Chapter 4

Instruction-Level Parallelism (ILP)

The Classic Five-Stage Pipeline for a RISC Processor

- A Simple Implementation of RISC-V
- Instructions Dependences
  - Data Dependences
  - Name Dependences
    - Anti-dependence
    - Output-dependence
  - Control Dependences

Dependences are a property of *programs.*

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

Hazard are properties of the *pipeline organization.*

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
```

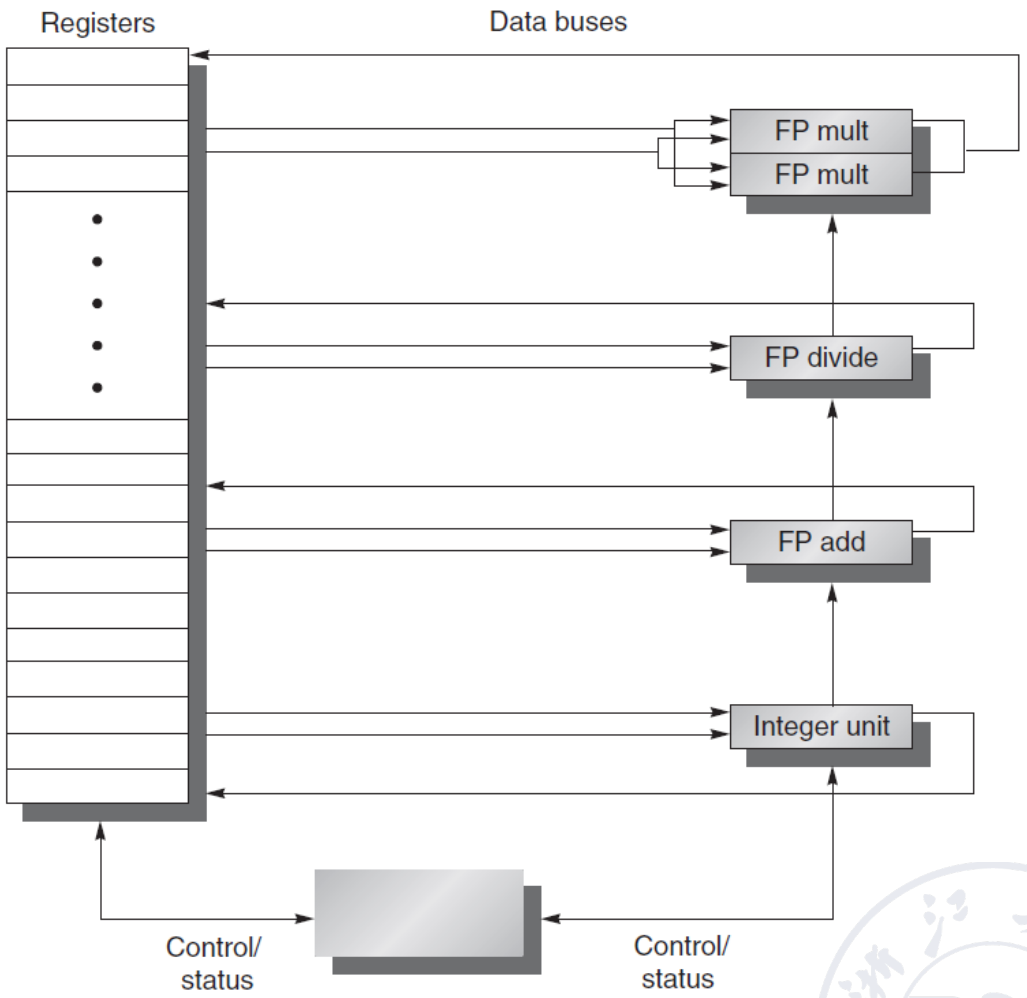
The FADD.D instruction cannot execute because the dependence of FSUB.D on FDIV.D causes the pipeline to stall; yet, FADD.D is not data dependent on anything in the pipeline.

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

Dynamic Scheduling

Idea: Dynamic Scheduling

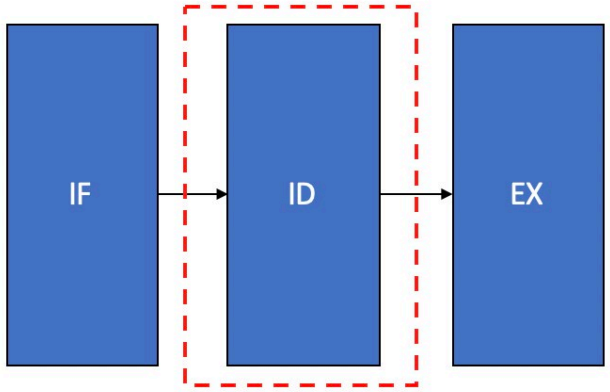
Method: out-of-order execution



Dynamic Scheduling with a Scoreboard

Dynamic Scheduling

- A Simple Implementation of RISC-V



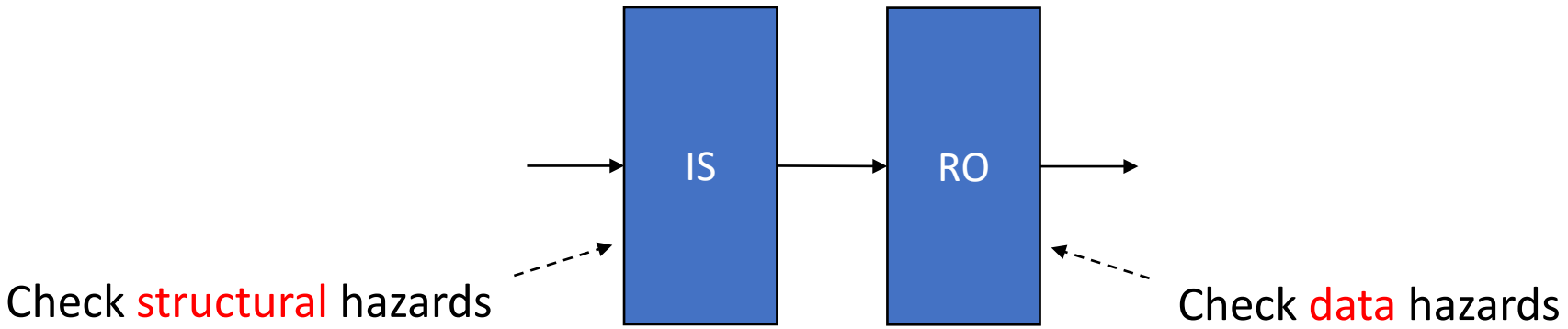
Check structural hazards

Check data hazards

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

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)



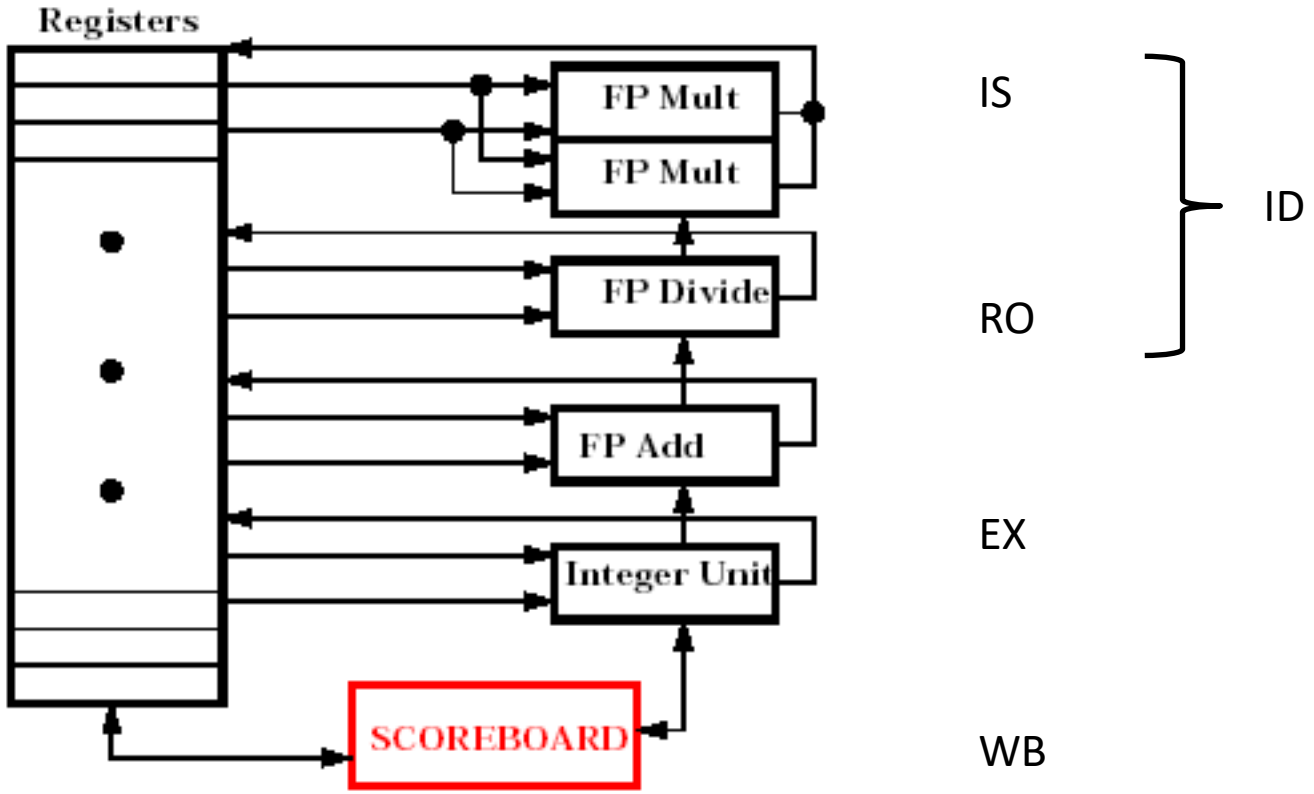
Dynamic Scheduling

- 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   } WAR
      FADD.D   F6, F8, F14   }
```

- 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.

Dynamic Scheduling: Scoreboard algorithm



The basic structure of a processor with scoreboard

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				

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		

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.

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.

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.

Dynamic Scheduling: The Idea

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

Anti-dependence
WAR hazards
(F8)

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

Output-dependence
WAR hazards
(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:

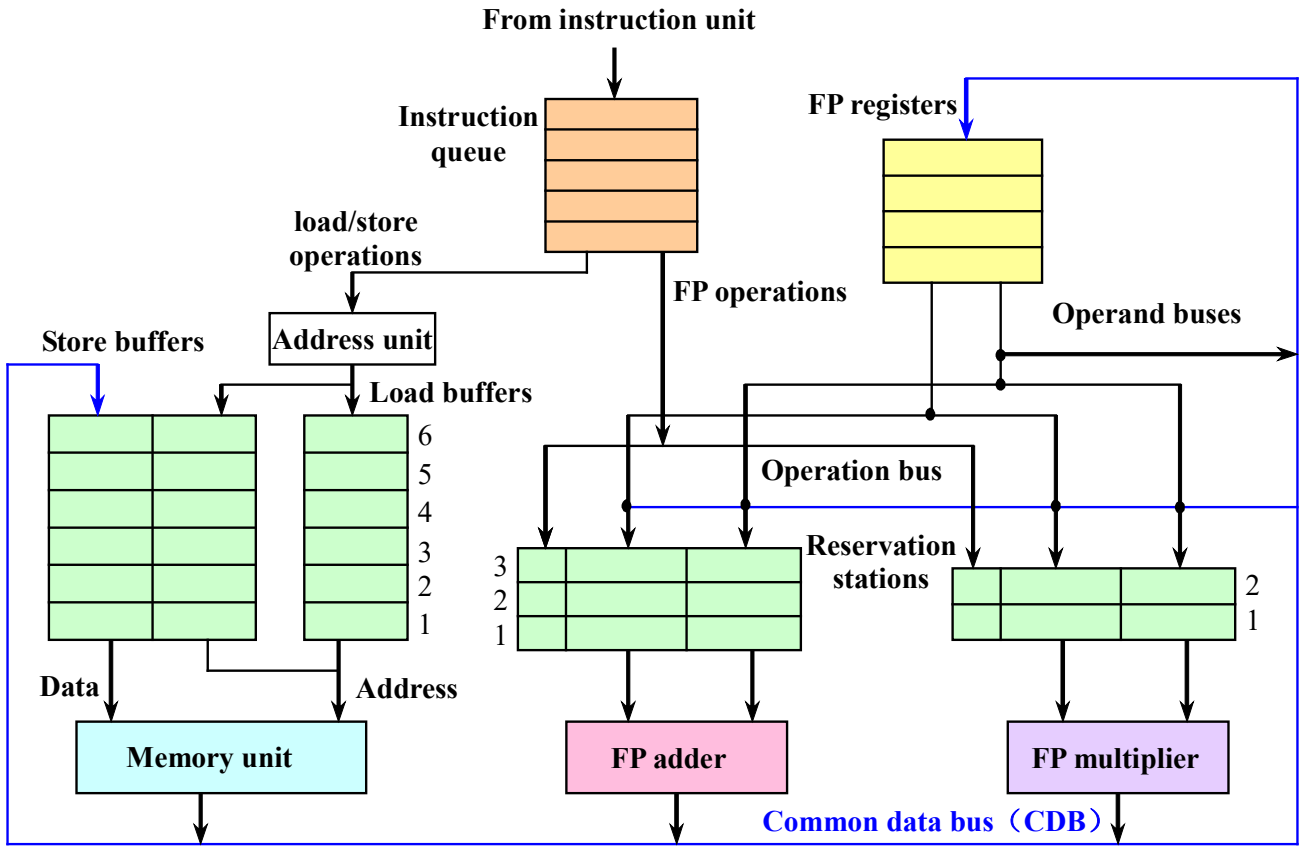
```
FDIV.D   F0, F2, F4
FADD.D   S, F0, F8
FSD      S, 0(R1)
FSUB.D   T, F10, F14
FMUL.D   F6, F10, T
```

F6 change as S

F8 change as T

Who finish the register renaming and how?

Tomasulo's Approach



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

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.

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.

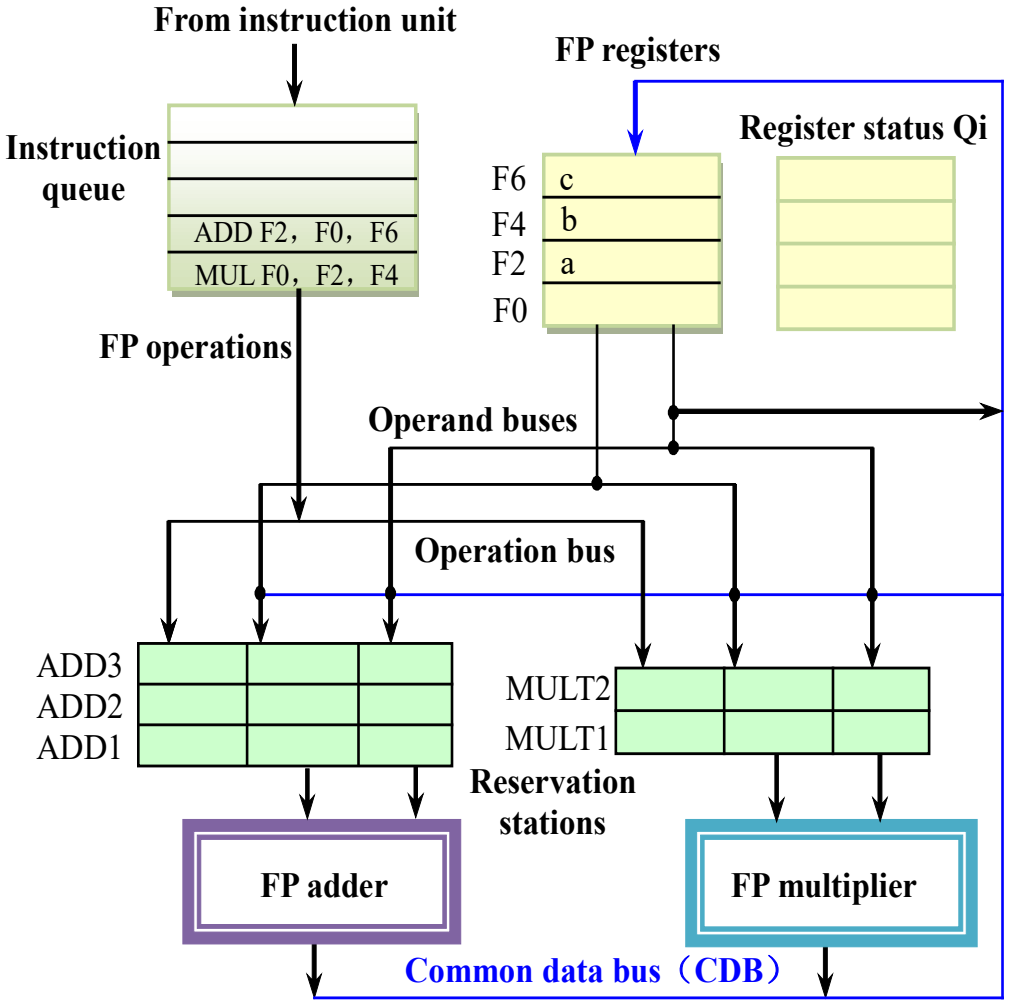
Tomasulo's Approach

Write results

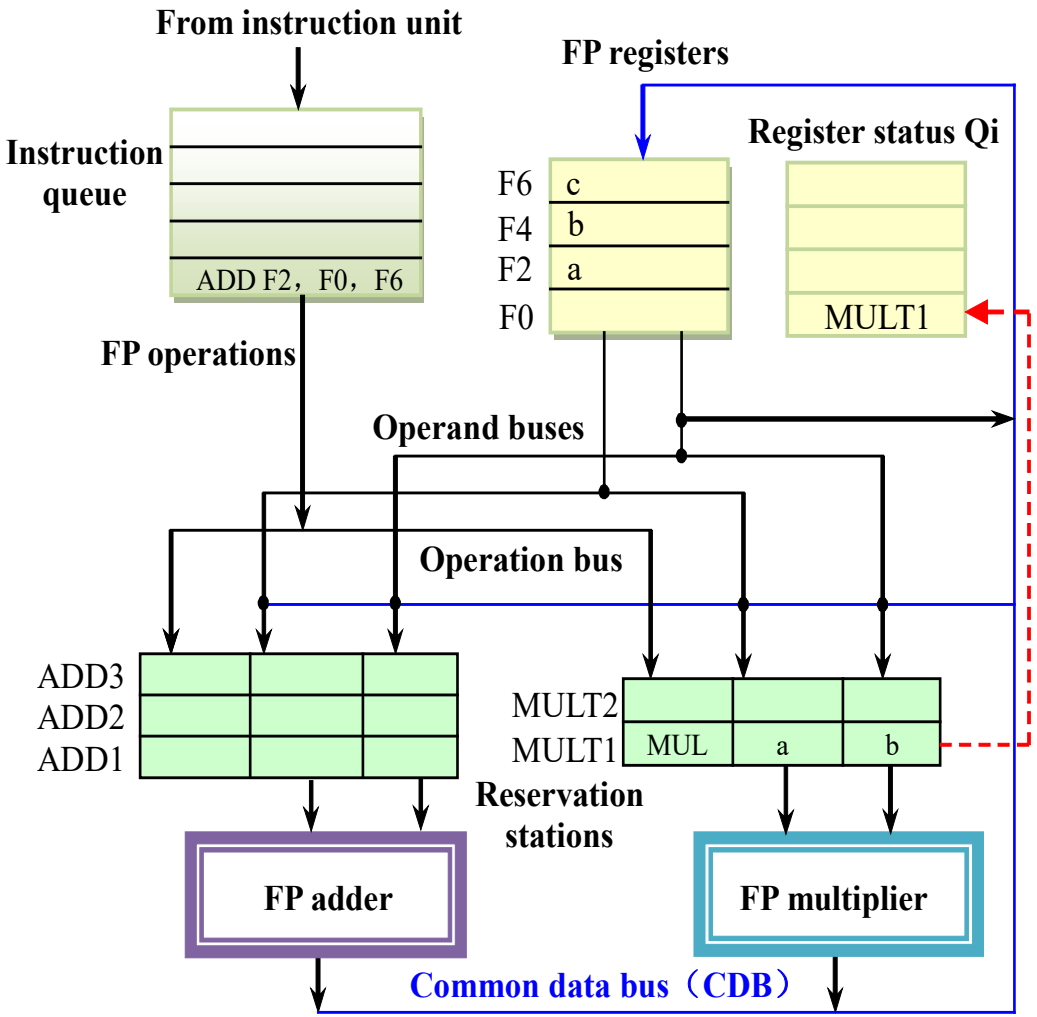
- When the result is available, write it on the CDB and from there into the registers and into any reservation stations (including store buffers).

- 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.

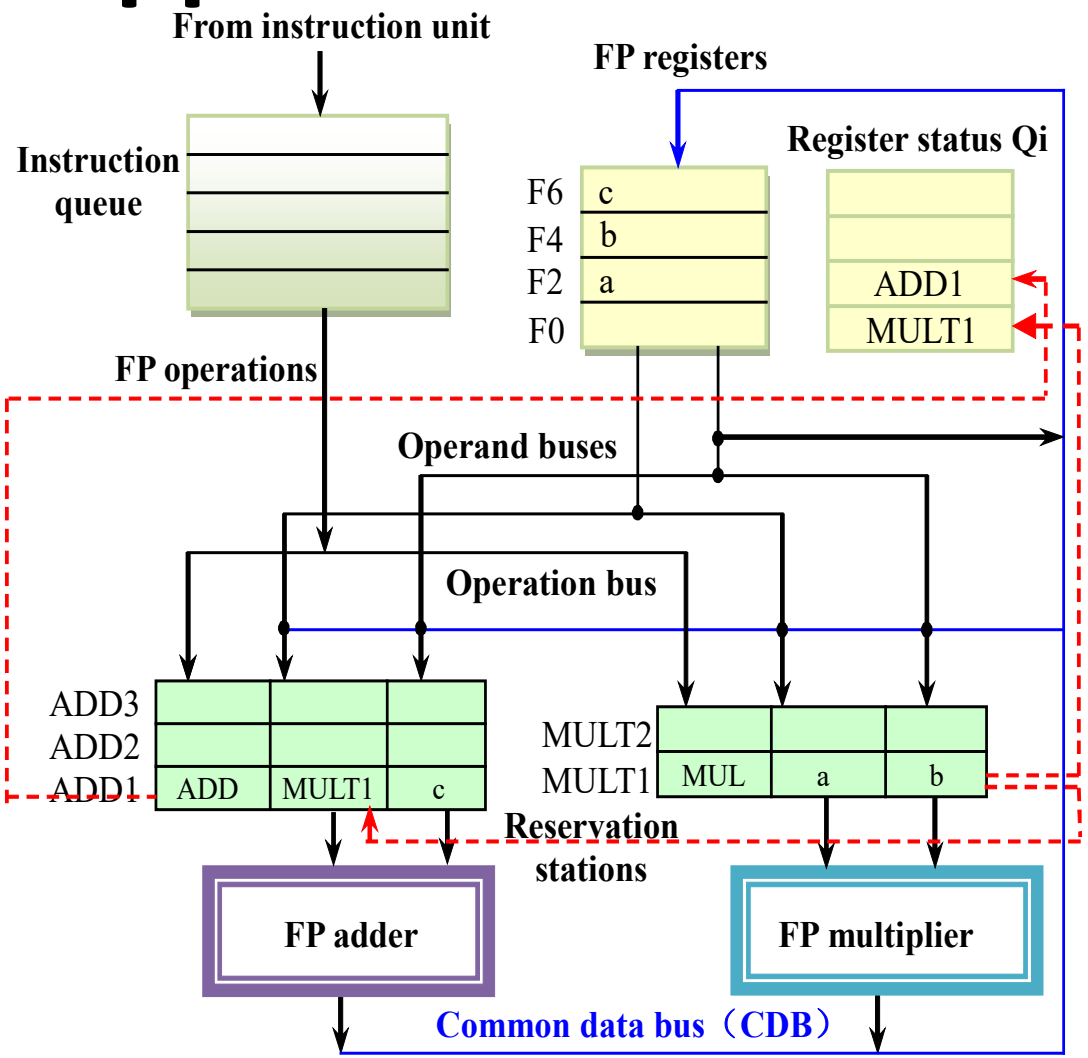
Tomasulo's Approach



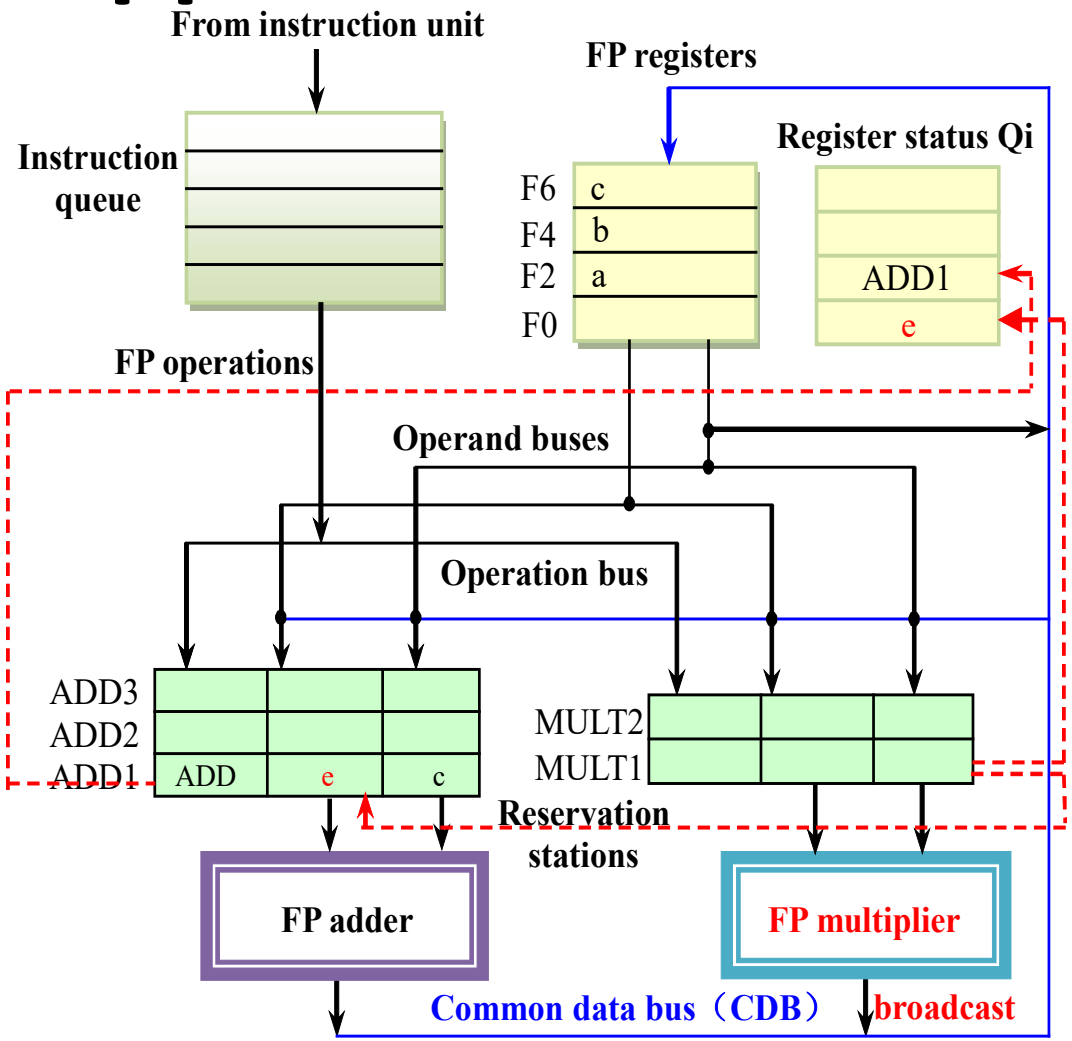
Tomasulo's Approach



Tomasulo's Approach



Tomasulo's Approach



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.

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	√		

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		

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.

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		

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

(1) Scoreboard algorithm

	IS	RO	EX	WB
LD	1	1	1	1
MUL	1	1	10	1
SUB	1	1	2	1
DIV	1	1	40	1
ADD	1	1	2	1

inst	Fi	Fj	Fk	is	ro	ex	wb
L.D	F6	34+R2		1	2	3	4
L.D	F2	45+R3		5	6	7	8
MUL.D	F0	F2	F4	6	9	10~19	20
SUB.D	F8	F2	F6	7	9	10~11	12
DIV.D	F10	F0	F6	8	21	22~61	62
ADD.D	F6	F8	F2	13	14	15~16	22

Instruction	1	2	3	4	5	6	7	8
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								

Instruction	9	10	11	12	13	14	15	16
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								

Instruction	17	18	19	20	21	22	···	···	61	62
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										

(2) Tomasulo's approach

	IS	EX	WB
LD	1	1	1
MUL	1	10	1
SUB	1	2	1
DIV	1	40	1
ADD	1	2	1

inst	Fi	Fj	Fk	is	ex	wb
L.D	F6	34+R2		1	3	4
L.D	F2	45+R3		2	4	5
MUL.D	F0	F2	F4	3	6~15	16
SUB.D	F8	F2	F6	4	6~7	8
DIV.D	F10	F0	F6	5	17~56	57
ADD.D	F6	F8	F2	6	9~10	11

Instruction	1	2	3	4	5	6	7	8
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								

Instruction	9	10	11	12	13	14	15	16	17~56	57
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										

Hardware-Based Speculation

Cache for uncommitted instruction results:

3 fields: instruction type, destination address, value

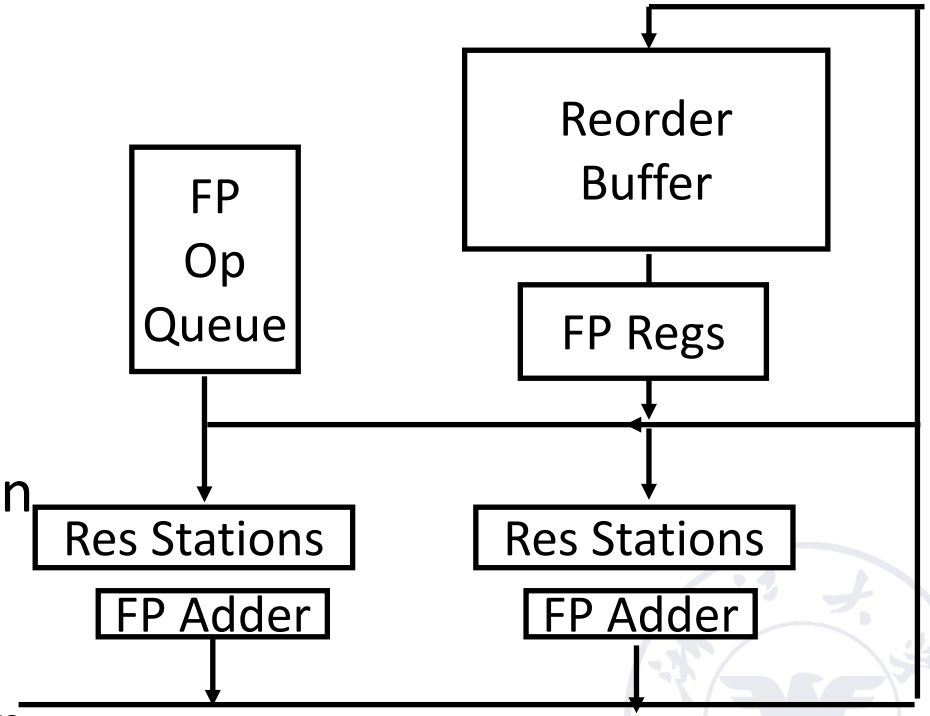
1. When the program execution phase is completed, replace the value in RS with the number of ROB

2. Increase instruction submission stage

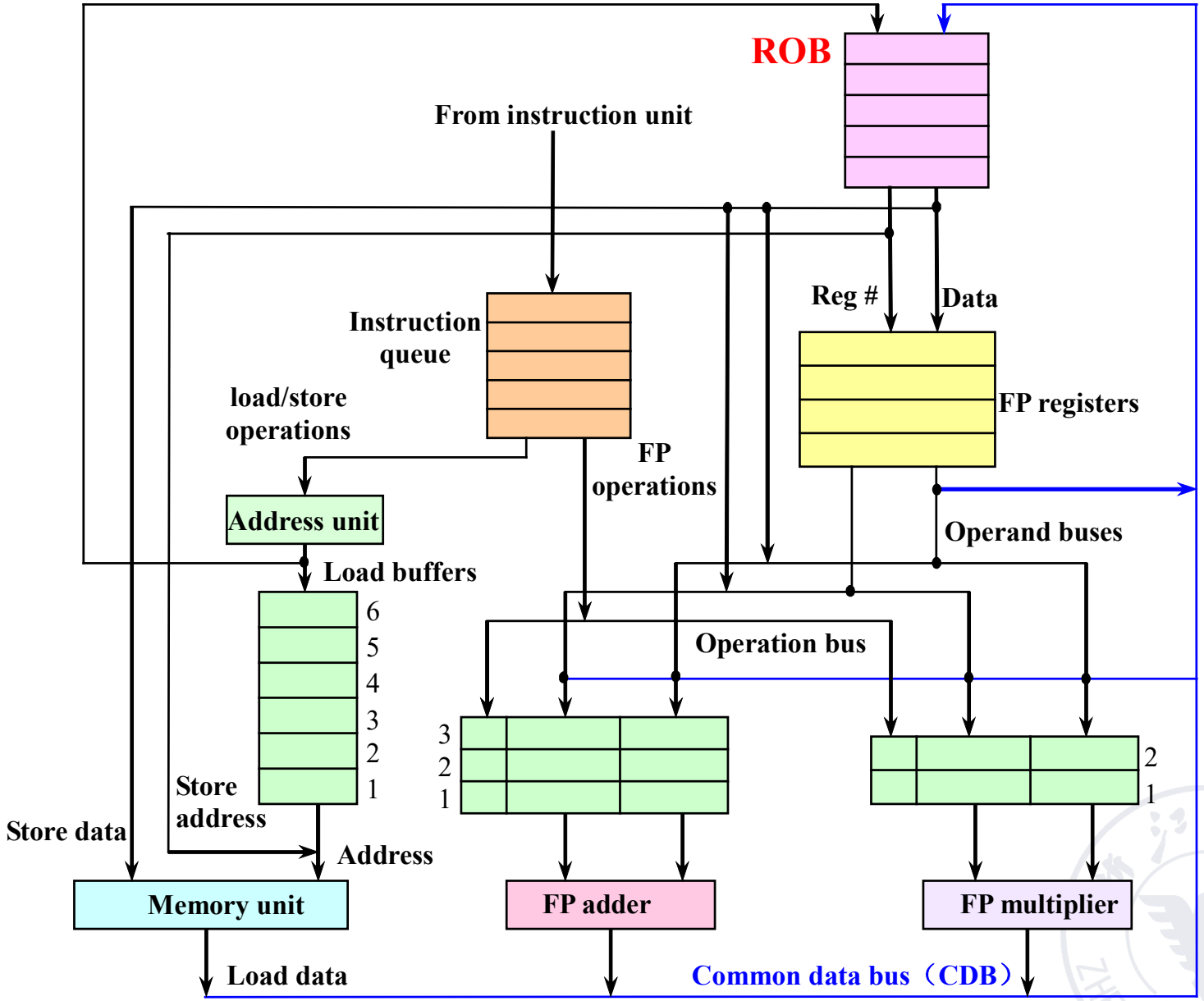
3. ROB provides the number of operations in the completion phase and the commit phase

4. Once the operand is submitted, the result is written to the register

5. In this way, when the prediction fails, it is easy to restore the inferred execution instruction, or when an exception occurs, it is easy to restore the state



The basic structure of a FP unit using Tomasulo's algorithm and extended to handle speculation.



Hardware-Based Speculation

1. Issue—get instruction from FP Op Queue

2. Execution—operate on operands (EX)

3. Write result—finish execution (WB)

4. Commit—update register with reorder result

Hardware-Based Speculation

- Hardware-based speculation combines three key ideas
  - dynamic branch prediction to choose which instructions to execute
  - speculation to allow the execution of instructions before the control dependences are resolved (with the ability to undo the effects of an incorrectly speculated sequence)
  - dynamic scheduling to deal with the scheduling of different combinations of basic blocks

Hardware-Based Speculation

- WB
  - The ROB holds the result of an instruction between the time the operation associated with the instruction completes and the time the instruction commits
  - The ROB is a source of operands for instructions, just as the reservation stations provide operands in Tomasulo's algorithm.
- instruction commit
  - The key idea behind implementing speculation is to allow instructions to execute out of order but to force them to commit in order and to prevent any irrevocable action (such as updating state or taking an exception) until an instruction commits.
  - The reorder buffer (ROB) provides additional registers in the same way as the reservation stations in Tomasulo's algorithm extend the register set.

Show what the status tables look like when the FMUL.D is ready to commit

```
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
```

Hardware-Based Speculation

Name	Reservation Station							
	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	no							
Add2	no							
Add3	no							
Mult1	no	MUL	Mem[45+Reg[R2]]	Regs[F4]			#3	
Mult2	yes	DIV		Mem[34+Reg[R2]]	#3		#5	

NO.	ROB				
	Busy	Instruction	Status	Object	Value
1	no	FLD F6, 34(R2)	Commit	F6	Mem[34+Regs[R2]]
2	no	FLD F2, 45(R3)	Commit	F2	Mem[45+Regs[R3]]
3	no	FMUL.D F0, F2, F4	WB	F0	#2 × Regs[F4]
4	yes	FSUB.D F8, F6, F2	WB	F8	#1 - #2
5	yes	FDIV.D F10, F0, F6	EX	F10	
6	yes	FADD.D F6, F8, F2	WB	F6	#4 + #2

Name	Register Status							
	F0	F2	F4	F6	F8	F10	…	F30
ROB	3			6	4	5		
Busy	yes	no	no	yes	yes	yes	…	no

Practice in Class

1. 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 by hardware-based speculation?

Hardware-Based Speculation

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, F8
FDIV.D   F10, F0, F6
FADD.D   F6, F8, F2
```

- How many cycles does it take to finish each instruction using Hardware-Based Speculation?

Tomasulo with Reorder Buffer - Cycle 0

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No							
Add3	No							
Mult1	No							
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Object	Value
1	no				
2	no				
3	no				
4	no				
5	no				
6	no				

	Busy	Instruction
Load1		
Load2		
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB								
Busy	no	no	no	no	no	no	...	no

Tomasulo with Reorder Buffer - Cycle 1

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No							
Add3	No							
Mult1	No							
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	Yes	FLD F6, 34(R2)	Issue	F6	
2	no				
3	no				
4	no				
5	no				
6	no				

	Busy	Instruction
Load1	Yes	34 + Regs[R2]
Load2		
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#1				
Busy	no	no	no	Yes	no	no	...	no

Tomasulo with Reorder Buffer - Cycle 2

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No							
Add3	No							
Mult1	No							
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	Yes	FLD F6, 34(R2)	Issue	F6	Mem[load1]
2	Yes	FLD F2, 45(R3)	Issue	F2	
3	no				
4	no				
5	no				
6	no				

	Busy	Instruction
Load1	Yes	34 + Regs[R2]
Load2	Yes	45 + Regs[R3]
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB		#2		#1				
Busy	no	Yes	no	Yes	no	no	...	no

Tomasulo with Reorder Buffer - Cycle 3

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No							
Add3	No							
Mult1	Yes	Mul		Regs[F4]	#2		#3	
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	Yes	FLD F6, 34(R2)	Ex1	F6	Mem[load1]
2	Yes	FLD F2, 45(R3)	Issue	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Issue	F0	
4	no				
5	no				
6	no				

	Busy	Instruction
Load1	Yes	34 + Regs[R2]
Load2	Yes	45 + Regs[R3]
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3	#2		#1				
Busy	Yes	Yes	no	Yes	no	no	...	no

Tomasulo with Reorder Buffer - Cycle 4

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	Yes	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No							
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	Yes	FLD F6, 34(R2)	write	F6	Mem[load1]
2	Yes	FLD F2, 45(R3)	Ex1	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Issue	F0	
4	Yes	FSUB.D F8, F6, F2	Issue	F8	
5	no				
6	no				

	Busy	Instruction
Load1	Yes	34 + Regs[R2]
Load2	Yes	45 + Regs[R3]
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3	#2		#1	#4			
Busy	Yes	Yes	no	Yes	Yes	no	...	no

Tomasulo with Reorder Buffer - Cycle 5

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	Yes	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No							
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	Yes	FLD F2, 45(R3)	write	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Issue	F0	
4	Yes	FSUB.D F8, F6, F2	Issue	F8	
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	no				

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3	#2			#4	#5		
Busy	Yes	Yes	no	no	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 6

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	Yes	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add		Regs[F2]	#4		#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex1	F0	
4	Yes	FSUB.D F8, F6, F2	Ex1	F8	
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	Issue	F6	

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 7

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex2	F0	
4	Yes	FSUB.D F8, F6, F2	Ex2	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	Issue	F6	

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 8

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex3	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	Issue	F6	

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 9

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex4	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	Ex1	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 10

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	Yes	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex5	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	Ex2	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 11

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex6	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 12

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex7	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 13

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div		Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex8	F0	
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 14

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex9	F0	#2 * Regs[F4]
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 15

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	Yes	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	Ex10	F0	#2 * Regs[F4]
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 16

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No	Mul	Mem(45+Regs[R3])	Regs[F4]			#3	
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	Yes	FMUL.D F0, F2, F4	write	F0	#2 * Regs[F4]
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Issue	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB	#3			#6	#4	#5		
Busy	Yes	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 17

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No	Sub	Regs[F6]	Mem[45+Regs[R3]]			#4	
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No							
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	Yes	FSUB.D F8, F6, F2	write	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Ex1	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

Need 39 more EX cycles for DIV to finish

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#6	#4	#5		
Busy	no	no	no	Yes	Yes	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 18

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No							
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	no	FSUB.D F8, F6, F2	commit	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Ex2	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

Need 38 more EX cycles for DIV to finish

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#6		#5		
Busy	no	no	no	Yes	no	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 56

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No							
Mult2	Yes	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	no	FSUB.D F8, F6, F2	commit	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	Ex40	F10	
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#6		#5		
Busy	no	no	no	Yes	no	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 57

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No							
Mult2	No	Div	#2 * Regs[F4]	Regs[F6]	#3		#5	

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	no	FSUB.D F8, F6, F2	commit	F8	F6 - #2
5	Yes	FDIV.D F10, F0, F6	write	F10	#3/F6
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#6		#5		
Busy	no	no	no	Yes	no	Yes	...	no

Tomasulo with Reorder Buffer - Cycle 58

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No	Add	#4	Regs[F2]			#6	
Add3	No							
Mult1	No							
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	no	FSUB.D F8, F6, F2	commit	F8	F6 - #2
5	no	FDIV.D F10, F0, F6	commit	F10	#3/F6
6	Yes	FADD.D F6, F8, F2	write	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB				#6				
Busy	no	no	no	Yes	no	no	...	no

Tomasulo with Reorder Buffer - Cycle 59

	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Add1	No							
Add2	No							
Add3	No							
Mult1	No							
Mult2	No							

Reservation Stations

	Busy	Instruction	Status	Dest	Value
1	no	FLD F6, 34(R2)	commit	F6	Mem[load1]
2	no	FLD F2, 45(R3)	commit	F2	Mem[load2]
3	no	FMUL.D F0, F2, F4	commit	F0	#2 * Regs[F4]
4	no	FSUB.D F8, F6, F2	commit	F8	F6 - #2
5	no	FDIV.D F10, F0, F6	commit	F10	#3/F6
6	no	FADD.D F6, F8, F2	commit	F6	#4 + F2

	Busy	Instruction
Load1	no	
Load2	no	
Load3		

Reorder Buffer

	F0	F2	F4	F6	F8	F10	...	F30
ROB								
Busy	no	no	no	no	no	no	...	no

Tomasulo with Reorder Buffer - Summary

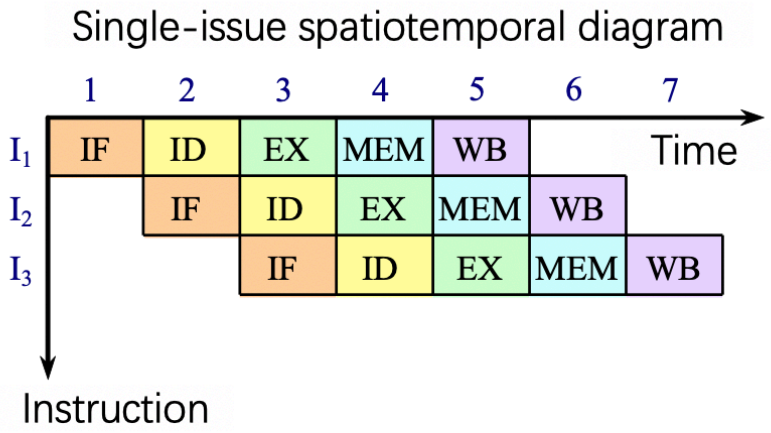
Instruction	Issue	Exec Comp	Writeback	Commit
FLD F6, 34(R2)	1	3	4	5
FLD F2, 45(R3)	2	4	5	6
FMUL.D F0, F2, F4	3	6-15	16	17
FSUB.D F8, F6, F2	4	6-7	8	18
FDIV.D F10, F0, F6	5	17-56	57	58
FADD.D F6, F8, F2	6	9-10	11	59

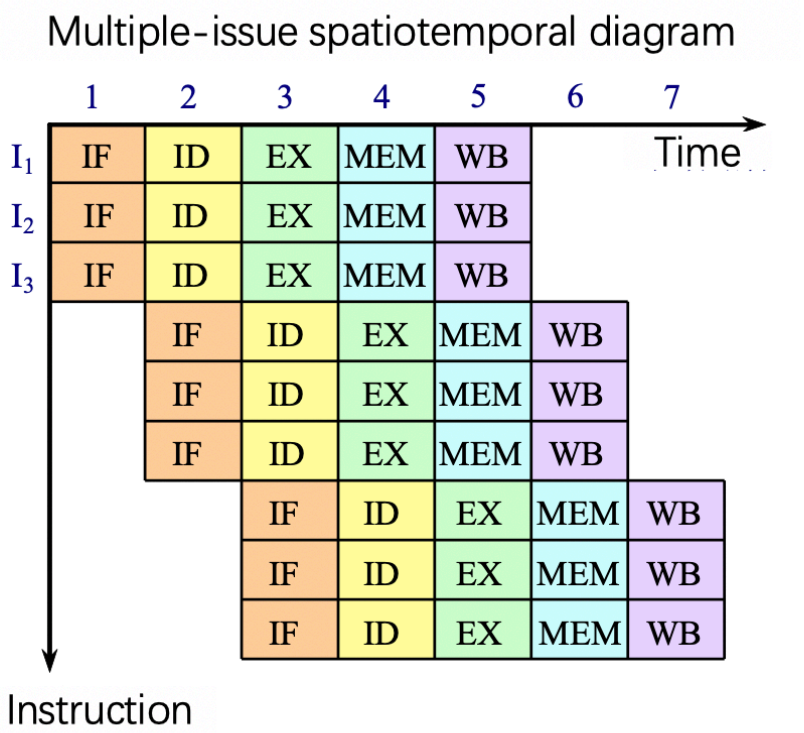
- In-order Issue/Commit, Out-of-Order Execution/Writeback

Hardware-Based Speculation

- Instructions are finished in order according to ROB
- It can be precise exception.
- It is easily extended to integer register and integer function unit.
- But the hardware is too complex.

Comparison of the spatiotemporal diagrams of instructions executed by single-issue and multiple-issue processors





Two types of multiple-issue processor

Superscalar

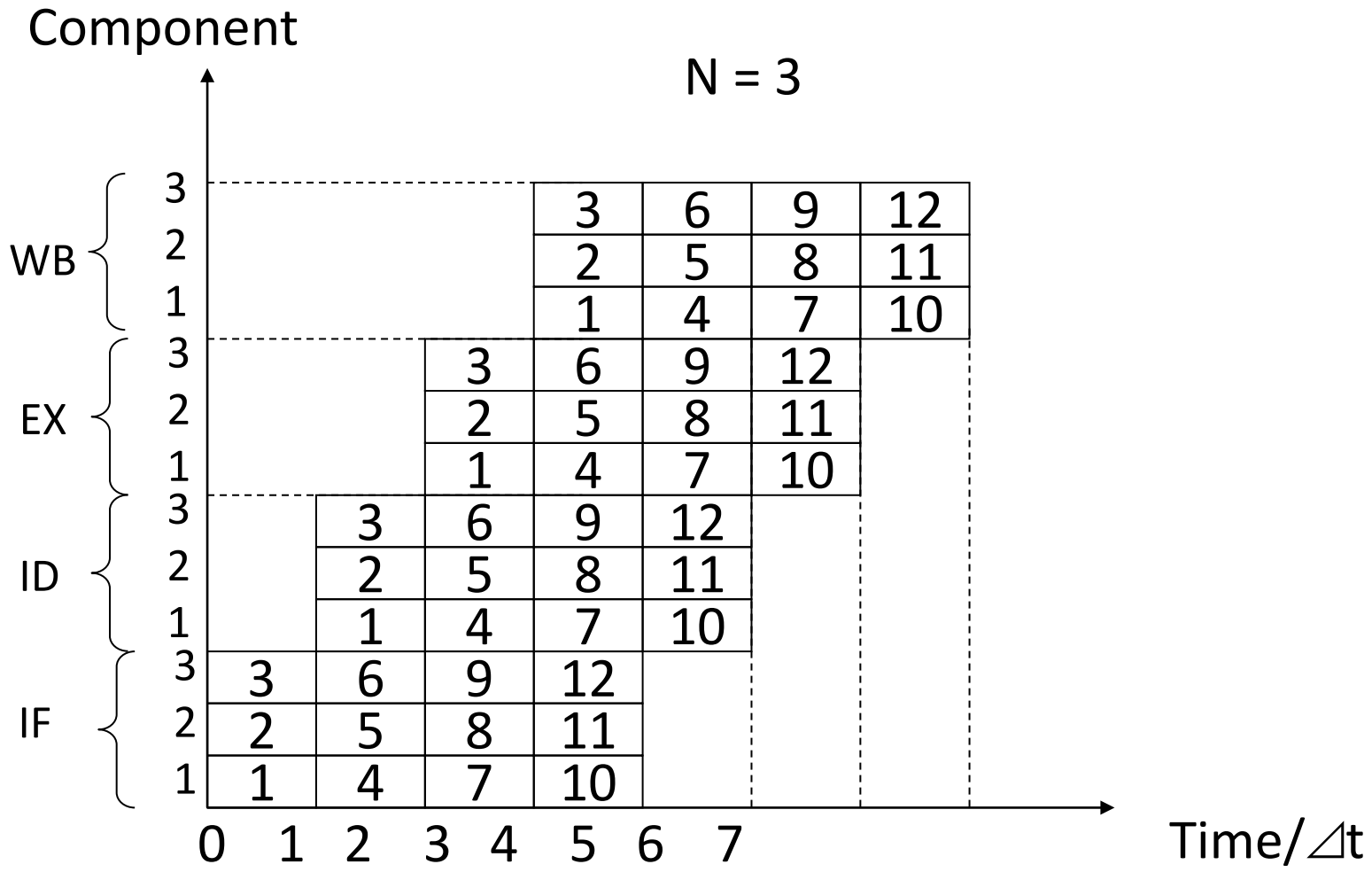
- The number of instructions which are issued in each clock cycle is not fixed. It depends on the specific circumstances of the code. (1-8, with upper limit)
- Suppose this upper limit is n, then the processor is called n-issue.
- It can be statically scheduled through the compiler, or dynamically scheduled based on the Tomasulo algorithm.
- This method is the most successful method for general computing at present.

Two types of multiple-issue processor

Superscalar

- The number of instructions which are issued in each clock cycle is not fixed. It depends on the specific circumstances of the code. (1-8, with upper limit)
- Suppose this upper limit is n, then the processor is called n-issue.
- It can be statically scheduled through the compiler, or dynamically scheduled based on the Tomasulo algorithm.
- This method is the most successful method for general computing at present.

Two types of multiple-issue processor



Two types of multiple-issue processor

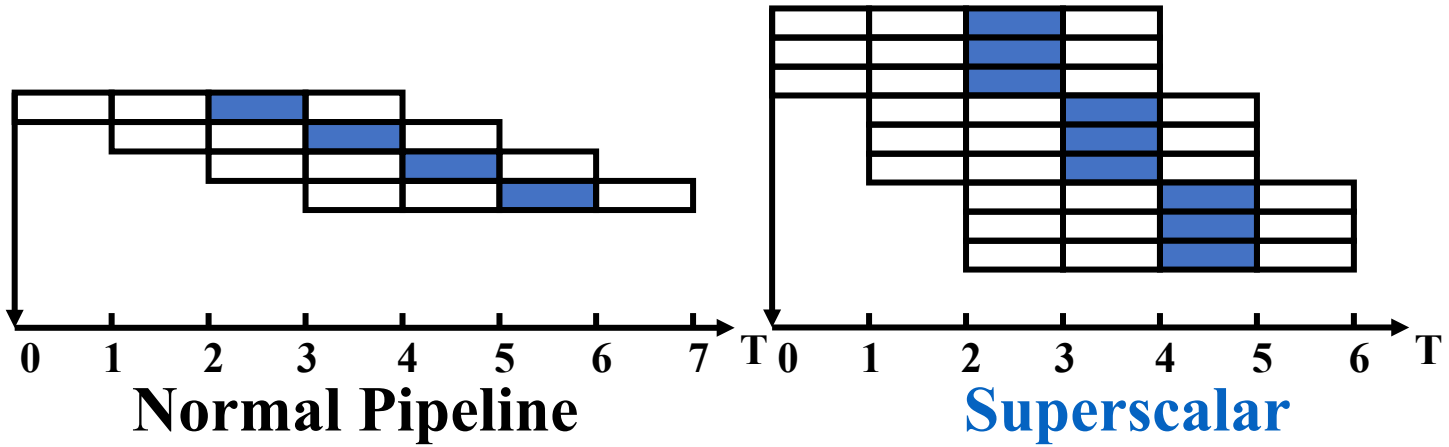
VLIW (Very Long Instruction Word)

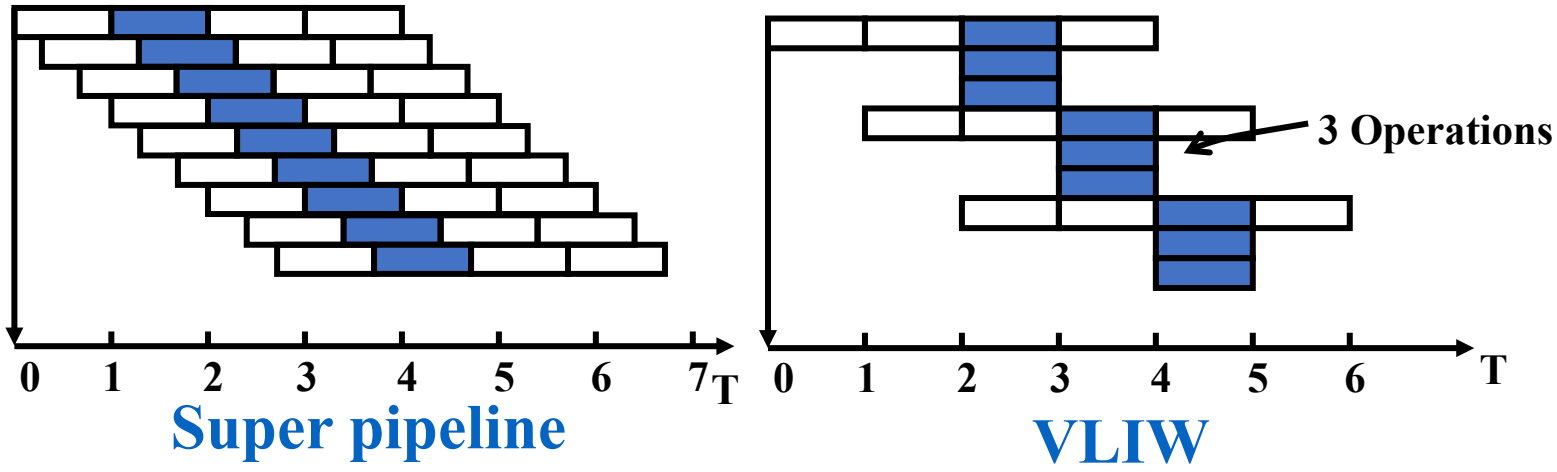
- The number of instructions which are issued in each clock cycle is fixed (4-16), and these instructions constitute a long instruction or an instruction packet.
- In the instruction packet, the parallelism between instructions is explicitly expressed through instructions.
- Instruction scheduling is done statically by the compiler.
- It has been successfully applied to digital signal processing and multimedia applications.

Superscalar & VLIM

- The superscalar structure is transparent to the programmer, because the processor can detect whether the next instruction can flow out, so there is no need to rearrange instructions to satisfy the issue of instructions.
- Even the code that has not been optimized by the compiler for scheduling and optimization of the superscalar structure or the code generated by the old compiler can run, of course, the running effect will not be very good.
- To achieve good results, one of the methods:
  - Use dynamic superscalar scheduling technology.

Superscalar & VLIM





Common name	Issue structure	Hazard detection	Scheduling	Distinguishing characteristic	Examples
Superscalar (static)	Dynamic	Hardware	Static	In-order execution	Mostly in the embedded space: MIPS and ARM, including the ARM Cortex-A8
Superscalar (dynamic)	Dynamic	Hardware	Dynamic	Some out-of-order execution, but no speculation	None at the present
Superscalar (speculative)	Dynamic	Hardware	Dynamic with speculation	Out-of-order execution with speculation	Intel Core i3, i5, i7; AMD Phenom; IBM Power 7
VLIW/LIW	Static	Primarily software	Static	All hazards determined and indicated by compiler (often implicitly)	Most examples are in signal processing, such as the TI C6x
EPIC	Primarily static	Primarily software	Mostly static	All hazards determined and indicated explicitly by the compiler	Itanium

Multi-issue technology based on static scheduling

- In a typical superscalar processor, 1 to 8 instructions can be issued per clock cycle.
- Instructions flow out in order, and conflict detection is performed when they flow out.
  - In the current sequence of instructions, there is no data conflict or Close conflict.

Example: A statically scheduled superscalar processor with 4 issues

- In the instruction fetch stage, the pipeline will receive 1 to 4 instructions (called issue packets) from the instruction fetch component.
- In one clock cycle, all of these instructions may be able to flow out, or only a part of them may flow out.

Multi-issue technology based on static scheduling

The outgoing component detects structural conflicts or data conflicts.

- Generally implemented in two stages:
  - The first stage: Carry out the conflict detection in the outgoing package, and select the instructions that can be outflowed initially.
  - The second stage: Check whether the selected instruction conflicts with the instruction being executed.

How does the MIPS processor achieve superscalar?

- Assumption: Two instructions flow out every clock cycle:
  - 1 integer instruction + 1 floating-point operation instruction
- Among them, load instructions, store instructions, and branch instructions are classified as integer instructions.

Multi-issue technology based on static scheduling

Claim:

- Fetch two instructions (64 bits) at the same time and decode two instructions (64 bits).

- The processing of instructions includes the following steps:
  - Fetch two instructions from Cache.
  - Determine which instructions can flow out (0~2 instructions).
  - Send them to the corresponding functional components.
- The execution process of instructions in a multiple-issue superscalar pipeline
  - Assumption: All floating-point instructions are addition instructions, and their execution time is two clock cycles.
  - For simplicity, integer instructions are always placed before floating-point instructions in the figure below.

Type	Pipeline work bench							
Integer Instruction	IF	ID	EX	MEM	WB			
Floating-Point Instruction	IF	ID	EX	EX	MEM	WB		
Integer Instruction		IF	ID	EX	MEM	WB		
Floating-Point Instruction		IF	ID	EX	EX	MEM	WB	
Integer Instruction			IF	ID	EX	MEM	WB	
Floating-Point Instruction			IF	ID	EX	EX	MEM	WB
Integer Instruction				IF	ID	EX	MEM	WB
Floating-Point Instruction				IF	ID	EX	EX	MEM

Multi-issue technology based on static scheduling

- With the parallel outflow method of "1 integer instruction + 1 floating point instruction", the amount of hardware that needs to be increased is small.
- Floating-point load or floating-point store instructions will use integer parts, which will increase access conflicts to floating point registers.
  - Add a read/write port for floating-point registers.
- Since the number of instructions in the pipeline has doubled, the directional path has to be increased.

Multi-issue technology based on dynamic scheduling

- Extended Tomasulo algorithm: supports two-way superscalar
  - Two instructions are issued every clock cycle;
  - One is an integer instruction and the other is a floating-point instruction.
- Use a relatively simple method:
  - Instructions flow to the RS in order, otherwise the program semantics will be destroyed.
  - Separate the table structure used for integers from the table structure used for floating-point, and process them separately, so that one floating-point instruction and one integer instruction can be sent to their respective reservation stations at the same time.

Multi-issue technology based on dynamic scheduling

- For the RISC-V pipeline that uses the Tomasulo algorithm and multi-issue technology, consider the following simple loop execution. This program adds the scalar in F2 to each element of a vector.

```
Loop:   FLD      F0, 0 (R1)      // Take an array element and put it into F0
        FADD.D   F4, F0, F2      // add the scalar in F2
        FSD      F4, 0 (R1)      // store result
        ADDI     R1, R1, 8       // increment pointer by 8
                                 // (each data occupies 8 bytes)
        BNE      R1, R2, Loop    // If R1 is not equal to R2, it means it is
                                 // not over yet, move to Loop to continue
```

Multi-issue technology based on dynamic scheduling

Now make the following assumptions:

- One integer instruction and one floating-point instruction can flow out every clock cycle, even if they are related.
- There is an integer component for integer ALU operations and address calculations; and for each type of floating-point operation, there is an independent pipelined floating-point functional component.
- The instruction flow and the write result each take one clock cycle.
- It has a dynamic branch prediction component and an independent functional component for calculating branch conditions.
- Branch instructions flowed out separately, no delayed branch was used, but branch prediction was perfect. Before the branch instruction is completed, its subsequent instructions can only be fetched and flowed out, but cannot be executed.

Multi-issue technology based on dynamic scheduling

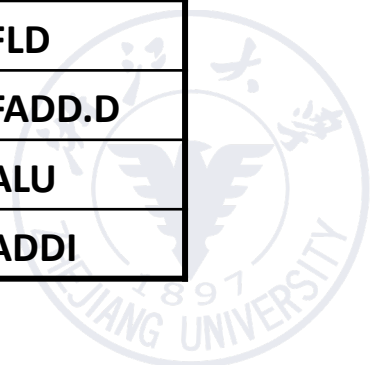
Because the write result occupies one clock cycle, the delay in generating the result is: one cycle for integer operations, two cycles for load, and three cycles for floating-point addition.

List the issue of each instruction in the first three loops of the program, start execution, and write the results to the CDB.

Solution:

When execution, the loop will be dynamically unrolled, and two instructions will issue whenever possible. For ease of analysis, the time when the memory fetch occurs is listed in the table. The running result is shown in the figure below.

	Instruction	IS	EX	MEM	Write CDB	Explanation
1	FLD F0, 0(R1)	1	2	3	4	Issue the first instruction
1	FADD.D F4, F0, F2	1	5		8	Wait for the result of FLD
1	FSD F4, 0(R1)	2	3	9		Wait for the result of FADD.D
1	ADDI R1, R1, 8	2	4		5	Wait for ALU
1	BNE R1, R2, Loop	3	6			Wait for the result of ADDI
2	FLD F0, 0(R1)	4	7	8	9	Wait for the result of BNE
2	FADD.D F4, F0, F2	4	10		13	Wait for the result of FLD
2	FSD F4, 0(R1)	5	8	14		Wait for the result of FADD.D
2	ADDI R1, R1, 8	5	9		10	Wait for the result of ALU
2	BNE R1, R2, Loop	6	11			Wait for the result of ADDI
3	FLD F0, 0(R1)	7	12	13	14	Wait for the result of BNE
3	FADD.D F4, F0, F2	7	15		18	Wait for the result of FLD
3	FSD F4, 0(R1)	8	13	19		Wait for the result of FADD.D
3	ADDI R1, R1, 8	8	14		15	Wait for the result of ALU
3	BNE R1, R2, Loop	9	16			Wait for the result of ADDI



Multi-issue technology based on dynamic scheduling

It can be seen from the figure:

- The program can basically reach 3 beats and 5 instructions
  - IPC = 5/3 = 1.67 items/beat
- Although the outflow rate of instructions is relatively high, the execution efficiency is not very high.
  - A total of 15 instructions were executed in 16 beats.
  - The average command execution speed is 15/16=0.94 per beat.
- The reason is that there are few floating-point operations, and ALU components have become a bottleneck.
- Solution: Add an adder to separate the ALU function from the address calculation function.

Multi-issue technology based on dynamic scheduling

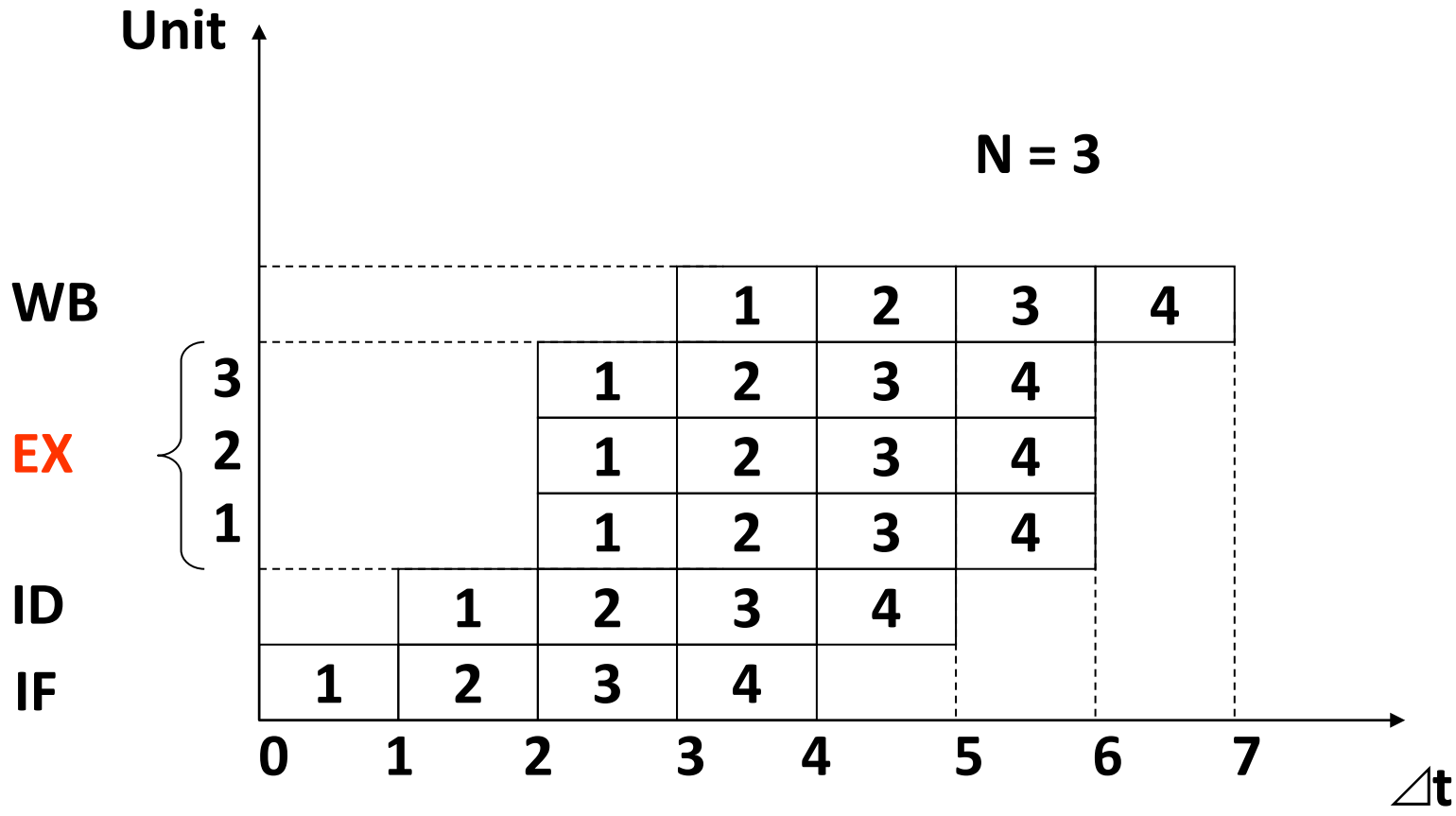
The performance of the double-issue dynamic scheduling pipeline is limited by the following 3 factors:

- The work load of integer components and floating-point components is not balanced, and the role of floating-point components is not fully utilized.
  - Try to reduce the number of integer instructions in the loop.
- The control overhead in each loop iteration is too large.
  - Two of the five instructions are auxiliary instructions.
  - Efforts should be made to reduce or eliminate these instructions.
- Control correlation makes the processor must wait until the result of the branch instruction comes out before starting the execution of the next L.D instruction.

Very long instruction word technology(VLIW)

- Assemble multiple instructions that can be executed in parallel into a very long instruction (more than 100 bits to hundreds of bits).
- Set up multiple features.
- The instruction word is divided into several fields, and each field is called an operation slot, which directly and independently controls a functional unit.
- In the VLIW processor, all processing and instruction arrangement are completed by the compiler.
- At compile time, multiple unrelated or unrelated operations that can be executed in parallel are combined to form a very long instruction word with multiple operation segments.

VLIW



Very long instruction word technology(VLIW)

Example 4.5

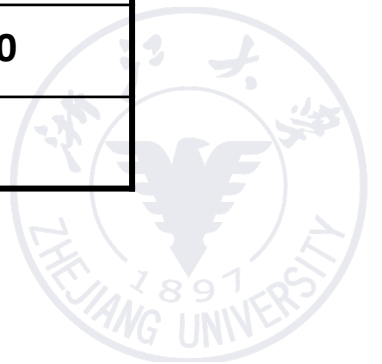
- Assume that the VLIW processor can simultaneously stream 5 instructions per clock cycle: two memory access instructions, two floating-point operation instructions, and one integer instruction or branch instruction. For the code after loop unrolling in Example 4.4, give its code sequence in the VLIW. The delay slot of branch instructions is not considered.

Solution: The code sequence is shown in the figure below.

- The running time is 8 clock cycles.
- An average of 1.6 clock cycles per cycle.
- 17 instructions issued in 8 clock cycles, 2.1 instructions per clock cycle.
- There are 8 * 5 = 40 operating slots in 8 clock cycles, and the ratio of effective slots is 42.5%.

Very long instruction word technology(VLIW)

L/S Instruction1	L/S Instruction2	FP Instruction1	FP Instruction2	Integer/branch Instruction
L.D F0,0(R1)	L.D F6,-8(R1)			
L.D F10,-16(R1)	L.D F14,-24(R1)			
L.D F18,-32(R1)		ADD.D F4,F0,F2	ADD.D F8,F6,F2	
		ADD.D F12,F10,F2	ADD.D F16,F14,F2	
		ADD.D F20,F18,F2		
S.D F4,0(R1)	S.D F8,-8(R1)			
S.D F12,-16(R1)	S.D F16,–24(R1)			DADDIU R1,R1,#-40
S.D F20,8(R1)				BNE R1,R2,Loop



Very long instruction word technology(VLIW)

Some problems with VLIW

- Program code length increased
  - A large number of loop unrolling to improve parallelism.
  - The operation slot in the instruction word cannot always be filled.
  - Solution: Use the method of command sharing the immediate digital field, or use the method of command compression storage, transfer to Cache or expansion during decoding.
- Lockstep mechanism
  - When any operating part is paused, the entire processor must be paused.
- Machine code incompatibility

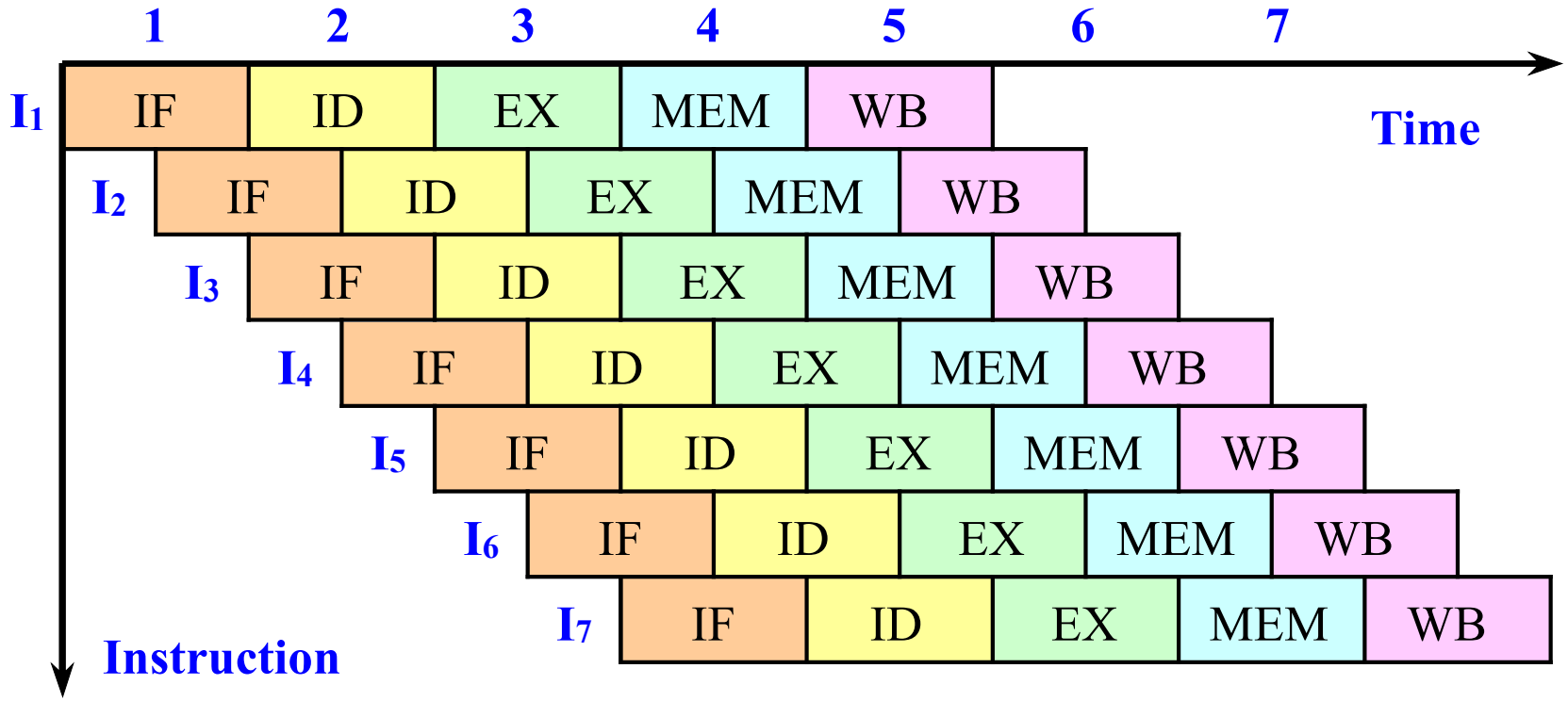
Very long instruction word technology(VLIW)

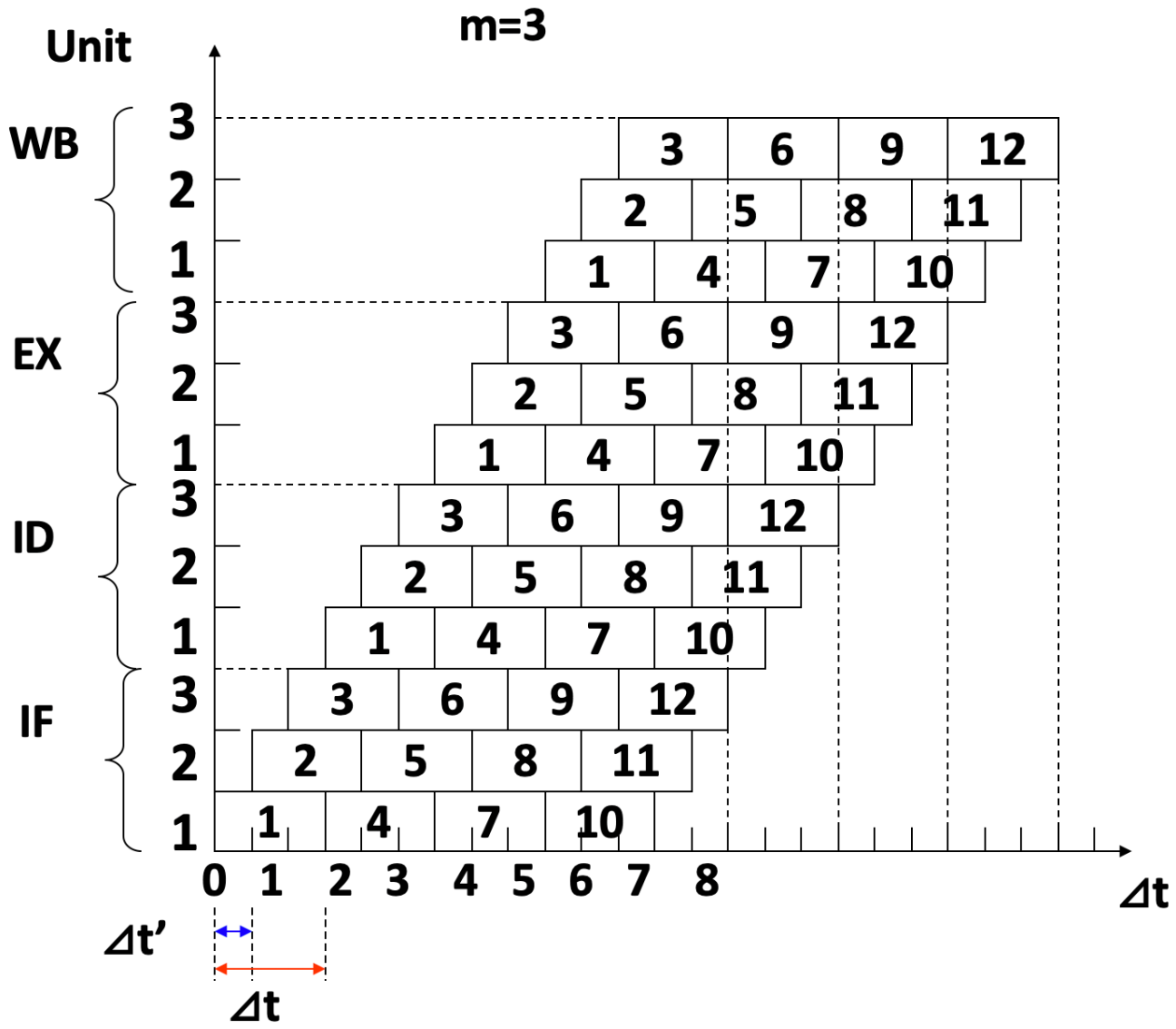
What are the limitations of the instruction multi-flow processor?

- Mainly affected by the following three aspects:
  - Instruction-level parallelism inherent in the program.
  - Difficulties in hardware implementation.
  - Technical limitations inherent in superscalar and super long instruction word processors.

Very long instruction word technology(VLIW)

- Each pipeline stage is further subdivided, so that multiple instructions can be time-shared in one clock cycle. This kind of processor is called a super-pipelined processor.
- For a super-pipelined computer that can flow out n instructions per clock cycle, these n instructions are not flowed out at the same time, but one instruction is flowed out every 1/n clock cycle.
  - In fact, the pipeline cycle of the super-pipeline computer is 1/n clock cycles.
- The time-space diagram of a super-pipelined computer that issues two instructions in time-sharing every clock cycle.

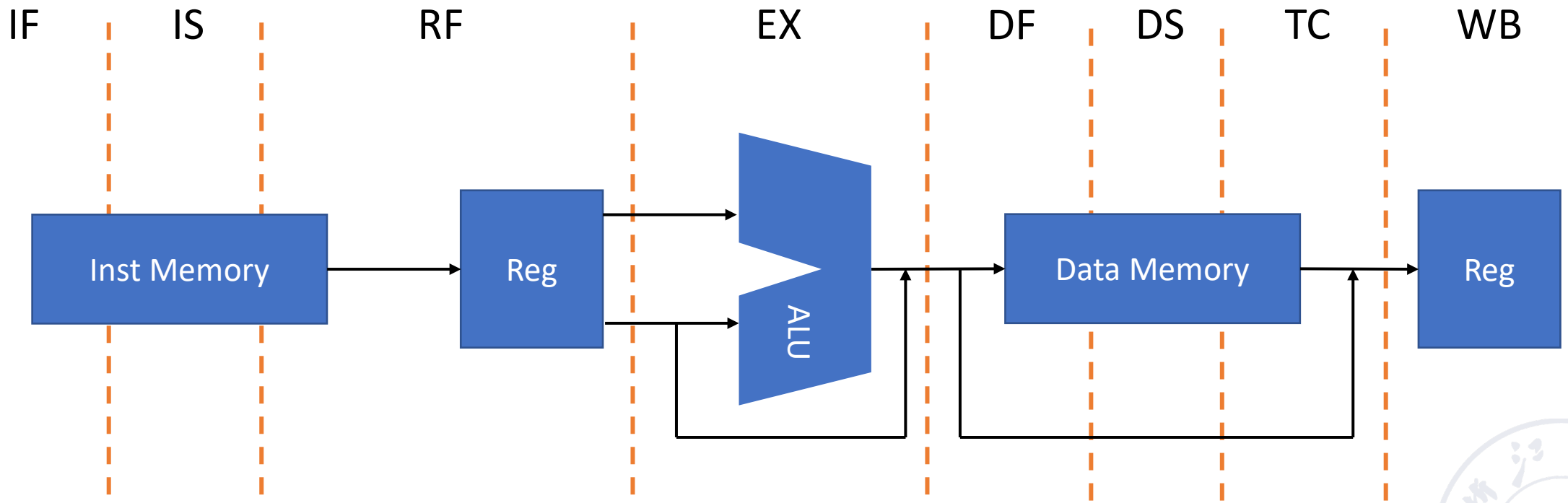

1
2
3
4
5
6
7
Time
I1
I2
I3
I4
I5
I6
I7
IF
ID
EX
MEM
WB
Instruction


m=3
Unit
WB
EX
ID
IF
3
2
1
0 1 2 3 4 5 6 7 8
Δt
Δt'
Δt

Superpipelining processor

- A pipeline processor with 8 or more instruction pipeline stages is called a superpipelining processor.
- Typical superpipelining processor: SGI's MIPS series R4000
  - There are 2 Caches in the R4000 microprocessor chip:
    - Instruction Cache and Data Cache
    - The capacity is 8 KB
    - The data width of each Cache is 64 b
  - R4000's core processing components: integer components
    - A 32×32 bit general register bank
    - An arithmetic logic unit (ALU)
    - A dedicated multiplication/division unit

R4000 Pipeline Structure



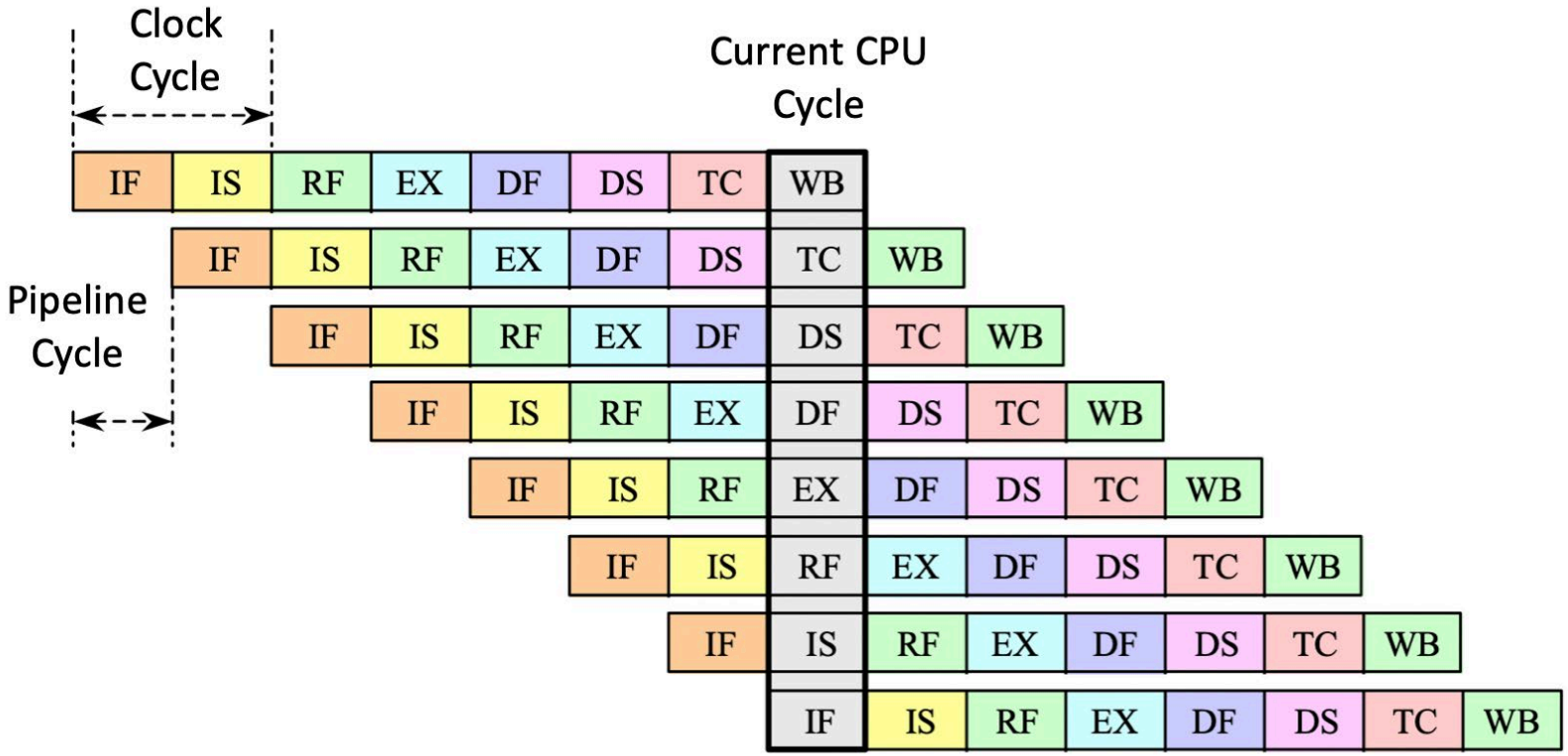
Function of Each Stage

- IF—First half of instruction fetch; PC selection actually happens here, together with initiation of instruction cache access.
- IS—Second half of instruction fetch, complete instruction cache access.
- RF—Instruction decode and register fetch, hazard checking, and instruction cache hit detection.
- EX—Execution, which includes effective address calculation, ALU operation, and branch-target computation and condition evaluation.

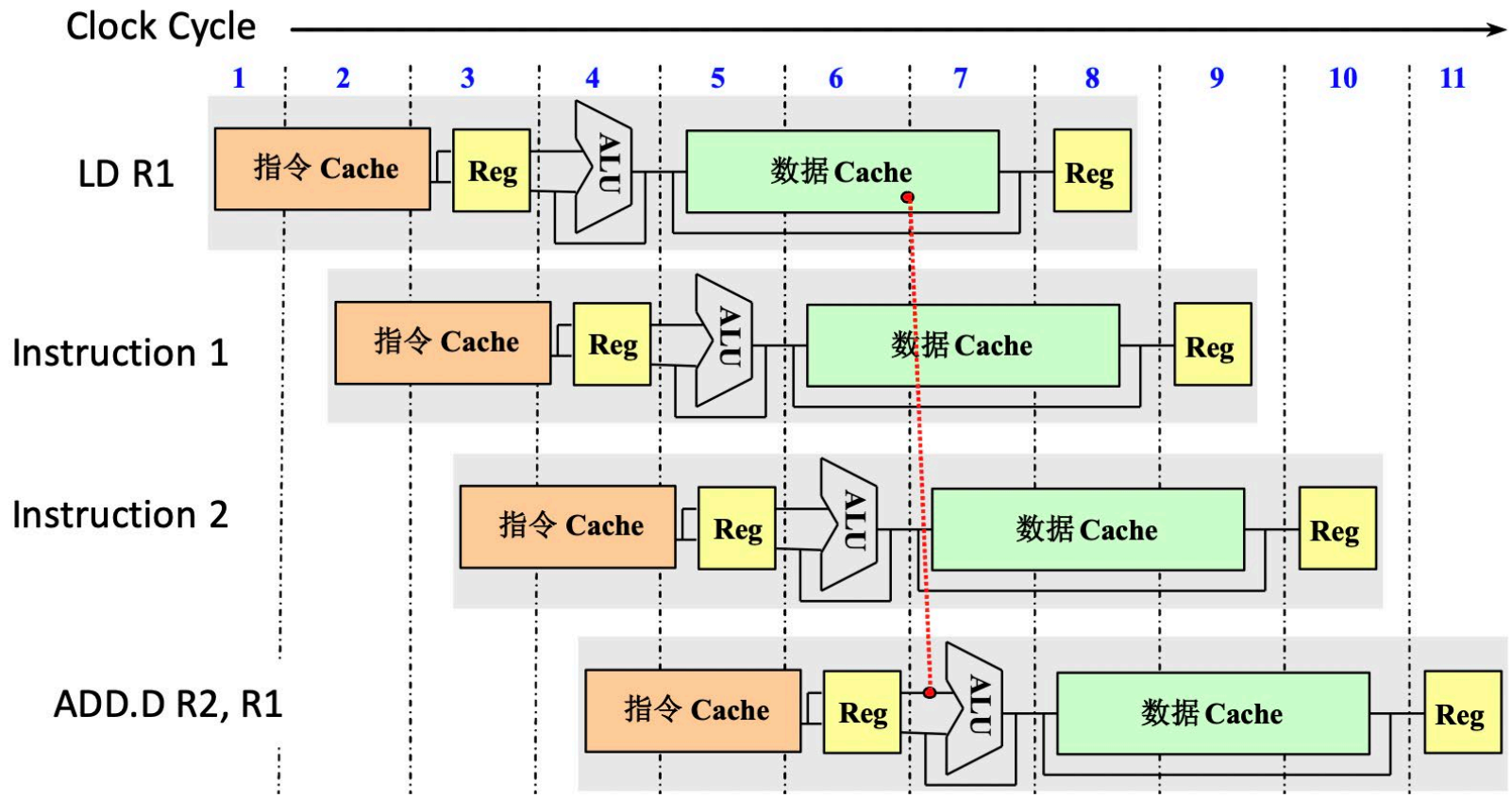
Function of Each Stage

- DF—Data fetch, first half of data cache access.
- DS—Second half of data fetch, completion of data cache access.
- TC—Tag check, to determine whether the data cache access hit.
- WB—Write-back for loads and register-register operations.

Spatiotemporal Diagram of MIPS R4000 Pipeline

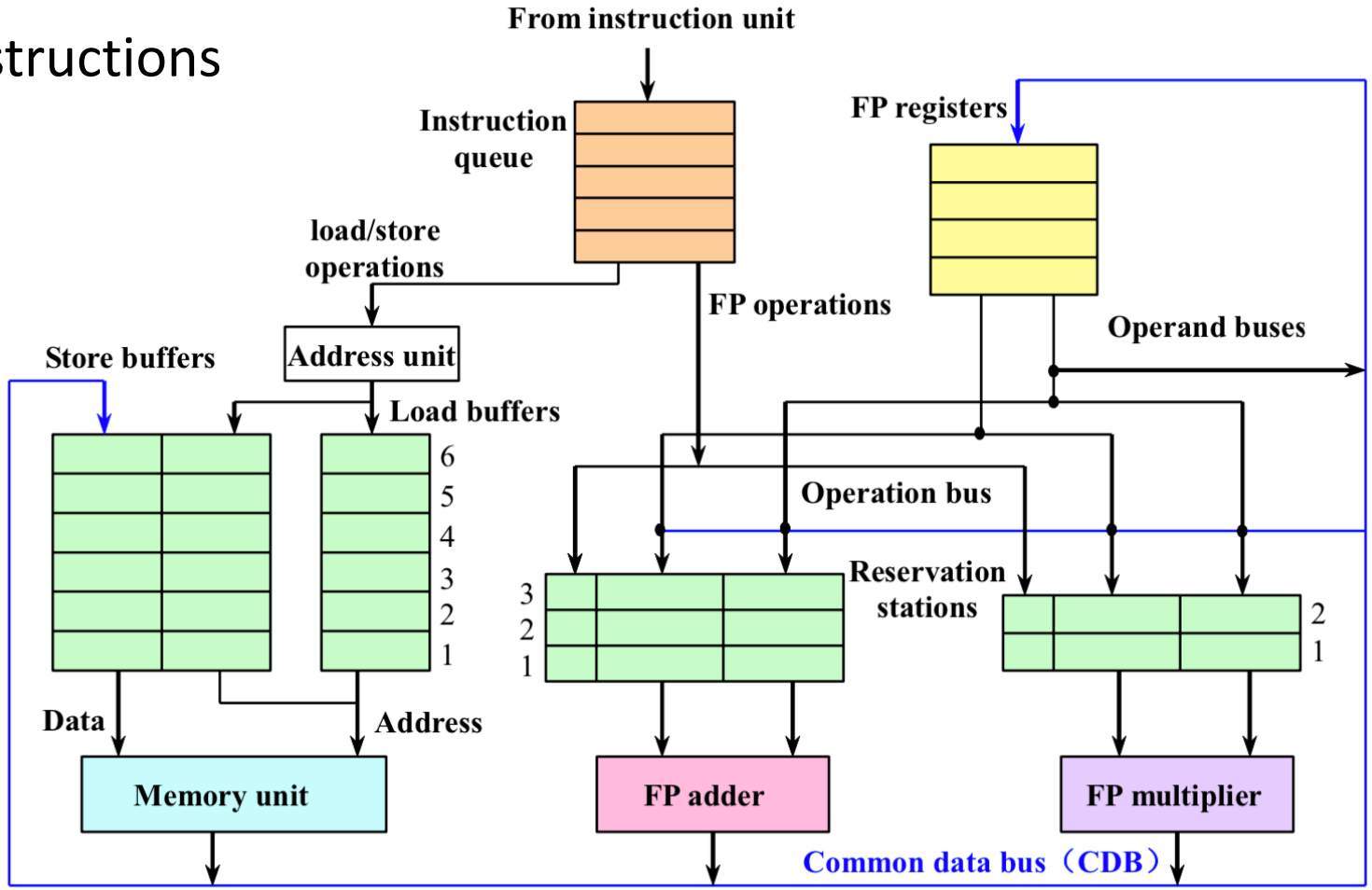


Two Clock Cycles for Load Delay



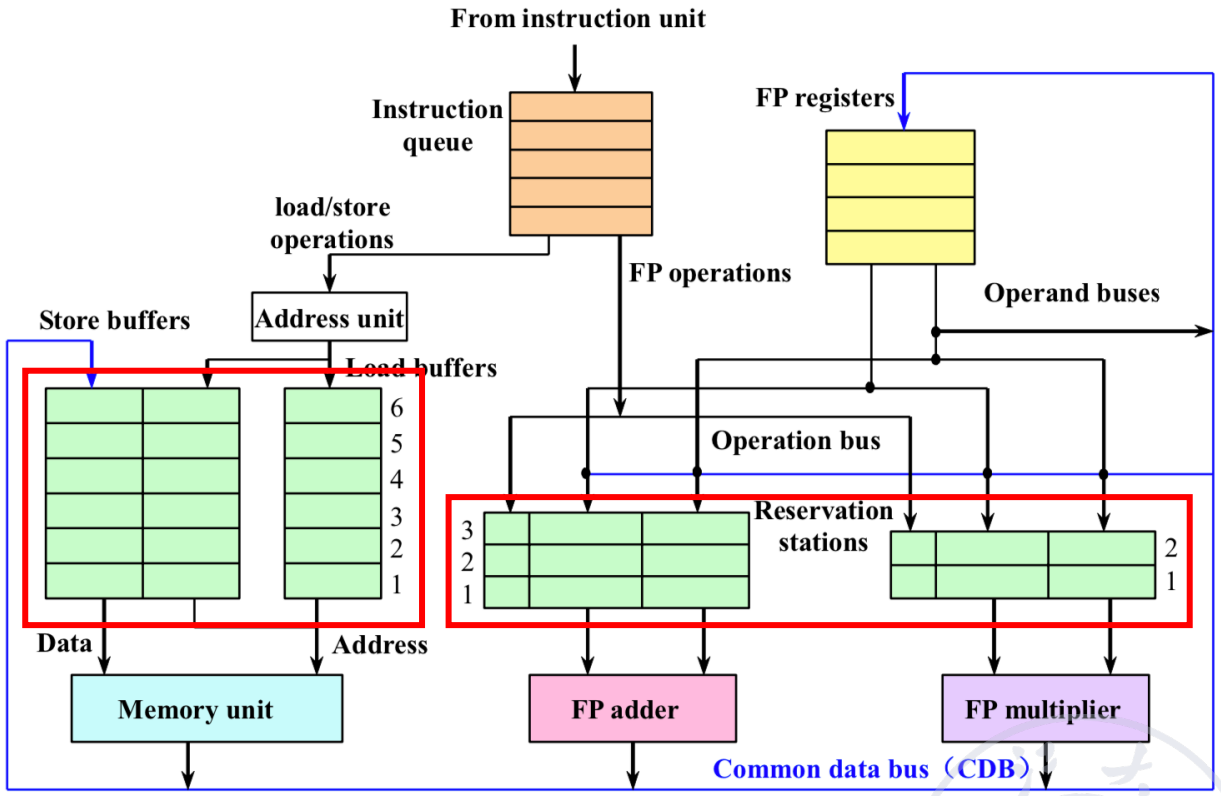
Summary

1. Dynamic scheduling of instructions
   - Scoreboard algorithm
   - Tomaluso algorithm



Summary

- 2. Dynamic branch prediction
  - Branch history table
  - Branch target buffer
  - Hardware-based speculation



Chapter 4

End