Instruction-Level Parallelism: Part I

(Chapter 3)

Outline

- Instruction-Level Parallelism: Concepts and Challenges
- Overcoming Data Hazards with Dynamic Scheduling
- Dynamic Scheduling: Examples and the Algorithm
- Reducing Branch Costs with Dynamic hardware Prediction
- High-Performance Instruction Delivery
- Taking Advantage of More ILP with Multiple issue
- Hardware-Based Speculation
- Explicit Register Renaming

Instruction-Level Parallelism: Concepts and Challenges

Processors use pipelining to overlap the execution of instructions and improve performance. This potential overlap among instructions is called **instruction-level parallelism (ILP).**

Determining how one instruction depends on another is critical to determining how much parallelism exists in a program and how that parallelism can be exploited.

There are 3 different types of dependences: **data dependences, name dependences, and control dependences.**

Data dependences

An instruction j is data dependent on instruction i if either of the following holds:

1. Instruction i produces a result that may be used by instruction j.

2. Instruction j is data dependent on instruction k, and instruction k is data dependent on instruction i.

Name dependences

A name dependence occurs when two instructions use the same register or memory location, called a name, but **there is no flow of data between the instructions associated with the name**.

There are two types of name dependences between an instruction i precedes instruction j in program order:

1. An **antidependence** between instruction i and instruction j occurs when instruction j writes a register or memory location that instruction i reads. The original ordering must be preserved to ensure that i reads the correct value.

2. An **output dependence** occurs when instruction i and instruction j write the same register or memory location. The ordering between instructions must be preserved to ensure that the value finally written corresponds j.

A name dependence is not a true dependence. Therefore, instructions involved in a name dependence can execute simultaneously or be re-ordered, if the name (register number of memory location) used in the instruction is changed so that the instructions do not conflict.

This renaming can be more easily done for register operands, where it is called **register renaming**.

Instruction segment before register renaming:

ADD.D is stalled because it waits for F0

DIV.D F0,F2,F4

ADD.D **F6**,F0,**F8**

S.D F6,0(R1)

SUB.D **F8**,F10,F14

MUL.D **F6**,F10,F8

SUB.D may be completed before ADD.D (WAR hazard)

MUL.D may be completed before ADD.D (WAW hazard)

Instruction segment after register renaming:

```
DIV.D    F0,F2,F4
ADD.D    S,F0,F8
S.D      S, 0(R1)
SUB.D    T,F10,F14
MUL.D    F6,F10,T
```

No name dependence

No name dependence

Control dependences

A control dependence determines the ordering of an instruction, i, with respect to a branch instruction so that the instruction i is executed in correct program order and only when it should be.

Overcoming Data Hazards with Dynamic Scheduling

In a dynamically scheduled pipeline, all instructions pass through the issue stage in order. However, they can be stalled or bypass each other out of order.

Scoreboarding is a dynamically scheduled pipelining technique. However, it does not use register renaming for eliminating WAR and WAW hazards.

Here we present an algorithm, termed Tomasulo algorithm, which employs the register renaming technique for out of order execution.

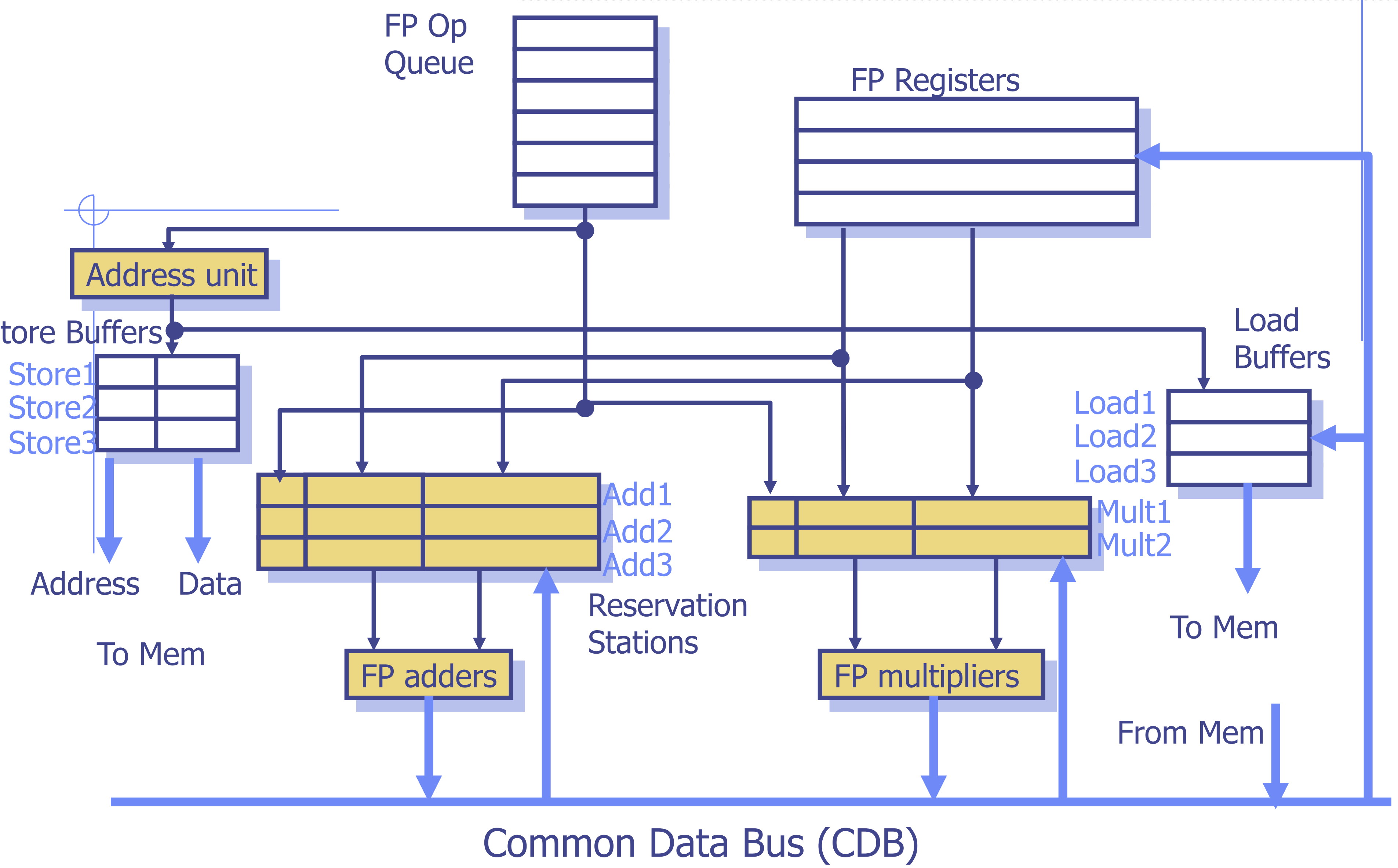
In Tomasulo's scheme, the register renaming is accomplished using the **reservation stations**.

The basic idea of the algorithm:

1. A reservation station fetches and buffers **an operand as soon as it is available**, eliminating the need to get the operand from a register.

2. Pending instruction designate the reservation station that will provide their inputs.

3. When successive writes to a register overlap in execution, only the last one is actually used to update the register.



The basic structure of a RISC-V floating-point unit using Tomasulo’s algorithm.

There are only three steps in the Tomasulo's algorithm:

1. Issue →

   Get the next instruction from the head of the instruction queue.
   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 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 registers, keep track of the reservation stations that will produce the operands**.

2. Execute→

If one or more of the operands is not yet available, monitor the common data bus while waiting for it to be computed.

When an operand becomes available, it is placed into the corresponding reservation station.

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

3. Write result→

When the result is available, write it on the CDB and from there into registers and into any reservation station waiting for the result. Stores also write data to memory during this step.

Once an instruction has issued and is waiting for a source operand, it refers to the operand by the reservation station number where the instruction that will write the register (operand) has been assigned.

Because there are more reservation stations than actual register numbers, WAW and WAR hazards are eliminated by **renaming results using reservation station numbers**.

Each reservation station has seven fields:

Op→The operation to perform on source operands S1 and S2.

Qj, Qk → The reservation stations that will produce the corresponding source operand; a zero value indicates that the source operand is already available in Vj or Vk, or is unnecessary.

Vj, Vk → The value of the source operands. Note that only one of the V field or the Q field is valid for each operand. For loads, the Vk field is used to hold the offset field.

A → used to hold information for the memory address calculation for a load or store. Initially, the immediate field of the instruction is stored here; after the address calculation, the effective address is stored here.

Busy → indicates that this reservation station and its accompanying functional unit are occupied.

The register file has a field, Qi:

Qi → the number of reservation station that contains the operation whose result should be stored into this register. If the value Qi is blank (or 0), no currently active instruction is computing a result destined for this register, meaning that the value is simply the register content.

Example:

Consider again the following code sequence

```
L.D     F6,34(R2)
L.D     F2,45(R3)
MUL.D F0,F2,F4
SUB.D F8,F6,F2
DIV.D  F10,F0,F6
ADD.D F6,F8,F2
```

Show what the status table looks like for the execution of each instruction.

Assume 2 clocks for Fl.pt. ADD, SUB;
10 clocks for MULT;
40 clocks for DIV.

Tomasulo Example

Instruction stream

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2			
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

3 Load/Buffers

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

FU count down

3 FP Adder R.S.
2 FP Mult R.S.

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
0	FU									

Clock cycle counter

Tomasulo Example Cycle 1

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3			
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Reservation Stations:

				S1	S2	RS	RS
Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	FU				Load1					

Tomasulo Example Cycle 2

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1		
LD	F2	45+	R3	$\boxed{2}$		
MULTD	F0	F2	F4			
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
2	*FU*		Load2		Load1					

Note: Can have multiple loads outstanding

Tomasulo Example Cycle 3

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	
LD	F2	45+	R3	2		
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2			
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	Yes	34+R2
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

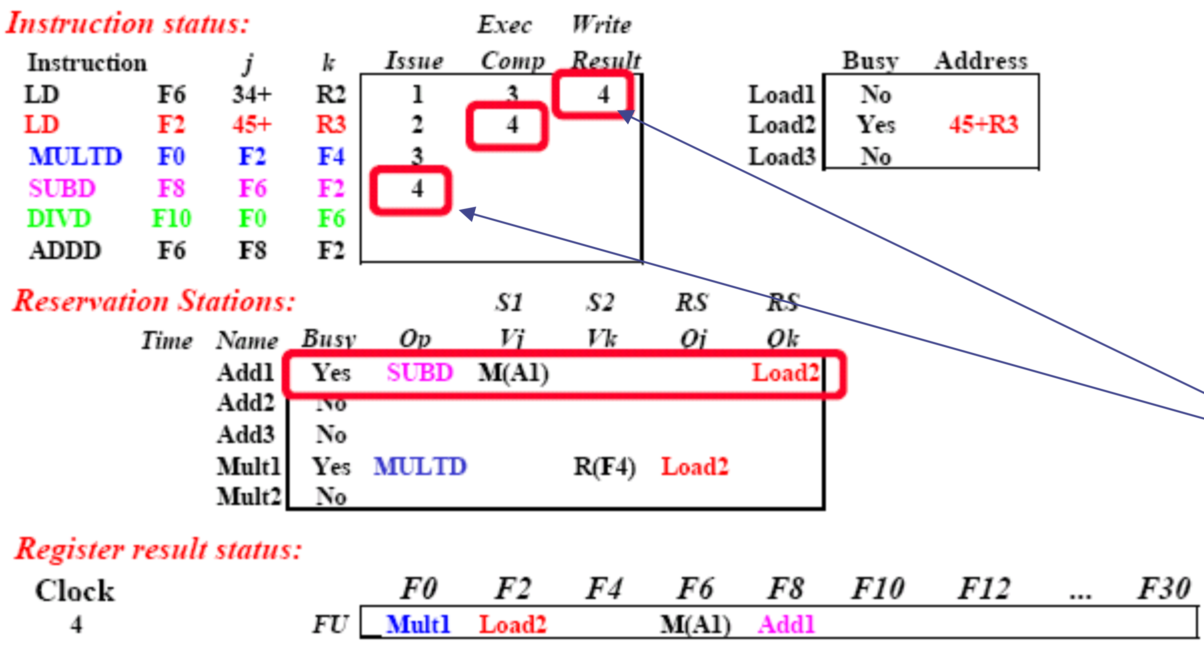
Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
3	FU	Mult1	Load2		Load1					

- Load1 completing; what is waiting for Load1?

Tomasulo Example Cycle 4



Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6			
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	Yes	45+R3
Load3	No	

Reservation Stations:

				S1	S2	RS	RS
Time	Name	Busy	Op	Vj	Vk	Qj	Qk
	Add1	Yes	SUBD	M(A1)			Load2
	Add2	No					
	Add3	No					
	Mult1	Yes	MULTD		R(F4)	Load2	
	Mult2	No					

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
4	FU	Mult1	Load2		M(A1)	Add1				

"Two ships passing in the night" problem

- Load2 completing; what is waiting for Load2?

"Two ships passing in the night" problem

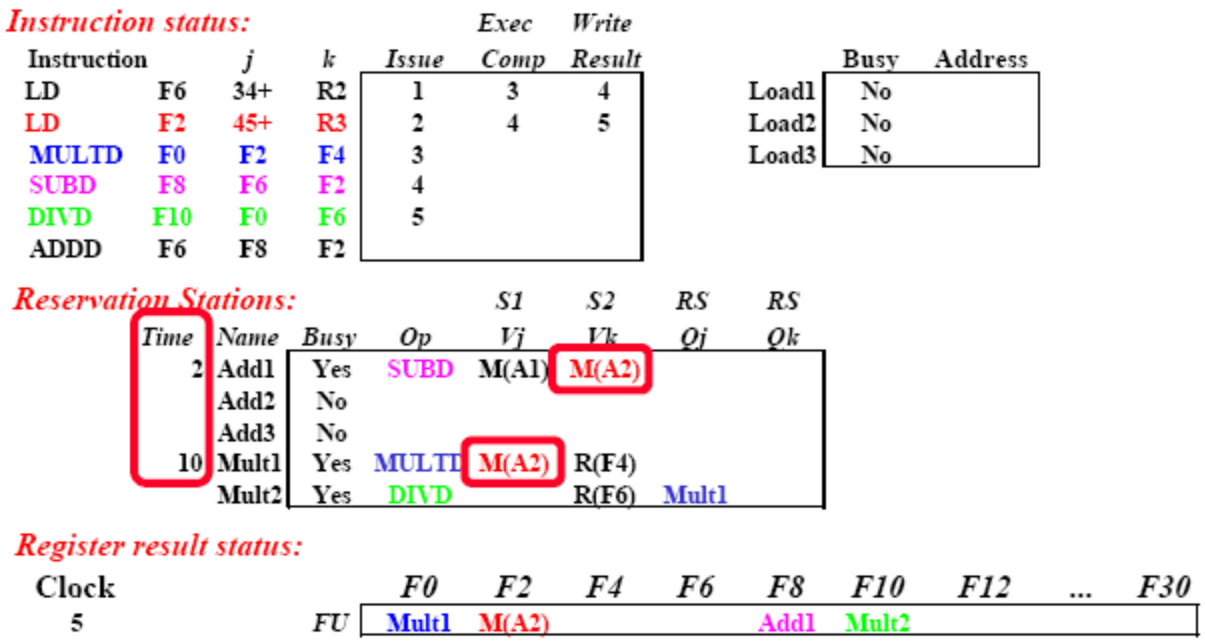
What happens if an instruction is being passed to a reservation station during the same clock period as one of its operands is going onto the CDB?

Before an instruction is in a reservation station, the operands can be fetched from the register file; but once it is in the station, the operands are **always** obtained from the CDB.

Since the instruction and its operand tag are in the transit to the reservation station, the tag cannot be matched against the tag on the CDB.

So there is a possibility that the instruction will then sit in the reservation station forever waiting for its operand, which it just missed.

Tomasulo Example Cycle 5



Instruction status:

					Exec	*Write*
Instruction		*j*	*k*	*Issue*	*Comp*	*Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2			

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

				S1	*S2*	*RS*	*RS*
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*	Mult1	M(A2)			Add1	Mult2			

- Timer starts down for Add1, Mult1

Tomasulo Example Cycle 6

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4		
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
1	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		R(F2)	Add1	
	Add3	No					
9	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	Mult1			Add2	Add1	Mult2			

Tomasulo Example Cycle 7

Instruction status:

					Exec	Write
Instruction		j	k	Issue	Comp	Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	$\boxed{7}$	
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

				S1	S2	RS	RS
Time	Name	Busy	Op	Vj	Vk	Qj	Qk
0	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	Yes	ADDD		M(A2)	Add1	
	Add3	No					
8	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
7	FU	Mult1			Add2	Add1	Mult2			

- Add1 (SUBD) completing; what is waiting for it?

Tomasulo Example Cycle 8

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

				S1	*S2*	*RS*	*RS*
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	Mult1			Add2	(M-M)	Mult2			

Tomasulo Example Cycle 9

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6		

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

				S1	S2	RS	RS
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
9	*FU*	Mult1			Add2		Mult2			

Tomasulo Example Cycle 10

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	Mult1			Add2		Mult2			

- Add2 (ADDD) completing; what is waiting for it?

Tomasulo Example Cycle 11

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

				S1	S2	RS	RS
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
11	FU	Mult1			(M-M+M		Mult2			

- Write result of ADDD here?
- All quick instructions complete in this cycle!

Tomasulo Example Cycle 12

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
3	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	*FU*	Mult1					Mult2			

Tomasulo Example Cycle 13

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
2	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
13	FU	Mult1					Mult2			

Tomasulo Example Cycle 14

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3		
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
1	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
14	FU	Mult1					Mult2			

Tomasulo Example Cycle 15

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
0	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		R(F6)	Mult1	

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	Mult1					Mult2			

- Mult1 (MULTD) completing; what is waiting for it?

Tomasulo Example Cycle 16

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
40	Mult2	Yes	DIVD	M*F4	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	*FU*	M*F4					Mult2			

- Just waiting for Mult2 (DIVD) to complete

Tomasulo Example Cycle 55

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5		
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU						Mult2			

Tomasulo Example Cycle 56

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write Result
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	R(F6)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU						Mult2			

- Mult2 (DIVD) is completing; what is waiting for it?

Tomasulo Example Cycle 57

Instruction status:

Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	3	4
LD	F2	45+	R3	2	4	5
MULTD	F0	F2	F4	3	15	16
SUBD	F8	F6	F2	4	7	8
DIVD	F10	F0	F6	5	56	57
ADDD	F6	F8	F2	6	10	11

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*RS Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	R(F6)		

Register result status:

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
56	*FU*						Result			

- Once again: In-order issue, out-of-order execution and out-of-order completion.

Example:

Consider the following simple sequence for multiplying the elements of an array by a scalar in F2:

```
Loop:  L.D       F0,0(R1)
       MUL.D     F4,F0,F2
       S.D       F4,0(R1)
       SUBI      R1,R1,8
       BNEZ      R1,Loop;
```

Assume we have issued all instructions in two successive iterations of the loop, but none of the floating-point load-stores operations has completed. The reservation stations, register status tables, and load-store buffers at this point are shown below.

Loop Example

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1			
1	MULTD F4	F0	F2			
1	SD F4	0	R1			
2	LD F0	0	R1			
2	MULTD F4	F0	F2			
2	SD F4	0	R1			

Iter-ation Count

	Busy	Addr	Fu
Load1	No		
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Added Store Buffers

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Instruction Loop

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
0	80	*Fu*									

Value of Register used for address, iteration control

Loop Example Cycle 1

Instruction status:

ITER	Instruction	*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD F0	0	R1	1		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	No					

Code:

```
LD     F0   0     R1
MULTD  F4   F0    F2
SD     F4   0     R1
SUBI   R1   R1    #8
BNEZ   R1   Loop
```

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
1	80	*Fu*	Load1								

Loop Example Cycle 2

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	No		
Store2	No		
Store3	No		

Reservation Stations:

				S1	*S2*	*RS*	
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	80	*Fu*	Load1		Mult1						

Loop Example Cycle 3

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

				S1	*S2*	*RS*	
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	80	*Fu*	Load1		Mult1						

- Implicit renaming sets up data flow graph

Loop Example Cycle 4

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	80	*Fu*	Load1		Mult1						

- Dispatching SUBI Instruction (not in FP queue)

Loop Example Cycle 5

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	No		
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	S1 Vj	S2 Vk	RS Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

```
LD     F0   0     R1
MULTD  F4   F0    F2
SD     F4   0     R1
SUBI   R1   R1    #8
BNEZ   R1   Loop
```

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
5	72	Fu	Load1		Mult1						

- And, BNEZ instruction (not in FP queue)

Loop Example Cycle 6

Instruction status:

ITER	Instruction		*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD	F0	0	R1	1		
1	MULTD	F4	F0	F2	2		
1	SD	F4	0	R1	3		
2	LD	F0	0	R1	6		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

					S1	*S2*	*RS*
Time	*Name*	*Busy*	*Op*	*Vj*	*Vk*	*Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	No					

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
6	72	*Fu*	Load2		Mult1						

- F0 never sees Load from location 80; no WAW

Loop Example Cycle 7

Instruction status:

ITER	Instruction	j	k	Issue	Exec Comp	Write Result
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		

	Busy	Addr	Fu
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	No		
Store3	No		

Reservation Stations:

Time	Name	Busy	Op	Vj	S1 Vk	S2 Qj	RS Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

LD	F0	0	R1
MULTD	F4	F0	F2
SD	F4	0	R1
SUBI	R1	R1	#8
BNEZ	R1	Loop	

Register result status

Clock	R1		F0	F2	F4	F6	F8	F10	F12	...	F30
7	72	Fu	Load2		Mult2						

- Register file completely detached from computation
- First and Second iteration completely overlapped

Loop Example Cycle 8

Instruction status:

ITER	Instruction	*j*	*k*	*Issue*	*Exec Comp*	*Write Result*
1	LD F0	0	R1	1		
1	MULTD F4	F0	F2	2		
1	SD F4	0	R1	3		
2	LD F0	0	R1	6		
2	MULTD F4	F0	F2	7		
2	SD F4	0	R1	8		

	Busy	*Addr*	*Fu*
Load1	Yes	80	
Load2	Yes	72	
Load3	No		
Store1	Yes	80	Mult1
Store2	Yes	72	Mult2
Store3	No		

Reservation Stations:

Time	*Name*	*Busy*	*Op*	*S1 Vj*	*S2 Vk*	*RS Qj*	*Qk*
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	Yes	Multd		R(F2)	Load1	
	Mult2	Yes	Multd		R(F2)	Load2	

Code:

```
LD     F0   0     R1
MULTD  F4   F0    F2
SD     F4   0     R1
SUBI   R1   R1    #8
BNEZ   R1   Loop
```

Register result status

Clock	R1		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	72	*Fu*	Load2		Mult2						

A load and a store can safely be done in a different order, provided they access different addresses. If a load and a store access the same address, then either

1. The load is before the store in program order and interchanging them results in a WAR hazard, or
2. The store is before the load in program order and interchanging them results in a RAW hazard.

Similarly, interchanging two stores to the same address result in WAW hazard.

Hence, to determine if a load can be executed at a given time, the processor can check whether any uncompleted store that precedes the load in program order shares the same data memory address as the load.

Similarly, a store must wait until there are no unexecuted loads or stores that are earlier in program order and share the same data memory address.

Reducing Branch Costs with Dynamic Hardware Prediction

- Basic Branch Prediction and Branch-Prediction Buffers

The simplest dynamic branch-prediction scheme is a branch-prediction buffer or branch history table.

A branch-prediction buffer is a small memory indexed by the **lower portion of the address of the branch instruction**. The memory contains a bit that says whether the branch was recently taken or not. This can be used as the hint for prediction. If the prediction turns out to be wrong, the prediction bit is inverted and stored back.

Example:

Consider a loop branch whose behavior is taken nine times in a row, then not taken once. What is the prediction accuracy for this branch, assuming the prediction bit for this branch remains in the prediction buffer ?

Sol:

The steady-state prediction behavior will mispredict on the first and last loop iterations. Thus, the prediction efficiency is only 80%.

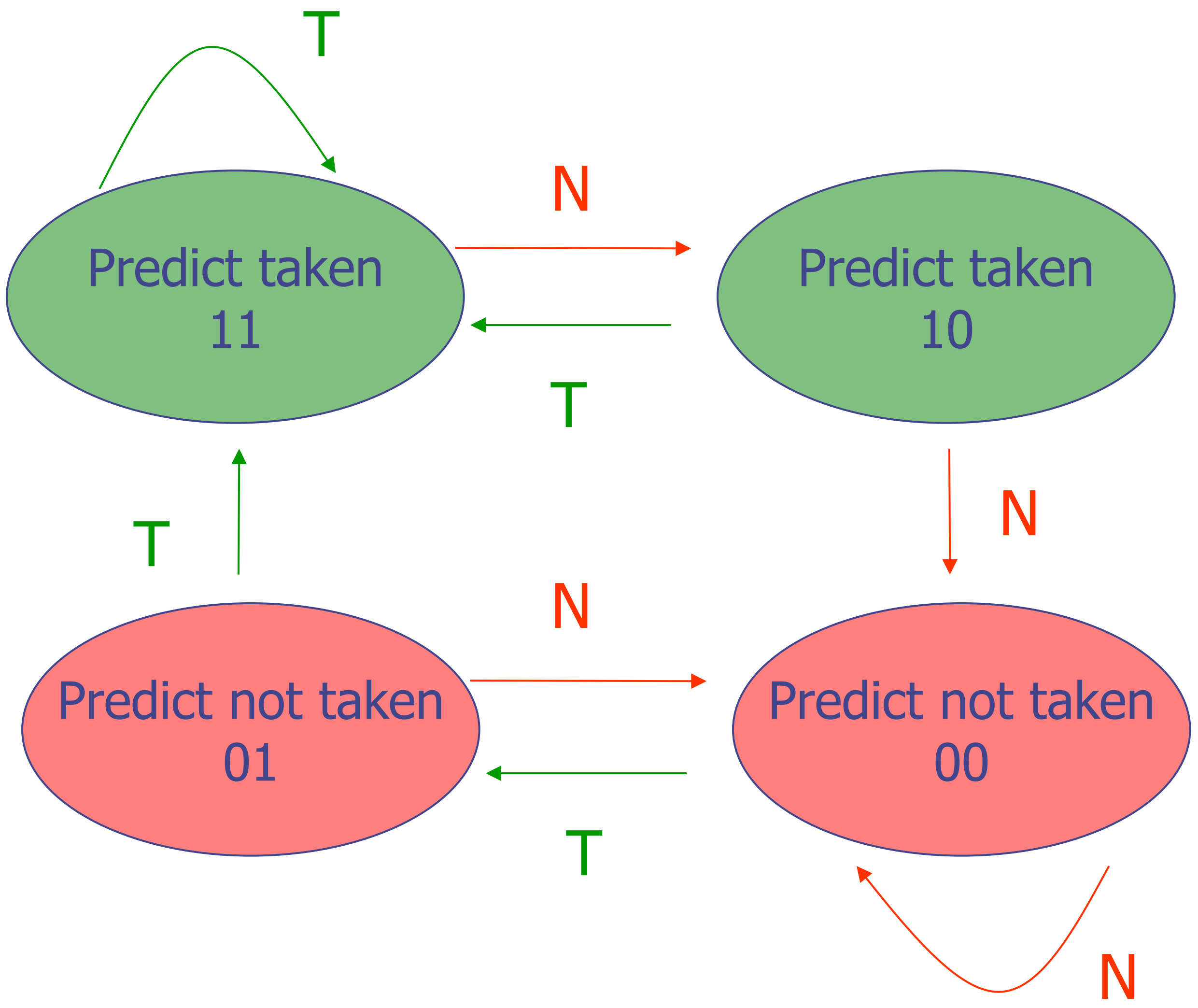
Prediction	Action
N	T
T	T
T	T
T	T
T	T
T	T
T	T
T	T
T	T
T	N

N: Not taken
T: Taken

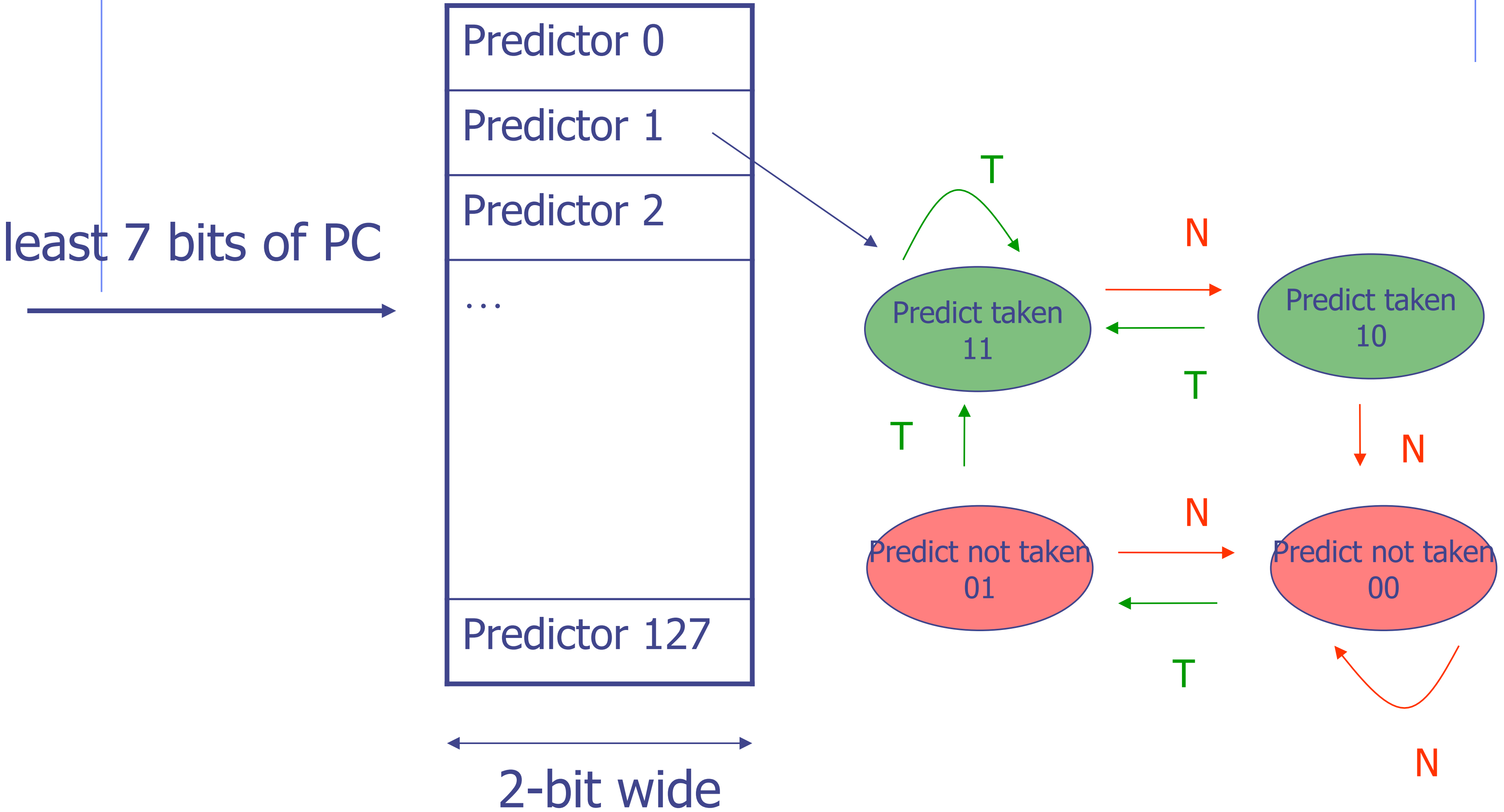
In general, for branches used to form loops—a branch is taken many times in a row, and then not once—a 1-bit predictor will mispredict at twice the rate that the branch is not taken.

To remedy this, 2-bit prediction schemes are often used. In a 2-bit scheme, a prediction must miss twice before it is changed. The following shows the finite-state processor for a 2-bit prediction scheme.

The states in a 2-bit prediction scheme



An example of branch prediction buffer with 2-bit prediction is shown below. Here there are 128 branch entries. Therefore, the branch address is 7 bits. Total size of buffer is 128 $\times$2=256 bits.



The 2-bit scheme is a specialization of a more general scheme that has an n-bit counter for each entry in the prediction buffer. The counter can take on values between 0 and 2^n-1. When the counter is greater than or equal to one-half of its maximum value, the branch is predicted as taken; otherwise, it is predicted untaken.

Correlating Branch Predictors

It may be possible to improve the prediction accuracy if we also look at the recent behavior of **other** branches rather than just the branch we are trying to predict.

Branch predictors that use the behavior of other branches to make a prediction are called **correlating predictors**.

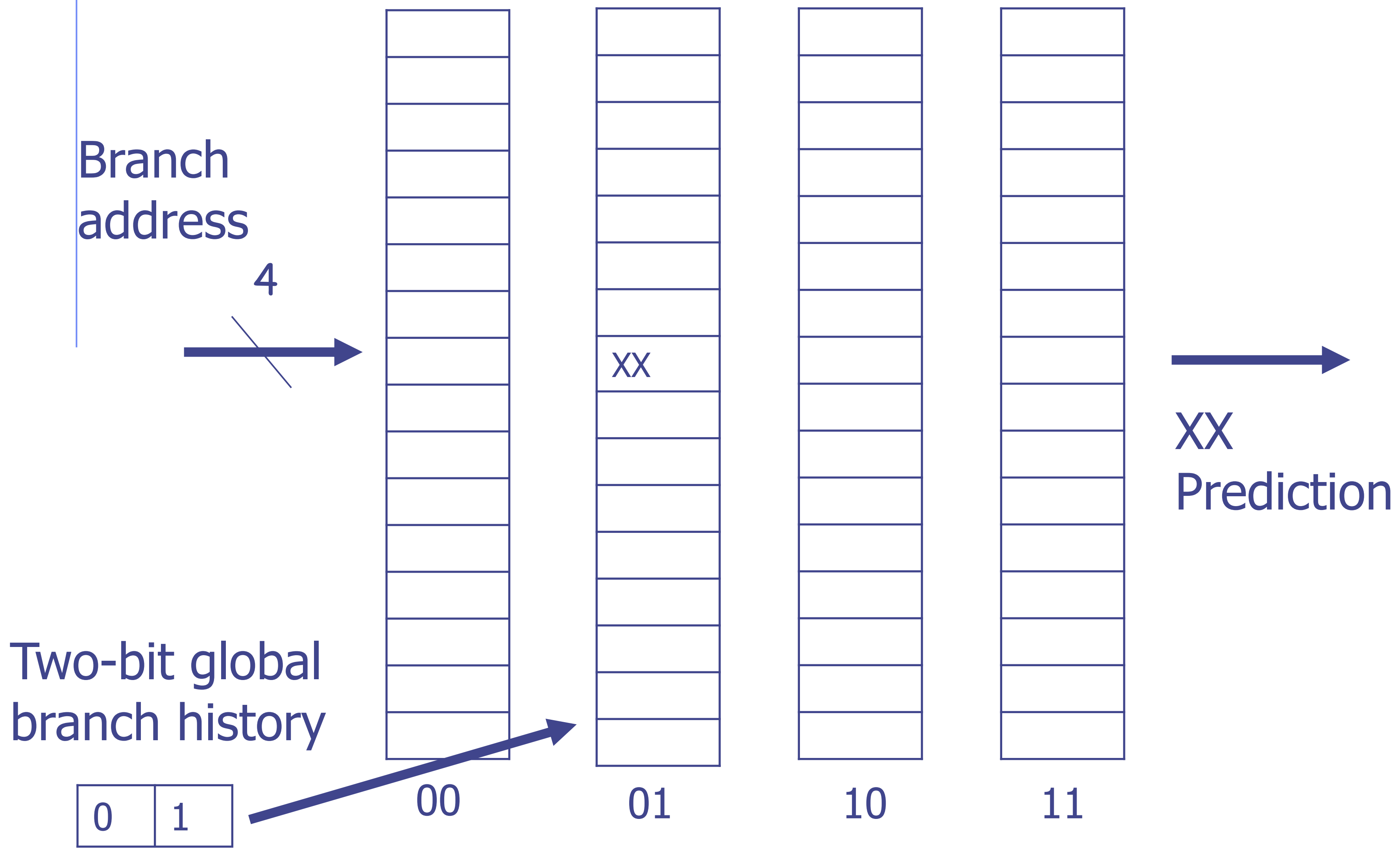
In the general case an (m,n) predictor uses the behavior of the **last m branches to choose from 2^m predictors**, each of which is an **n-bit predictor**.

The number of bits in an (m,n) predictor therefore is

$2^m \times n \times$ Number of prediction entries selected by the branch address

Example:

Consider a (2,2) branch predictor with 16 branch entries. The branch address therefore is 4 bits. The number of bits in the branch buffer is $2^2 \times 2 \times 16 = 128$ bits.

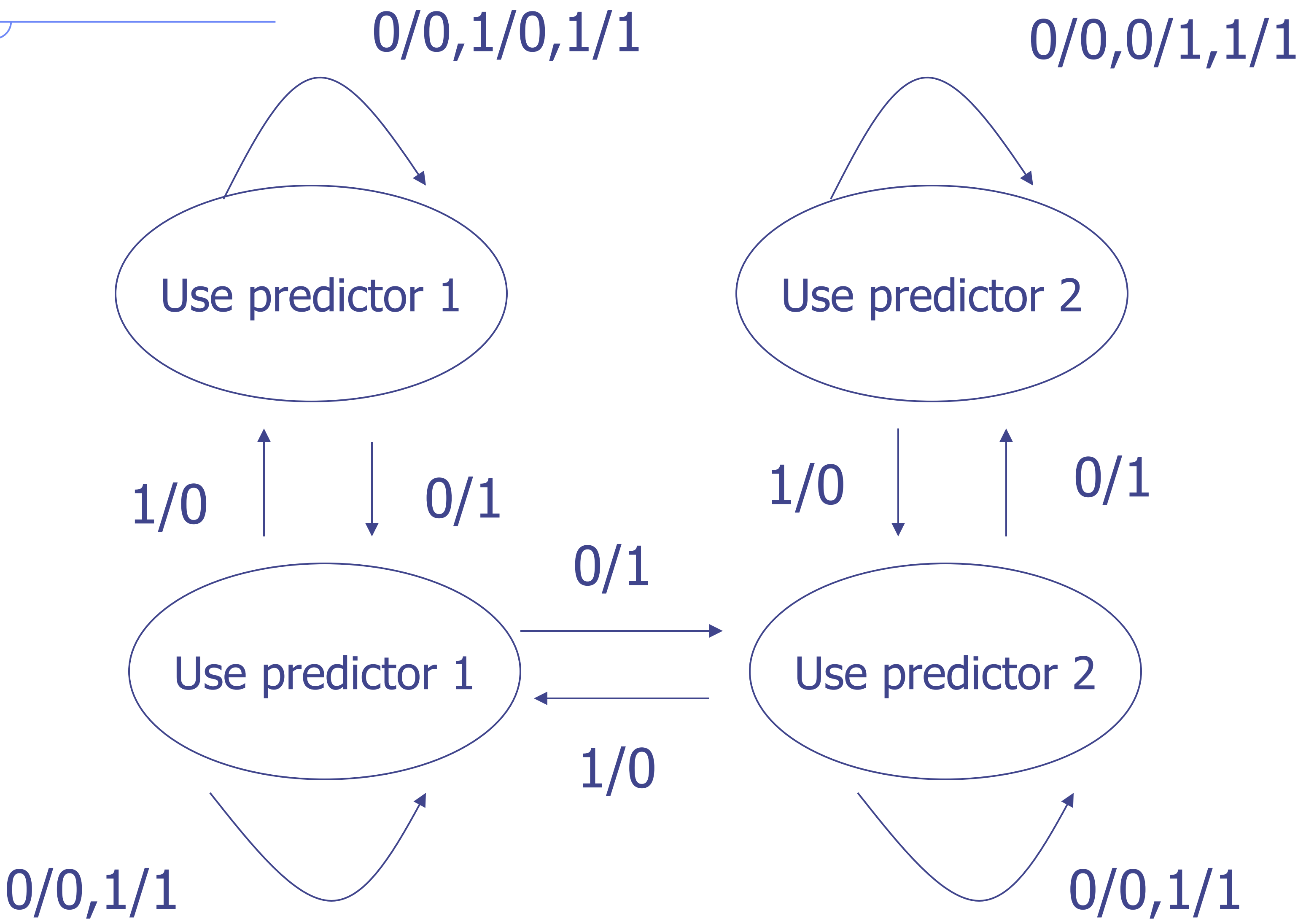


Tournament Predictors: Adaptively Combining Local and Global Predictors

A tournament predictor consists of many predictors. Some are the local predictors. The other are the global predictors. A selector is then used to choose a predictor for branch prediction.

A simple example of the tournament predictor consisting of 2 predictors is shown below.

0/0 :both predictors make wrong guess,
1/1 :both predictors make correct guess,
0/1 :predictor 1 makes wrong guess; whereas, predictor 2 makes correct guess,
1/0 :predictor 1 makes correct guess; whereas, predictor 2 makes wrong guess.



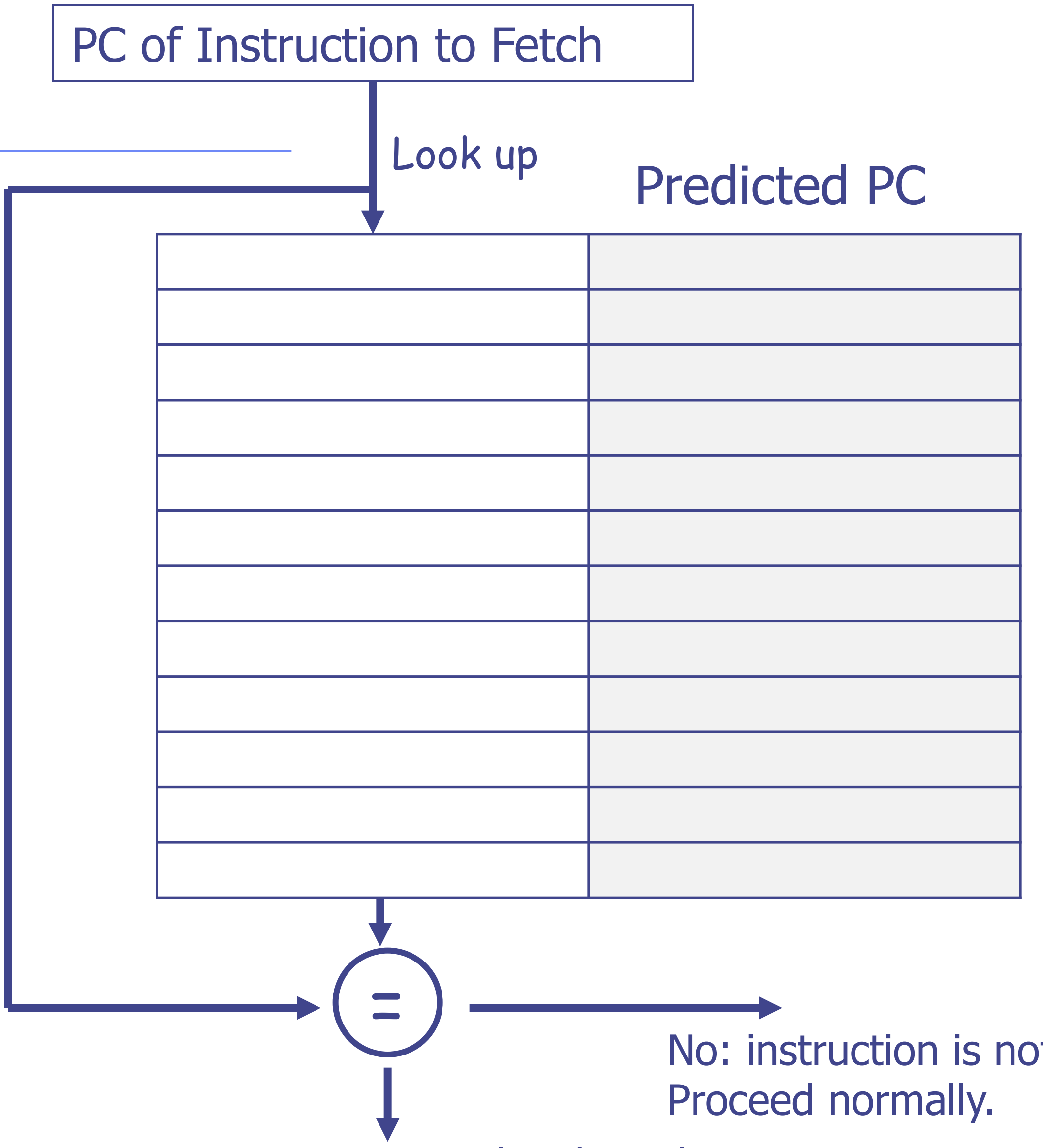
High-Performance Instruction Delivery

- Branch-target Buffers (BTB)

If the instruction is a **branch** and we know what the next PC should be **by the end of IF**, we can have a branch penalty of zero. This may be accomplished by using a **branch target buffer**, which stores the predicted address for the next instruction after a branch.

The PC of the instruction being fetched is matched against a set of instruction addresses stored in the first column; these represent the addresses of known branches.

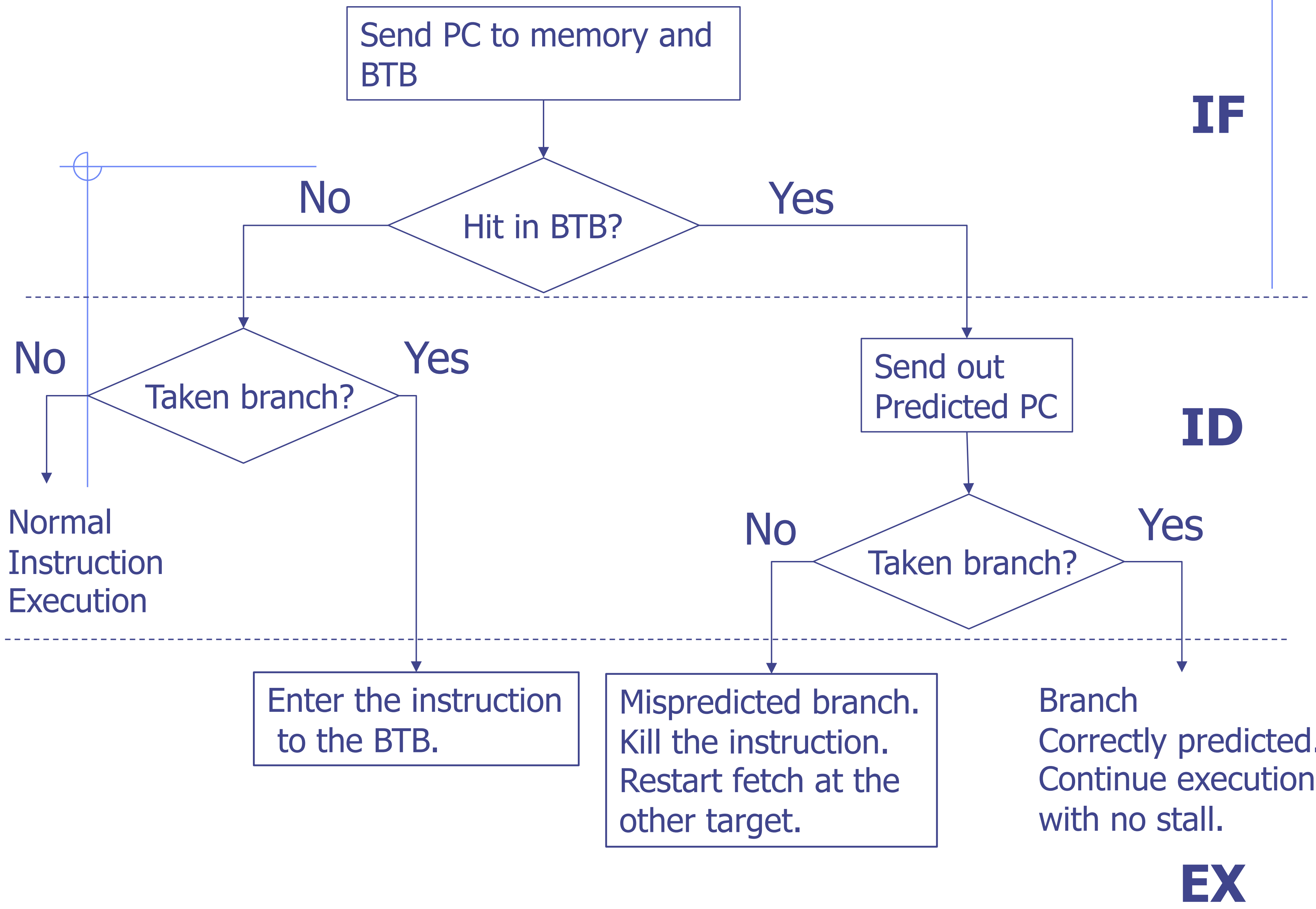
If the PC matches one of these entries, then the instruction being fetched is a taken branch, and the second field, predicted PC, contains the prediction for the next PC after the branch. Fetching begins immediately at that address.



A branch-target buffer (BTB)

No: instruction is not predicted to be a taken branch.
Proceed normally.

Yes: instruction is a taken branch.
The predicted PC is used as the next PC.

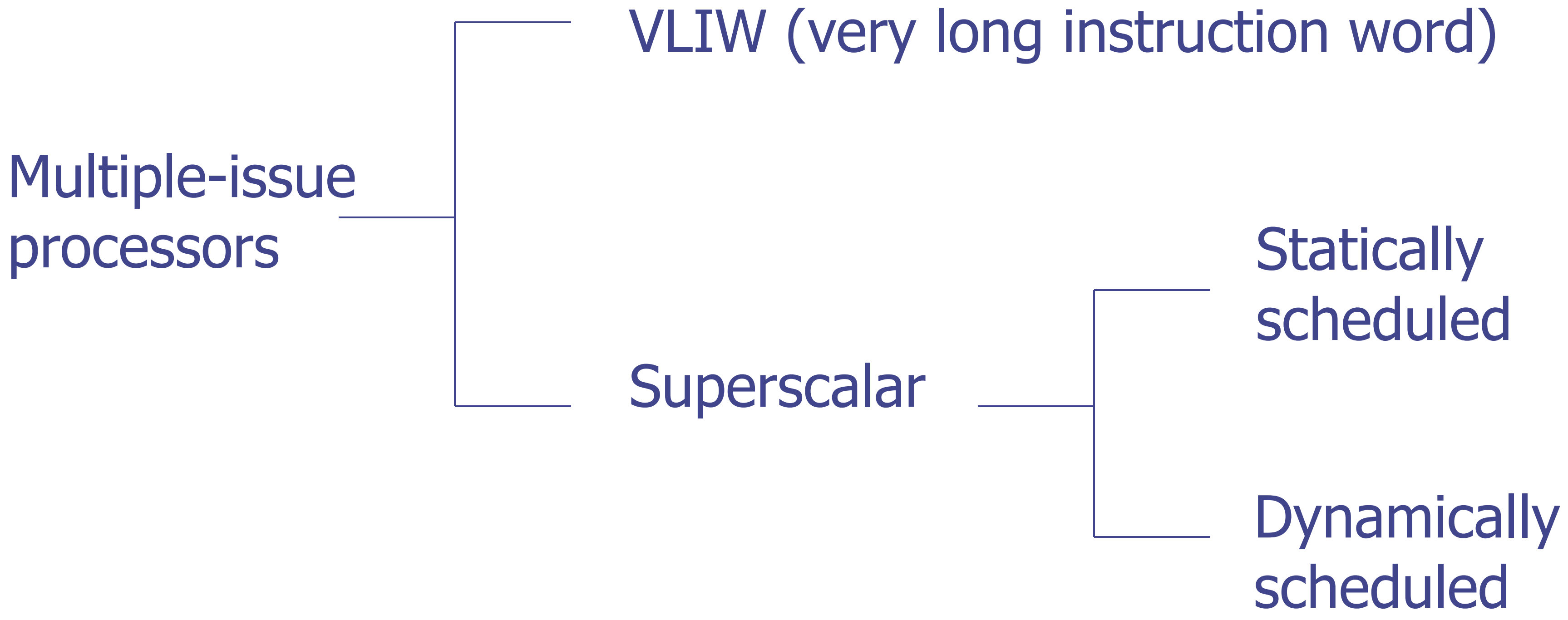

Send PC to memory and BTB
IF
Hit in BTB?
No
Yes
No
Taken branch?
Yes
Normal Instruction Execution
Send out Predicted PC
ID
No
Taken branch?
Yes
Enter the instruction to the BTB.
Mispredicted branch. Kill the instruction. Restart fetch at the other target.
Branch Correctly predicted. Continue execution with no stall.
EX

To evaluate how well a branch-target buffer works, we must determine the penalties in all possible cases. The following contains this information.

Instruction in buffer	Prediction	Actual branch	Penalty cycles
Yes	Taken	Taken	0
Yes	Taken	Not taken	2
No		Taken	2
No		Not taken	0

Taking Advantage of More ILP with Multiple Issue

The goal of the multiple-issue processors is to allow multiple instructions to issue in a clock cycle.



Superscalar processors issue varying numbers of instructions per clock and are either statically scheduled or dynamically scheduled. Statically scheduled processors use in-order execution, while dynamically scheduled processors use out-of-order execution.

VLIW processors issue a fixed number of instructions formatted as one large instruction.

A statically scheduled superscalar RISC-V processor

For simplicity, let’s us assume at most two instructions can be issued per clock cycle and that one of the instruction can be a load, store, or integer ALU operation, and the other can be any floating-point operation. Note that we consider loads and stores, including those to floating point registers, as integer operations.

We call the group of instructions received from the fetch unit that could potentially issue in one clock cycle the **issue packet**.

Conceptually, the instruction fetch unit examines each instruction in the issue packet in the program order. If an instruction would cause a structural hazard or a data hazard either due to an earlier instruction already in execution or due to an instruction earlier in the issue packet, then the instruction is not issued.

This issue limitation results in 0 to 2 instructions from the issue packet actually being issued in a given clock cycle.

For this simple superscalar fetching two instructions at a clock cycle, doing the hazard check is relatively straightforward, since the restriction of one integer and one FP instruction eliminates most hazard possibilities within the issue packet.

Assuming no hazards, the following shows how the instructions look as they go into the pipeline in pairs.

Instruction type	Pipe stages							
Integer instruction	IF	ID	EX	MEM	WB			
FP instruction	IF	ID	EX	EX	EX	WB		
Integer instruction		IF	ID	EX	MEM	WB		
FP instruction		IF	ID	EX	EX	EX	WB	
Integer instruction			IF	ID	EX	MEM	WB	
FP instruction			IF	ID	EX	EX	EX	WB
Integer instruction				IF	ID	EX	MEM	WB
FP instruction				IF	ID	EX	EX	EX

Maintaining the peak throughput for this dual-issue pipeline is much harder than it is for a single-issue pipeline. For example, in our 5-stage pipeline, the result of the load instruction can not be used on the same clock cycle or on the next clock cycle. Therefore, the next 3 instructions can not use the load result without stalling.

Multiple instruction issue with dynamic scheduling

Dynamic scheduling is one method for improving performance in a multiple instruction issue processor. When applied to a super scalar processor, dynamic scheduling has traditional benefit of boosting performance in the face of data hazards, but **it also allows the processor to potentially overcome the issue restriction.**

Although the hardware may not be able to initiate execution of more than one integer and one FP operation in a clock cycle, dynamic scheduling can eliminate this restriction at instruction issue, at least until the hardware runs out of reservation stations.

Example:

Consider the execution of the following simple loop, which adds a scalar in F2 to each element of a vector in memory. Use a pipeline extended with Tomasulo's algorithm and with multiple issue:

```
Loop: L.D        F0,0(R1)
      ADD.D      F4,F0,F2
      S.D        F4,0(R1)
      ADDI       R1,R1,-8
      BNE        R1,R2,Loop
```

Assume one integer functional unit is used for both ALU operations and effective address calculations, and a separate pipelined FP functional unit for each operation type.

The number of cycles of latency is one cycle for ALU operations, 2 cycles for loads, and 3 cycles for FP add. Create a table showing when each instruction issues, begins execution, and writes its result to the CDB for the first three iterations of the loop.

# of latency= clock number in WB – clock number in EX

Integer Operations

Iteration Number	Instructions	Issues at	EXE at	MEM at	Write CDB at	Comments
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for LD
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	ADDI R1,R1,-8	2	4		5	Wait for ALU
1	BNE,R1,R2,loop	3	6			Wait for ADDI
2	L.D F0,0(R1)	4	7	8	9	Wait for BNE
2	ADD.D F4,F0,F2	4	10		13	Wait for LD
2	S.D F4,0(R1)	5	8	14		Wait for ADD.D
2	ADDI R1,R1,-8	5	9		10	Wait for ALU
2	BNE,R1,R2,loop	6	11			Wait for ADDI
3	L.D F0,0(R1)	7	12	13	14	Wait for BNE
3	ADD.D F4,F0,F2	7	15		18	Wait for LD
3	S.D F4,0(R1)	8	13	19		Wait for ADD.D
3	ADDI R1,R1,-8	8	14		15	Wait for ALU
3	BNE,R1,R2,loop	9	16			Wait for ADDI

The throughput improvement versus a single-issue pipeline is small because there is only one floating-point operation per iteration and, thus, the integer pipeline becomes a bottleneck.

If the processor could execute more integer operations per cycle, larger improvements would be possible.

Example:

Consider the execution of the same loop on a two-issue processor, but, in addition, assume that there are separate integer functional units for effective address calculation and for ALU operations. Create a table for the first three iterations.

Iteration Number	Instructions	Issues at	EXE at	MEM at	Write CDB at	Comments
1	L.D F0,0(R1)	1	2	3	4	First issue
1	ADD.D F4,F0,F2	1	5		8	Wait for LD
1	S.D F4,0(R1)	2	3	9		Wait for ADD.D
1	ADDI R1,R1,-8	2	3		4	Execute earlier
1	BNE,R1,R2,loop	3	5			Wait for ADDI
2	L.D F0,0(R1)	4	6	7	8	Wait for BNE
2	ADD.D F4,F0,F2	4	9		12	Wait for LD
2	S.D F4,0(R1)	5	7	13		Wait for ADD.D
2	ADDI R1,R1,-8	5	6		7	Execute earlier
2	BNE,R1,R2,loop	6	8			Wait for ADDI
3	L.D F0,0(R1)	7	9	10	11	Wait for BNE
3	ADD.D F4,F0,F2	7	12		15	Wait for LD
3	S.D F4,0(R1)	8	10	16		Wait for ADD.D
3	ADDI R1,R1,-8	8	9		10	Execute earlier
3	BNE,R1,R2,loop	9	11			Wait for ADDI

Hardware-Based Speculation

Branch prediction reduces the direct stalls attributable to branches, but for a processor executing multiple instructions per clock, just predicting branches accurately may not be sufficient to generate the desired amount of instruction-level parallelism.

Overcoming the control dependence in this case can be done by speculating on the outcome of branches and **executing** the program as if our guess were correct.

The hardware that implements Tomasulo's algorithm can be extended to support speculation. To do so we **must separate the bypassing of results among instructions from the actual completion** of an instruction By making this separation, we can allow an instruction to execute and to bypass its results to other instructions, without allowing the instruction to perform any updates that can not be done, until we know that instruction is no longer speculative.

When an instruction is no longer speculative, we allow it to update the register file or memory; we call this additional step **instruction commit**.

The key idea behind implementation speculation is to allow instructions to execute out of order but to force them to **commit in order** and to prevent any irrevocable action until an instruction commits.

The **reorder buffer (ROB)** is use for instruction commit. The ROB holds the result of an instruction between the time the operation associated with the instruction completes and the time the instruction commits. Therefore, ROB supplies operands in the interval between completion of instruction execution and instruction commit.

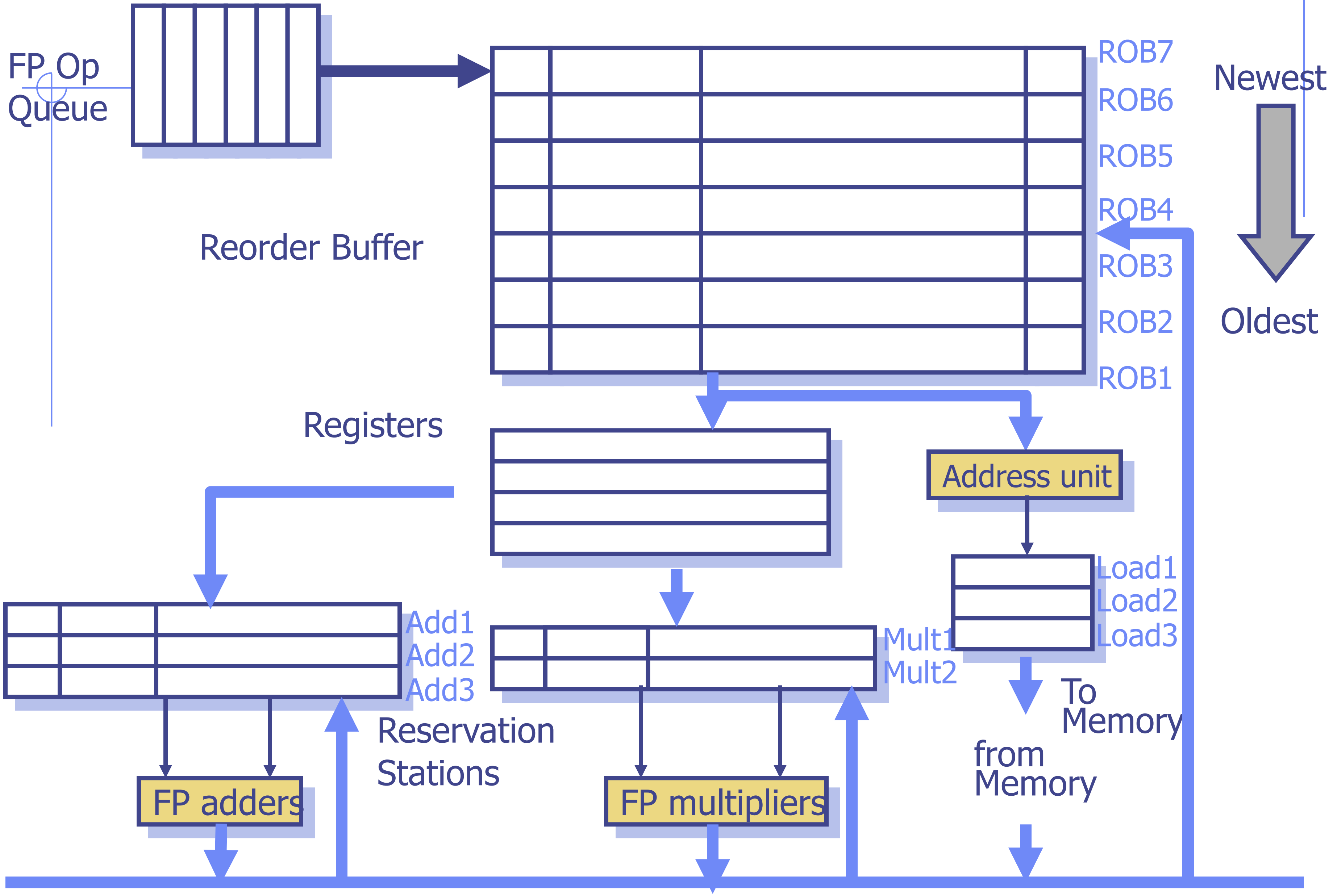
Each entry in the ROB contains four fields: the instruction type, the destination field, the value field, and the ready field.

- Instruction type
  - Branch
  - Store
  - Register operation
- Destination field
  - Register number
  - Memory address

The value field is used to hold the value of the instruction result until the instruction commits.

The ready field indicates that the instruction has completed execution, and the value is ready.

Simplified RISC-V Architecture for Tomasulo with ROB



Here are four steps involved in instruction execution:

1. Issue→ get an instruction from the instruction queue. Issue the instruction if there is an **empty reservation station** and an **empty slot in the ROB**. Send the operands to the reservation station if they are available in either the registers or the ROB.

2. Execute→ If one or more of the operands is not yet available, monitor the CDB. When both operands are available at a reservation station, execute the operation.

3. Write result→ When the result is available, write it on the **CDB** and from the **CDB into ROB,** as well as to **any reservation stations** waiting for the result.

Special actions are required for store instructions. If the value to be stored is available, it is written into the Value field of ROB entry before the store. If the value to be stored is not available yet, the CDB must be monitored until that value is broadcast, at which time the Value field of the ROB entry of the store is updated.

4. Commit→ There are three different sequences of actions at commit:

(a) A branch with incorrect prediction: when a branch with incorrect prediction reaches the head of the ROB, it indicates that the speculation was wrong. The ROB is flushed and the execution is restarted at the correct successor of the branch.

(b) A normal commit: The normal commit case occurs when an instruction reaches the head of the ROB and its result is present in the buffer; at this point, the processor updates the register with the result and removes the instruction from the ROB.

(c) A store: Committing a store is similar except that the memory is updated rather than a register.

Exceptions are handled by not recognizing the exception until it is ready to commit.

If a speculated instruction raises an exception, the exception is recorded in the ROB. If a branch misprediction arises and the instruction should not have been executed, the exception is flushed along with the instruction when the ROB is cleared.

◆Multiple issue with speculation

To show how the speculation can improve performance in a multiple-issue processor, consider the following example using speculation.

Example:

Consider the execution of the following loop, which searches an array, on a two-issue processor, once without speculation and once with speculation.

```
Loop:   LD      R2,0(R1)
        ADDI    R2,R2,#1
        SD      R2,0(R1)
        ADDI    R1,R1,#4
        BNE     R2,R3,Loop
```

Assume that there are separate integer functional units for effective address calculation, for ALU operations, and for branch condition evaluation. Create a table for the first three iterations of this loop for both machines. Assume that up to two instructions of any type can commit per block.

Without speculation, LD following the BNE can not start execution earlier, because it must wait until the branch outcome is determined.

Iteration Number	Instructions	Issues at	EXE at	MEM at	Write CDB at	Comments
1	L.D R2,0(R1)	1	2	3	4	First issue
1	ADDI R2,R2,1	1	5		6	Wait for LD
1	S.D R2,0(R1)	2	3	7		Wait for ADDI
1	ADDI R1,R1,4	2	3		4	Execute directly
1	BNE R2,R3,loop	3	7			Wait for ADDI
2	L.D R2,0(R1)	4	8	9	10	Wait for BNE
2	ADDI R2,R2,1	4	11		12	Wait for LD
2	S.D R2,0(R1)	5	9	13		Wait for ADDI
2	ADDI R1,R1,4	5	8		9	Wait for BNE
2	BNE R2,R3,loop	6	13			Wait for ADDI
3	L.D R2,0(R1)	7	14	15	16	Wait for BNE
3	ADDI R2,R2,1	7	17		18	Wait for LD
3	S.D R2,0(R1)	8	15	19		Wait for ADDI
3	ADDI R1,R1,4	8	14		15	Wait for BNE
3	BNE R2,R3,loop	9	19			Wait for ADDI

With speculation, LD following the BNE can start execution earlier, because it is speculative.

Iteration Number	Instructions	Issues at	EXE at	MEM at	Writes CDB at	Commits at	Comments
1	L.D R2,0(R1)	1	2	3	4	5	First issue
1	ADDI R2,R2,1	1	5		6	7	Wait for LD
1	S.D R2,0(R1)	2	3			7	Wait for ADDI
1	ADDI R1,R1,4	2	3		4	8	
1	BNE R2,R3,loop	3	7			8	Wait for ADDI
2	L.D R2,0(R1)	4	5	6	7	9	No delay
2	ADDI R2,R2,1	4	8		9	10	Wait for LD
2	S.D R2,0(R1)	5	6			10	Wait for ADDI
2	ADDI R1,R1,4	5	6		7	11	
2	BNE R2,R3,loop	6	10			11	Wait for ADDI
3	L.D R2,0(R1)	7	8	9	10	12	
3	ADDI R2,R2,1	7	11		12	13	Wait for LD
3	S.D R2,0(R1)	8	9			13	Wait for ADDI
3	ADDI R1,R1,4	8	9		10	14	
3	BNE R2,R3,loop	9	13			14	Wait for ADDI

Explicit Register Renaming

In addition to Tomasulo's algorithm, we can use explicit register renaming for eliminating WAW and WAR hazards.

The explicit register renaming makes use of a *physical* register file that is larger than number of architecturally visible registers (R0,…,R31, F0, …, F31).

The explicit register renaming contains four steps:

1. Issue—decode instructions & check for structural hazards & allocate new physical register for result.
   Instructions issued in program order (for hazard checking)
      Don't issue if no free physical registers
      Don't issue if structural hazard
2. Read operands—wait until no hazards, read operands
      All real dependencies (RAW hazards) resolved in this stag
      since we wait for instructions to write back data.
3. Execution—operate on operands
      The functional unit begins execution upon receiving operar
      When the result is ready, it notifies the scoreboard
4. Write result—finish execution

Explicit register renaming + Scoreboard = Renamed Scoreboard

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2				
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
	FU	P0	P2	P4	P6	P8	P10	P12		P30

Renamed Scoreboard 1

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1			
LD	F2	45+	R3				
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	Yes	Load	P32		R2				Yes
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
1	*FU*	P0	P2	P4	P32	P8	P10	P12		P30

- Each instruction allocates free register
- Similar to single-assignment compiler transformation

Renamed Scoreboard 2

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2		
LD	F2	45+	R3	2			
MULTD	F0	F2	F4				
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	No								
	Add	No								
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
2	*FU*	P0	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 3

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	
LD	F2	45+	R3	2	3		
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	Yes	Load	P32		R2				Yes
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	No								
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
3	*FU*	P36	P34	P4	P32	P8	P10	P12		P30

Renamed Scoreboard 4

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	Yes	Load	P34		R3				Yes
	Mult1	Yes	Multd	P36	P34	P4	Int2		No	Yes
	Add	Yes	Sub	P38	P32	P34		Int2	Yes	No
	Divide	No								

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
4	*FU*	P36	P34	P4	P32	P38	P10	P12		P30

Renamed Scoreboard 5

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3			
SUBD	F8	F6	F2	4			
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
5	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 6

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
10	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
6	FU	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 7

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6		
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
9	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
7	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 8

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
8	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Sub	P38	P32	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
8	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 9

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2				

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
7	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
9	*FU*	P36	P34	P4	P32	P38	P40	P12		P30

Renamed Scoreboard 10

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10			

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
6	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	Yes	Addd	P42	P38	P4			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

WAR Hazard gone!

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
10	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

- Notice that P32 not listed in Rename Table
  - Still live. Must not be reallocated by accident

Renamed Scoreboard 11

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
5	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
2	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
11	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 12

Instruction status:

Instruction		j	k	*Read*	*Exec*	*Write*	
				Issue	*Oper*	*Comp*	*Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11		

Functional unit status:

				dest	*S1*	*S2*	*FU*	*FU*	*Fj?*	*Fk?*
Time	*Name*	*Busy*	*Op*	*Fi*	*Fj*	*Fk*	*Qj*	*Qk*	*Rj*	*Rk*
	Int1	No								
	Int2	No								
4	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
1	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
12	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 13

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
3	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
0	Add	Yes	Addd	P42	P38	P34			Yes	Yes
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
13	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 14

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
2	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
14	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 15

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6		
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
1	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
15	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 16

Instruction status:

Instruction		j	k	Issue	Read Oper	Exec Comp	Write Result
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	Name	Busy	Op	dest Fi	S1 Fj	S2 Fk	FU Qj	FU Qk	Fj? Rj	Fk? Rk
	Int1	No								
	Int2	No								
0	Mult1	Yes	Multd	P36	P34	P4			Yes	Yes
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		No	Yes

Register Rename and Result

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
16	FU	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 17

Instruction status:

Instruction		*j*	*k*	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
17	*FU*	P36	P34	P4	P42	P38	P40	P12		P30

Renamed Scoreboard 18

Instruction status:

Instruction		j	k	*Issue*	*Read Oper*	*Exec Comp*	*Write Result*
LD	F6	34+	R2	1	2	3	4
LD	F2	45+	R3	2	3	4	5
MULTD	F0	F2	F4	3	6	16	17
SUBD	F8	F6	F2	4	6	8	9
DIVD	F10	F0	F6	5	18		
ADDD	F6	F8	F2	10	11	13	14

Functional unit status:

Time	*Name*	*Busy*	*Op*	*dest Fi*	*S1 Fj*	*S2 Fk*	*FU Qj*	*FU Qk*	*Fj? Rj*	*Fk? Rk*
	Int1	No								
	Int2	No								
	Mult1	No								
	Add	No								
40	Divide	Yes	Divd	P40	P36	P32	Mult1		Yes	Yes

Register Rename and Result

Clock		*F0*	*F2*	*F4*	*F6*	*F8*	*F10*	*F12*	...	*F30*
18	*FU*	P36	P34	P4	P42	P38	P40	P12		P30