Computer Architecture
计 算 机 体 系 结 构

Lecture 4. Instruction-Level Parallelism II

第四讲、指令级并行 II

Chao Li, PhD.

李超 博士

SJTU-SE346, Spring 2019

Review

- Hazards (data/name/control)
- RAW, WAR, WAW hazards
- Different types of functional units
- Dynamic scheduling and out-of-order execution
- Scoreboarding approach vs. Tomasulo's approach

Outlines

- Superscalar Pipeline
- Speculation
- VLIW and EPIC

Limitations of Scalar Pipelines

- A scalar instruction pipeline is a single pipeline with multiple stages organized in a linear sequential order

Three key Limitations

Upper bound on scalar pipeline throughput

- Only one instruction per machine cycle
- Cost-effectiveness of deeper pipelining

Inefficient unification into a single pipeline

- Different hardware resource support needed
- Instructions require long/variable latencies

Efficiency issue due to a rigid execution

- One stalled instruction affect all trailing instructions (backward propagation)

Improving Simple Scalar Pipelines

Involve extensions to alleviate the three limitations

- Ext. 1: **Make it parallel** **(**Superscalar Pipelines**)**
  - Fetch multiple instructions in every machine cycle
  - Decode instructions in parallel
- Ext. 2: **Make it diversified**
  - Multiple and heterogeneous FUs in the EX stage
  - With appropriate number and mix of function units
- Ext. 3: **Make it dynamic**
  - Trailing instructions can bypass a stalled leading instruction
  - Complex multi-entry buffers for buffering instructions in flight

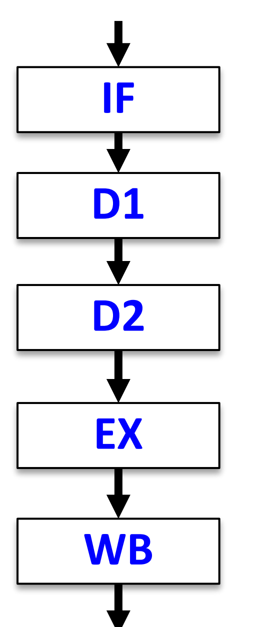
Multiple Issue Processor

- To get a CPI<1, the CPU must be capable of issuing more than one instruction per cycle.

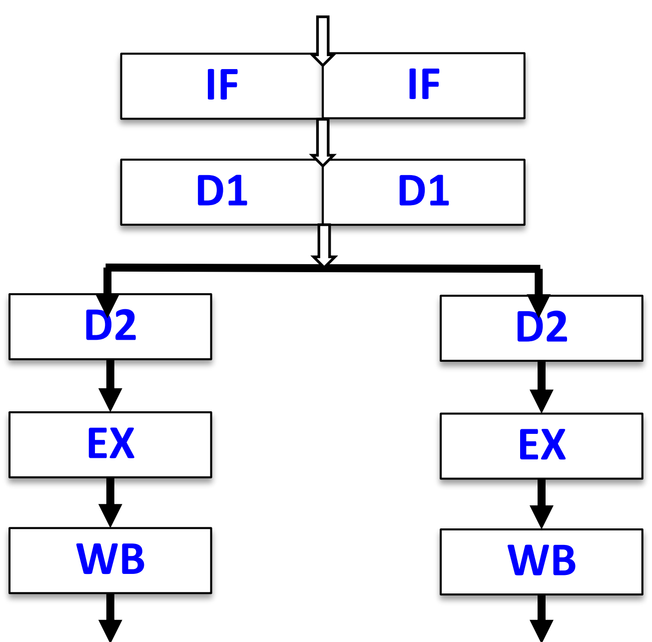
- A ***k*-issue** processor: each stages accept *k* instructions
  - Alpha 21264: four-issue
  - MIPS R1000: four-issue
  - Intel P4: three-Issue

- Multi-issue processors come in two basic flavors:
  - Superscalar processors
  - VLIW (very long instruction word) processors

From Scalar to Superscalar



5-stage i486 scalar pipeline



5-stage Pentium superscalar pipeline: Width = 2

- Performance speedup:
  - Scalar: measured with respect to a no pipelined design
    - Primarily determined by the depth of the scalar pipeline
  - Superscalar: measured with respect to a scalar pipeline
    - Primarily determined by the width of the parallel pipeline

Superscalar Pipeline Example

Width = 3
4 pipelined FUs
2 multi-entry buffers:

- Dispatch Buffer (DB)
  - Centralized
    - Pentium Pro
  - Distributed
    - PowerPC 604

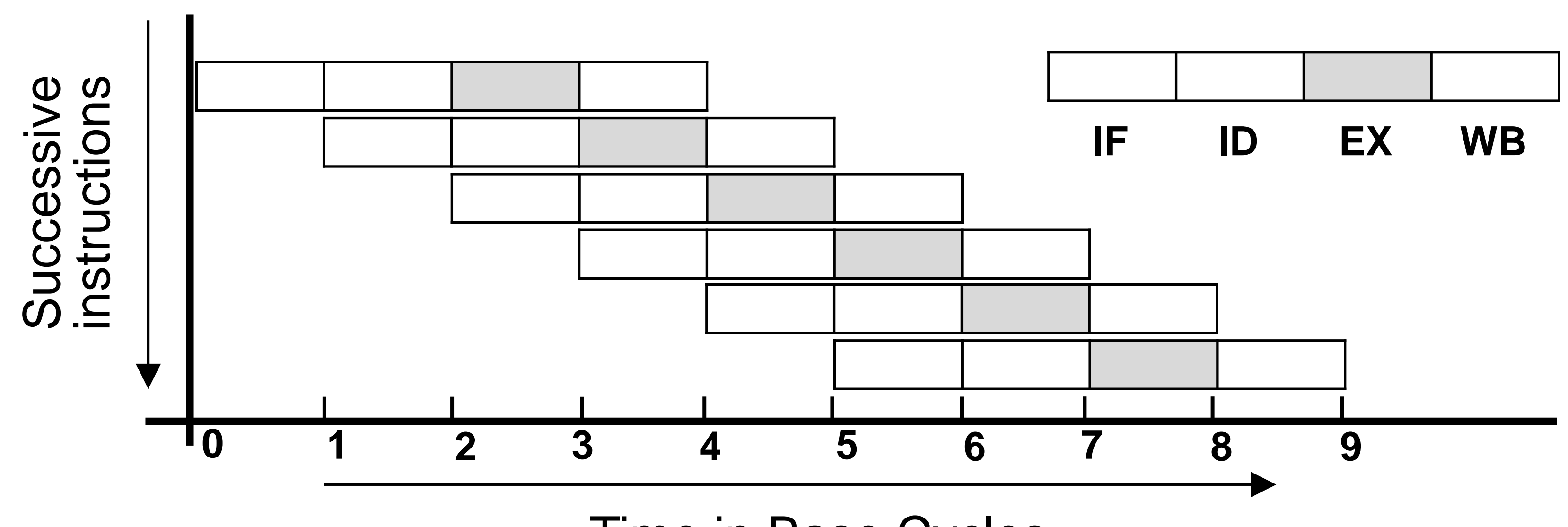
- Completion Buffer

IF
ID
(in order)
Dispatch Buffer
(out of order)
EX
ALU
MEM1
FP1
BR
MEM2
FP2
FP3
(out of order)
WB
Completion Buffer
(in order)

A parallel diversified pipeline

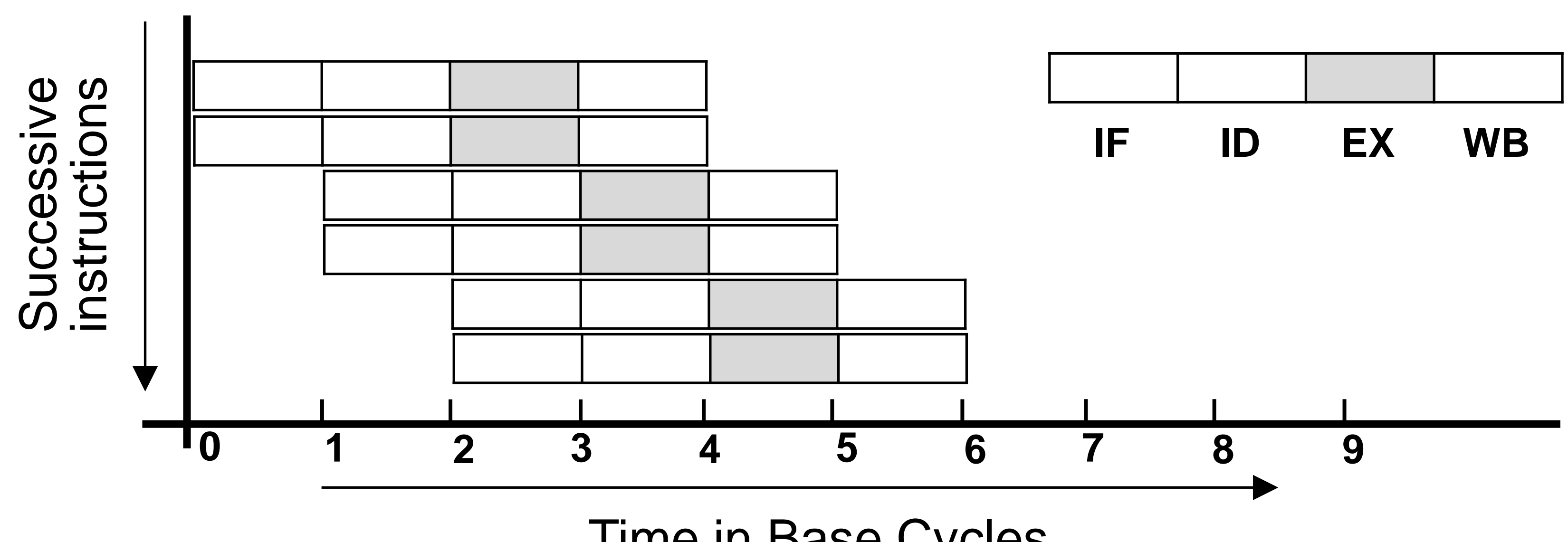
Classifying ILP Machines

- Baseline machine (simple scalar)
  - Instruction-level parallelism required to fully utilize = 1 / cycle
  - Instruction issued per cycle (IPC) = 1
  - Simple operation latency = 1



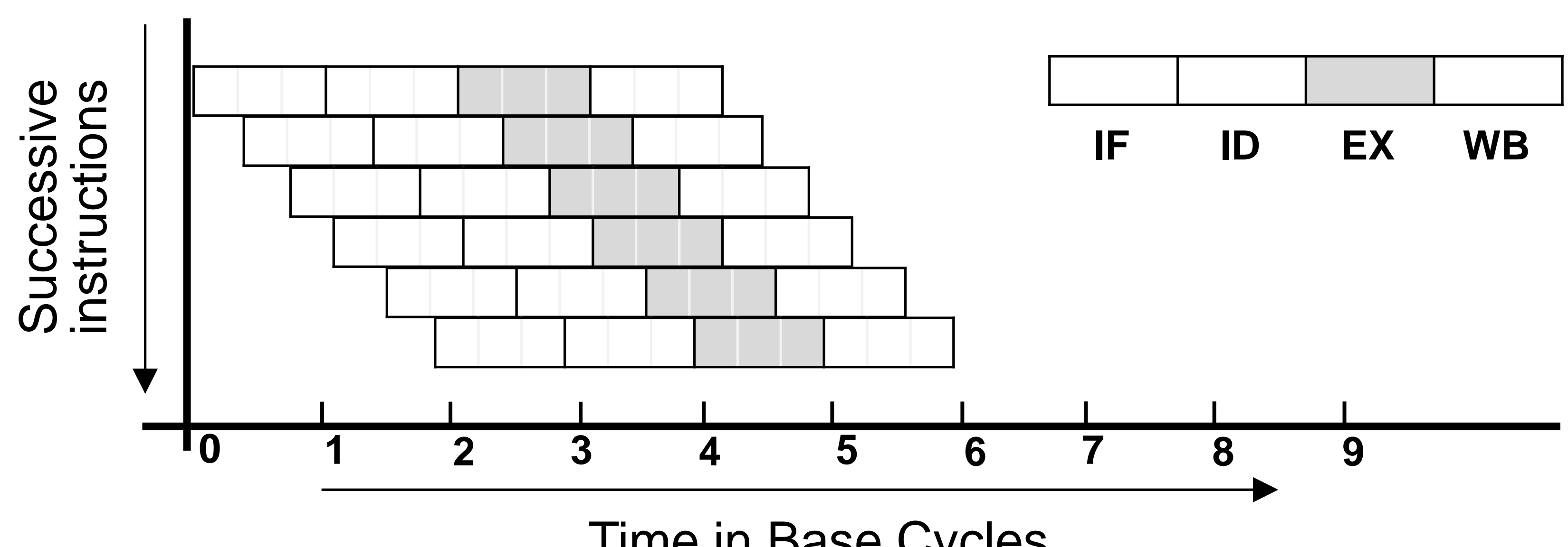
Classifying ILP Machines

- Superscalar machine (n width)
  - Instruction-level parallelism required to fully utilize = n / cycle
  - Instruction issued per cycle (IPC) = n
  - Simple operation latency = 1



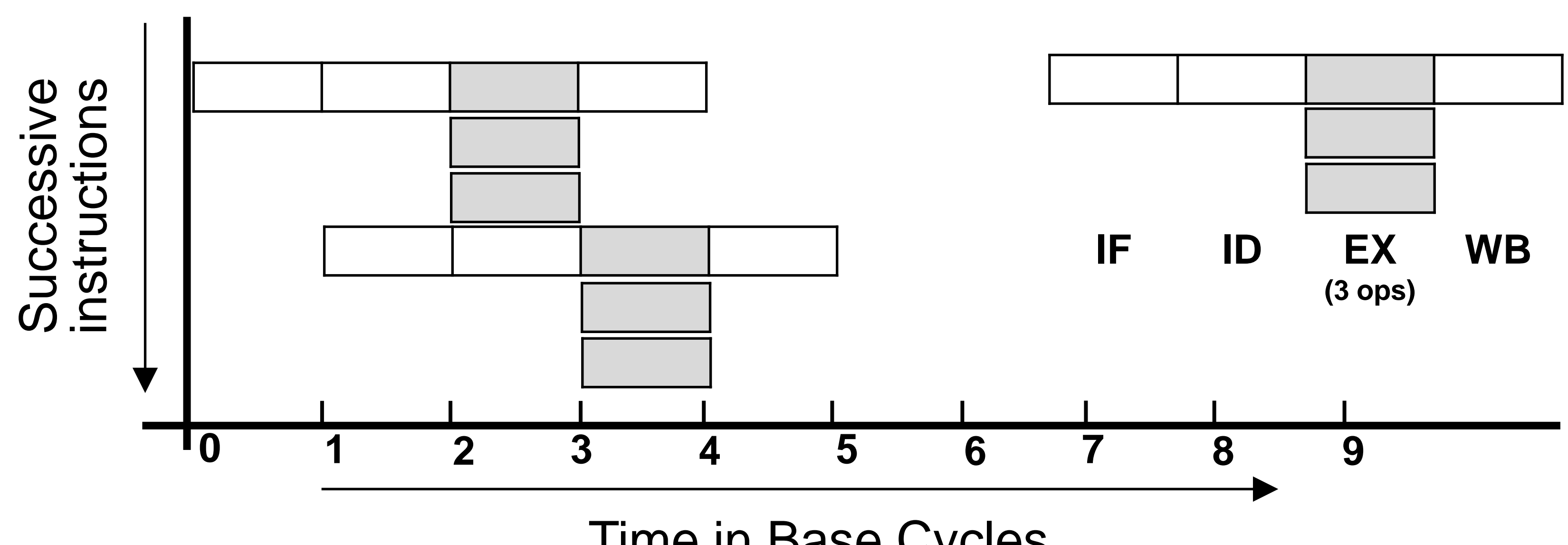
Classifying ILP Machines

- Superpipelined machine (degree m)
  - Instruction-level parallelism required to fully utilize = m / base cycle
  - IPC = 1, but the cycle time is 1/m of the base machine
  - Simple operation latency = m



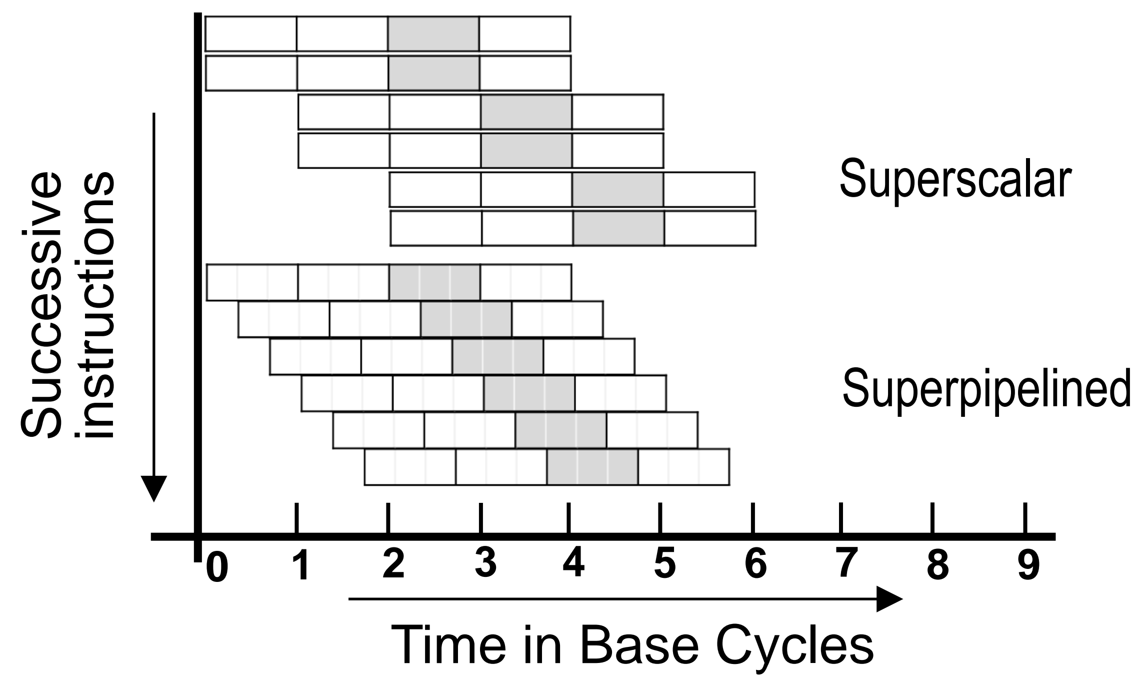
Classifying ILP Machines

- VLIW machine
  - Instruction-level parallelism required to fully utilize = n / cycle
  - IPC = n instructions / cycle = 1 VLIW / cycle
  - Simple operation latency = 1



Superscalar vs. Superpipelined

- Roughly equivalent performance
  - If n = m, then both have about the same throughput
  - Parallelism exposed in different way (spatial vs. temporal)



Outlines

- Superscalar Pipeline
- Speculation
- VLIW and EPIC

Branch Prediction

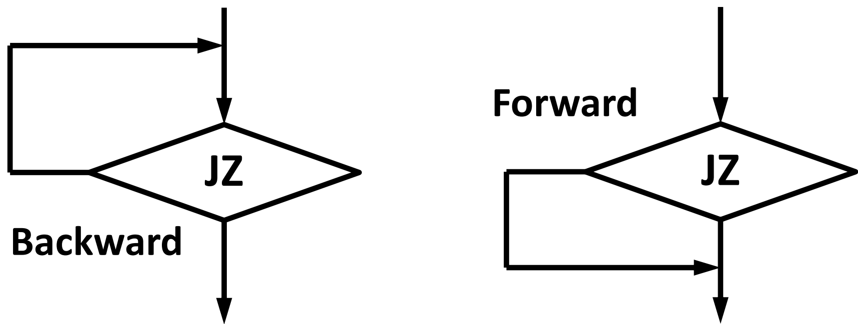
- 17% instructions of *gcc* is conditional branch

Instruction	Taken known?	Target know?
BEQZ/BNEZ	After Reg. Fetch	After Inst. Fetch
J	Always taken	After Inst. Fetch
JR	Always taken	After Reg. Fetch

- Two fundamental components of branch prediction
  - Branch target speculation and Branch condition speculation
- Branch penalties limit performance
  - Performance = f(accuracy, cost of misprediction)

Static Branch Prediction

- Static: predict branch behavior at compile-time
  - Individual branch is often highly biased toward taken or untaken
    - Predict a branch as always taken
    - Predict on the basis of branch direction
    - Predict on the basis of profile information collected earlier



- Dynamic: based on the behavior of each branch
  - May change predictions for a branch over the life of a program
  - HW support: branch history tables, branch target buffers, etc.

History-based Prediction

Fetch PC

x x

2^k entry
2 bits/entry

k

Instruction Cache

BHT Index

Branch History Table (BTH)

Opcode

Offset

+

Branch?

Target PC

Taken / not Taken

- 4K-entry BHT, 2 bits/entry: ~ 80-90% correct predictions

2-bit Branch Prediction

Taken

11 / 1

Not Taken

10 / 1

Taken

Taken

Not Taken

01 / 0

Not Taken

00 / 0

Taken

Not Taken

- Change prediction only if **get misprediction twice**
- n-bit Branch Prediction?
  - Not much more accurate than the 2-bit version (n>2)
  - Will likely predict incorrectly twice (n==1)

Branch Target Buffer

- BTB contains two fields
  - Branch instruction address: BIA
  - Branch target address: BTA

PC

BTB

I-Cache

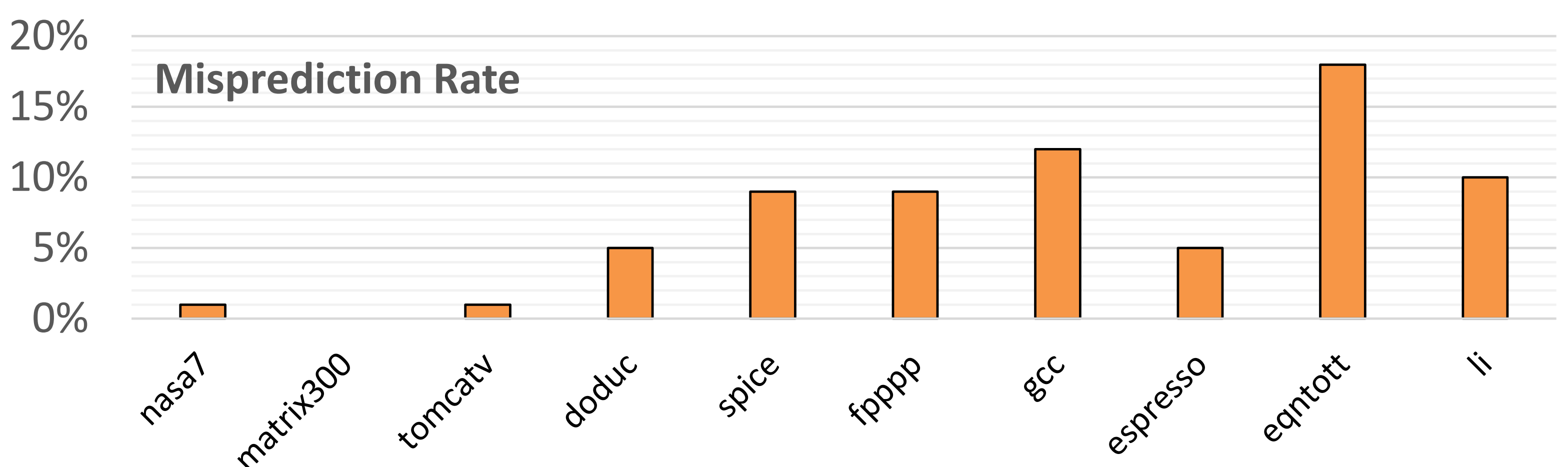
BIA field	BTA field	Branch History

Speculative target address

FSM

Predict taken/not taken

Mis-prediction Recovery



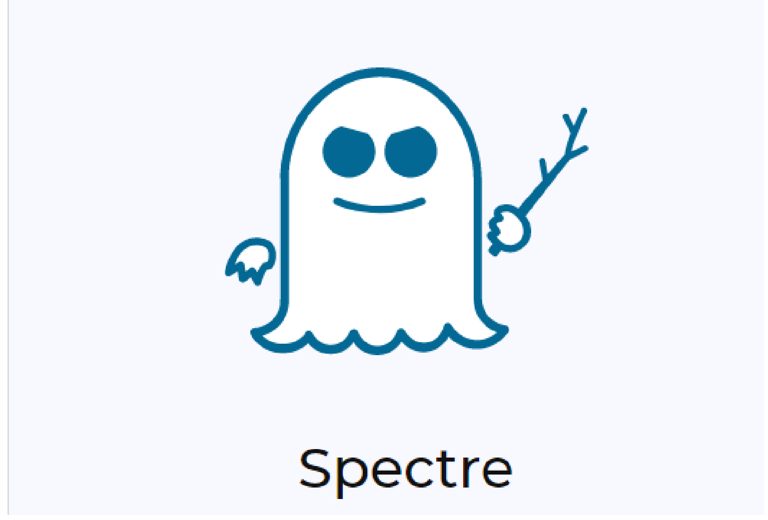
- In-order processor
  - Kill all instructions in pipeline behind branch: No instruction issued after branch can write-back before branch resolves
- Out-of-order processor
  - **Multiple instructions following branch in program order can complete before branch resolves**

Precise Exception

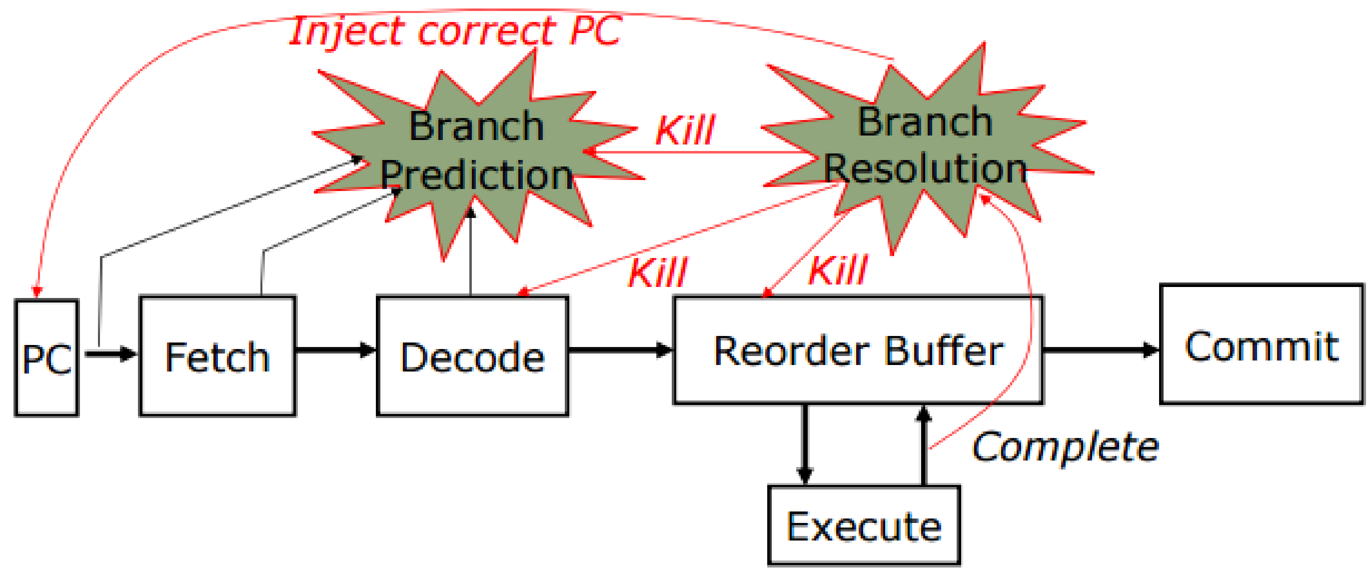
- An exception is precise if
  - The exception is associated with a particular faulting instruction ***Fi***
  - All instructions (in **program order**) prior to ***Fi*** complete execution
  - ***Fi*** and all subsequent instructions **appear to have never started**
- Precise exception & misprediction recovery: same problem
- Maintain precise exception implies in-order completion
- Need HW buffer for results of uncommitted instructions:
  - order in which instructions are completed
  - order in which memory is accessed

Discussion

- Although the instructions executed out of order do not have any visible architectural effect on registers or memory, they have micro-architectural side effects.



Hardware-based Speculation

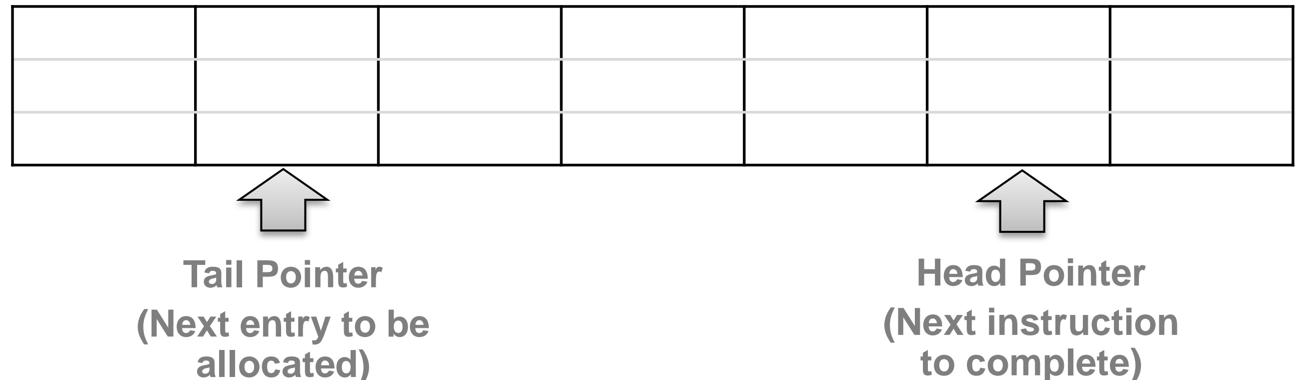


- **Hardware-based speculation** involves:
  - Dynamic branch prediction
  - Dynamic scheduling of basic blocks
  - OoO execution with precise exception (in-order commit)
- **Commit**: write instruction results to architectural state
  - To graduate: oldest instruction + valid results

Reorder Buffer (ROB)

- A completion buffer managed as circular queue (FIFO)
  - Contains all the instructions that are in flight
  - Holds result between complete and commit

- ROB fields
  - **Instruction type**: branch/store/ALU
  - **Destination**: supplies register number or memory address
  - **Value**: holds results until the instruction commits



Tomasulo + ROB

- Extended “Tomasulo hardware” to support speculation
  - PowerPC 603/604/G3/G4
  - MIPS R10000/R12000
  - Intel Pentium II/III/4
  - Alpha 21264
  - AMD K5/K6/Athlon
- Combined ROB-RS structure called instruction window
  - The size of the instruction window determines the degree of ILP that can be achieved by the machine
- Now 4 stages (issue, execute, write, commit) are needed

Tomasulo

Instruction Status

		j	*k*	IS	EX	WR
1. LD	F6	34	R2			
2. LD	F2	45	R3			
3. MUL.D	F0	F2	F4			
4. SUB.D	F8	F6	F2			
5. DIV. D	F10	F0	F6			
6. ADD.D	F6	F8	F2			

Register Results

F0	
F2	
F4	
F6	
F8	
F10	

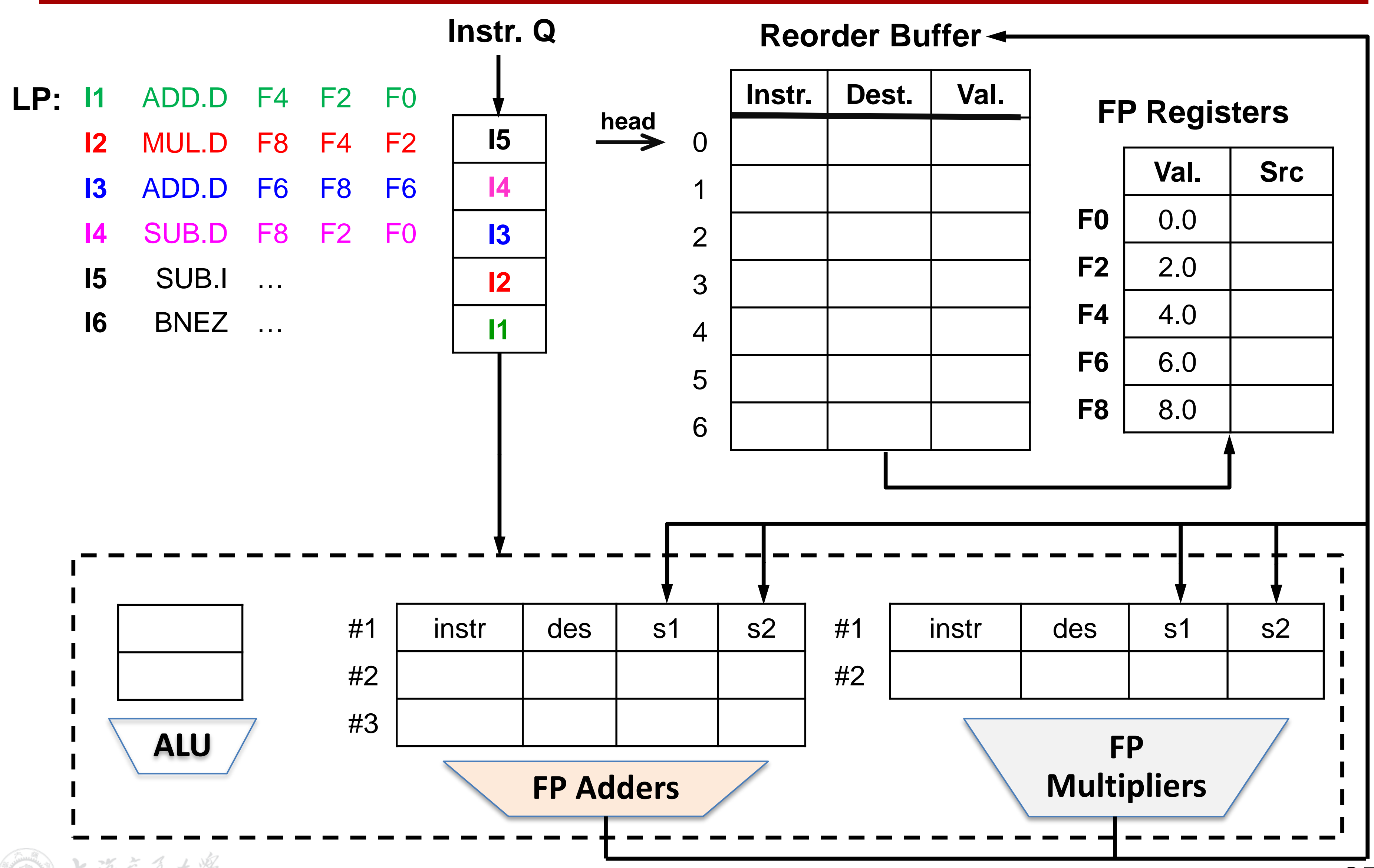
Reservation Stations

Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

	Busy	*Addr.*
Load1	No	
Load2	No	
Load3	No	

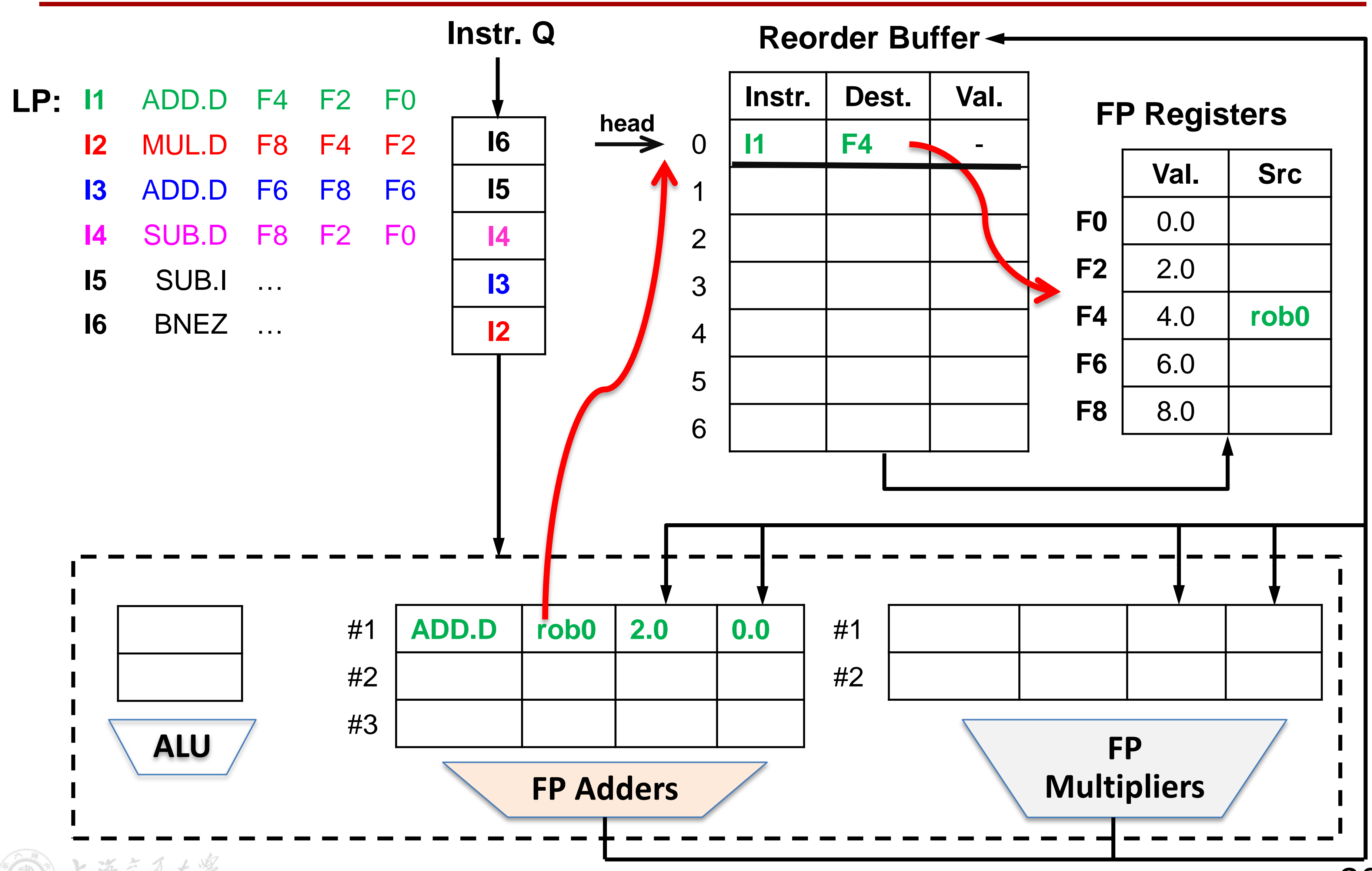
Load Buffer

“Tomasulo+ROB” Example



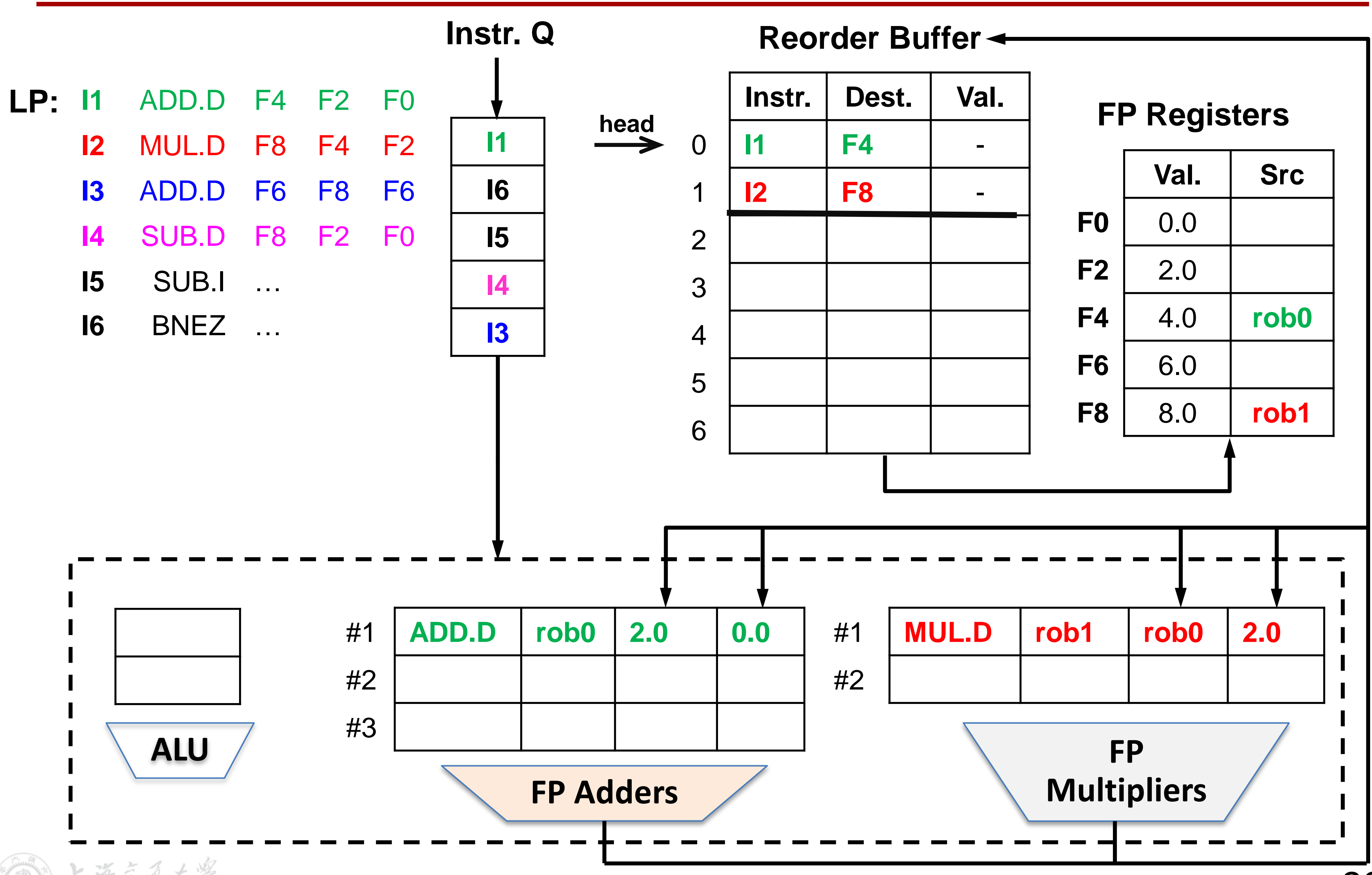
上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 1



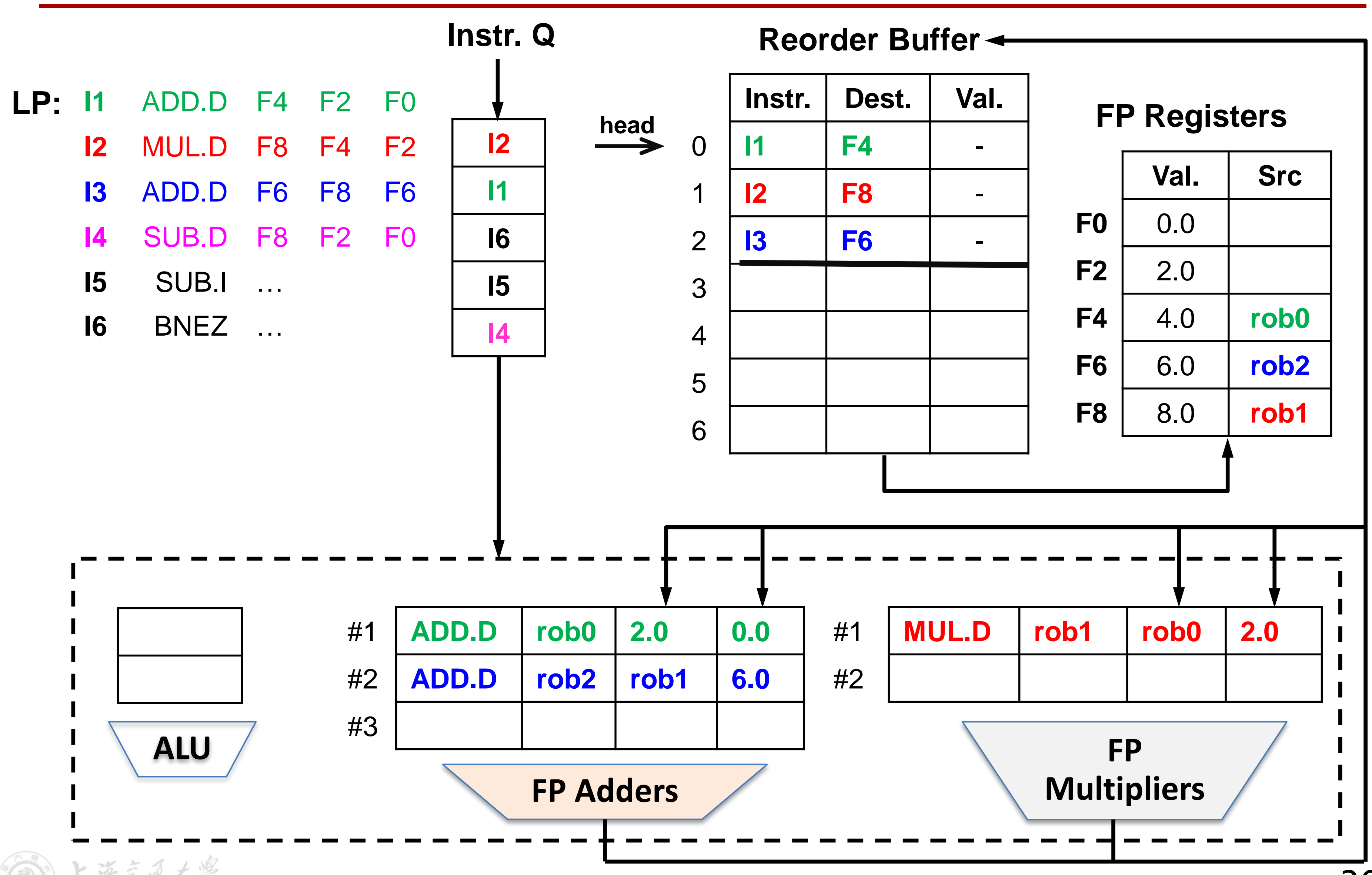
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 2

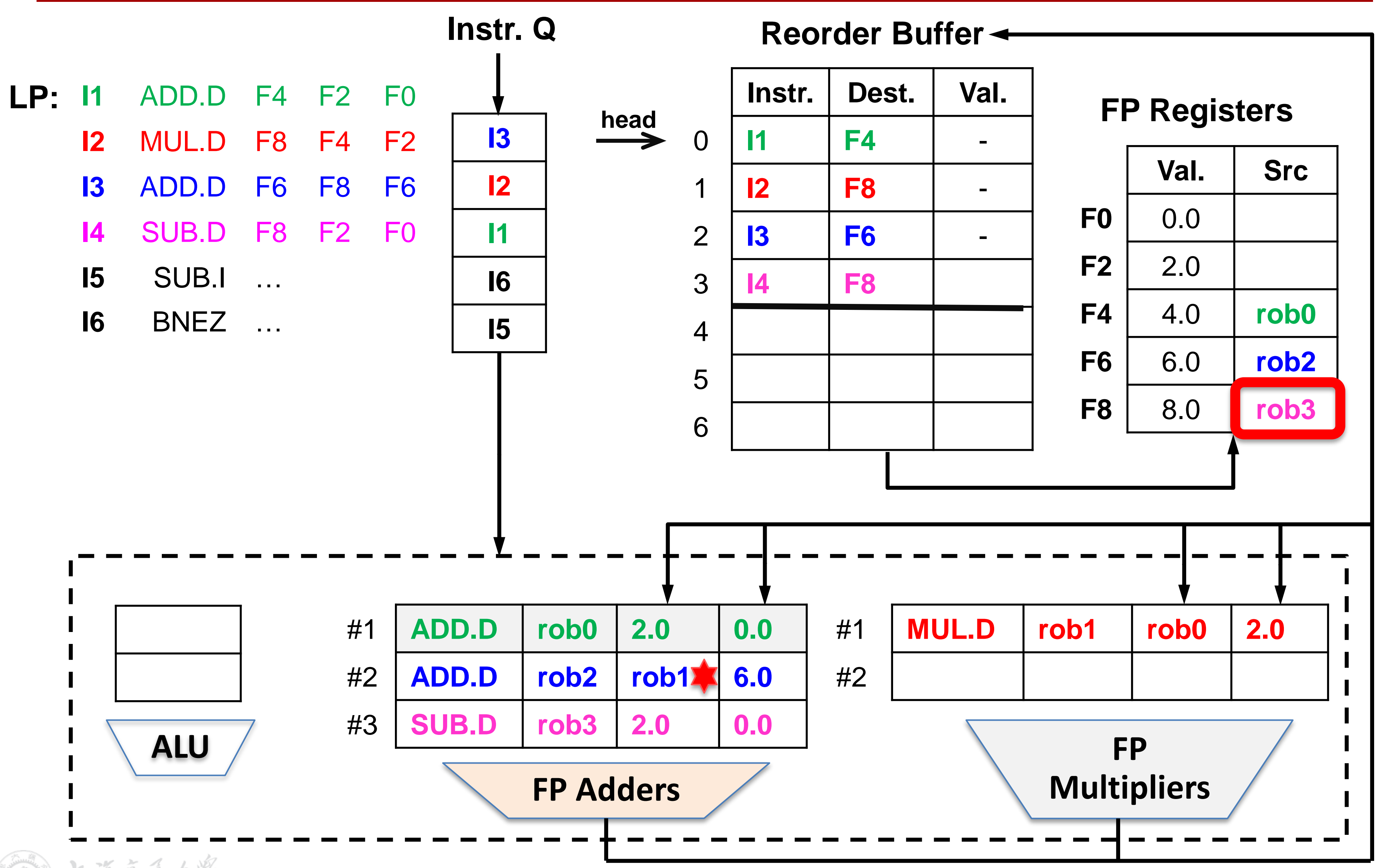


上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 3

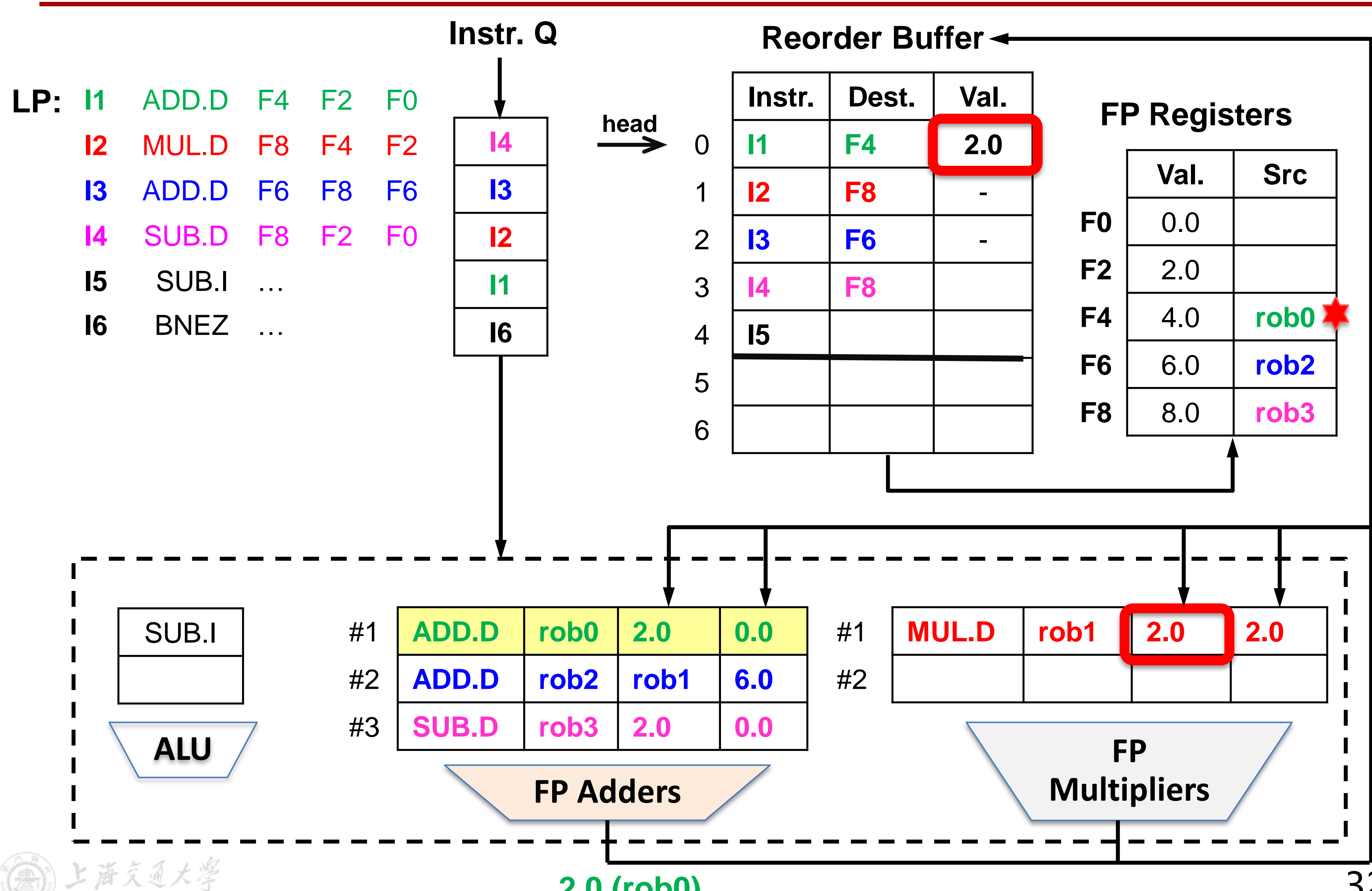


"Tomasulo+ROB" Example – Cycle 4



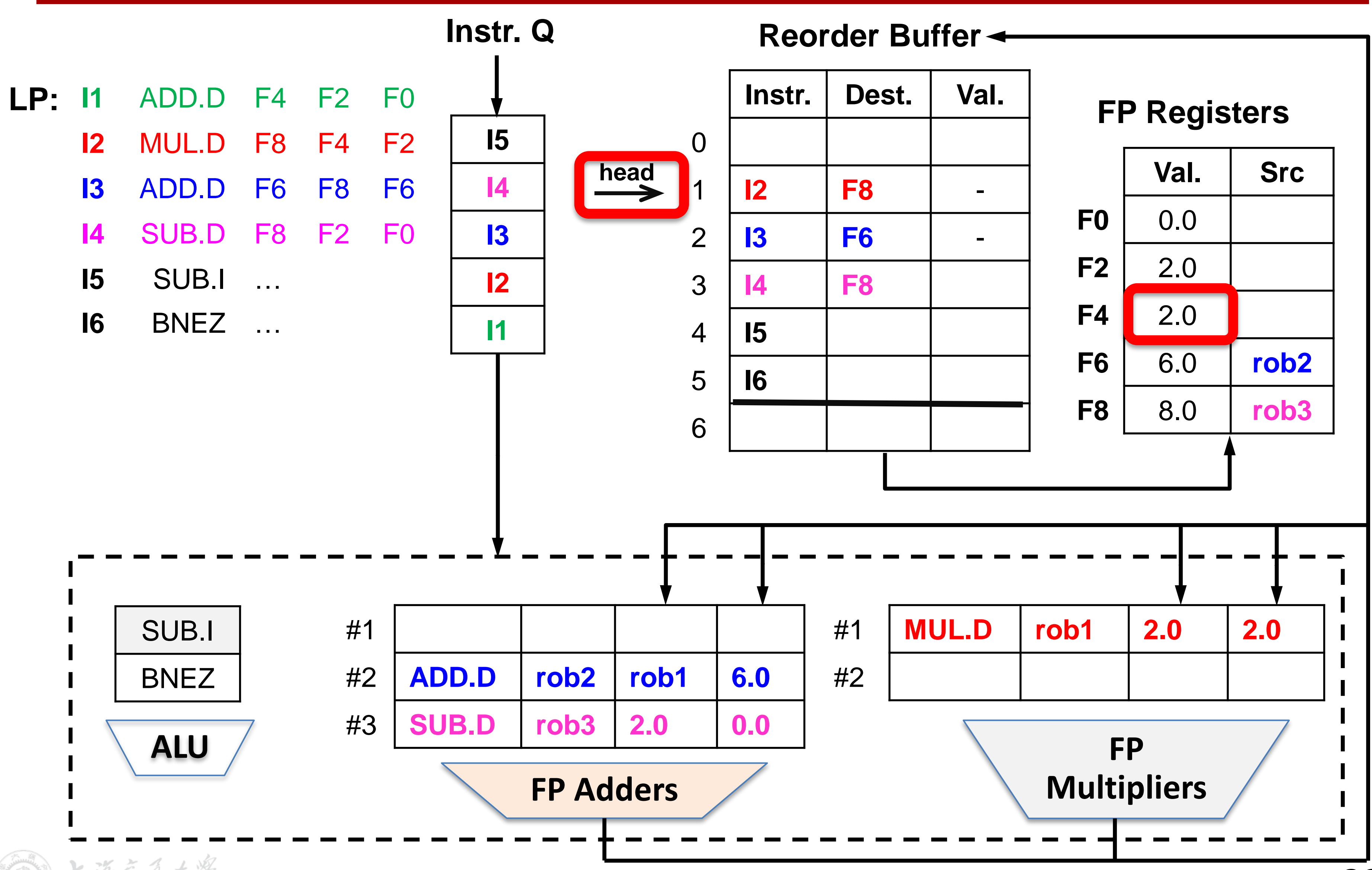
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 5



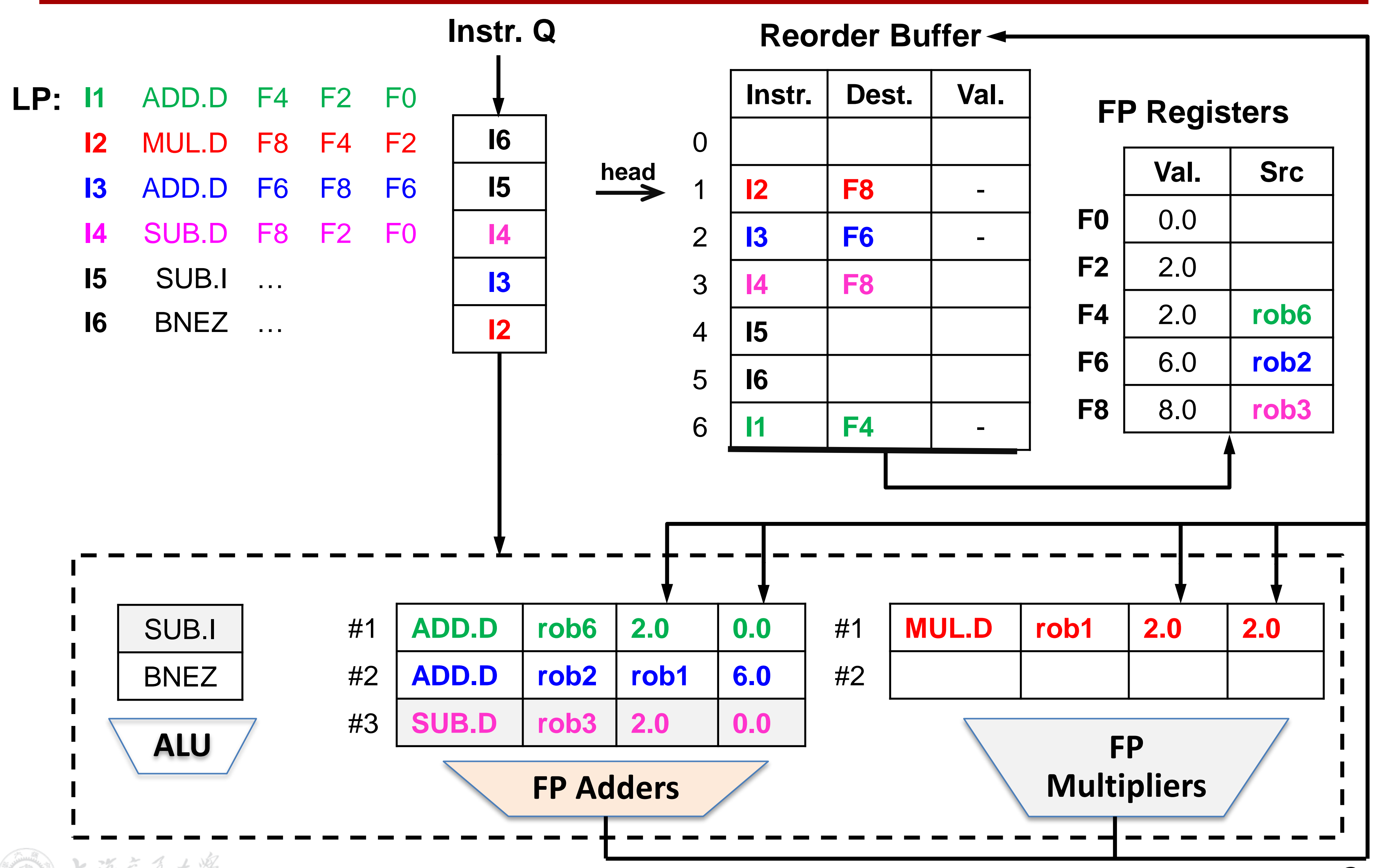
上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 6



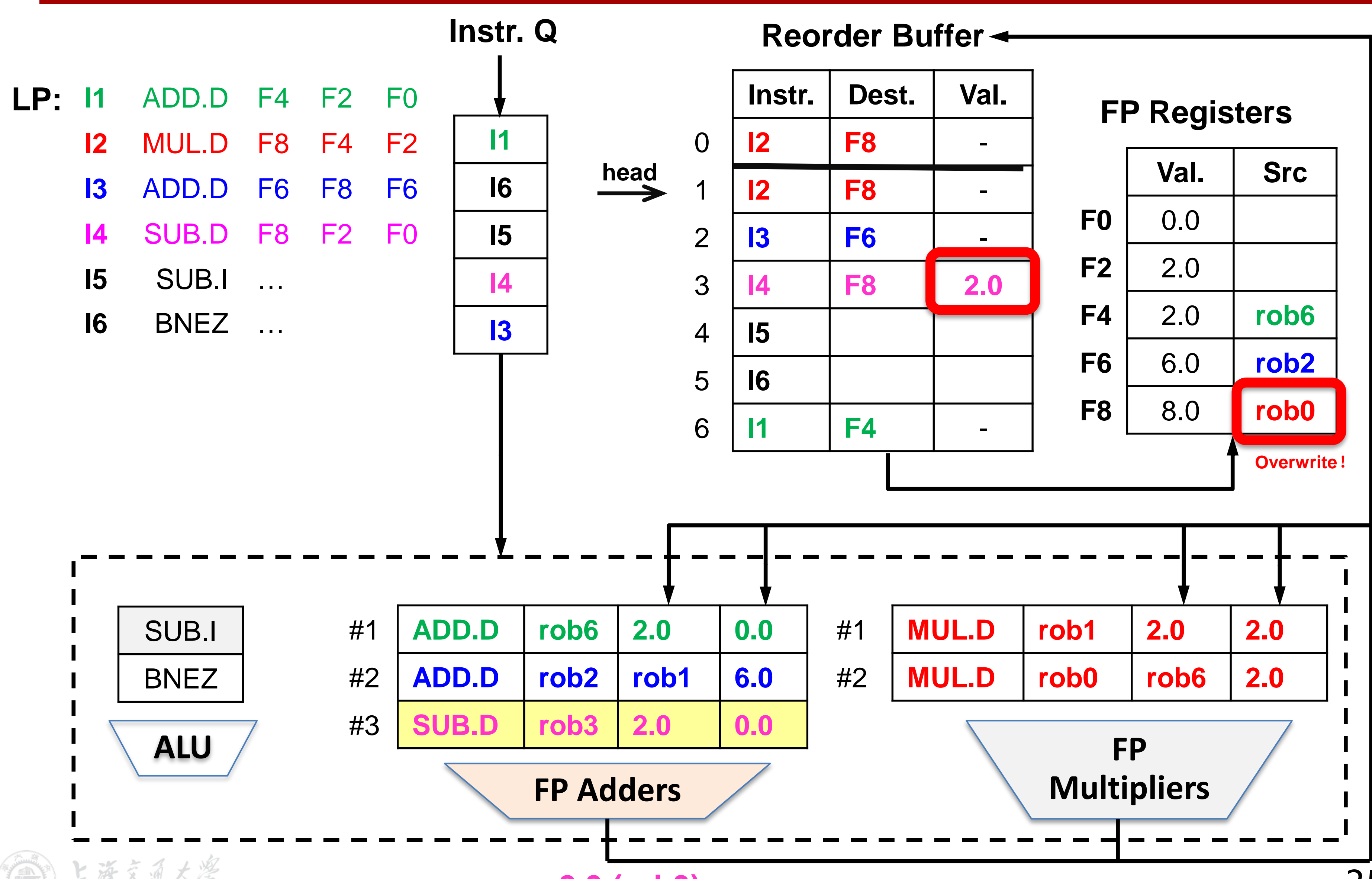
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 7



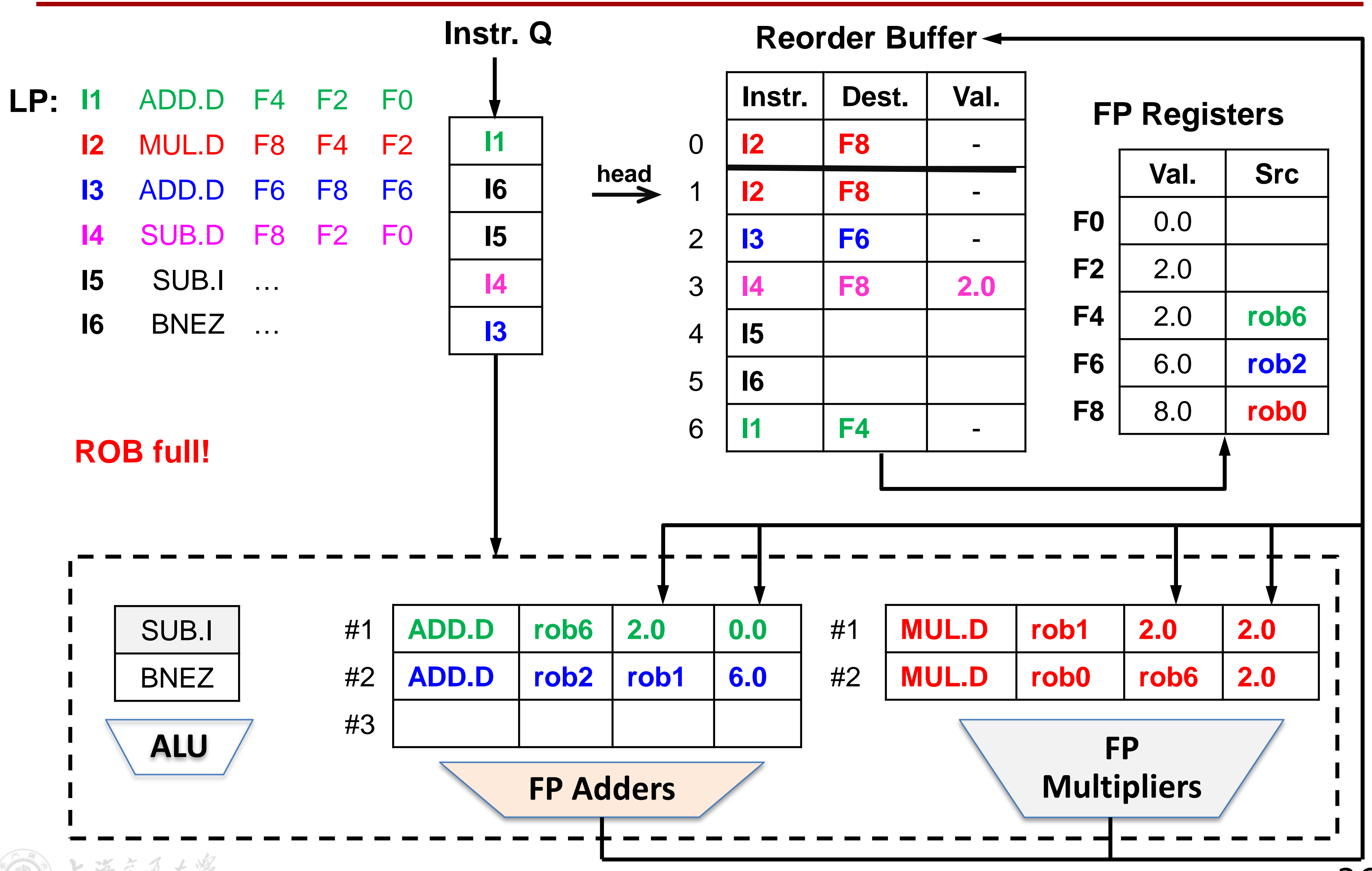
上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 8



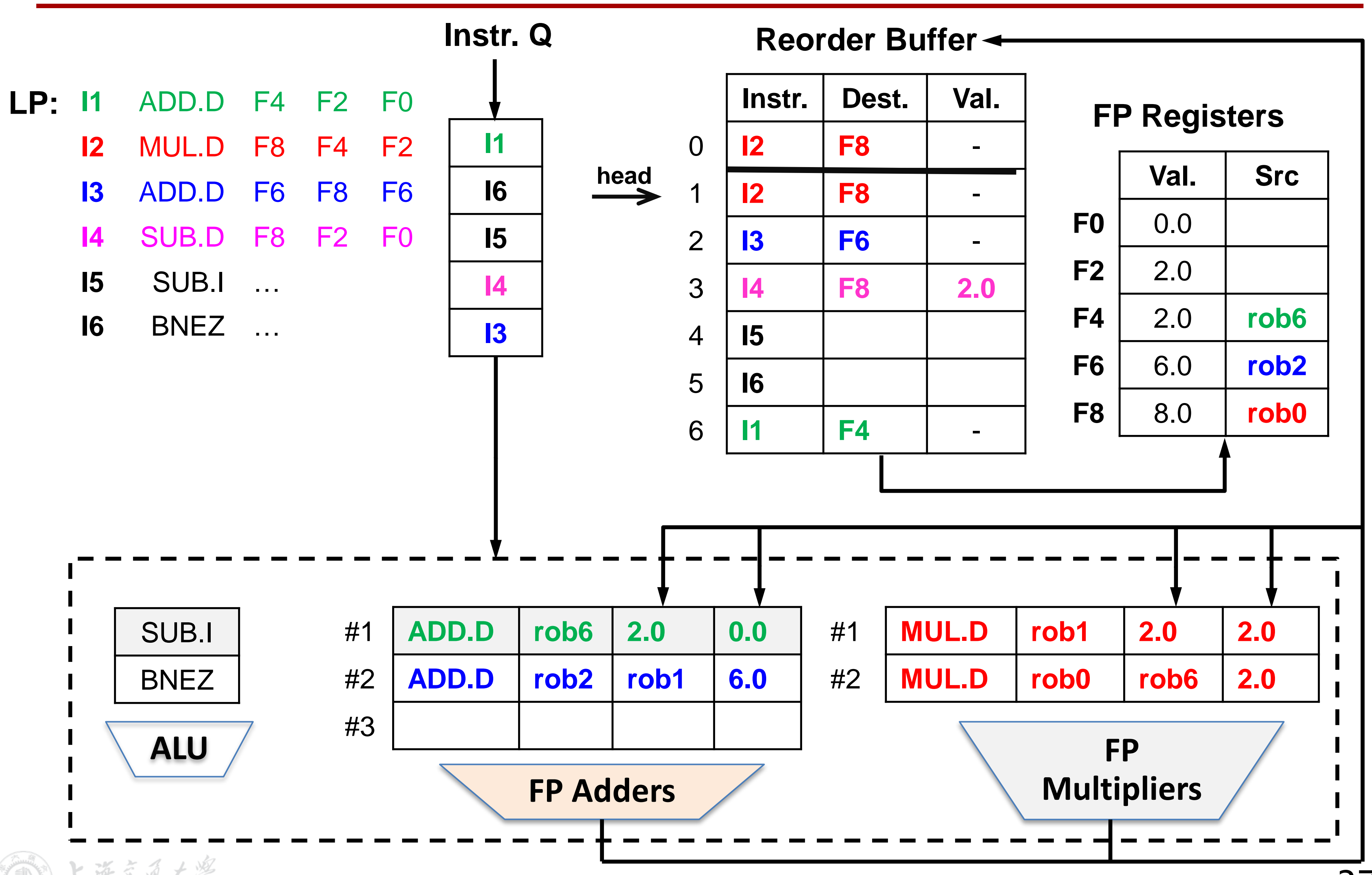
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 9



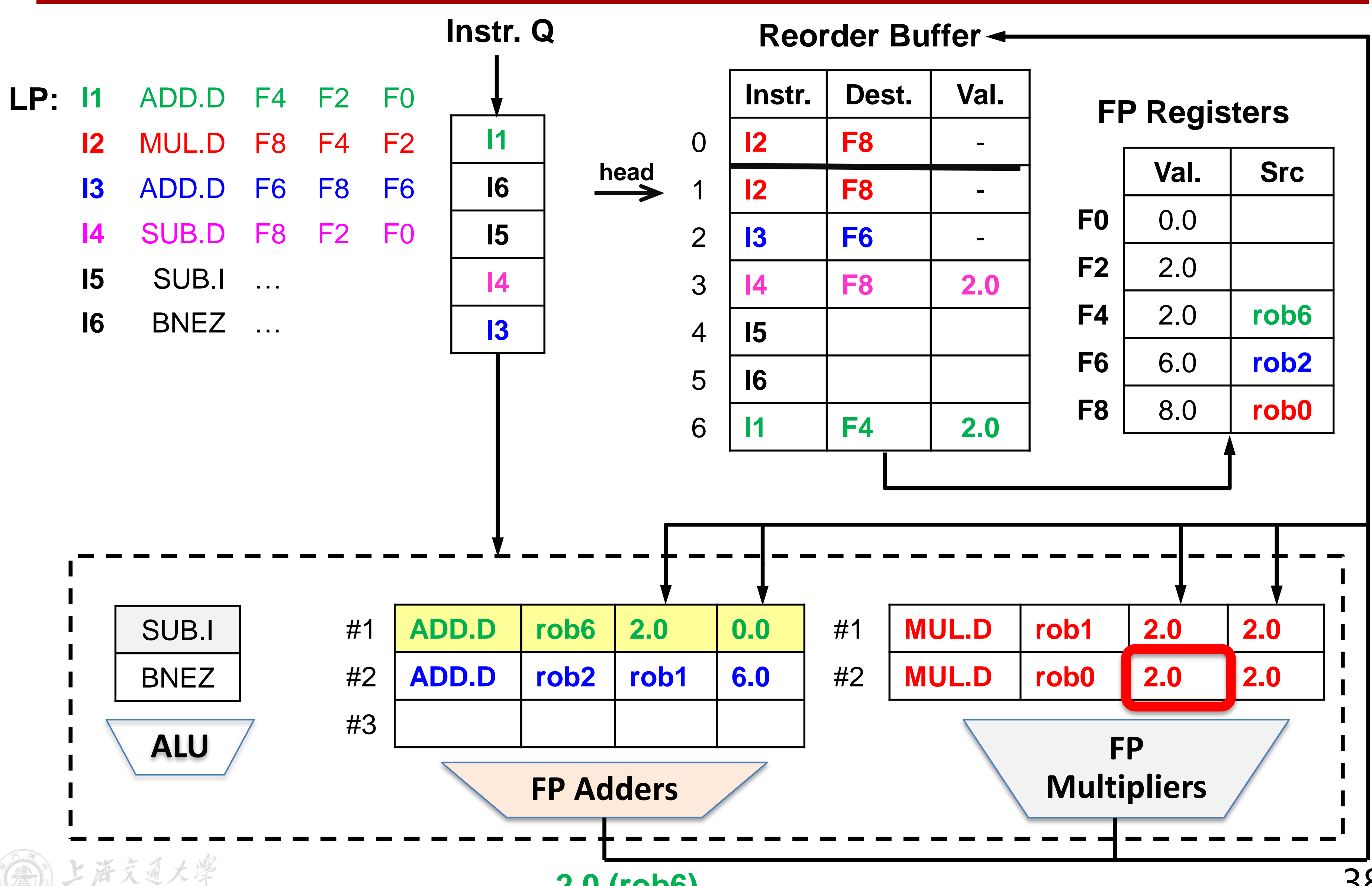
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 10

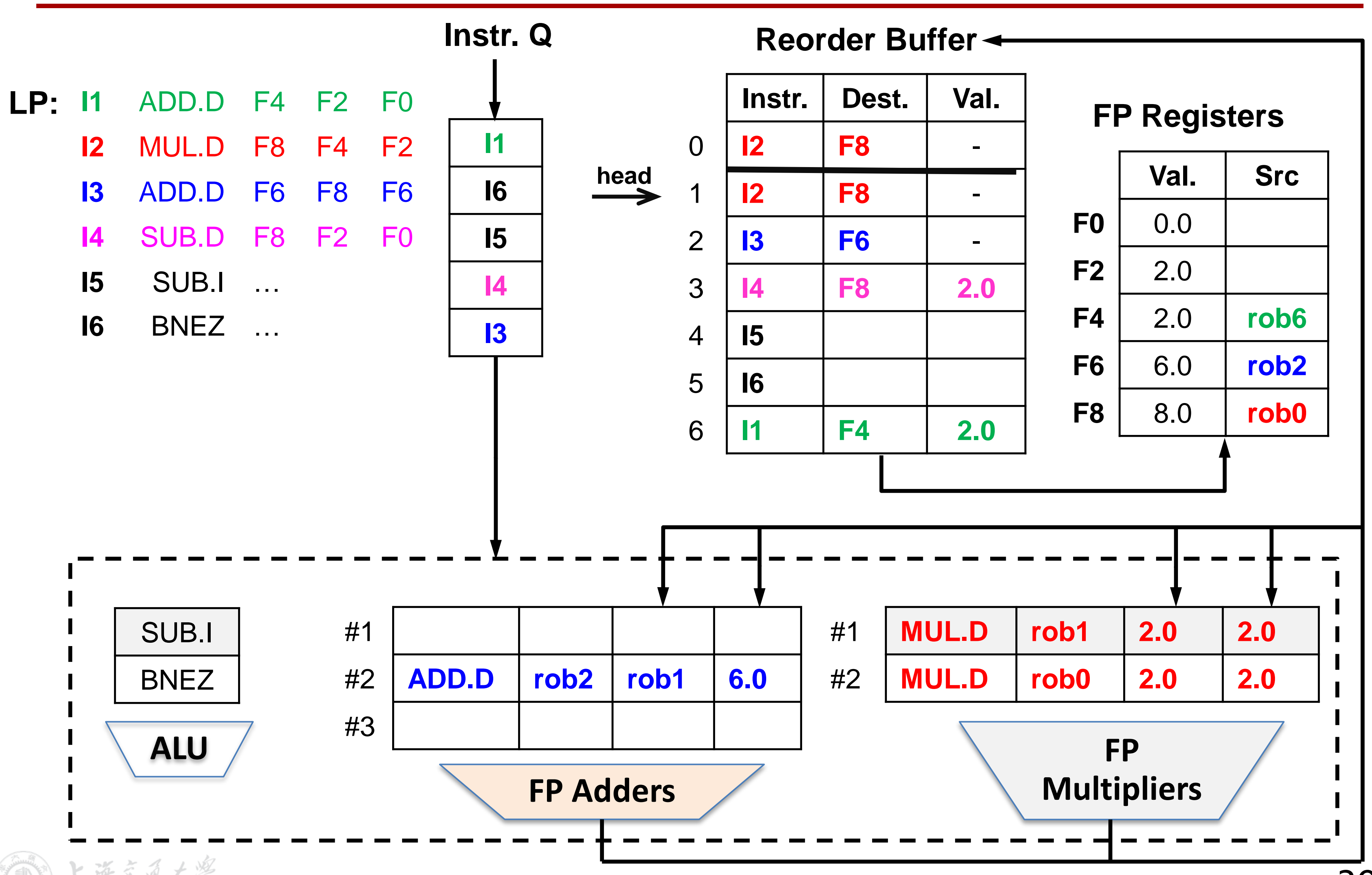


上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 11

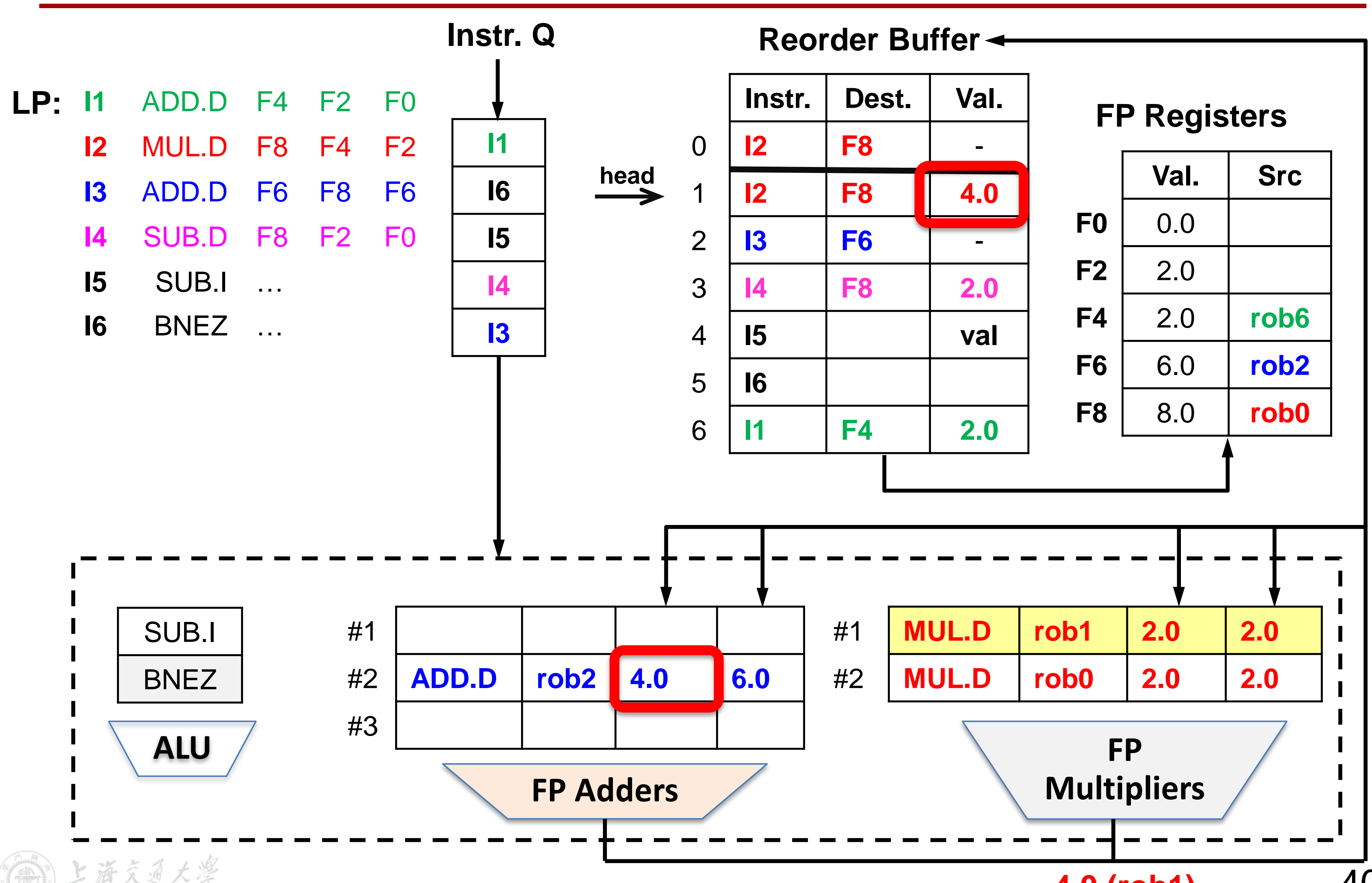


"Tomasulo+ROB" Example – Cycle 16



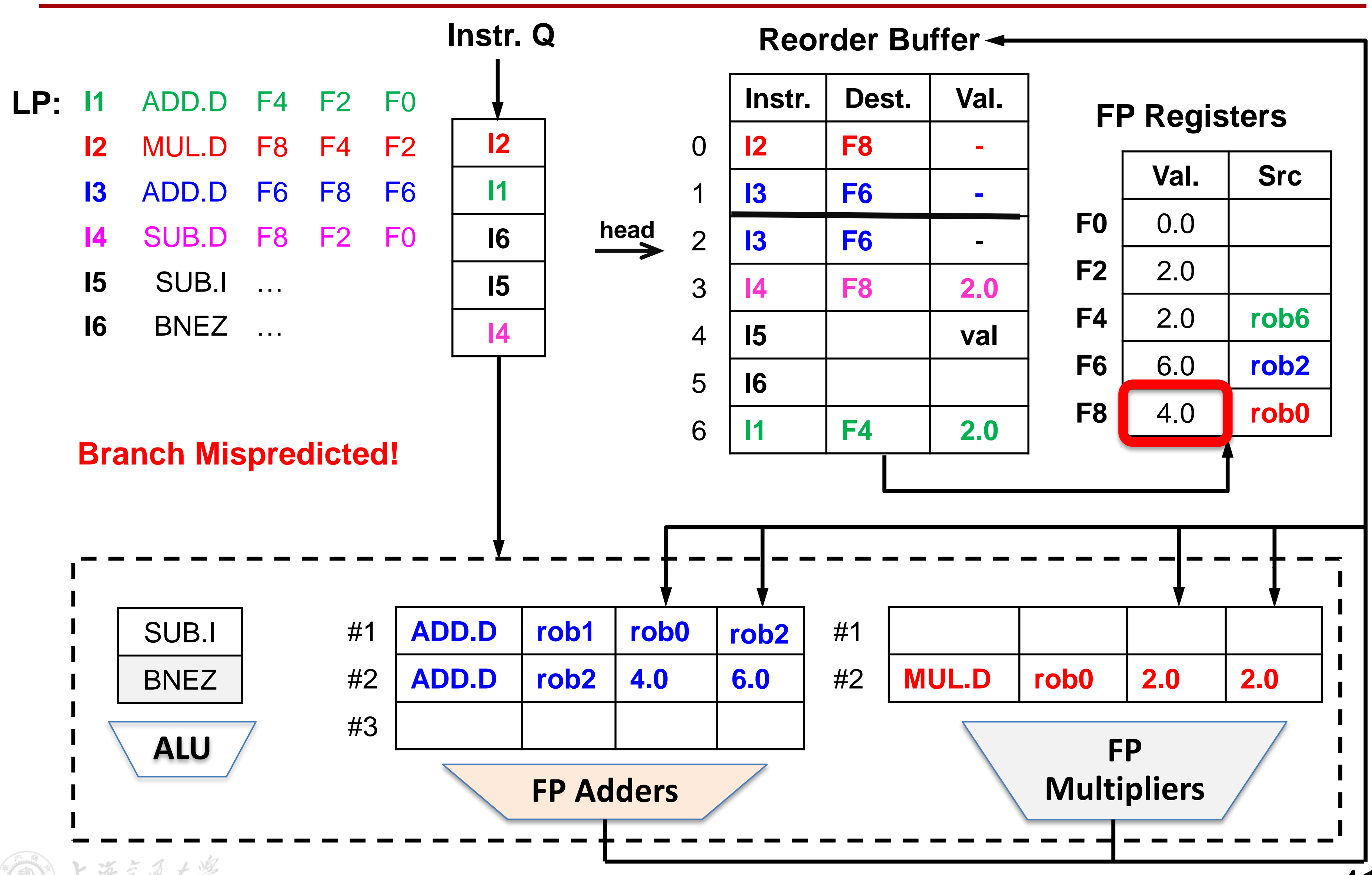
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 17



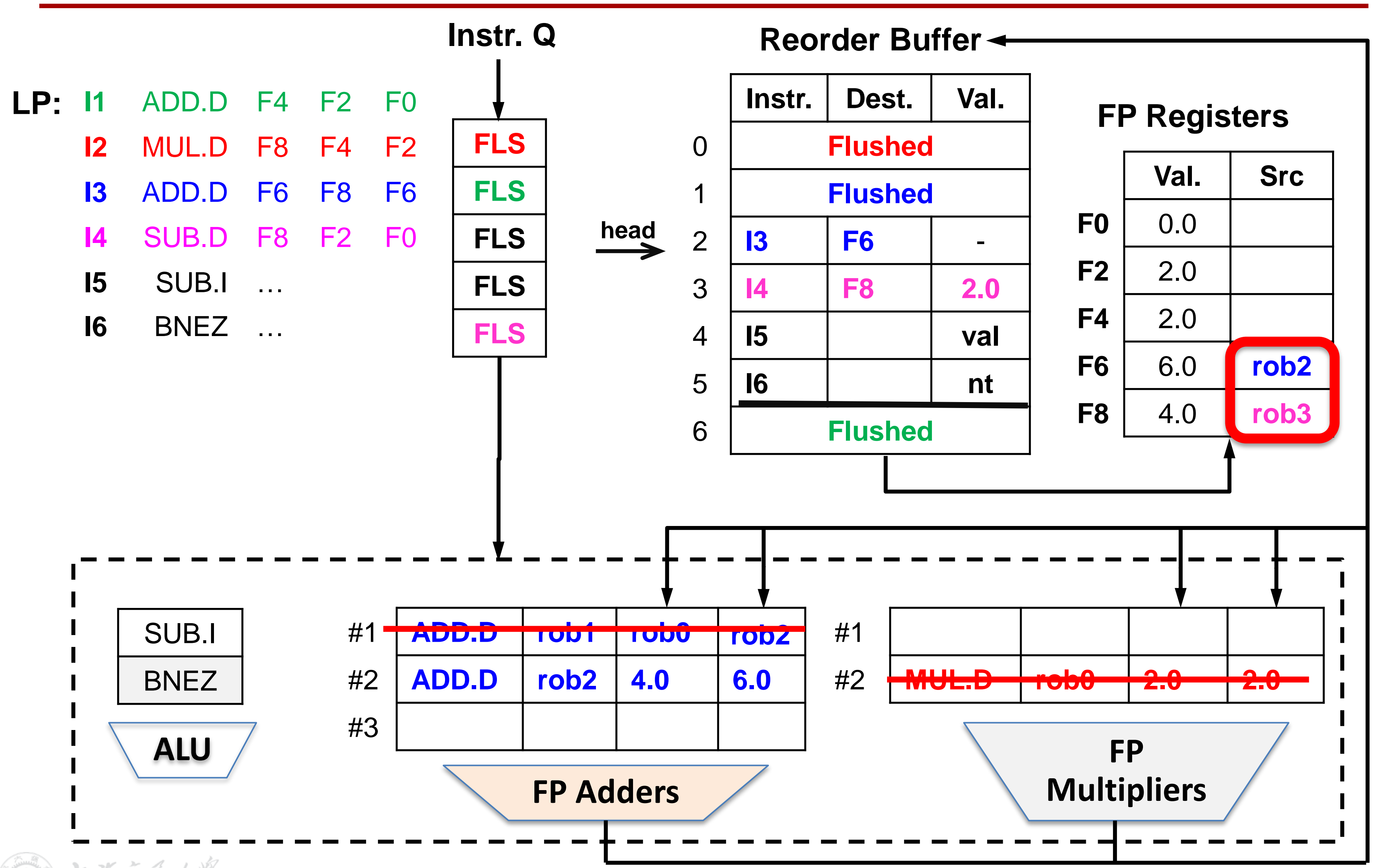
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 18



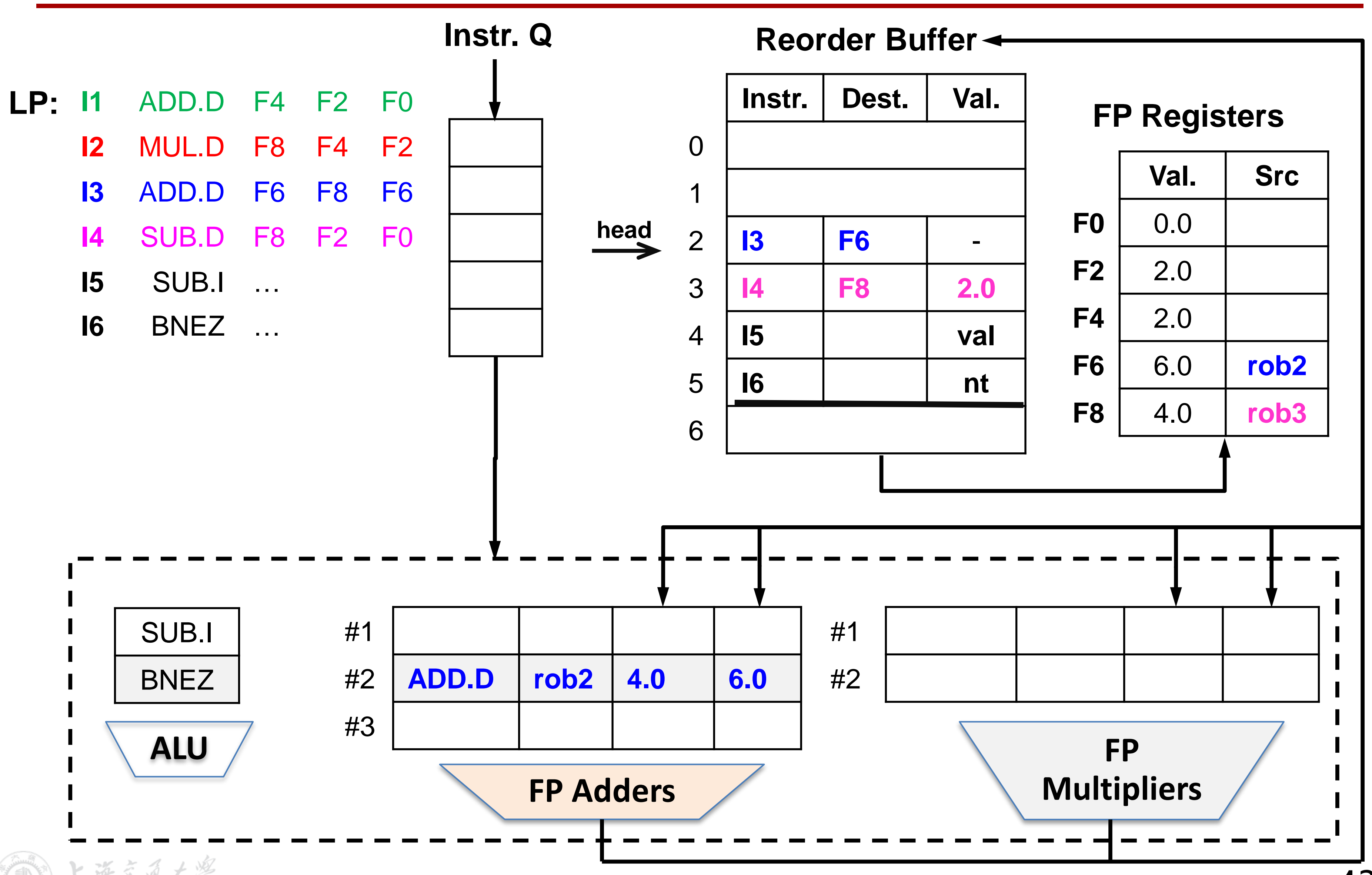
上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 19



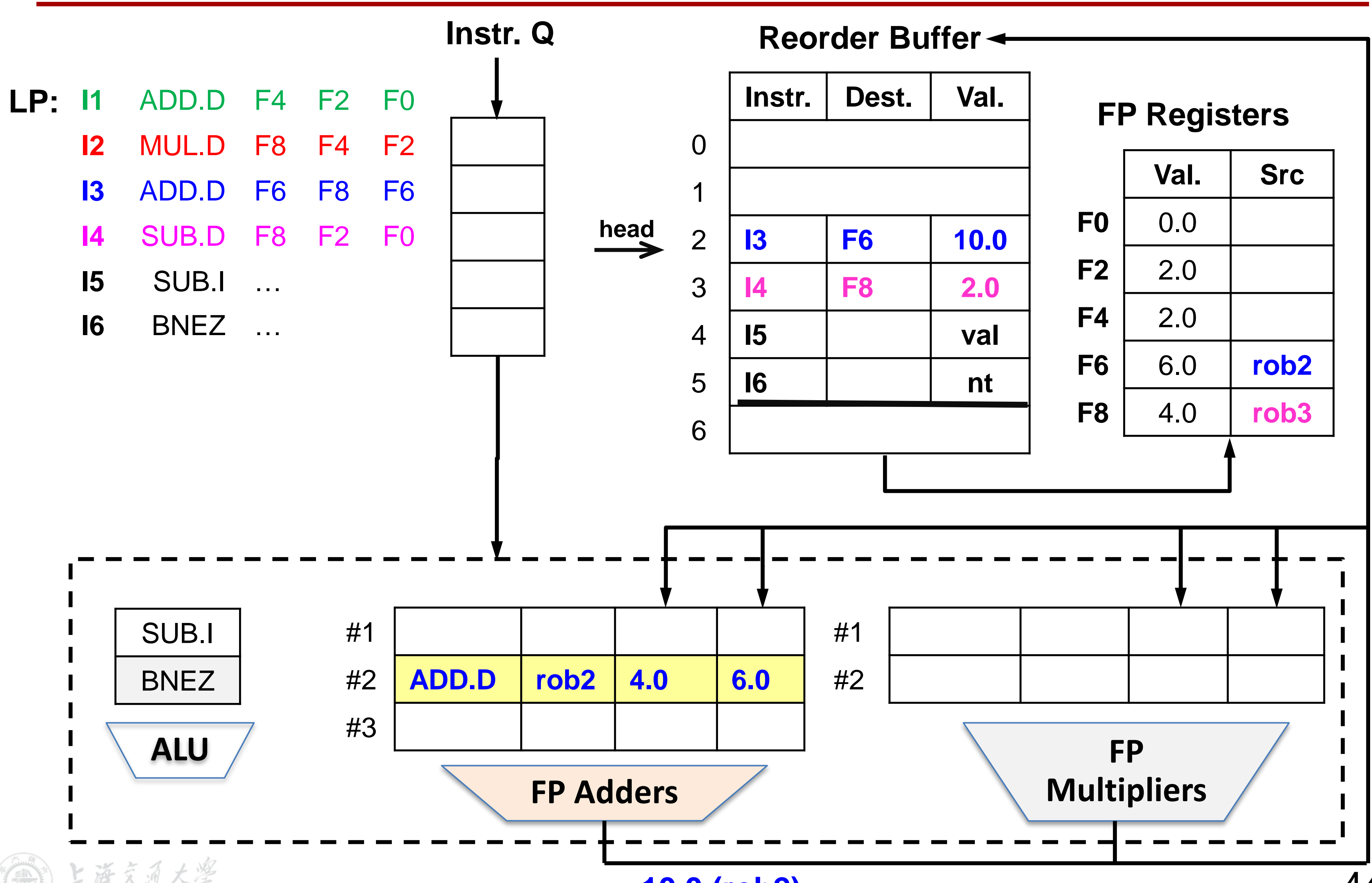
上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 20

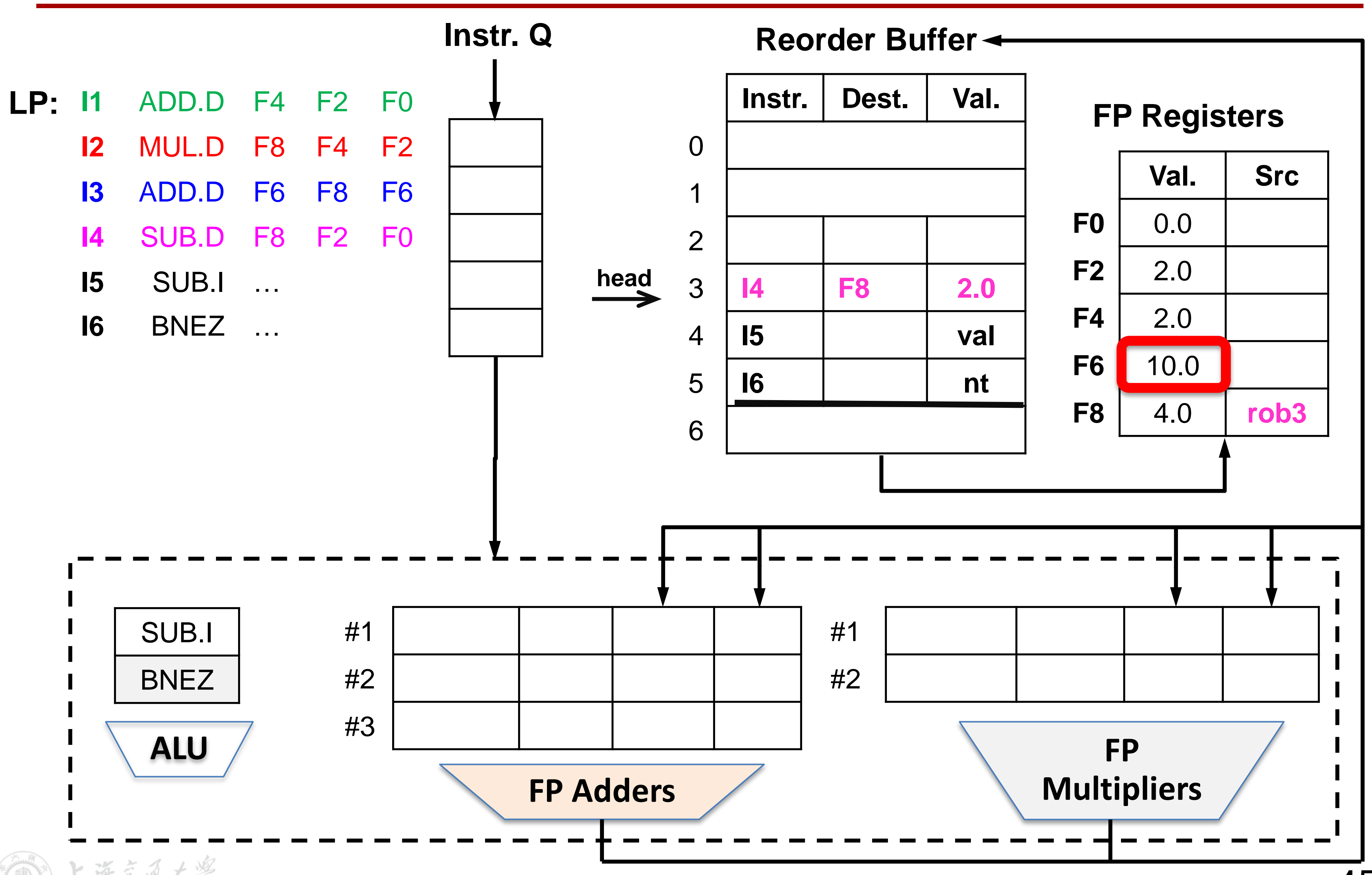


上海交通大学 SHANGHAI JIAO TONG UNIVERSITY

"Tomasulo+ROB" Example – Cycle 21

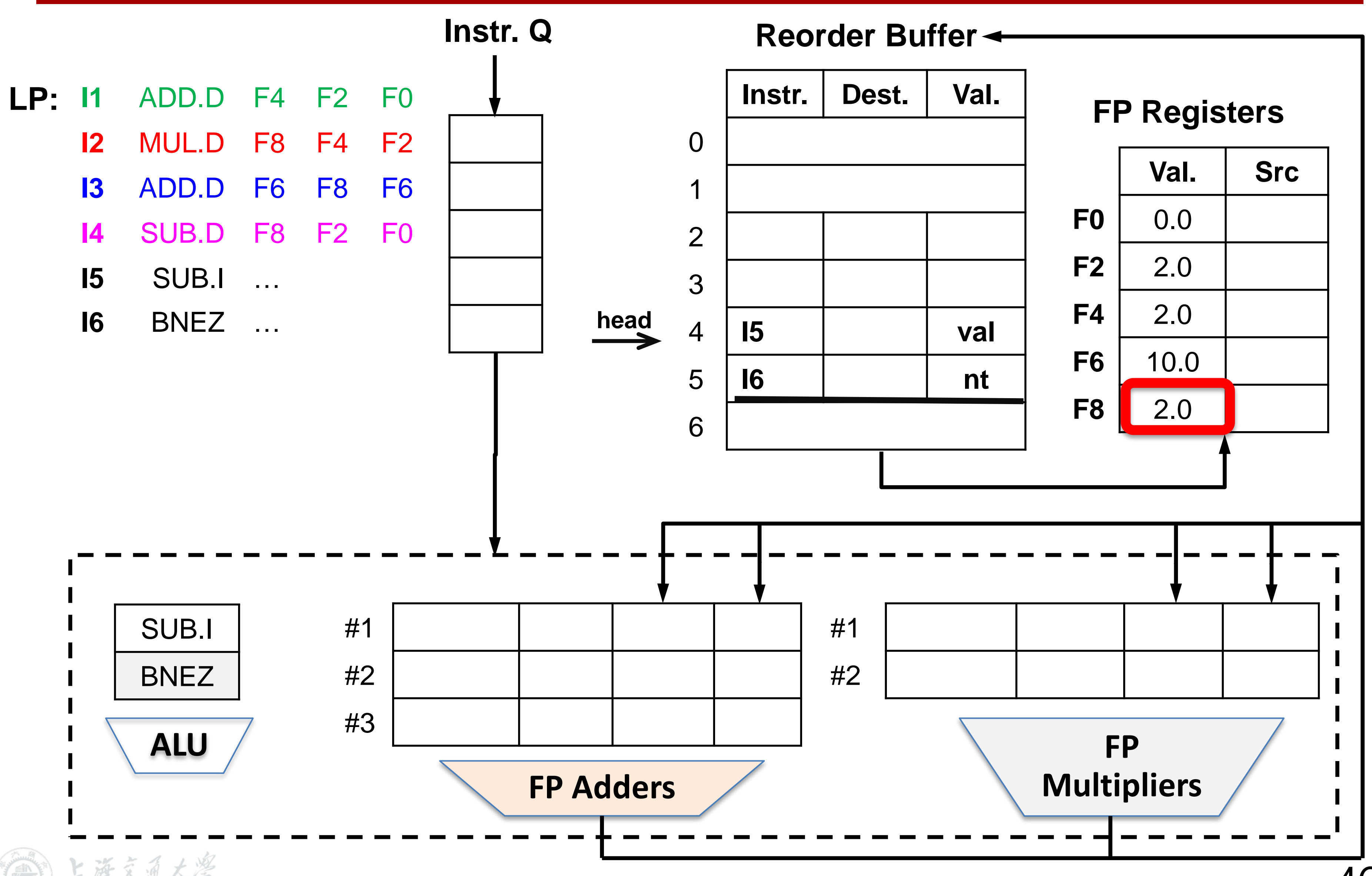


“Tomasulo+ROB” Example – Cycle 22



上海交通大學 SHANGHAI JIAO TONG UNIVERSITY

“Tomasulo+ROB” Example – Cycle 23



上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

Outlines

- Superscalar Pipeline
- Speculation
- VLIW and EPIC

VLIW Processor

- VLIW: (Very Long Instruction Word) Processors
  - A fixed number of operations are formatted as one big instruction
  - Typically 3 operations today (e.g. Intel Itanium)
- A few startup companies attempted to commercialize VLIW architecture in the market in the1980s
- Goal: High performance with lower hardware complexity
  - Less multiple-issue hardware
  - Simpler instruction dispatch
  - No structural hazard checking logic

Architecture Comparison

Category	CISC	RISC	VLIW
Inst Size	Varies	Fixed (typically 32bits	Fixed
Inst Format	Field placement varies	Regular, consistent placement of fields	
Inst Semantics	Complex; possibly many dependent ops per instr	Almost always one simple operation	Multiple, independent simple operations
Registers	Few, sometimes special	Many, general purpose	
Memory	Reg-Mem architecture	Load/Store architecture	
Hardware	Exploit microcode	Not microcoded implementation	
Example:			

Compiler Support for VLIW Processor

- Complexity of scheduling is moved into the compiler
  - Compiler creates each VLIW word
  - Detects hazards and hides latencies
  - Optimizations to fill the slots in instruction

- **Structural hazards?**
  - No two operations to the same functional unit
  - No two operations to the same memory bank
- **Data hazards?**
  - no data hazards among instructions in a bundle
- **Control hazards?**
  - Predicated execution
  - Static branch prediction

Loop Unrolling

```
for (i=1000; i>0; i=i-1)
        x[i] = x[i] + s;
```

```
Loop:   L.D     F0,0(R1)      ;F0=array element
        ADD.D   F4,F0,F2      ;add scalar in F2
        S.D     F4,0(R1)      ;store result
        DADDUI  R1,R1,#-8     ;decrement pointer
                              ;8 bytes (per DW)
        BNE     R1,R2,Loop    ;branch R1!=zero
```

Without any scheduling:

	Instruction	Operands	Clock Cycle Issued
Loop:	L.D	F0,0(R1)	1
	stall		2
	ADD.D	F4,F0,F2	3
	stall		4
	stall		5
	S.D	F4,0(R1)	6
	DADDUI	R1,R1,#-8	7
	stall		8
	BNE	R1,R2,Loop	9
	stall		10

Loop Unrolling

- Loop unrolling simply replicates the loop body multiple times, adjusting the loop termination code.

```
1 Loop: L.D     F0,0(R1)
2       L.D     F6,-8(R1)
3       L.D     F10,-16(R1)
4       L.D     F14,-24(R1)
5       ADD.D   F4,F0,F2
6       ADD.D   F8,F6,F2
7       ADD.D   F12,F10,F2
8       ADD.D   F16,F14,F2
9       S.D     0(R1),F4
10      S.D     -8(R1),F8
11      S.D     -16(R1),F12
12      DSUBUI  R1,R1,#32
13      BNEZ    R1,LOOP
14      S.D     8(R1),F16      ; 8-32 = -24
```

L.D to ADD.D: 1 Cycle
ADD.D to S.D: 2 Cycles

14 clock cycles, or 3.5 per iteration

Loop Unrolling in VLIW

Memory reference 1	*Memory reference 2*	*FP operation 1*	*FP operation 2*	*Int. op/ branch*	*clk*
LD F0, 0 (R1)	LD F6, -8 (R1)				1
LD F10, -16 (R1)	LD F14, -24 (R1)				2
LD F18, -32 (R1)	LD F22, -40 (R1)	ADDD F4, F0, F2	ADDD F8, F6, F2		3
LD F26, -48 (R1)		ADDD F12, F10, F2	ADDD F16, F14, F2		4
		ADDD F20, F18, F2	ADDD F24, F22, F2		5
SD 0(R1), F4	SD -8(R1), F8	ADDD F28, F26, F2			6
SD -16(R1), F12	SD -24(R1), F16				7
SD -32(R1), F20	SD -40(R1), F24			SUBI R1, R1, #48	8
SD -0(R1), F28				BNEZ R1, LOOP	9

- Unrolled 7 times to avoid delays
- 7 results in 9 cycles, or **1.3 clocks per iteration (1.8X)**
- Average: 2.5 ops per clock, 50% efficiency

Explicitly Parallel Instruction Computing

EPIC (style of architecture) is an evolution of VLIW which has absorbed many of the best ideas of superscalar processors

EPIC Philosophy:

- Providing the ability to design the desired plan of execution (POE) at compile-time
- Providing features that permit the compiler to "play the statistics".
- Providing the ability to communicate the POE to the hardware.

- Intel/HP Itanium (IA-64) follows EPIC philosophy
  - 6-wide, 10-stage pipeline at 800Mhz

Summary

- Superscalar Pipeline
  - Limitations of scalar processor
  - Basic feature of superscalar pipeline
  - Multi-Issue Processor
  - Dispatch buffer and completion buffer
  - Classification of ILP Machines
  - Rationale of branch prediction; 2-bit prediction
- Speculation
  - Precise exception
  - Reorder buffer (ROB)
  - Tomasulo Algorithm with ROB
- VLIW and EPIC
  - CISC vs RISC vs VLIW
  - Loop unrolling
  - The concept of EPIC