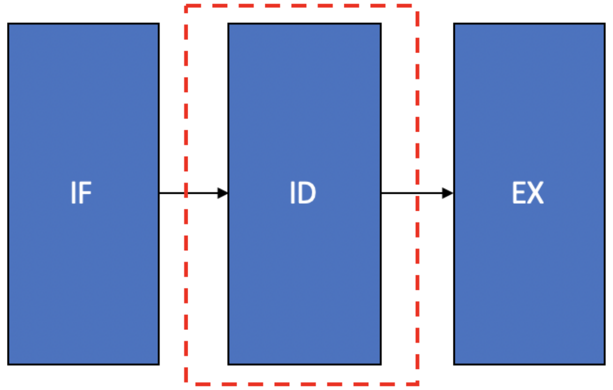
Method: out-of-order execution



Dynamic Scheduling with a Scoreboard

Dynamic Scheduling

- A Simple Implementation of RISC-V



Check structural hazards

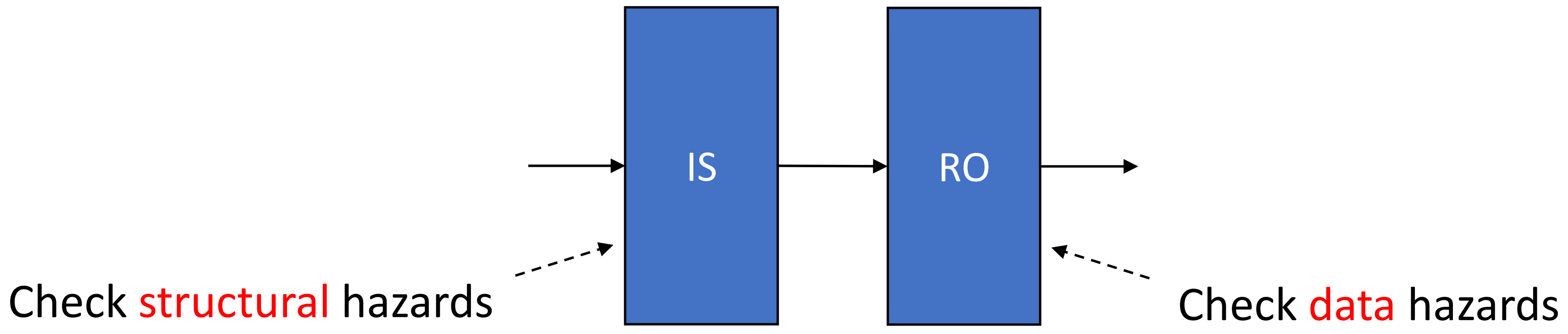
Check data hazards

- When an instruction could execute without hazards, it was issued from ID knowing that all data hazards had been resolved.

浙江大学 1897 ZHEJIANG UNIVERSITY

Dynamic Scheduling

- To allow out-of-order execution, we essentially split the ID pipe stage of our simple five-stage pipeline into two stages：
  - Issue(IS): Decode instructions, check for structural hazards. (in-order issue)
  - Read Operands(RO): Wait until no data hazards, then read operands. (out of order execution)



ZHEJIANG UNIVERSITY 1897 浙江大学

Dynamic Scheduling

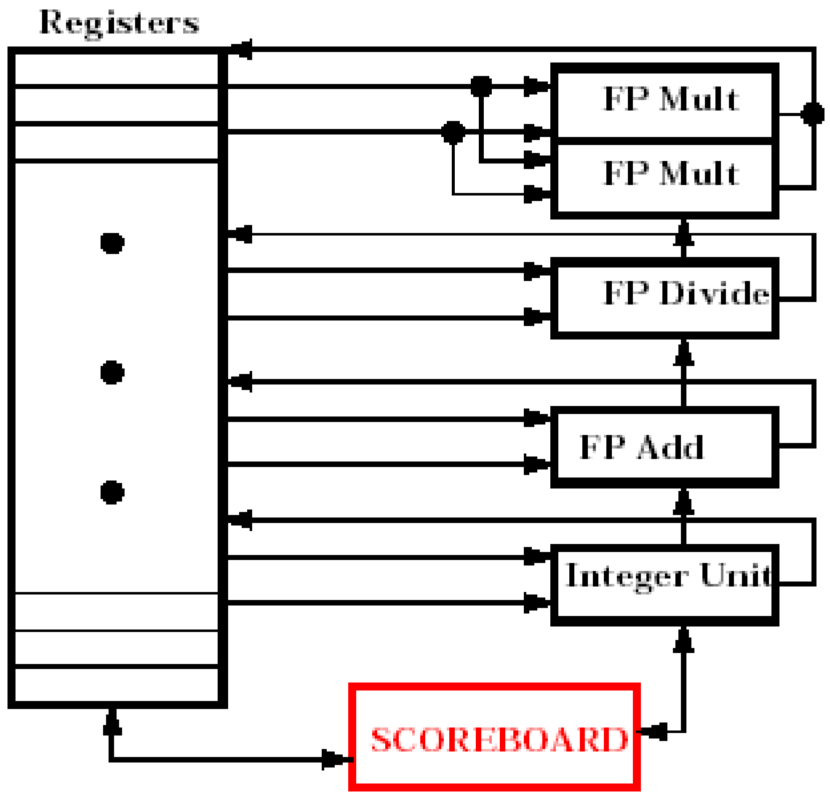
- Out-of-order execution introduces the possibility of WAR and WAW hazards, which do not exist in the five-stage integer pipeline and its logical extension to an in-order floating-point pipeline.
  - Consider the following RISC-V floating-point code sequence:

```
WAW ┌ FDIV.D   F10, F0, F2
    └ FSUB.D   F10, F4, F6  ┐
      FADD.D   F6, F8, F14  ┘ WAR
```

- Scoreboard algorithm is an approach to schedule the instructions.
- Robert Tomasulo introduces register renaming in hardware to minimize WAW and WAR hazards, named Tomasulo's Approach.

1897 ZHEJIANG UNIVERSITY

Dynamic Scheduling: Scoreboard algorithm



IS

ID

RO

EX

WB

The basic structure of a processor with scoreboard

ZHEJIANG UNIVERSITY 1897

Dynamic Scheduling: Scoreboard algorithm

- Show what the information tables look like for the following code sequence when only the first load has completed and written its result:

```
FLD        F6, 34 (R2)
FLD        F2, 45 (R3)
FMUL.D     F0, F2, F4
FSUB.D     F8, F2, F6
FDIV.D     F10, F0, F6
FADD.D     F6, F8, F2
```

Dynamic Scheduling: Scoreboard algorithm

Instruction		Instruction Status			
		IS	RO	EX	WB
FLD	F6, 34(R2)	√	√	√	√
FLD	F2, 45(R3)	√	√	√	
FMUL.D	F0, F2, F4	√			
FSUB.D	F8, F6, F2	√			
FDIV.D	F10, F0, F6	√			
FADD.D	F6, F8, F2				

ZHEJIANG UNIVERSITY 1897

Name	Function Component Status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	yes	Load	F2	R3				no	
Mult1	yes	MUL	F0	F2	F4	Integer		no	yes
Mult2	no								
Add	yes	SUB	F8	F6	F2		Integer	yes	no
Divide	yes	DIV	F10	F0	F6	Mult1		no	yes

Rj, Rk : “ yes” ——operand is ready but not read;
“no” & “Qj = null” ——operand is read;
“no” & “Qj != null” ——operand is not ready.

	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1	Integer			Add	Divide		

浙江大学 1897 ZHEJIANG UNIVERSITY

Dynamic Scheduling: Scoreboard algorithm

Instruction		Instruction Status			
		IS	RO	EX	WB
FLD	F6, 34(R2)	√	√	√	√
FLD	F2, 45(R3)	√	√	√	√
FMUL.D	F0, F2, F4	√	√	√	
FSUB.D	F8, F6, F2	√	√	√	√
FDIV.D	F10, F0, F6	√			
FADD.D	F6, F8, F2	√	√	√	

Show what the status tables look like when the FMUL.D is ready to write its result.

1897 ZHEJIANG UNIVERSITY

Name	Function Component Status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	yes	MUL	F0	F2	F4			no	no
Mult2	no								
Add	yes	ADD	F8	F6	F2			no	no
Divide	yes	DIV	F10	F0	F6	Mult1		no	yes

	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1			Add		Divide		

Show what the status tables look like when the FMUL.D is ready to write its result.

Dynamic Scheduling: Scoreboard algorithm

Instruction		Instruction Status			
		IS	RO	EX	WB
FLD	F6, 34(R2)	√	√	√	√
FLD	F2, 45(R3)	√	√	√	√
FMUL.D	F0, F2, F4	√	√	√	√
FSUB.D	F8, F6, F2	√	√	√	√
FDIV.D	F10, F0, F6	√	√	√	
FADD.D	F6, F8, F2	√	√	√	√

Show what the status tables look like when the FDIV.D is ready to write its result.

§2.1 Dynamic Scheduling

Name	Function Component Status								
	Busy	Op	Fi	Fj	Fk	Qj	Qk	Rj	Rk
Integer	no								
Mult1	no								
Mult2	no								
Add	no								
Divide	yes	DIV	F10	F0	F6			no	no

	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
Qi						Divide		

Show what the status tables look like when the FDIV.D is ready to write its result.

浙江大学 1897 ZHEJIANG UNIVERSITY

Dynamic Scheduling: The Idea

- Consider the following RISC-V floating-point code sequence:

```
                     FDIV.D   F0, F2, F4
Anti-dependence    { FADD.D   F6, F0, F8    }
  WAR hazards      { FSD      F6, 0(R1)     }  Output-dependence
     (F8)          { FSUB.D   F8, F10, F14  }     WAR hazards
                     FMUL.D   F6, F10, F8   }        (F6)
```

Tomasulo’s Approach

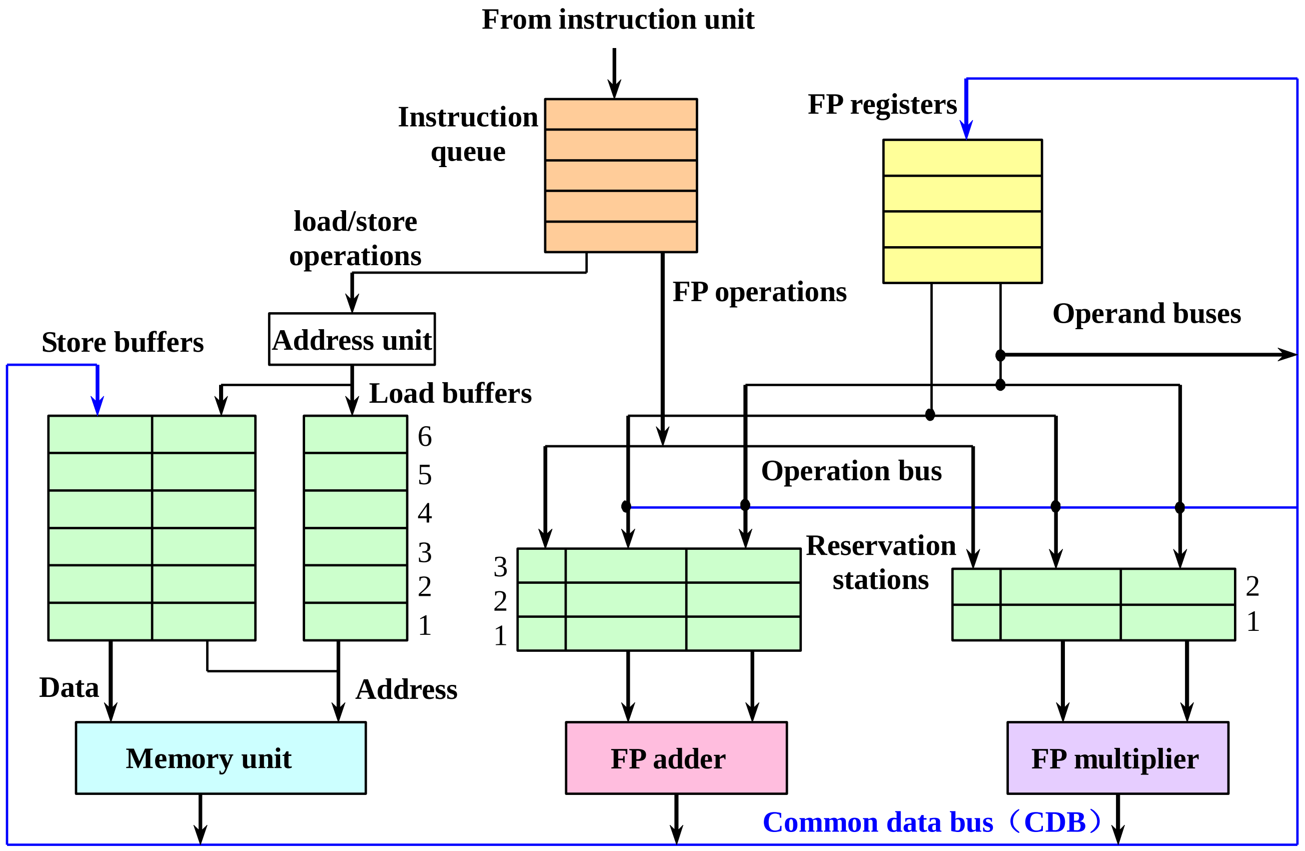
- These name dependences can all be eliminated by register renaming.
  - Assume the existence of two temporary registers, S and T.
  - The sequence can be rewritten without any dependences as：

```
FDIV.D  F0, F2, F4
FADD.D  S, F0, F8       }
FSD     S, 0(R1)        }  F6 change as S
FSUB.D  T, F10, F14     }
FMUL.D  F6, F10, T      }  F8 change as T
```

Who finish the register renaming and how?

浙江大学 1897 ZHEJIANG UNIVERSITY

Tomasulo's Approach



The basic structure of a floating-point unit using Tomasulo's algorithm

浙江大学 1897 ZHEJIANG UNIVERSITY

Tomasulo’s Approach: Main Idea

- It tracks when operands for instructions are available to minimize RAW hazards;
- It introduces register renaming in hardware to minimize WAW and WAR hazards.

Tomasulo’s Approach

- Let’s look at the three steps an instruction goes through：
  - Issue： Get the next instruction from the head of the instruction queue (FIFO)
  - If there is a matching reservation station that is empty, issue the instruction to the station with the operand values, if they are currently in the registers.
  - If there is not an empty reservation station, then there is a structural hazard and the instruction stalls until a station or buffer is freed.
  - If the operands are not in the registers, keep track of the functional units that will produce the operands.

- This step renames registers, eliminating WAR and WAW hazards not in the registers.

浙江大学 1897 ZHEJIANG UNIVERSITY

Tomasulo’s Approach

Execute

- When all the operands are available, the operation can be executed at the corresponding functional unit.

- Load and store require a two-step execution process：
  - It computes the effective address when the base register is available.
  - The effective address is then placed in the load or store buffer.

Tomasulo’s Approach

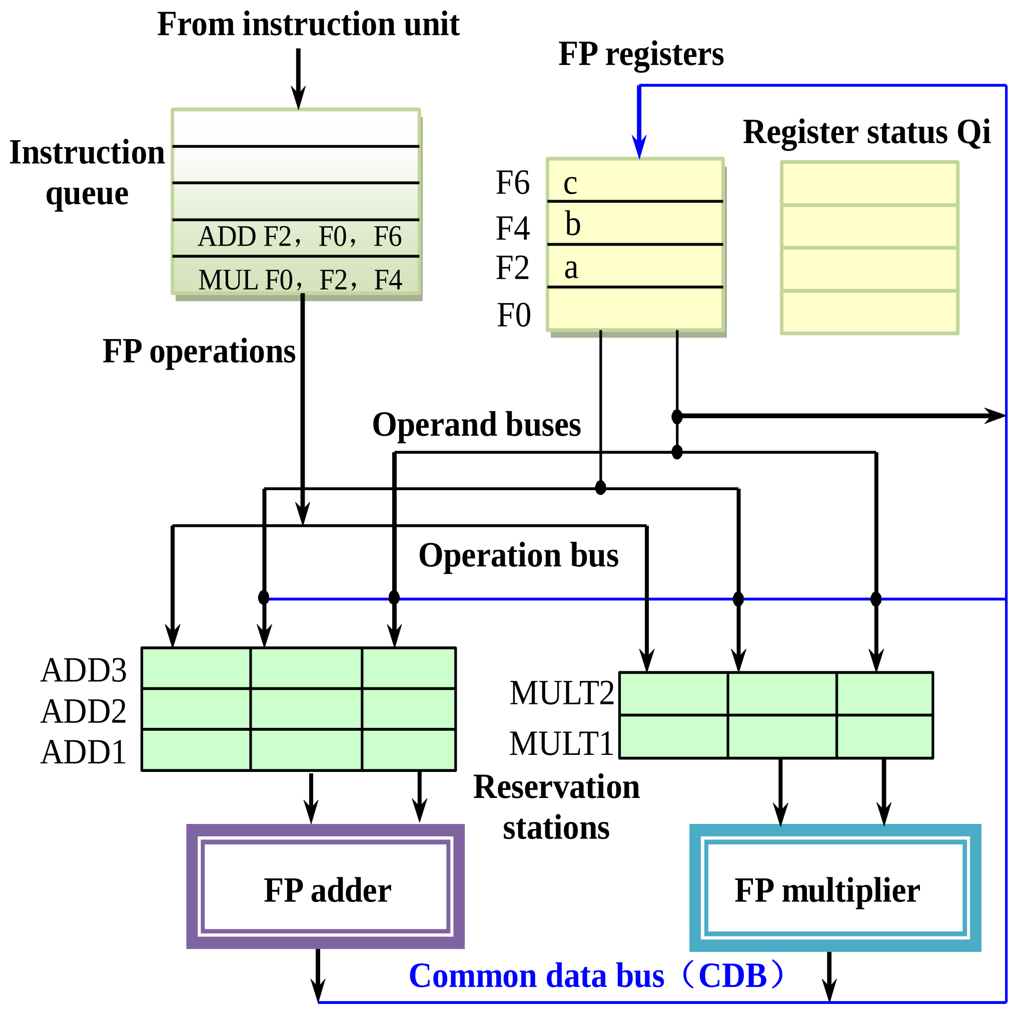
Write results

- When the result is available, write it on the CDB and from there into the registers and into any reservation stations (including store buffers).

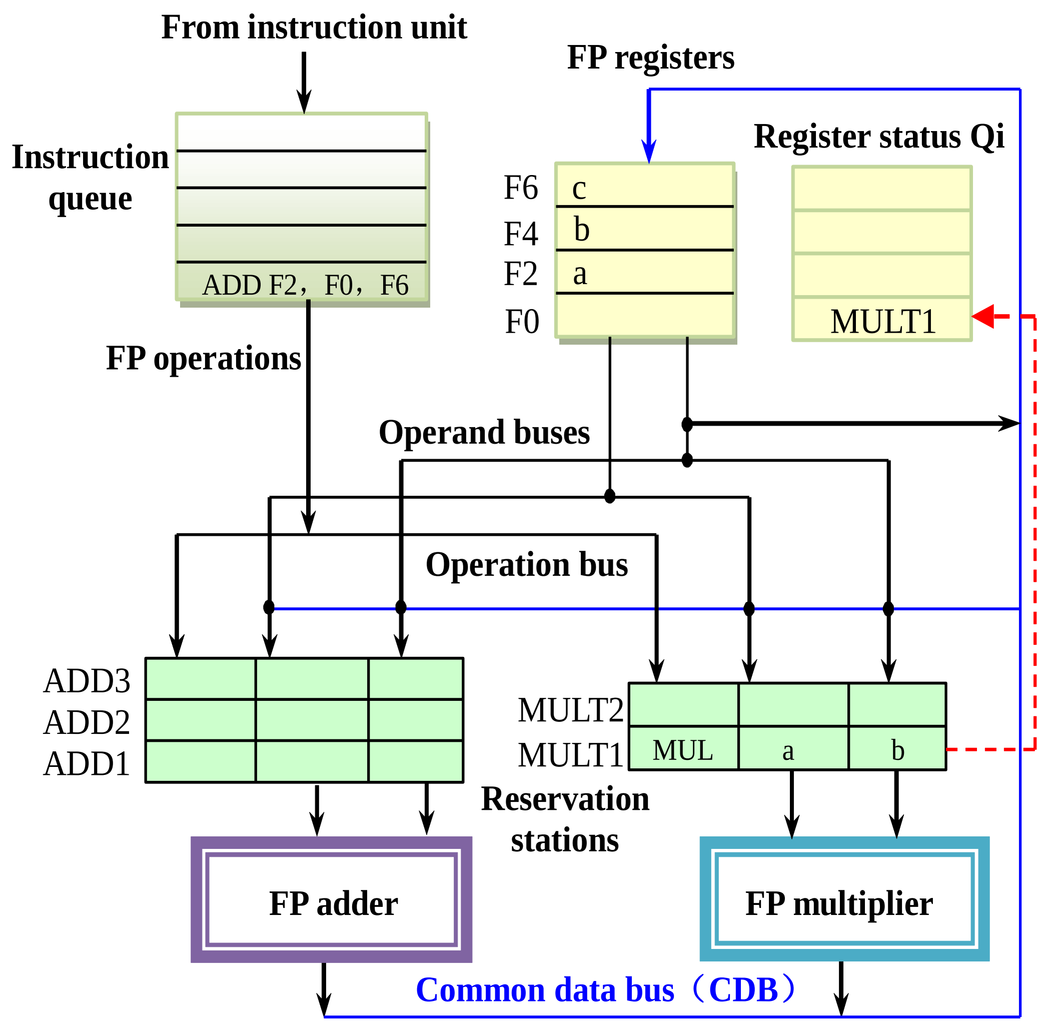
- Stores are buffered in the store buffer until both the value to be stored and the store address are available, then the result is written as soon as the memory unit is free.

ZHEJIANG UNIVERSITY 1897

Tomasulo’s Approach

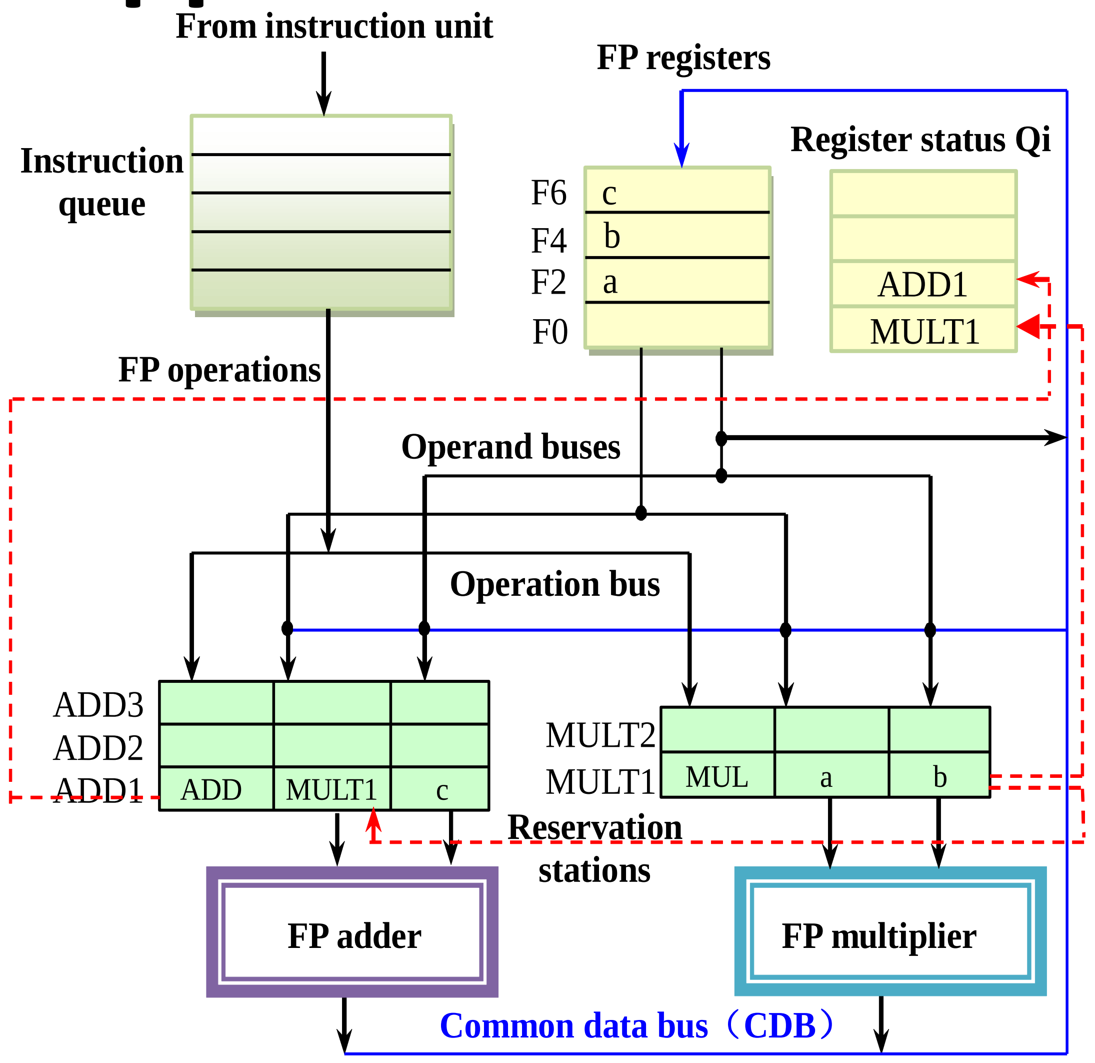


Tomasulo's Approach

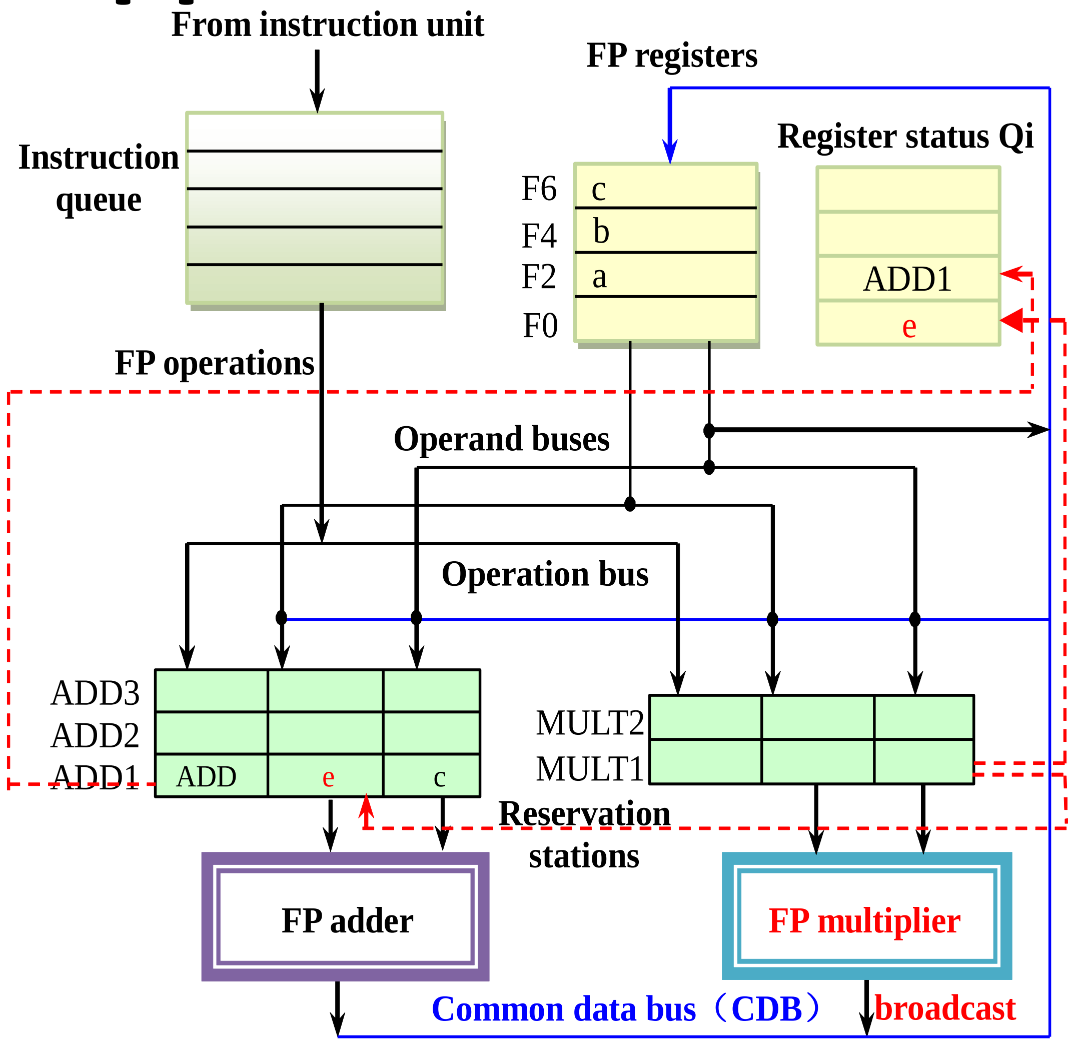


1897 ZHEJIANG UNIVERSITY

Tomasulo’s Approach



Tomasulo’s Approach



ZHEJIANG UNIVERSITY 1897

Tomasulo’s Approach

There are three tables for Tomasulo’s Approach.

- Instruction status table: This table is included only to help you understand the algorithm; it is not actually a part of the hardware.
- Reservation stations table: The reservation station keeps the state of each operation that has issued.
- Register status table (Field Qi): The number of the reservation station that contains the operation whose result should be stored into this register.

Tomasulo’s Approach

Each reservation station has seven fields:

Op: The operation to perform on source operands.

Qj, Qk: The reservation stations that will produce the corresponding source operand.

Vj, Vk: The value of the source operands.

Busy: Indicates that this reservation station and its accompanying functional unit are occupied.

A: Used to hold information for the memory address calculation for a load or store.

ZHEJIANG UNIVERSITY 1897

Tomasulo's Algorithm and Examples

- Show what the information tables look like for the following code sequence when only the first load has completed and written its result:

```
FLD        F6, 34 (R2)
FLD        F2, 45 (R3)
FMUL.D     F0, F2, F4
FSUB.D     F8, F2, F6
FDIV.D     F10, F0, F6
FADD.D     F6, F8, F2
```

Dynamic Scheduling: Tomasulo’s algorithm

Instruction		Instruction Status		
		Issue	Execute	Write Result
FLD	F6, 34(R2)	√	√	√
FLD	F2, 45(R3)	√	√	
FMUL.D	F0, F2, F4	√		
FSUB.D	F8, F6, F2	√		
FDIV.D	F10, F0, F6	√		
FADD.D	F6, F8, F2	√		

浙江大学 1897 ZHEJIANG UNIVERSITY

Name	Function Component Status						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					**45+Regs[R3]**
Add1	Yes	SUB		**Mem[34+Regs[R2]]**	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		**Reg[F4]**	Load2		
Mult2	Yes	DIV		**Mem[34+Regs[R2]]**	Mult1		

	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1	Load2		Add2	Add1	Mult2		

ZHEJIANG UNIVERSITY 1897 浙江大学

Dynamic Scheduling: Tomasulo's algorithm

Instruction		Instruction Status		
		Issue	Execute	Write Result
FLD	F6, 34(R2)	√	√	√
FLD	F2, 45(R3)	√	√	√
FMUL.D	F0, F2, F4	√	√	
FSUB.D	F8, F6, F2	√	√	√
FDIV.D	F10, F0, F6	√		
FADD.D	F6, F8, F2	√	√	√

Show what the status tables look like when the FMUL.D is ready to write its result.

浙江大学 ZHEJIANG UNIVERSITY 1897

Name	Function Component Status						
	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	No						
Add1	No						
Add2	No						
Add3	No						
Mult1	Yes	MUL	**Mem[45+Regs[R3]]**	**Reg[F4]**			
Mult2	Yes	DIV		**Mem[34+Regs[R2]]**	Mult1		

	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
Qi	Mult1					Mult2		

浙江大学 1897 ZHEJIANG UNIVERSITY

Summary

1. Tomasula’s Algorithm main contributions

- Dynamic scheduling
- Register renaming---eliminatining WAW and WAR hazards
- Load/store disambiguation
- Better than Scoreboard Algorithm

Summary

2. Tomasulo's Algorithm major defects

- Structural complexity.
- Its performance is limited by Common Data Bus.
- A load and a store can safely be done out of order, provided they access different addresses. If a load and a store access the same address, then either:
  - The load is before the store in program order and interchanging them results in a WAR hazard, or
  - The store is before the load in program order and interchanging them results in a RAW hazard
  - Interchanging two stores to the same address results in a WAW hazard

Summary

3. The limitations on ILP approaches directly led to the movement to multicore.

Question

Does out-of-order execution mean out-of-order completion?

Homework

Suppose：

Add instruction needs 2 clock cycles. Multiply instruction needs 10 clock cycles. Division instruction needs 40 clock cycles. LD instruction need 1 clock cycles.

```
 FLD         F6, 34（R2）
 FLD         F2, 45（R3）
FMUL.D       F0, F2, F4
FSUB.D       F8, F2, F6
FDIV.D       F10, F0, F6
FADD.D       F6, F8, F2
```

How many cycles does it take to finish each instruction using the following two methods?

(1) Scoreboard algorithm

(2) Tomasulo's approach