

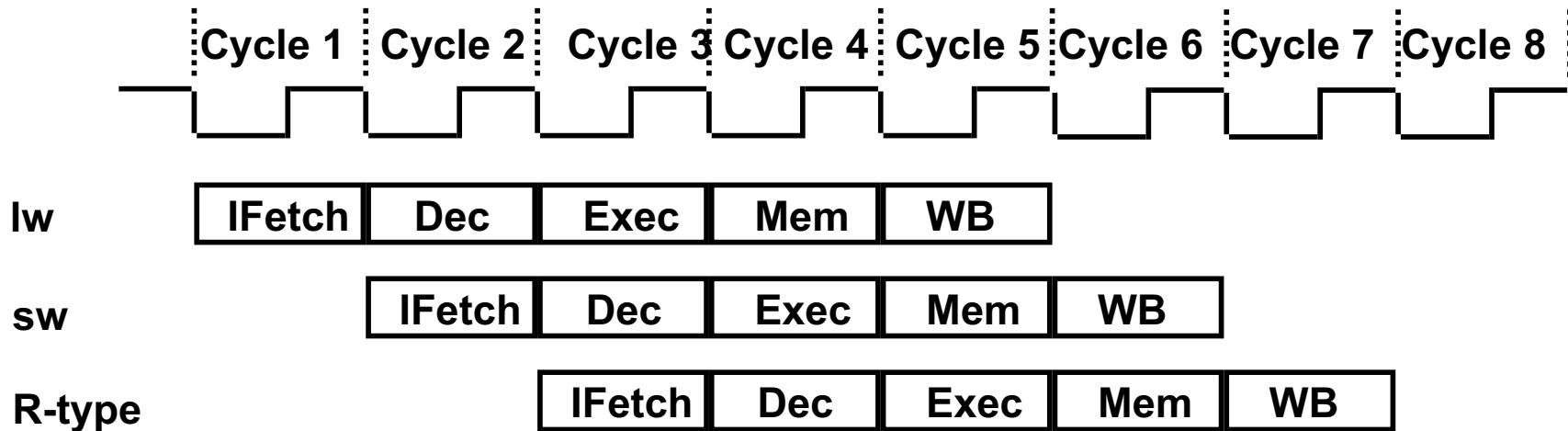
Chapter 3

Instruction-Level Parallelism and Its Exploitation

Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Introduction



- Pipelining became universal technique in 1985
 - Overlaps execution of instructions
 - Exploits “Instruction Level Parallelism”
- Beyond this, there are two main approaches:
 - Hardware-based dynamic approaches
 - Used in server and desktop processors
 - Not used as extensively in PMD processors
 - Compiler-based static approaches
 - Not as successful outside of scientific applications

Three Classes of Hazards

- Hazards: prevent the next instruction in the instruction stream from executing during its designated clock cycle
 - Structural Hazards:
 - Arise from resource conflicts when the hardware cannot support all possible combinations of instructions in simultaneous overlapped execution.
 - Data hazards
 - Arise when an instruction depends on the results of a previous instruction
 - Control hazards
 - Arise from the pipelining of branches and other instructions that change the PC.

Instruction-Level Parallelism

- When exploiting instruction-level parallelism, goal is to maximize CPI (Cycles per Instruction)
 - Pipeline CPI =
 - Ideal pipeline CPI +
 - Structural stalls +
 - Data hazard stalls +
 - Control stalls
 - *Ideal pipeline CPI* is a measure of the maximum performance attainable by the implementation

Major Techniques for Improving CPI

Technique	Reduces	Section
Forwarding and bypassing	Potential data hazard stalls	C.2
Delayed branches and simple branch scheduling	Control hazard stalls	C.2
Basic compiler pipeline scheduling	Data hazard stalls	C.2, 3.2
Basic dynamic scheduling (scoreboarding)	Data hazard stalls from true dependences	C.7
Loop unrolling	Control hazard stalls	3.2
Branch prediction	Control stalls	3.3
Dynamic scheduling with renaming	Stalls from data hazards, output dependences, and antidependences	3.4
Hardware speculation	Data hazard and control hazard stalls	3.6
Dynamic memory disambiguation	Data hazard stalls with memory	3.6
Issuing multiple instructions per cycle	Ideal CPI	3.7, 3.8
Compiler dependence analysis, software pipelining, trace scheduling	Ideal CPI, data hazard stalls	H.2, H.3
Hardware support for compiler speculation	Ideal CPI, data hazard stalls, branch hazard stalls	H.4, H.5

What is ILP?

- **Basic block:** a straight-line code sequence with no branches in except to the entry and no branches out except at the exit
- Parallelism with a basic block is limited
 - Typical size of basic block = 3-6 instructions
- Conclusion: Must optimize across branches

Dependences and Hazards

- Dependences

- Determining how one instruction depends on another
- How much parallelism exists in a program
- How that parallelism can be exploited!

- Hazards

- When there are hazards, the pipeline may be stopped in order to resolve the hazards

Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Data Dependence

- Loop-Level Parallelism
 - Unroll loop statically or dynamically
 - Use SIMD (vector processors and GPUs)
- Challenges:
 - Data dependency
 - Instruction j is data dependent on instruction i if
 - Instruction i produces a result that may be used by instruction j
 - Instruction j is data dependent on instruction k and instruction k is data dependent on instruction i
 - Dependent instructions cannot be executed simultaneously

Data Dependence

- Dependencies are a property of programs
- Pipeline organization determines if a dependence results in an actual hazard being detected and if the hazard causes a stall
- Data dependence conveys:
 - Possibility of a hazard
 - Order in which results must be calculated
 - Upper bound on exploitable instruction level parallelism

Name Dependence

- Two instructions use the same name but no flow of information
 - Not a true data dependence, *but is a problem when reordering instructions*
 - *Antidependence*: instruction j writes a register or memory location that instruction i reads
 - Initial ordering (i before j) must be preserved
 - *Output dependence*: instruction i and instruction j write the same register or memory location
 - Ordering must be preserved
- To resolve, use renaming techniques

Control Dependence

- Ordering of instruction i with respect to a branch instruction
 - Instruction control dependent on a branch **cannot be moved before** the branch so that its execution is no longer controlled by the branch
 - An instruction not control dependent on a branch **cannot be moved after** the branch so that its execution is controlled by the branch

Outline

- Instruction Level Parallelism
- Dependences
- **Data Hazards**
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Data Hazards

- Read after write (RAW)
 - j tries to read a source before i writes to it
 - Program order must be preserved
- Write after write (WAW)
 - j tries to write an operand before it is written by i
- Write after read (WAR)
 - j tries to write a destination before it is read by i , so i incorrectly gets the new value

Examples

Program order is what determines which predecessor will actually deliver a data value to an instruction.

- Example 1:

DADDU R1,R2,R3
BEQZ R4,L
DSUBU R1,R1,R6

L: ...

OR R7,**R1**,R8

- OR instruction dependent on DADDU and DSUBU

- Preserving data dependence alone is insufficient for current execution

- Example 2:

DADDU R1,R2,R3
BEQZ R12,skip
DSUBU R4,R5,R6
DADDU R5,R4,R9

skip:

OR R7,R8,R9

- Assume R4 isn't used after skip

- Possible to move DSUBU before the branch

Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- **Techniques for Exposing ILP**
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Compiler Techniques for Exposing ILP

- Pipeline scheduling
 - Separate dependent instruction from the source instruction by the pipeline latency of the source instruction
- Example:

```
for (i=999; i>=0; i=i-1)
    x[i] = x[i] + s;
```

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Stalls

Loop: L.D F0,0(R1)

stall

ADD.D F4,F0,F2

stall

stall

S.D F4,0(R1)

DADDUI R1,R1,#-8

stall (assume integer load latency is 1)

BNE R1,R2,Loop

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Pipeline Scheduling

Scheduled code:

Loop: L.D F0,0(R1)
DADDUI R1,R1,#-8
ADD.D F4,F0,F2
stall
stall
S.D F4,**8**(R1)
BNE R1,R2,Loop

Instruction producing result	Instruction using result	Latency in clock cycles
FP ALU op	Another FP ALU op	3
FP ALU op	Store double	2
Load double	FP ALU op	1
Load double	Store double	0

Loop Unrolling

- Loop unrolling
 - Unroll by a factor of 4 (assume # elements is divisible by 4)
 - Eliminate unnecessary instructions

Loop:

```
L.D F0,0(R1)
      ADD.D F4,F0,F2
      S.D F4,0(R1) ;drop DADDUI & BNE
      L.D F6,-8(R1)
      ADD.D F8,F6,F2
      S.D F8,-8(R1) ;drop DADDUI & BNE
      L.D F10,-16(R1)
      ADD.D F12,F10,F2
      S.D F12,-16(R1) ;drop DADDUI & BNE
      L.D F14,-24(R1)
      ADD.D F16,F14,F2
      S.D F16,-24(R1)
      DADDUI R1,R1,#-32
      BNE R1,R2,Loop
```

- note: number of live registers vs. original loop

Loop Unrolling/Pipeline Scheduling

- Pipeline schedule the unrolled loop:

Loop:

- L.D F0,0(R1)
- L.D F6,-8(R1)
- L.D F10,-16(R1)
- L.D F14,-24(R1)
- ADD.D F4,F0,F2
- ADD.D F8,F6,F2
- ADD.D F12,F10,F2
- ADD.D F16,F14,F2
- S.D F4,0(R1)
- S.D F8,-8(R1)
- DADDUI R1,R1,#-32**
- S.D F12,16(R1)
- S.D F16,8(R1)
- BNE R1,R2,Loop

Strip Mining

- Unknown number of loop iterations?
 - Number of iterations = n
 - Goal: make k copies of the loop body
 - Generate pair of loops:
 - First executes $n \bmod k$ times
 - Second executes n / k times
 - “Strip mining”

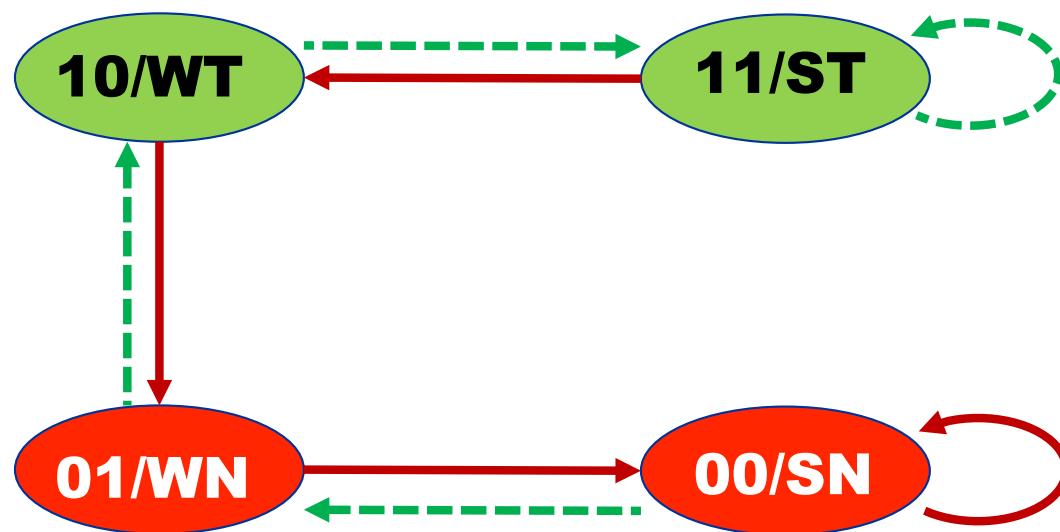
Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Branch Prediction

- Basic 2-bit predictor:
 - For each branch:
 - Predict taken or not taken
 - If the prediction is wrong two consecutive times, change prediction
- Correlating predictor:
 - Multiple 2-bit predictors for each branch
 - One for each possible combination of outcomes of preceding n branches
- Local predictor:
 - Multiple 2-bit predictors for each branch
 - One for each possible combination of outcomes for the last n occurrences of this branch
- Tournament predictor:
 - Combine correlating predictor with local predictor

Basic 2-bit Predictor



→ Taken

→ Not Taken

Predict Taken

Predict Not Taken

ST: Strong Taken

WT: Weak Taken

WN: Weak Not Taken

SN: Strong Not Taken

Correlating Branch Predictor - Motivation

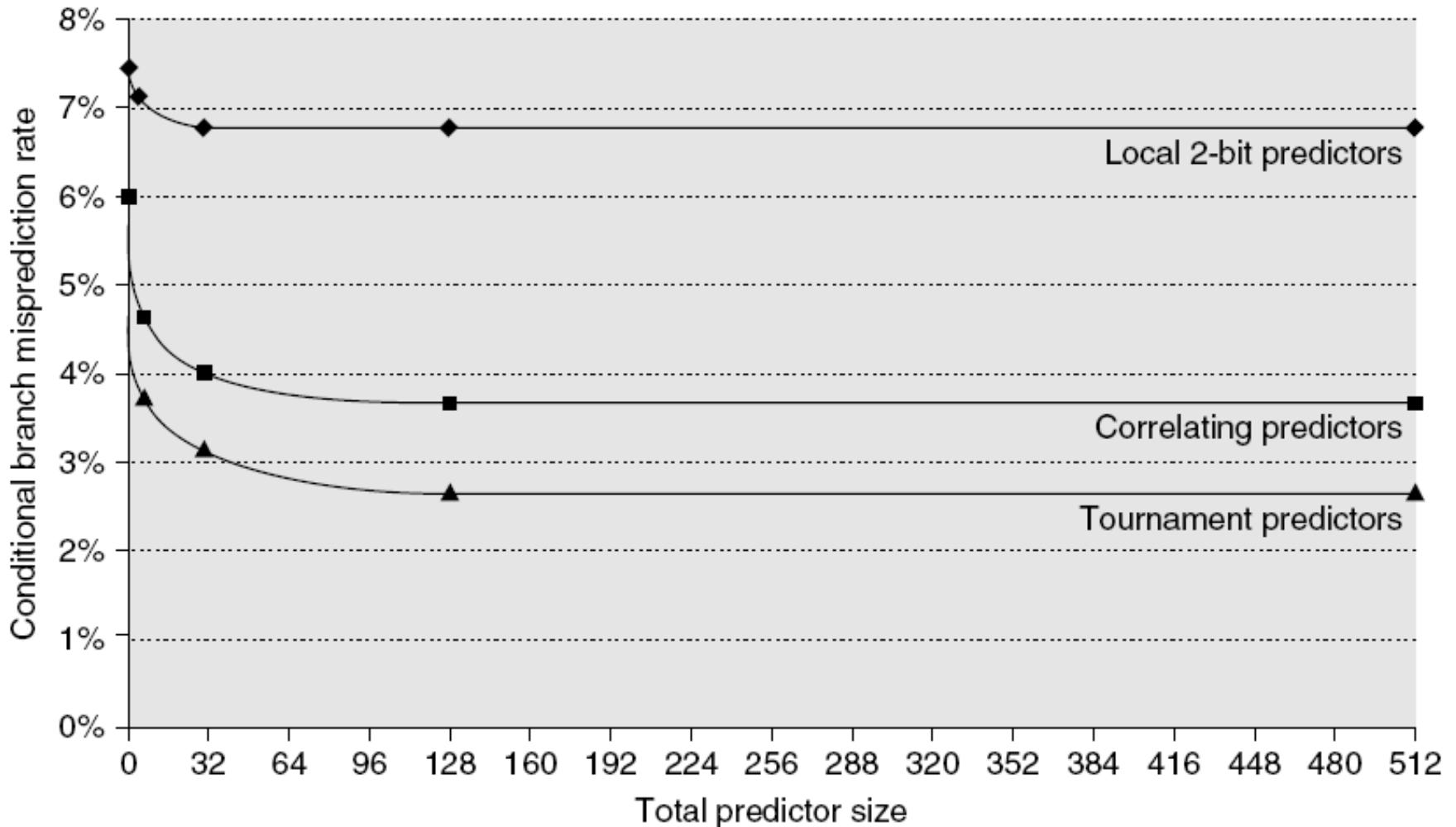
■ Global prediction

```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) {
```

is translated as

DSUBUIR3,R1,#2	
BNEZ R3,L1	;branch b1 (aa!=2)
DADD R1,R0,R0	;aa=0
L1: DSUBUIR3,R2,#2	
BNEZ R3,L2	;branch b2 (bb!=2)
DADD R2,R0,R0	;bb=0
L2: DSUBU R3,R1,R2	;R3=aa-bb
BEQZ R3,L3	;branch b3

Branch Prediction Performance



Branch predictor performance

Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- **Dynamic Scheduling**
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Dynamic Scheduling

- Rearrange order of instructions to reduce stalls while maintaining data flow
- Advantages:
 - Compiler doesn't need to have knowledge of microarchitecture
 - Handles cases where dependencies are unknown at compile time
- Disadvantage:
 - Substantial increase in hardware complexity
 - Complicates exceptions

Dynamic Scheduling

- Separate structural hazards and data hazards checking
- Dynamic scheduling implies:
 - In-order instruction issue
 - Out-of-order execution
 - Out-of-order completion
- Creates the possibility for WAR and WAW hazards
- Tomasulo's Approach
 - Tracks when operands are available
 - Introduces register renaming in hardware
 - Minimizes WAW and WAR hazards

Register Renaming

- Example:

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

antidependence

antidependence

+ name dependence with F6

Register Renaming

- Example:

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

- Now only RAW hazards remain, which can be strictly ordered

Imprecise Exceptions

- An exception is *imprecise* if the processor state when an exception is raised does not look exactly as if the instructions were executed sequentially in strict program order
 - Pipeline may have already completed instructions that are later in program order than the instruction causing the exception
 - Pipeline may have not yet completed some instructions that are earlier in program order than the instruction causing the exception

Solution: Speculation

Outline

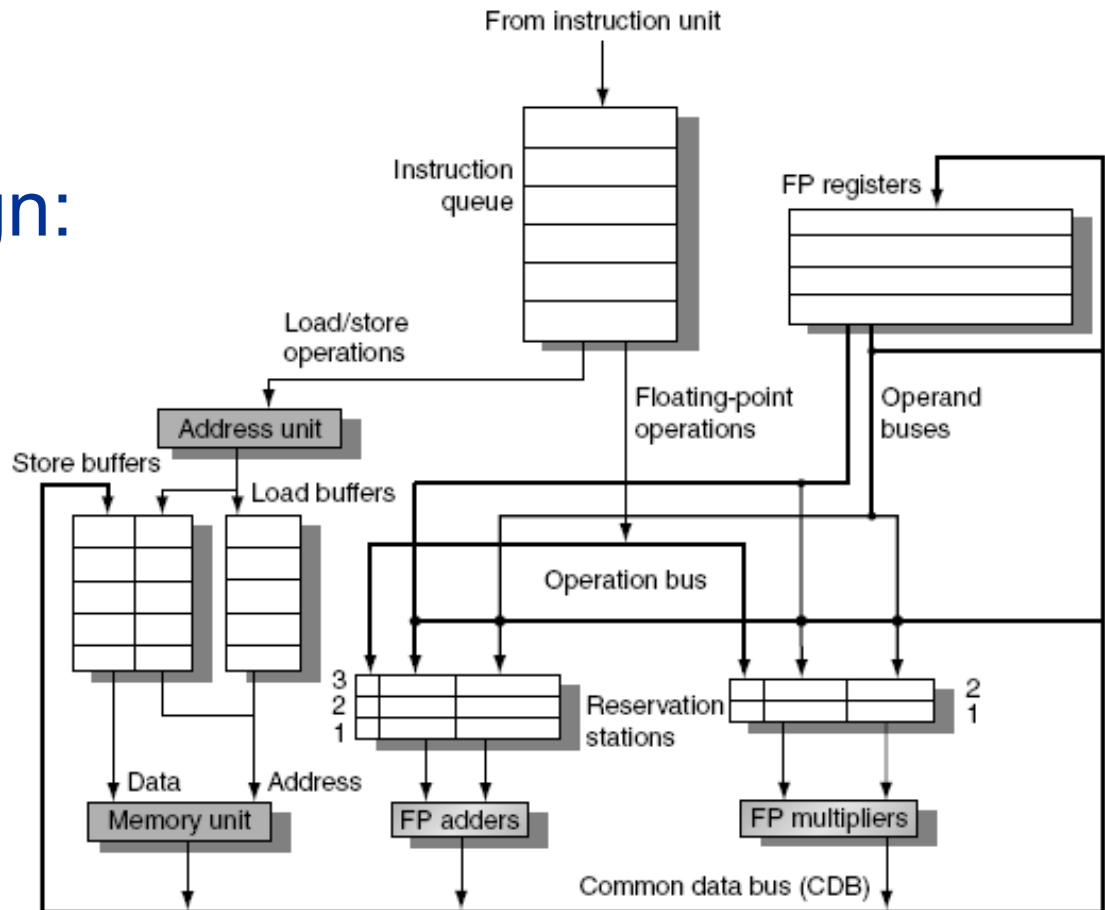
- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- **Tomasulo's Approach**
- Speculation
- Advanced Techniques

Implementing Register Renaming

- Register renaming is provided by reservation stations (RS)
 - Contains:
 - The instruction
 - Buffered operand values (when available)
 - Reservation station number of instruction providing the operand values
 - RS fetches and buffers an operand as soon as it becomes available (not necessarily involving register file)
 - Pending instructions designate the RS to which they will send their output
 - Result values broadcast on a result bus, called the common data bus (CDB)
 - Only the last output updates the register file
 - As instructions are issued, the register specifiers are renamed with the reservation station
 - May be more reservation stations than registers

Tomasulo's Algorithm

- Load and store buffers
 - Contain data and addresses, act like reservation stations
- Top-level design:



Tomasulo's Algorithm

- Three Steps:
 - Issue
 - Get next instruction from FIFO queue
 - If available RS, issue the instruction to the RS with operand values if available
 - If operand values not available, stall the instruction
 - Execute
 - When operand becomes available, store it in any reservation stations waiting for it
 - When all operands are ready, issue the instruction
 - Loads and store maintained in program order through effective address
 - No instruction allowed to initiate execution until all branches that proceed it in program order have completed
 - Write result
 - Write result on CDB into reservation stations and store buffers
 - (Stores must wait until address and value are received)

Reservation Station Components

Op: Operation to perform in the unit (e.g., + or -)

V_j, V_k: Value of Source operands

Store buffers has V field, result to be stored

Q_j, Q_k: Reservation stations producing source registers (value to be written)

Note: No ready flags as in Scoreboard; Q_j,Q_k=0 => ready

Store buffers only have Q_i for RS producing result

Busy: Indicates reservation station or FU is busy

Register result status - Indicates which functional unit will write each register, if one exists. Blank when no pending instructions that will write that register.

Example

Instruction status							
Instruction		Issue		Execute		Write Result	
L.D	F6,32(R2)		✓		✓		✓
L.D	F2,44(R3)		✓		✓		
MUL.D	F0,F2,F4		✓				
SUB.D	F8,F2,F6		✓				
DIV.D	F10,F0,F6		✓				
ADD.D	F6,F8,F2		✓				

Reservation stations							
Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	No						
Load2	Yes	Load					44 + Regs[R3]
Add1	Yes	SUB		Mem[32 + Regs[R2]]	Load2		
Add2	Yes	ADD			Add1	Load2	
Add3	No						
Mult1	Yes	MUL		Regs[F4]	Load2		
Mult2	Yes	DIV		Mem[32 + Regs[R2]]	Mult1		

Register status								
Field	F0	F2	F4	F6	F8	F10	F12	... F30
Qi	Mult1	Load2		Add2	Add1	Mult2		

Tomasulo Example

Instruction status:

Instruction	j	k	Issue	Exec	Write
				Comp	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	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				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
0	FU								

Tomasulo Example Cycle 1

Instruction status:

Instruction	j	k	Issue	Exec	Write
				Comp	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:

Time	Name	Busy	Op	S1	S2	RS	RS
				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:

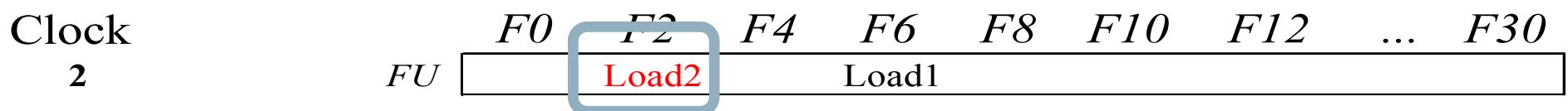
Instruction	j	k	Issue	Exec	Write
				Comp	Result
LD	F6	34+	R2	1	
LD	F2	45+	R3	2	
MULTD	F0	F2	F4		
SUBD	F8	F6	F2		
DIVD	F10	F0	F6		
ADDD	F6	F8	F2		

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

Reservation Stations:

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

Register result status:



Note: Allow multiple loads outstanding

Tomasulo Example Cycle 3

Instruction status:

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Exec</i>	<i>Write</i>	<i>Busy</i>	<i>Address</i>
				<i>Comp</i>	<i>Result</i>		
LD	F6	34+	R2	1	3	Load1	Yes 34+R2
LD	F2	45+	R3	2		Load2	Yes 45+R3
MULTD	F0	F2	F4	3		Load3	No
SUBD	F8	F6	F2				
DIVD	F10	F0	F6				
ADDD	F6	F8	F2				

Reservation Stations:

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
	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	<i>j</i>	<i>k</i>	Exec Write			Busy	Address	
			Issue	Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4		Load2	Yes 45+R3
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4				
DIVD	F10	F0	F6					
ADDD	F6	F8	F2					

Reservation Stations:

Time	Name	Busy	Op	S1		RS	
				<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
	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			

- Load2 completing; what is waiting for Load2?

Tomasulo Example Cycle 5

Instruction status:

Instruction	<i>j</i>	<i>k</i>	<i>Issue</i>	<i>Exec</i>	<i>Write</i>	<i>Busy</i>	<i>Address</i>	
				<i>Comp</i>	<i>Result</i>			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4				
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2					

Reservation Stations:

<i>Time</i>	<i>Name</i>	<i>Busy</i>	<i>Op</i>	<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
2	Add1	Yes	SUBD	M(A1)	M(A2)		
	Add2	No					
	Add3	No					
10	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
<i>FU</i>	Mult1	M(A2)		M(A1)	Add1	Mult2			
5									

Tomasulo Example Cycle 6

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Exec Write			Busy	Address	
			Issue	Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4				
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6				

Reservation Stations:

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

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
6	FU	Mult1	M(A2)		Add2	Add1	Mult2		

Tomasulo Example Cycle 7

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Exec Write			Busy	Address	
			Issue	Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7			
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6				

Reservation Stations:

Time	Name	Busy	Op	S1		RS	
				<i>Vj</i>	<i>Vk</i>	<i>Qj</i>	<i>Qk</i>
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		M(A1)	Mult1	

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
7	FU	Mult1	M(A2)		Add2	Add1	Mult2		

- Add1 completing; what is waiting for it?

Tomasulo Example Cycle 8

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6				

Reservation Stations:

Time	Name	Busy	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
	Add1	No					
2	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
7	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	<i>F₀</i>	<i>F₂</i>	<i>F₄</i>	<i>F₆</i>	<i>F₈</i>	<i>F₁₀</i>	<i>F₁₂</i>	...	<i>F₃₀</i>
8	FU	Mult1	M(A2)		Add2	(M-M)	Mult2		

Tomasulo Example Cycle 9

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6				

Reservation Stations:

Time	Name	Busy	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
	Add1	No					
1	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
6	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	<i>F₀</i>	<i>F₂</i>	<i>F₄</i>	<i>F₆</i>	<i>F₈</i>	<i>F₁₀</i>	<i>F₁₂</i>	...	<i>F₃₀</i>
9		Mult1	M(A2)		Add2	(M-M)	Mult2		

Tomasulo Example Cycle 10

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Exec Write			Busy	Address	
			Issue	Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10			

Reservation Stations:

Time	Name	Busy	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
	Add1	No					
0	Add2	Yes	ADDD	(M-M)	M(A2)		
	Add3	No					
5	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	<i>F₀</i>	<i>F₂</i>	<i>F₄</i>	<i>F₆</i>	<i>F₈</i>	<i>F₁₀</i>	<i>F₁₂</i>	...	<i>F₃₀</i>
10	<i>FU</i>	Mult1	M(A2)		Add2	(M-M)	Mult2		

- Add2 completing; what is waiting for it?

Tomasulo Example Cycle 11

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Exec Write			Busy	Address	
			Issue	Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
	Add1	No					
	Add2	No					
	Add3	No					
4	Mult1	Yes	MULTD	M(A2)	R(F4)		
	Mult2	Yes	DIVD		M(A1)	Mult1	

Register result status:

Clock	<i>F₀</i>	<i>F₂</i>	<i>F₄</i>	<i>F₆</i>	<i>F₈</i>	<i>F₁₀</i>	<i>F₁₂</i>	...	<i>F₃₀</i>
11	<i>FU</i>	Mult1	M(A2)		(M-M+M)	(M-M)	Mult2		

- All quick instructions complete in this cycle!

Tomasulo Example Cycle 12

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	S1		S2		RS	
			Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>	
	Add1	No						
	Add2	No						
	Add3	No						
3	Mult1	Yes	MULTD	M(A2)	R(F4)			
	Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
12	<i>FU</i>	Mult1	M(A2)	(M-M+N(M-M))	Mult2				

Tomasulo Example Cycle 13

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	S1		S2		RS	
			Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>	
	Add1	No						
	Add2	No						
	Add3	No						
2	Mult1	Yes	MULTD	M(A2)	R(F4)			
	Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
13	<i>FU</i>	Mult1	M(A2)	(M-M+N(M-M))	Mult2				

Tomasulo Example Cycle 14

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3			Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	S1		S2		RS	
			Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>	
	Add1	No						
	Add2	No						
	Add3	No						
1	Mult1	Yes	MULTD	M(A2)	R(F4)			
	Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
14	<i>FU</i>	Mult1	M(A2)	(M-M+N(M-M))	Mult2				

Tomasulo Example Cycle 15

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec Write			Busy	Address
				Issue	Comp	Result		
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3	15		Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5				
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	S1		S2		RS	
			Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>	
	Add1	No						
	Add2	No						
	Add3	No						
0	Mult1	Yes	MULTD	M(A2)	R(F4)			
	Mult2	Yes	DIVD		M(A1)	Mult1		

Register result status:

Clock	<i>F₀</i>	<i>F₂</i>	<i>F₄</i>	<i>F₆</i>	<i>F₈</i>	<i>F₁₀</i>	<i>F₁₂</i>	...	<i>F₃₀</i>
15		FU	Mult1	M(A2)		(M-M+N(M-M))	Mult2		

Tomasulo Example Cycle 16

Instruction status:

Instruction	<i>j</i>	<i>k</i>		Exec			Write		Busy	Address
				Issue	Comp	Result				
LD	F6	34+	R2	1	3	4			Load1	No
LD	F2	45+	R3	2	4	5			Load2	No
MULTD	F0	F2	F4	3	15	16			Load3	No
SUBD	F8	F6	F2	4	7	8				
DIVD	F10	F0	F6	5						
ADDD	F6	F8	F2	6	10	11				

Reservation Stations:

Time	Name	Busy	S1		S2		RS		RS	
			Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>			
	Add1	No								
	Add2	No								
	Add3	No								
	Mult1	No								
40	Mult2	Yes	DIVD	M*F4	M(A1)					

Register result status:

Clock	<i>F0</i>	<i>F2</i>	<i>F4</i>	<i>F6</i>	<i>F8</i>	<i>F10</i>	<i>F12</i>	...	<i>F30</i>
16	<i>FU</i>	M*F4	M(A2)	(M-M+N)	(M-M)	Mult2			

Tomasulo Example Cycle 55

- skip a couple of cycles...

Instruction status:

Instruction	j	k	Issue	Exec	Write	Busy	Address
				Comp	Result		
LD	F6	34+	R2	1	3	4	Load1
LD	F2	45+	R3	2	4	5	Load2
MULTD	F0	F2	F4	3	15	16	Load3
SUBD	F8	F6	F2	4	7	8	
DIVD	F10	F0	F6	5			
ADDD	F6	F8	F2	6	10	11	

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
1	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock		F0	F2	F4	F6	F8	F10	F12	...	F30
55	FU	M*F4	M(A2)	(M-M+M(M-M))	Mult2					

Tomasulo Example Cycle 56

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Issue	Exec	Write	Busy	Address	
				Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3	15	16	Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5	56			
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	Op	S1	S2	RS	RS
				Vj	Vk	Qj	Qk
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
0	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)	(M-M+N	(M-M)	Mult2			

- Mult2 is completing; what is waiting for it?

Tomasulo Example Cycle 57

Instruction status:

Instruction	<i>j</i>	<i>k</i>	Issue	Exec		Write	Busy	Address
				Comp	Result			
LD	F6	34+	R2	1	3	4	Load1	No
LD	F2	45+	R3	2	4	5	Load2	No
MULTD	F0	F2	F4	3	15	16	Load3	No
SUBD	F8	F6	F2	4	7	8		
DIVD	F10	F0	F6	5	56	57		
ADDD	F6	F8	F2	6	10	11		

Reservation Stations:

Time	Name	Busy	Op	<i>V_j</i>	<i>V_k</i>	<i>Q_j</i>	<i>Q_k</i>
	Add1	No					
	Add2	No					
	Add3	No					
	Mult1	No					
	Mult2	Yes	DIVD	M*F4	M(A1)		

Register result status:

Clock	F0	F2	F4	F6	F8	F10	F12	...	F30
56	FU	M*F4	M(A2)	(M-M+N	(M-M)	Result			

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

Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Hardware-Based Speculation

- Execute instructions along predicted execution paths but only commit the results if prediction was correct
- Instruction commit: allowing an instruction to update the register file when instruction is no longer speculative
- Need an additional piece of hardware to prevent any irrevocable action until an instruction commits
 - I.e. updating state or taking an execution

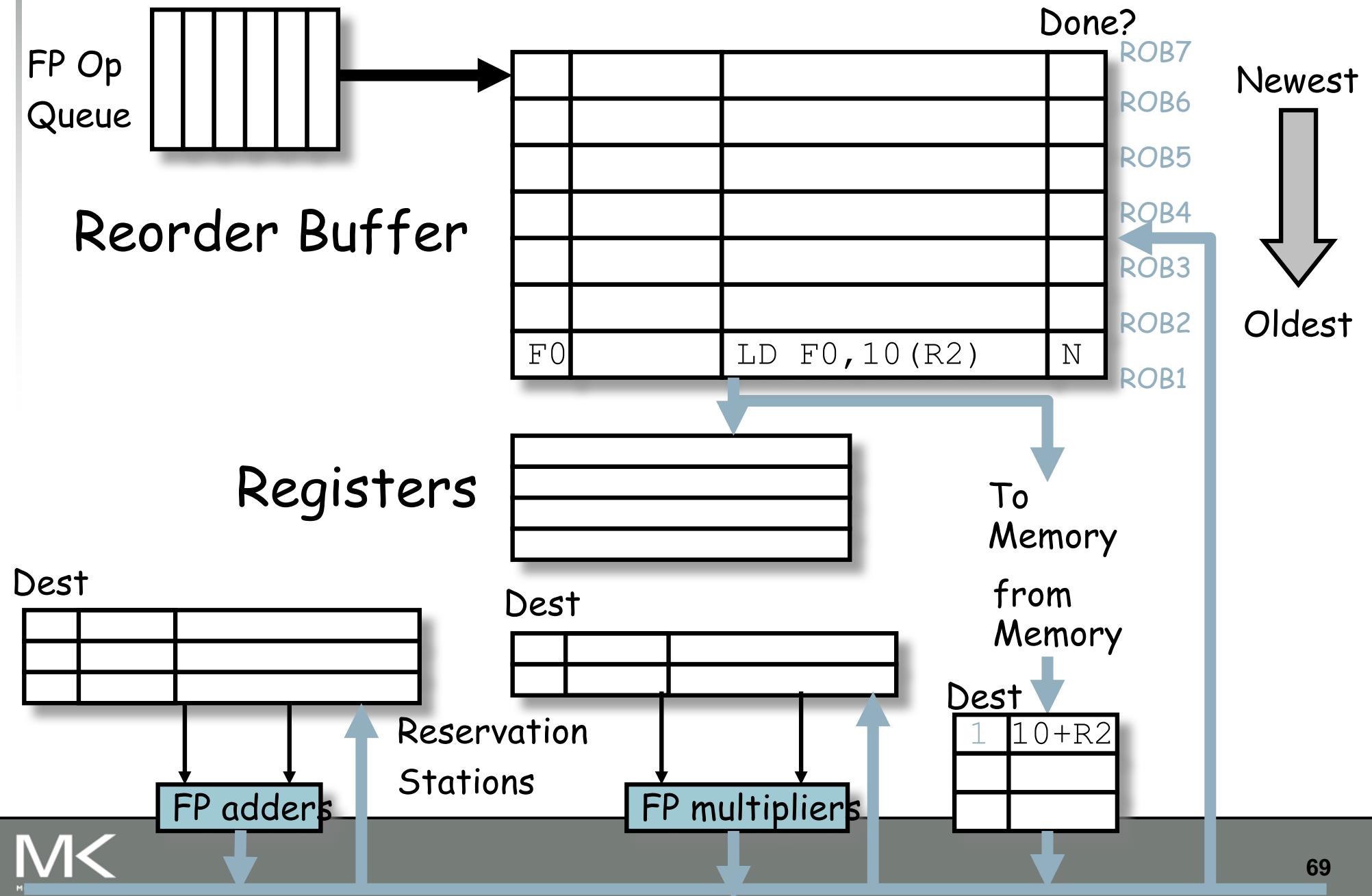
Reorder Buffer

- Reorder buffer – holds the result of instruction between completion and commit
- Four fields:
 - Instruction type: branch/store/register
 - Destination field: register number
 - Value field: output value
 - Ready field: completed execution?
- Modify reservation stations:
 - Operand source is now reorder buffer instead of functional unit

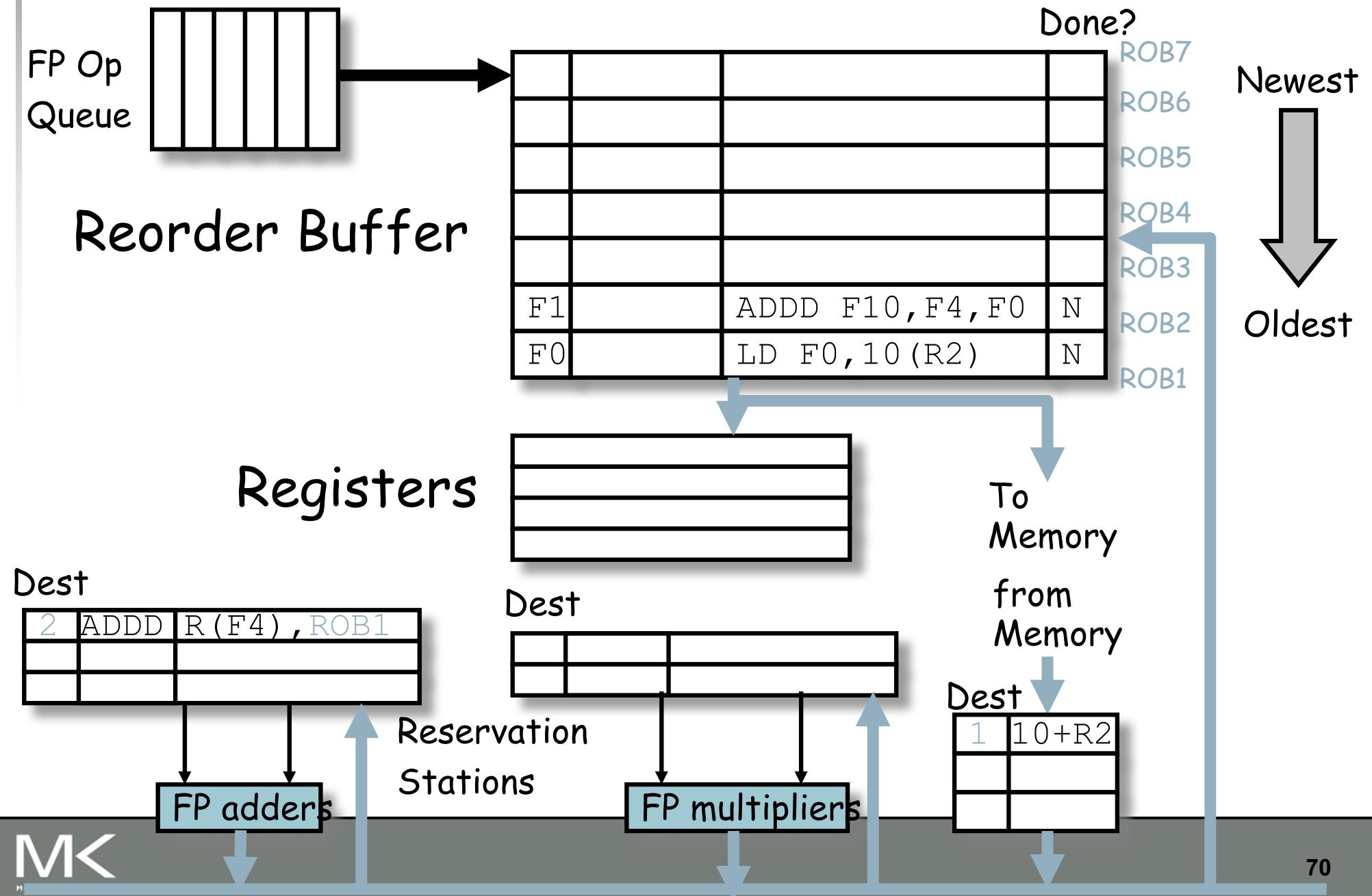
Reorder Buffer

- Register values and memory values are not written until an instruction commits
- On misprediction:
 - Speculated entries in ROB are cleared
- Exceptions:
 - Not recognized until it is ready to commit

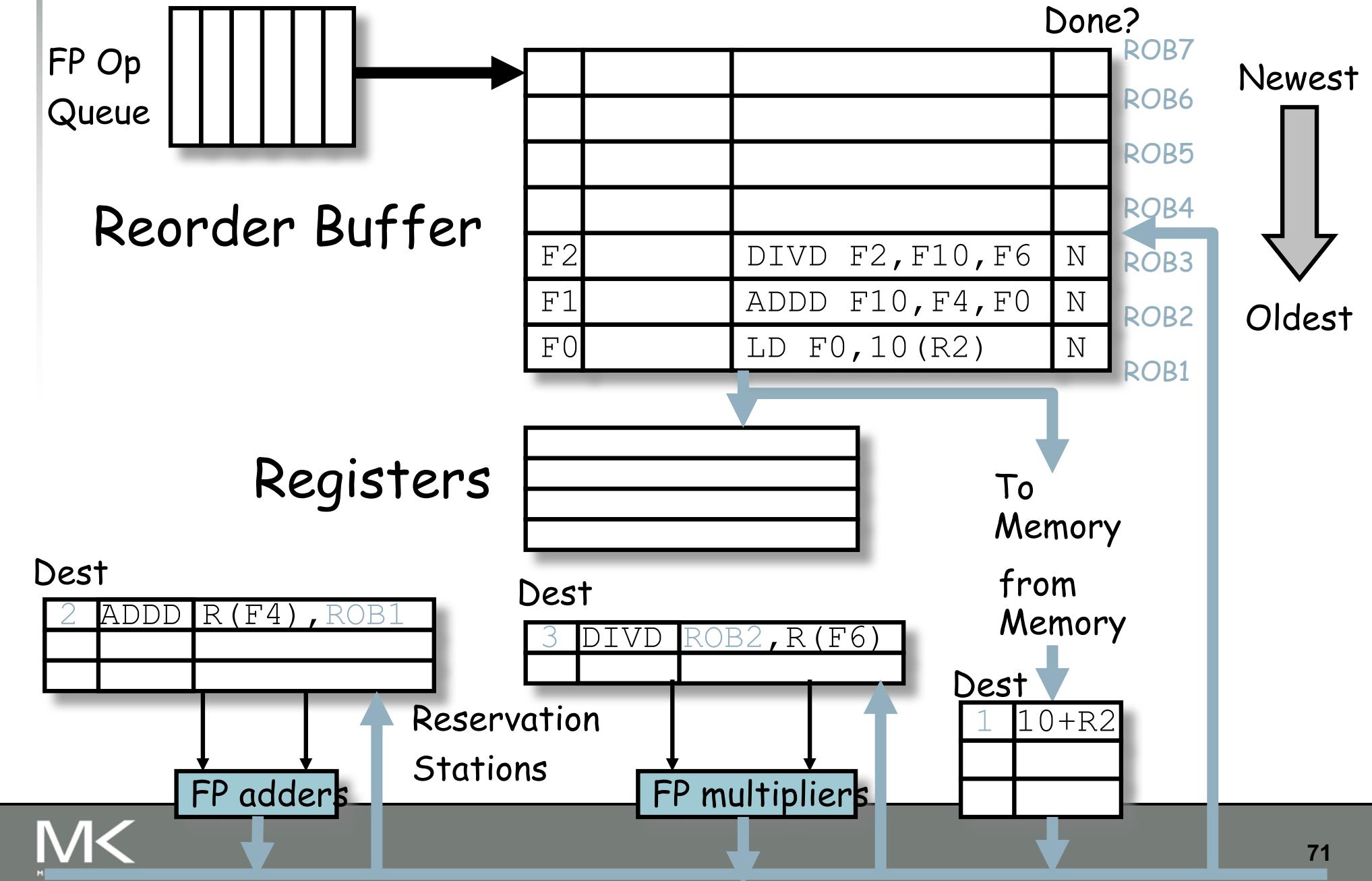
Tomasulo With Reorder buffer:



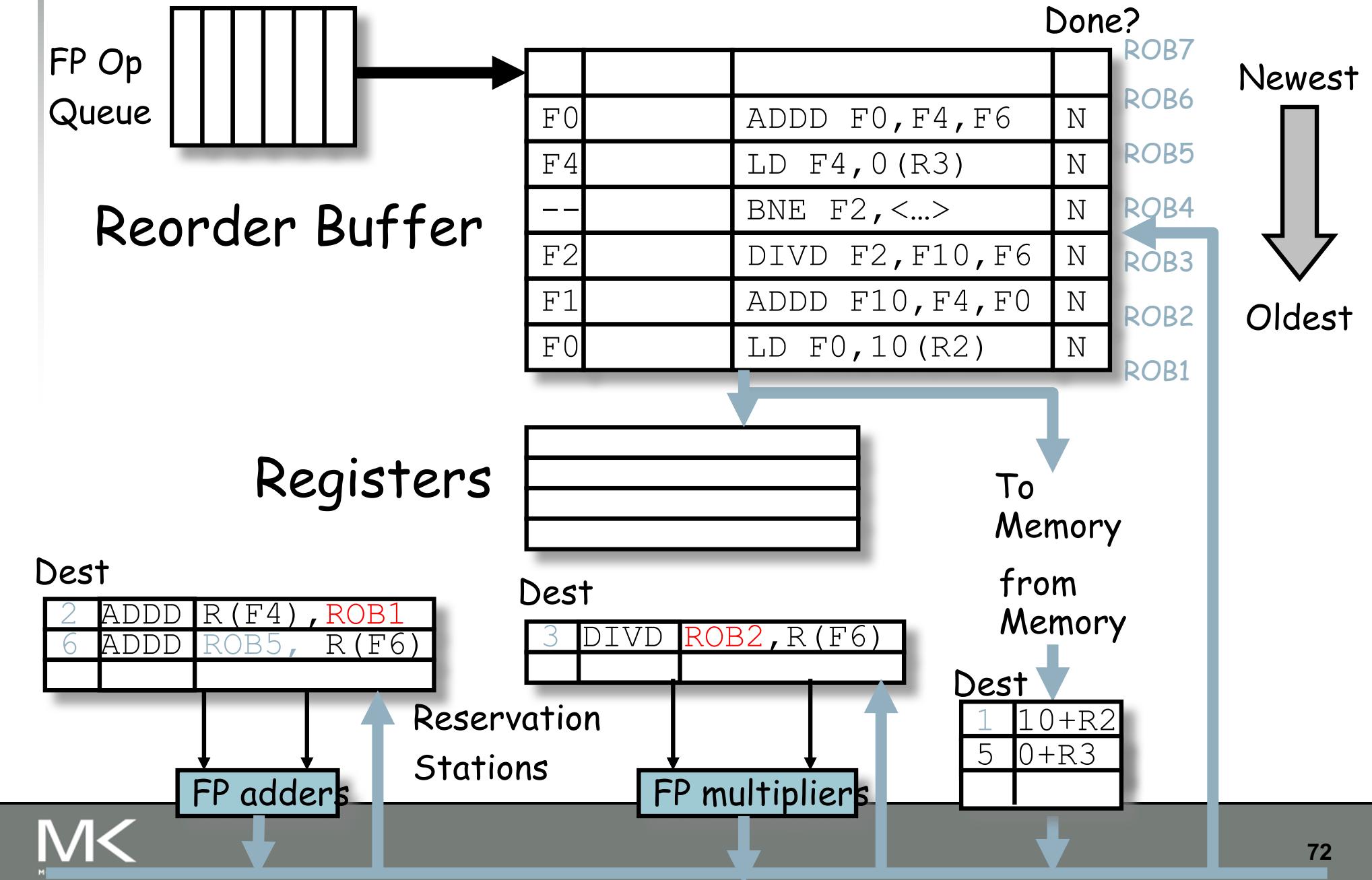
Tomasulo With Reorder buffer:



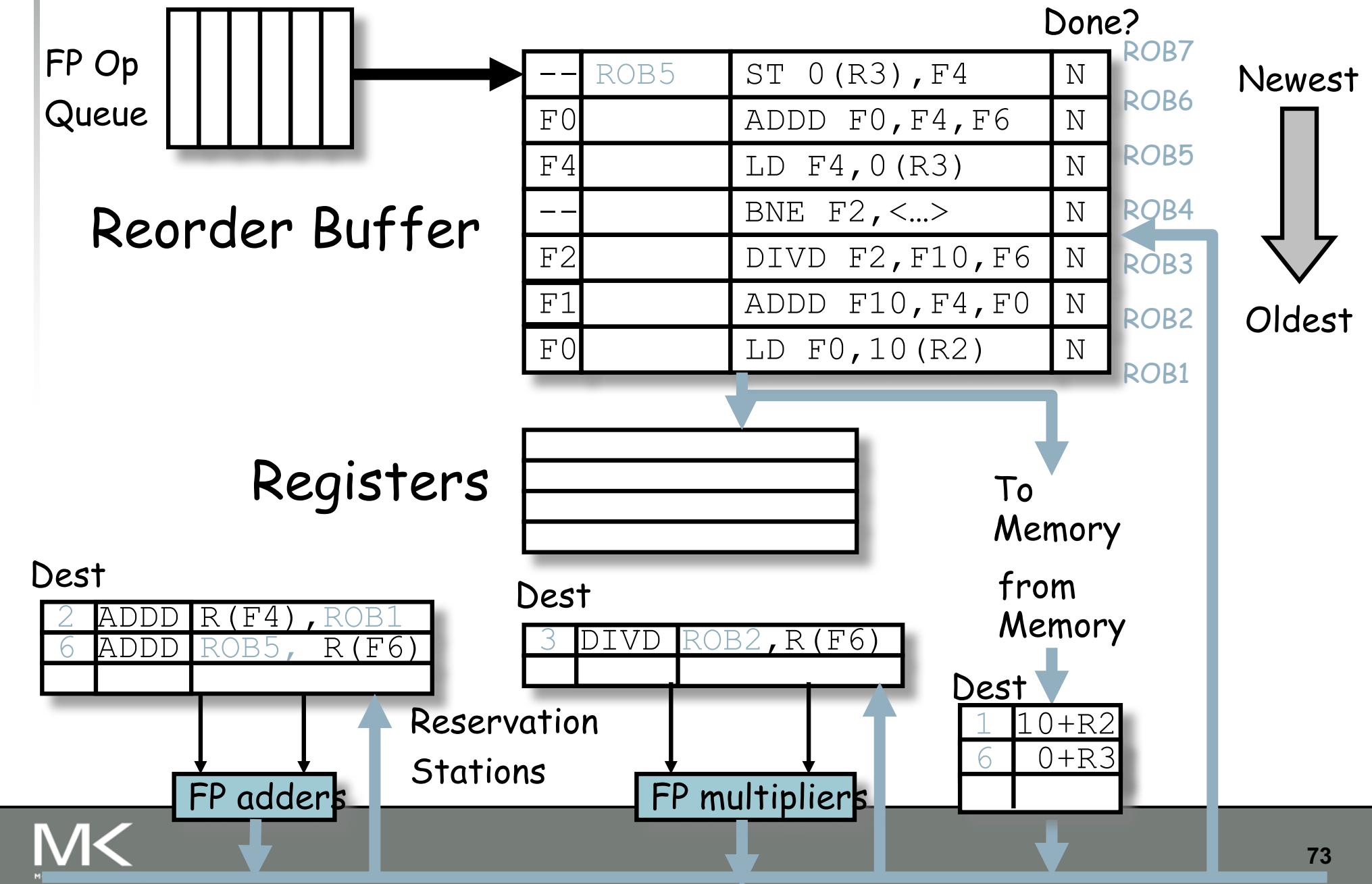
Tomasulo With Reorder buffer:



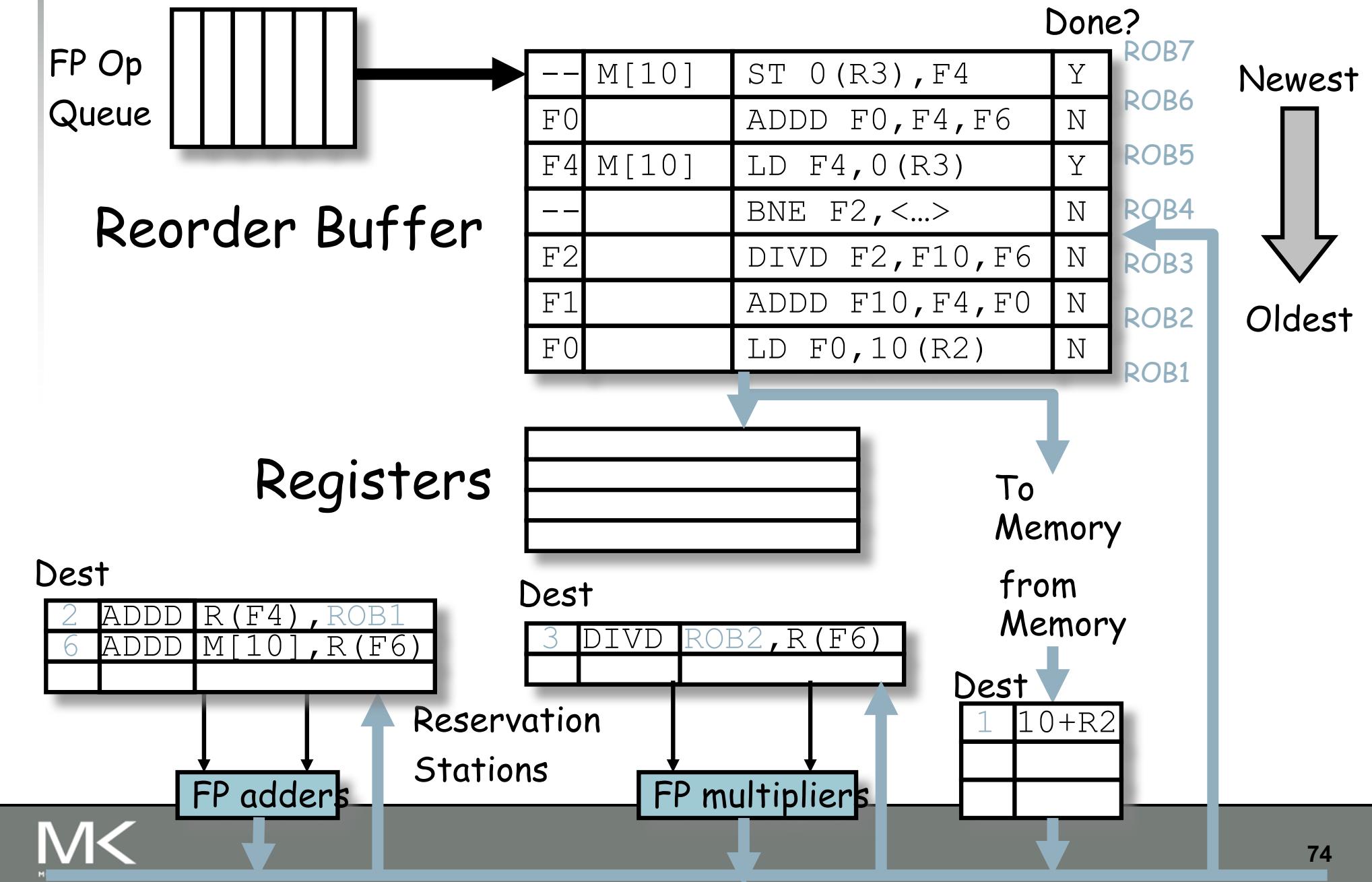
Tomasulo With Reorder buffer:



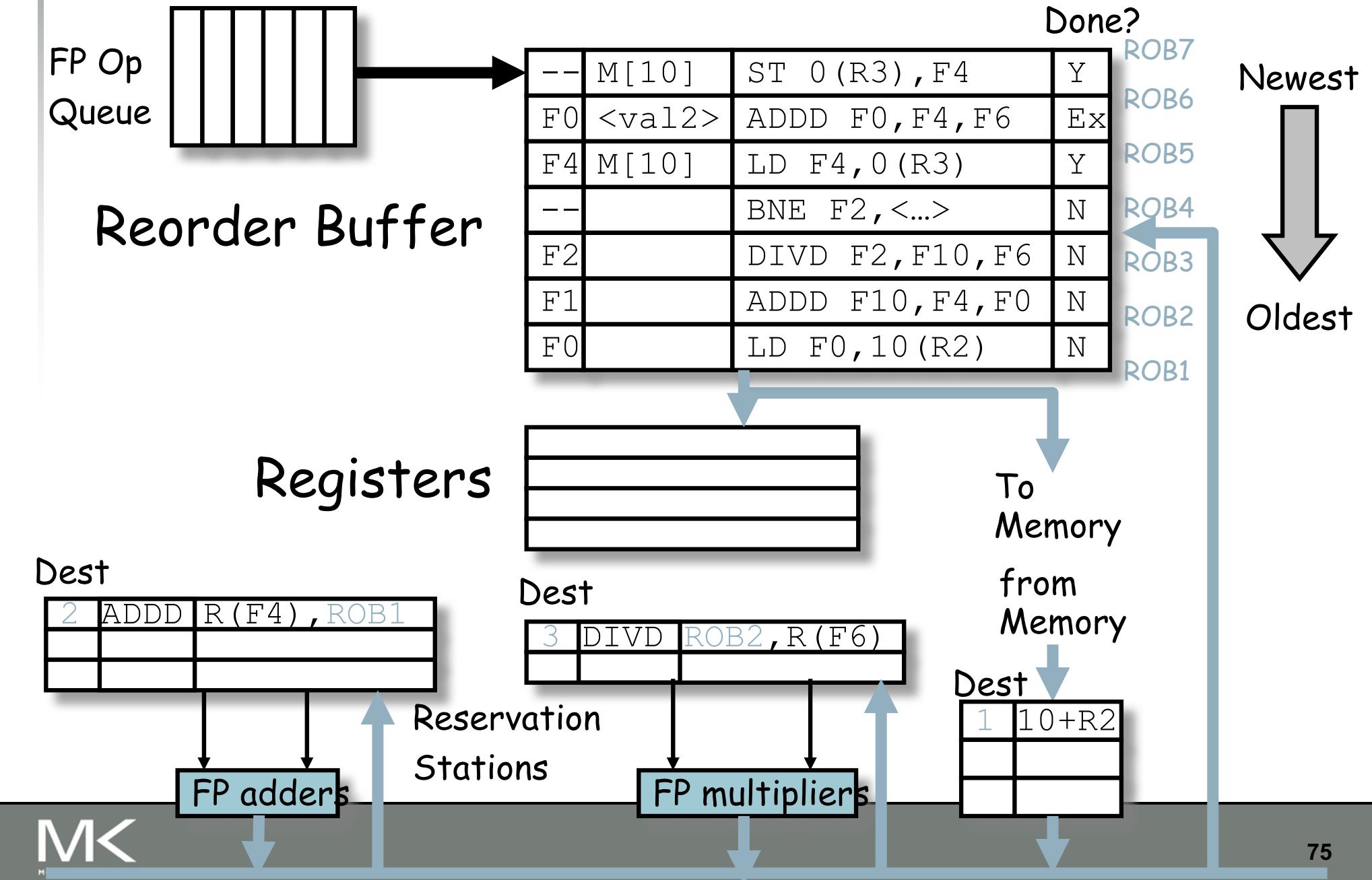
Tomasulo With Reorder buffer:



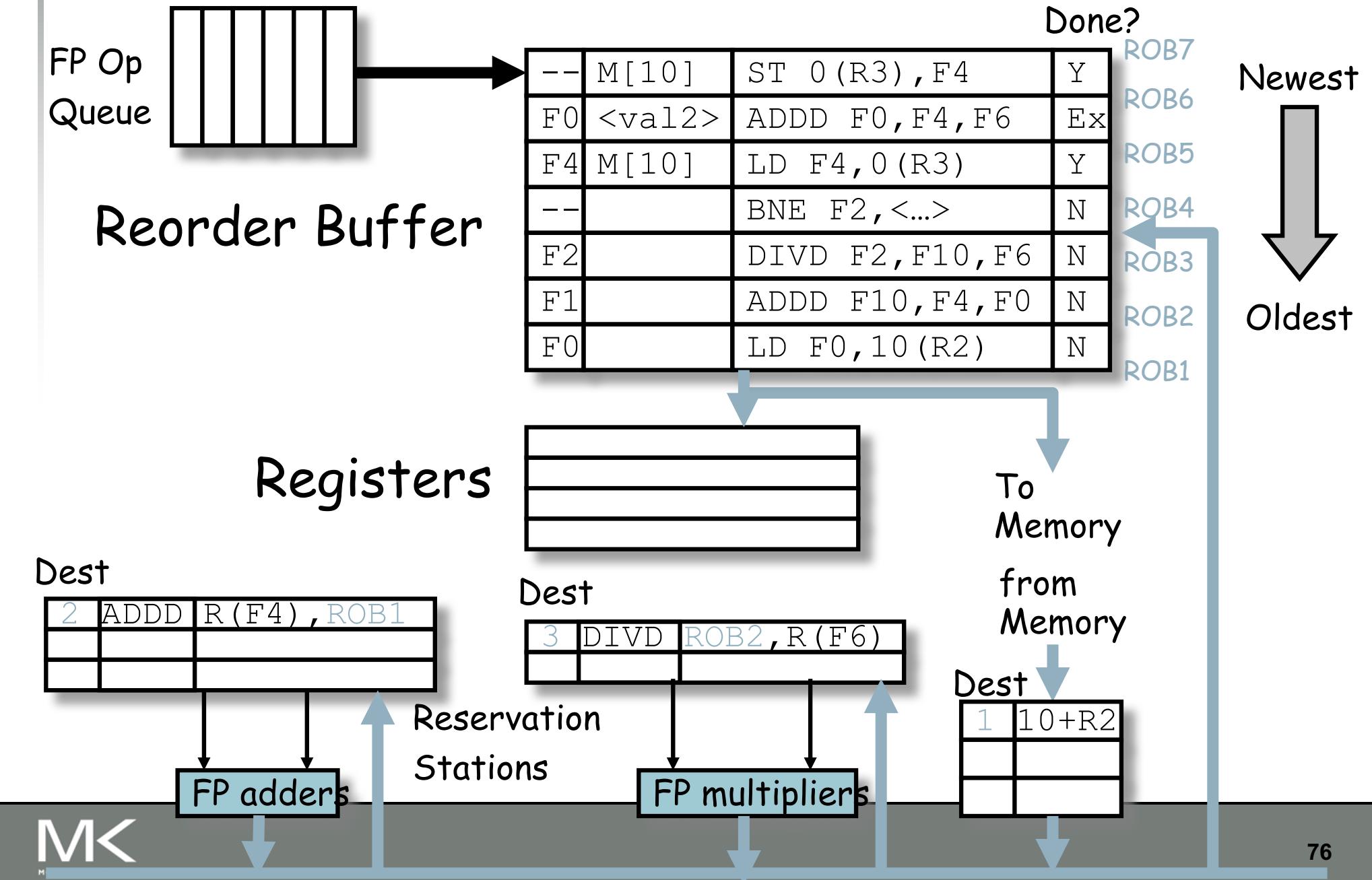
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Outline

- Instruction Level Parallelism
- Dependences
- Data Hazards
- Techniques for Exposing ILP
- Branch Prediction
- Dynamic Scheduling
- Tomasulo's Approach
- Speculation
- Advanced Techniques

Multiple Issue and Static Scheduling

- To achieve $CPI < 1$, need to complete multiple instructions per clock
- Solutions:
 - Statically scheduled superscalar processors
 - VLIW (very long instruction word) processors
 - dynamically scheduled superscalar processors

Multiple Issue

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

VLIW Processors

- Package multiple operations into one instruction
- Example VLIW processor:
 - One integer instruction (or branch)
 - Two independent floating-point operations
 - Two independent memory references
- Must be enough parallelism in code to fill the available slots

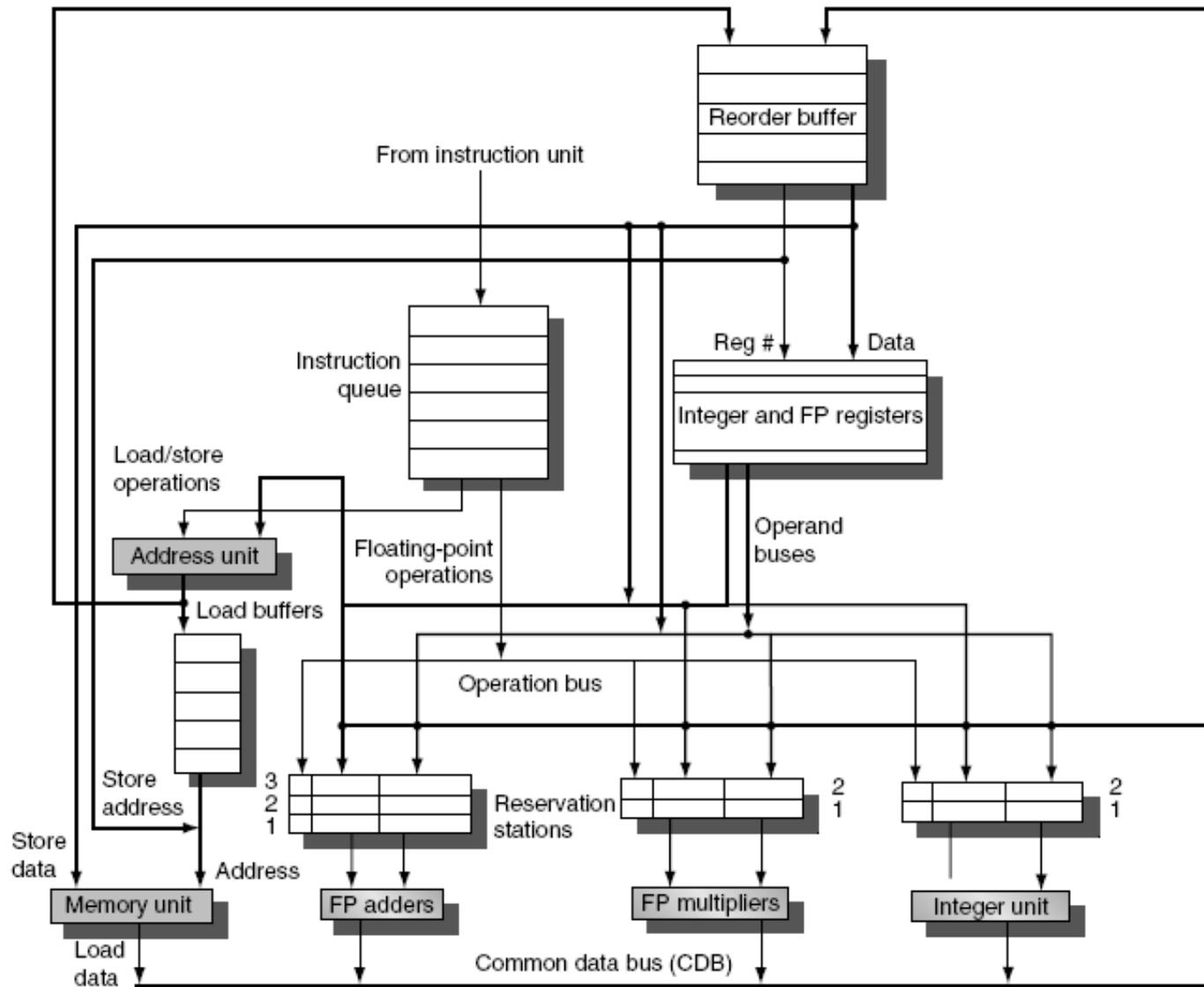
VLIW Processors

- Disadvantages:
 - Statically finding parallelism
 - Code size
 - No hazard detection hardware
 - Binary code compatibility

Dynamic Scheduling, Multiple Issue, and Speculation

- Modern microarchitectures:
 - Dynamic scheduling + multiple issue + speculation
- Two approaches:
 - Assign reservation stations and update pipeline control table in half clock cycles
 - Only supports 2 instructions/clock
 - Design logic to handle any possible dependencies between the instructions
 - Hybrid approaches
- Issue logic can become bottleneck

Overview of Design



Multiple Issue

- Limit the number of instructions of a given class that can be issued in a “bundle”
 - I.e. on FP, one integer, one load, one store
- Examine all the dependencies among the instructions in the bundle
- If dependencies exist in bundle, encode them in reservation stations
- Also need multiple completion/commit

Example

Increase each element of an integer array

Loop: LD R2,0(R1) ;R2=array element
DADDIU R2,R2,#1 ;increment R2
SD R2,0(R1) ;store result
DADDIU R1,R1,#8 ;increment pointer
BNE R2,R3,LOOP ;branch if not last element

Example (No Speculation)

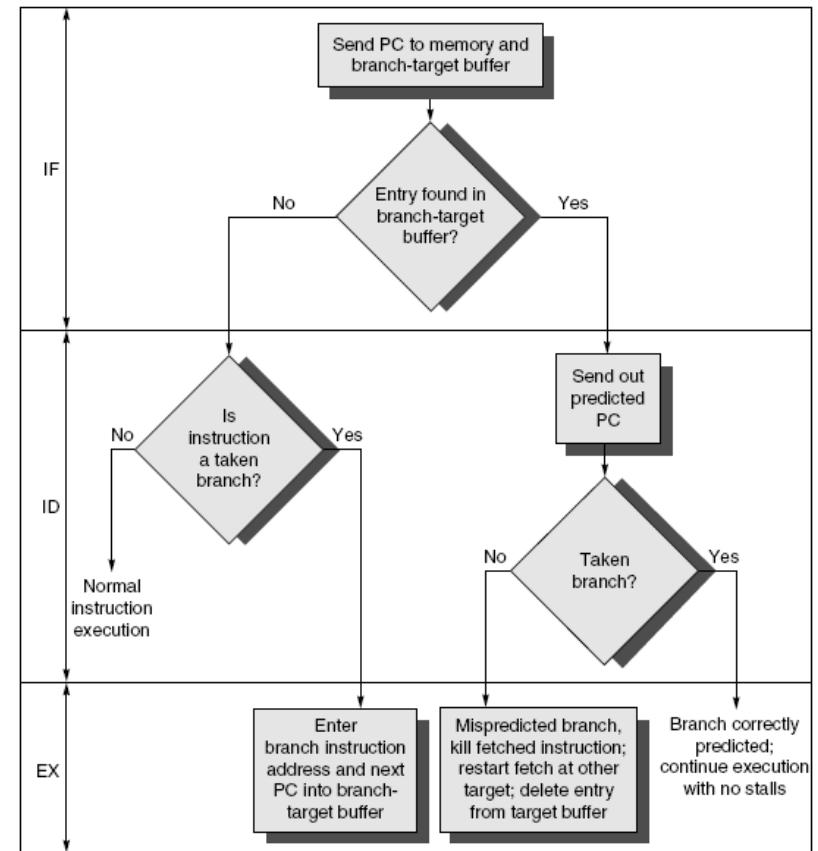
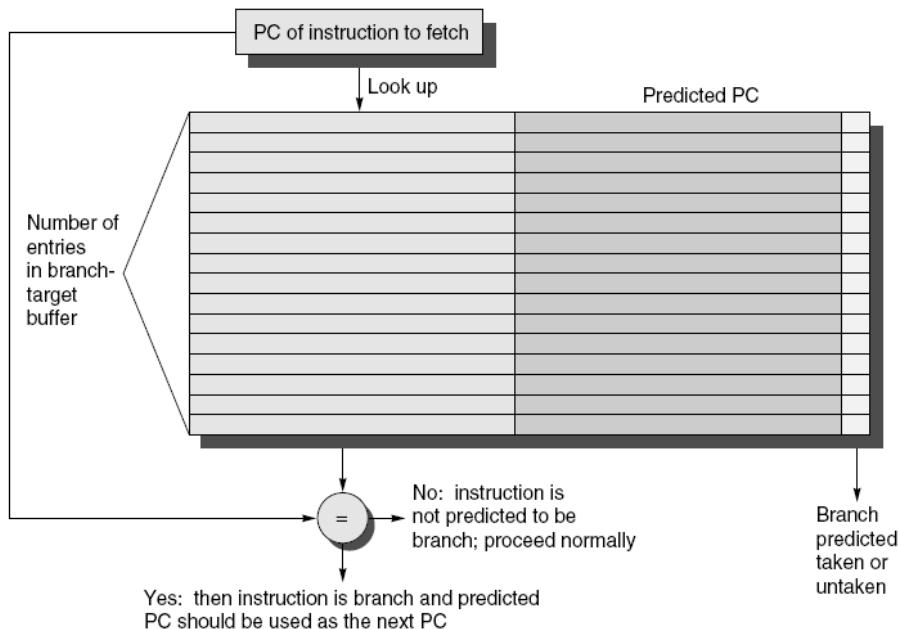
Iteration number	Instructions	Issues at clock cycle number	Executes at clock cycle number	Memory access at clock cycle number	Write CDB at clock cycle number	Comment
1	LD R2,0(R1)	1	2	3	4	First issue
1	DADDIU R2,R2,#1	1	5		6	Wait for LW
1	SD R2,0(R1)	2	3	7		Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	Execute directly
1	BNE R2,R3,LOOP	3	7			Wait for DADDIU
2	LD R2,0(R1)	4	8	9	10	Wait for BNE
2	DADDIU R2,R2,#1	4	11		12	Wait for LW
2	SD R2,0(R1)	5	9	13		Wait for DADDIU
2	DADDIU R1,R1,#8	5	8		9	Wait for BNE
2	BNE R2,R3,LOOP	6	13			Wait for DADDIU
3	LD R2,0(R1)	7	14	15	16	Wait for BNE
3	DADDIU R2,R2,#1	7	17		18	Wait for LW
3	SD R2,0(R1)	8	15	19		Wait for DADDIU
3	DADDIU R1,R1,#8	8	14		15	Wait for BNE
3	BNE R2,R3,LOOP	9	19			Wait for DADDIU

Example

Iteration number	Instructions	Issues at clock number	Executes at clock number	Read access at clock number	Write CDB at clock number	Commits at clock number	Comment
1	LD R2,0(R1)	1	2	3	4	5	First issue
1	DADDIU R2,R2,#1	1	5		6	7	Wait for LW
1	SD R2,0(R1)	2	3			7	Wait for DADDIU
1	DADDIU R1,R1,#8	2	3		4	8	Commit in order
1	BNE R2,R3,LOOP	3	7			8	Wait for DADDIU
2	LD R2,0(R1)	4	5	6	7	9	No execute delay
2	DADDIU R2,R2,#1	4	8		9	10	Wait for LW
2	SD R2,0(R1)	5	6			10	Wait for DADDIU
2	DADDIU R1,R1,#8	5	6		7	11	Commit in order
2	BNE R2,R3,LOOP	6	10			11	Wait for DADDIU
3	LD R2,0(R1)	7	8	9	10	12	Earliest possible
3	DADDIU R2,R2,#1	7	11		12	13	Wait for LW
3	SD R2,0(R1)	8	9			13	Wait for DADDIU
3	DADDIU R1,R1,#8	8	9		10	14	Executes earlier
3	BNE R2,R3,LOOP	9	13			14	Wait for DADDIU

Branch-Target Buffer

- Need high instruction bandwidth!
 - Branch-Target buffers
 - Next PC prediction buffer, indexed by current PC



Branch Folding

- Optimization:
 - Larger branch-target buffer
 - Add target instruction into buffer to deal with longer decoding time required by larger buffer
 - “Branch folding”

Return Address Predictor

- Most unconditional branches come from function returns
- The same procedure can be called from multiple sites
 - Causes the buffer to potentially forget about the return address from previous calls
- Create return address buffer organized as a stack

Integrated Instruction Fetch Unit

- Design monolithic unit that performs:
 - Branch prediction
 - Instruction prefetch
 - Fetch ahead
 - Instruction memory access and buffering
 - Deal with crossing cache lines

Register Renaming

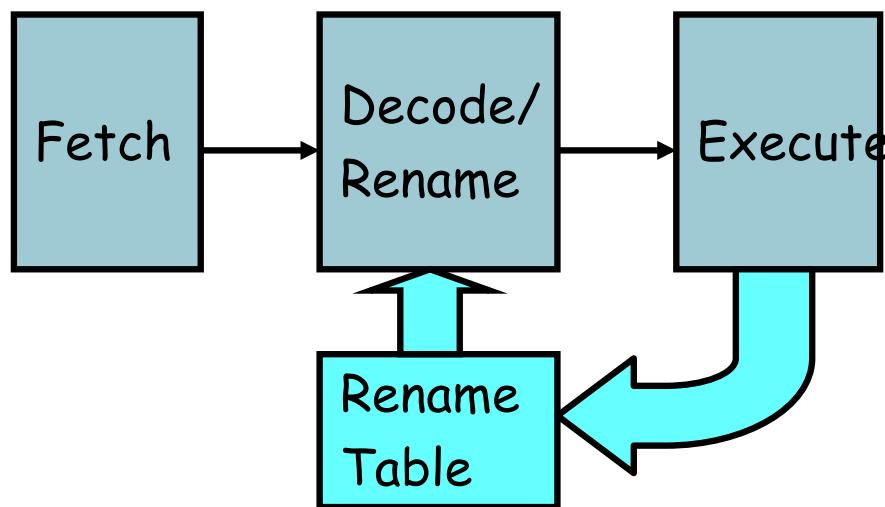
- Register renaming vs. reorder buffers
 - Instead of virtual registers from reservation stations and reorder buffer, create a single register pool
 - Contains visible registers and virtual registers
 - Use hardware-based map to rename registers during issue
 - WAW and WAR hazards are avoided
 - Speculation recovery occurs by copying during commit
 - Still need a ROB-like queue to update table in order
 - Simplifies commit:
 - Record that mapping between architectural register and physical register is no longer speculative
 - Free up physical register used to hold older value
 - In other words: SWAP physical registers on commit
 - Physical register de-allocation is more difficult

Integrated Issue and Renaming

- Combining instruction issue with register renaming:
 - Issue logic pre-reserves enough physical registers for the bundle (fixed number?)
 - Issue logic finds dependencies within bundle, maps registers as necessary
 - Issue logic finds dependencies between current bundle and already in-flight bundles, maps registers as necessary

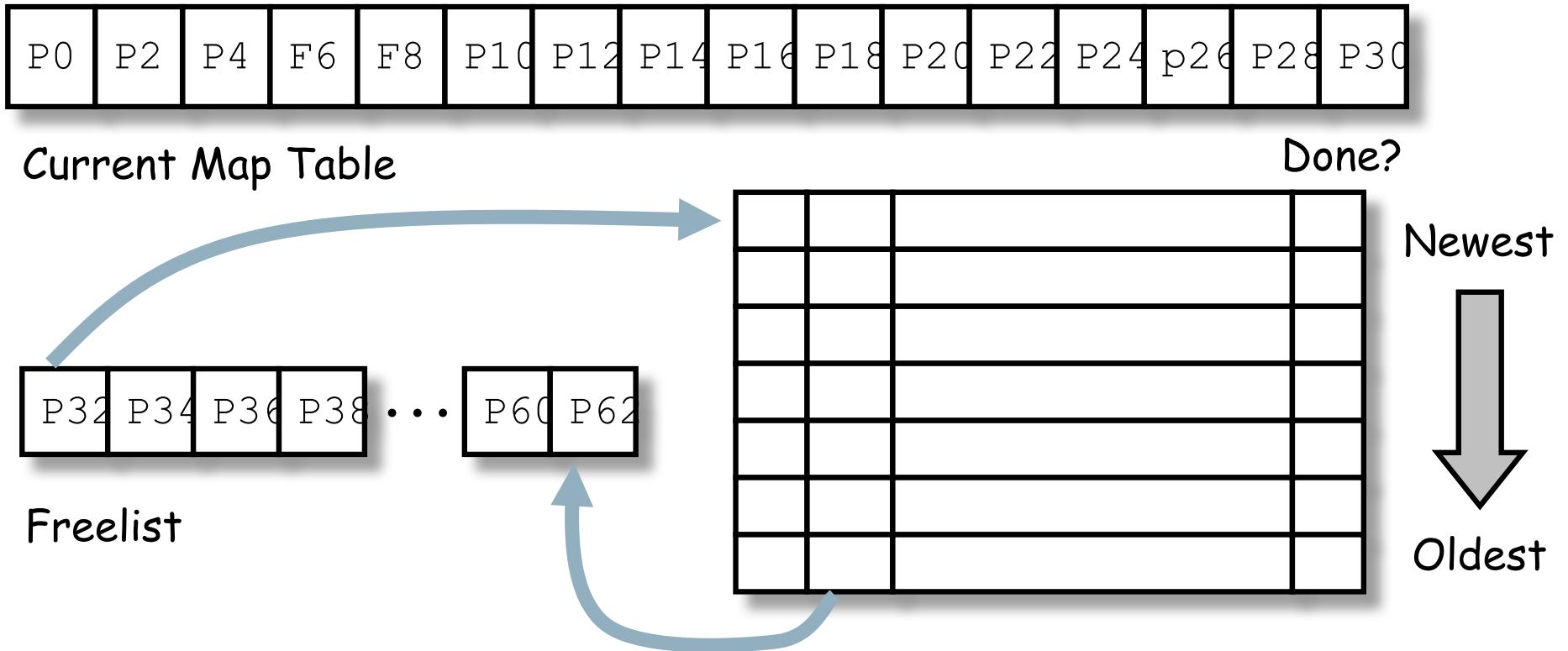
Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
 - ISA register => physical register mapping
 - When register is written, replace table entry with new register from freelist.
 - Physical register becomes free when not being used by any instructions in progress.



Explicit register renaming:

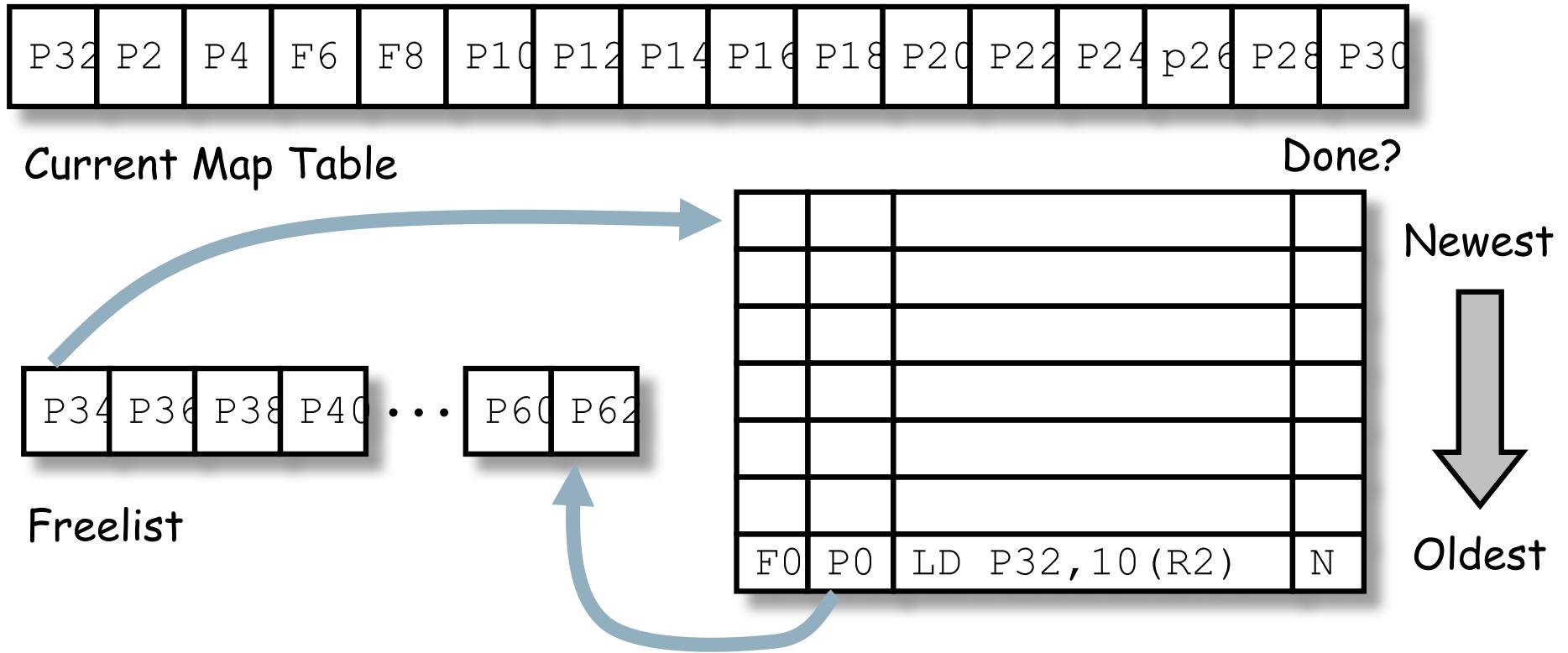
R10000 Freelist Management



- Physical register file larger than ISA register file
- On issue, each instruction that modifies a register is allocated new physical register from freelist
- Used on: R10000, Alpha 21264, HP PA8000

Explicit register renaming:

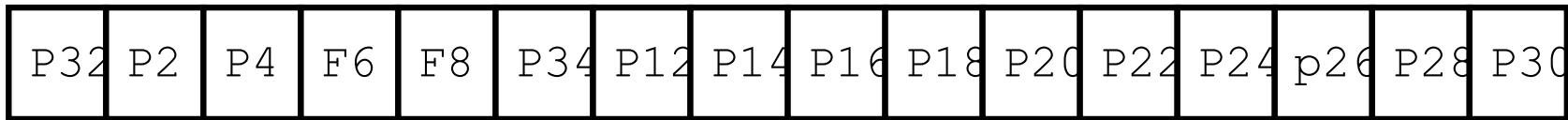
R10000 Freelist Management



- Note that physical register P0 is “dead” (or not “live”) past the point of this load.
 - When we go to commit the load, we free up

Explicit register renaming:

R10000 Freelist Management



Current Map Table



Freelist

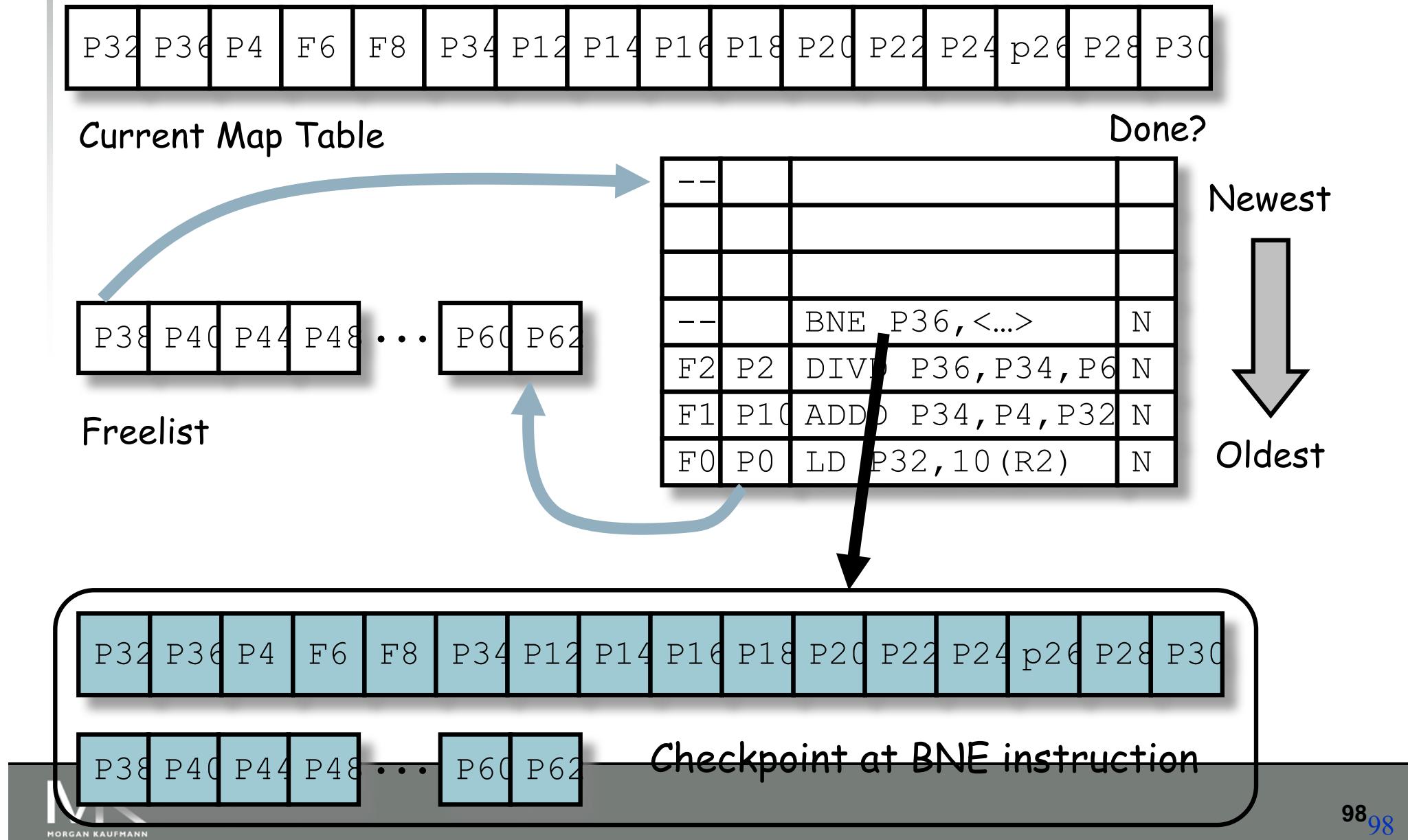
F1	P10	ADDD P34, P4, P32	N
F0	P0	LD P32, 10(R2)	N

Done?

Newest
↓
Oldest

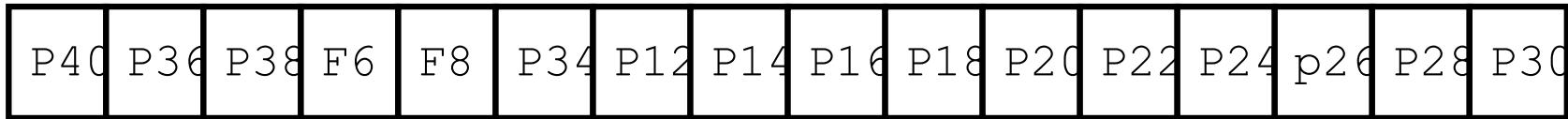
Explicit register renaming:

R10000 Freelist Management



Explicit register renaming:

R10000 Freelist Management



Current Map Table

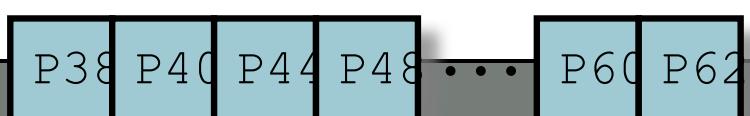
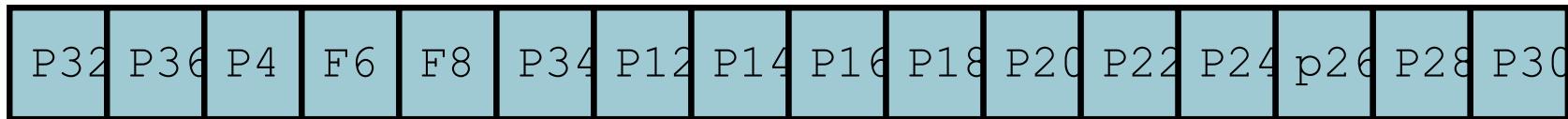
Done?



Freelist

--		ST 0 (R3) , P40	Y
F0	P32	ADDD P40, P38, P6	Y
F4	P4	LD P38, 0 (R3)	Y
--		BNE P36, <...>	N
F2	P2	DIVD P36, P34, P6	N
F1	P10	ADDD P34, P4, P32	y
F0	P0	LD P32, 10 (R2)	y

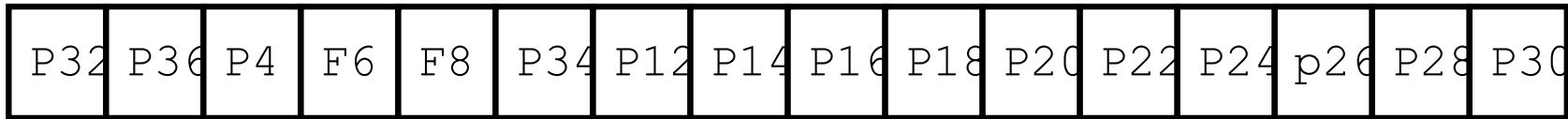
Newest
↓
Oldest



Checkpoint at BNE instruction

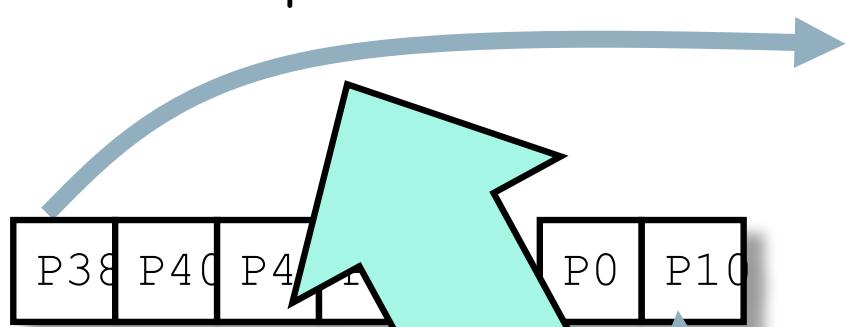
Explicit register renaming:

R10000 Freelist Management



Current Map Table

Done?



Freelist

F2	P2	DIVD	P36, P34, P6	N
F1	P10	ADDD	P34, P4, P32	y
F0	P0	LD	P32, 10 (R2)	y

Newest
↓
Oldest

Error fixed by restoring map table and merging freelist



Checkpoint at BNE instruction

How Much?

- How much to speculate
 - Mis-speculation degrades performance and power relative to no speculation
 - May cause additional misses (cache, TLB)
 - Prevent speculative code from causing higher costing misses (e.g. L2)
- Speculating through multiple branches
 - Complicates speculation recovery
 - No processor can resolve multiple branches per cycle

Energy Efficiency

- Speculation and energy efficiency
 - Note: speculation is only energy efficient when it significantly improves performance
- Value prediction
 - Uses:
 - Loads that load from a constant pool
 - Instruction that produces a value from a small set of values
 - Not been incorporated into modern processors
 - Similar idea--*address aliasing prediction*--is used on some processors