CS152 – Computer Architecture and Engineering
Lecture 18 – Advanced Pipelining: Modern Out-Of-Order Microarchitectures

2003-10-28

Dave Patterson
(www.cs.berkeley.edu/~patterson)

www-inst.eecs.berkeley.edu/~cs152/

Review

- Reservations stations: renaming to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
- Helps cache misses as well
- Lasting Contributions
  - Dynamic scheduling
  - Register renaming
  - Load/store disambiguation
- 360/91 descendants are Pentium II,III, 4; PowerPC 604; MIPS R10000; HP-PA 8000; Alpha 21264; ...

2003 Reflections on Tomasulo

- Can interrupts be made precise?
  - What happens with mis-predicted Branches?
- What about RAW hazards in memory?
- Implement out-of-order without reservation station registers?
- Out-of-order **and** superscalar?
- Limits to ILP?

Now what about exceptions?

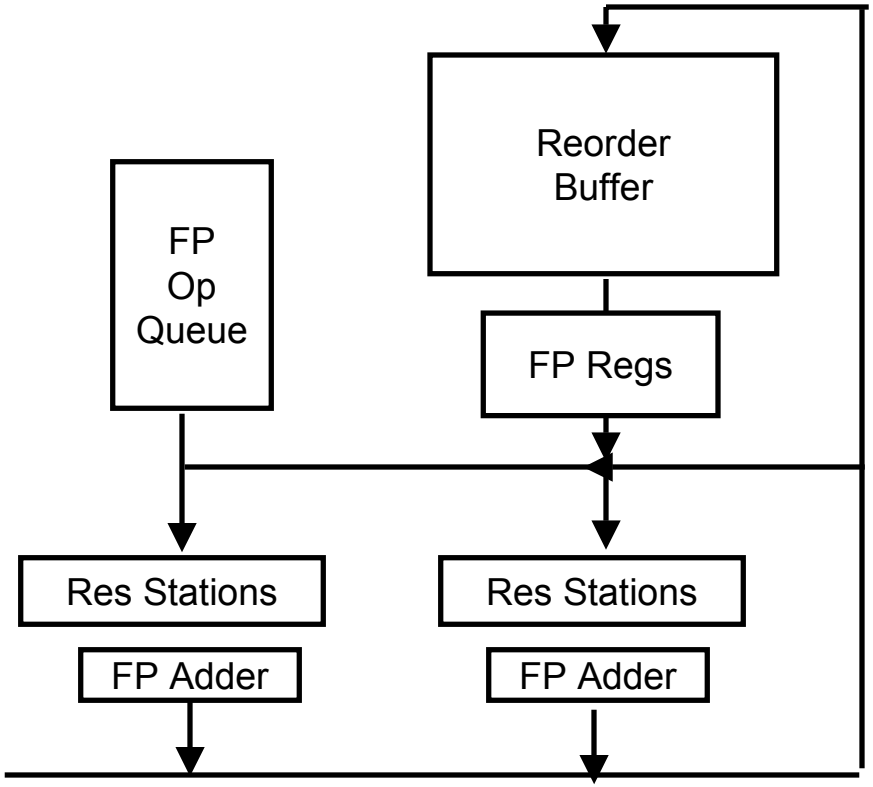
- Out-of-order commit really messes up our chance to get precise exceptions!
  - Register file contains results from later instructions while earlier ones have not completed yet.
  - What if need to cause exception on one of those early instructions??
- Need to “rollback” register file to consistent state:
  - Recall: “precise” interrupt means that there is some PC such that:
    - all instructions before have committed results
    - and none after have committed results.
- Technique for precise exceptions: *in-order completion or commit*
  - Must commit instruction results in same order as issue

Out-of-order execution vs. commit

- Basically, speculatively executing instructions
- Allow speculative instruction to execute and bypass results to other instructions
  - ~ a speculative register read
- When instruction no longer speculative, can update register file or memory
  - Called *instruction commit*
- Execute out-of-order but commit in-order
- Need HW to hold results of instructions executed but not committed
  => *Reorder Buffer (ROB)*

HW support for precise interrupts

- Need HW buffer for results of uncommitted instructions: *reorder buffer (ROB)*
  - 3 fields: instr, destination, value
  - Reorder buffer can be operand source => more registers like RS
  - Use reorder buffer number instead of reservation station
  - Instructions commit when done
  - ROB supplies operands between execution complete & commit
  - Once instruction commits, result is put into register
  - As a result, its easy to undo instructions on exceptions or on mispredicted branches



Comments on ROB

- ROB provides extra registers (like RS)
- ROB is source of operands (like RS)
- Tomasulo: once instruction writes result, subsequently issued instructions get result from register file
- ROB: register file not updated until instruction commits, so ROB supplies operands between completion of instruction execution and commit
  - ROB like Store Buffer in Tomasulo
- Exceptions handled by not recognizing exception until instruction is ready to commit

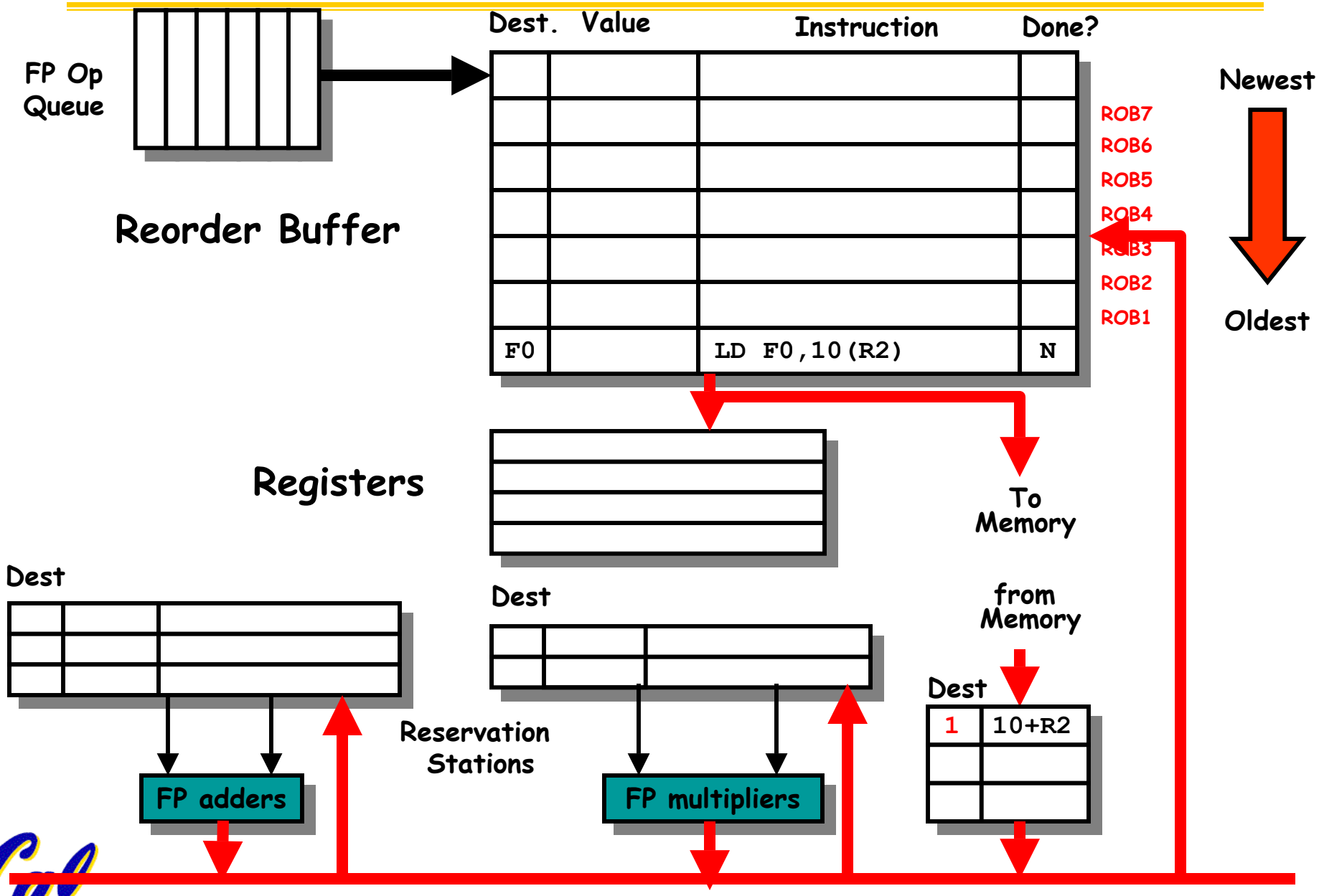
Four Steps of Speculative Tomasulo Algorithm

1. Issue—get instruction from FP Op Queue
   - If reservation station and reorder buffer slot free, issue instr & send operands & reorder buffer no. for destination
- 2. Execution—operate on operands (EX)
   - When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW
3. Write result—finish execution (WB)
   - Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available
4. Commit—update register with reorder result
   - When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr. from reorder buffer (predicted branch done)
   - Mispredicted branch at head of reorder buffer flushes reorder buffer and restarts at correct branch point. Similarly, ROB flushed for interrupts.

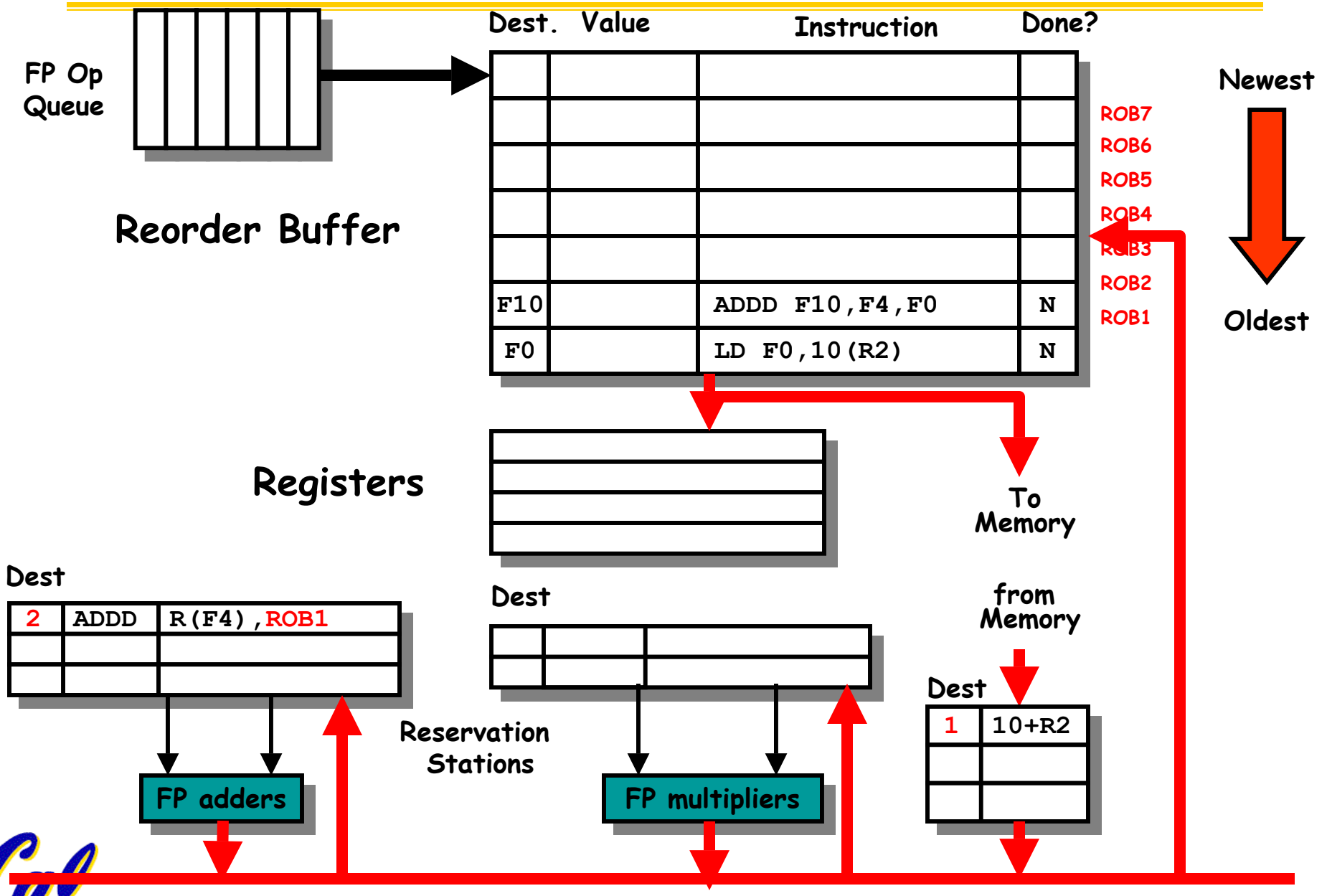
Speculative Tomasulo Example

```
        LD          F0    10    R2
        ADDD        F10   F4    F0
        DIVD        F2    F10   F6
        BNEZ        F2    Exit
        LD          F4    0     R3
        ADDD        F0    F4    F9
        SD          F4    0     R3
        …
Exit:
```

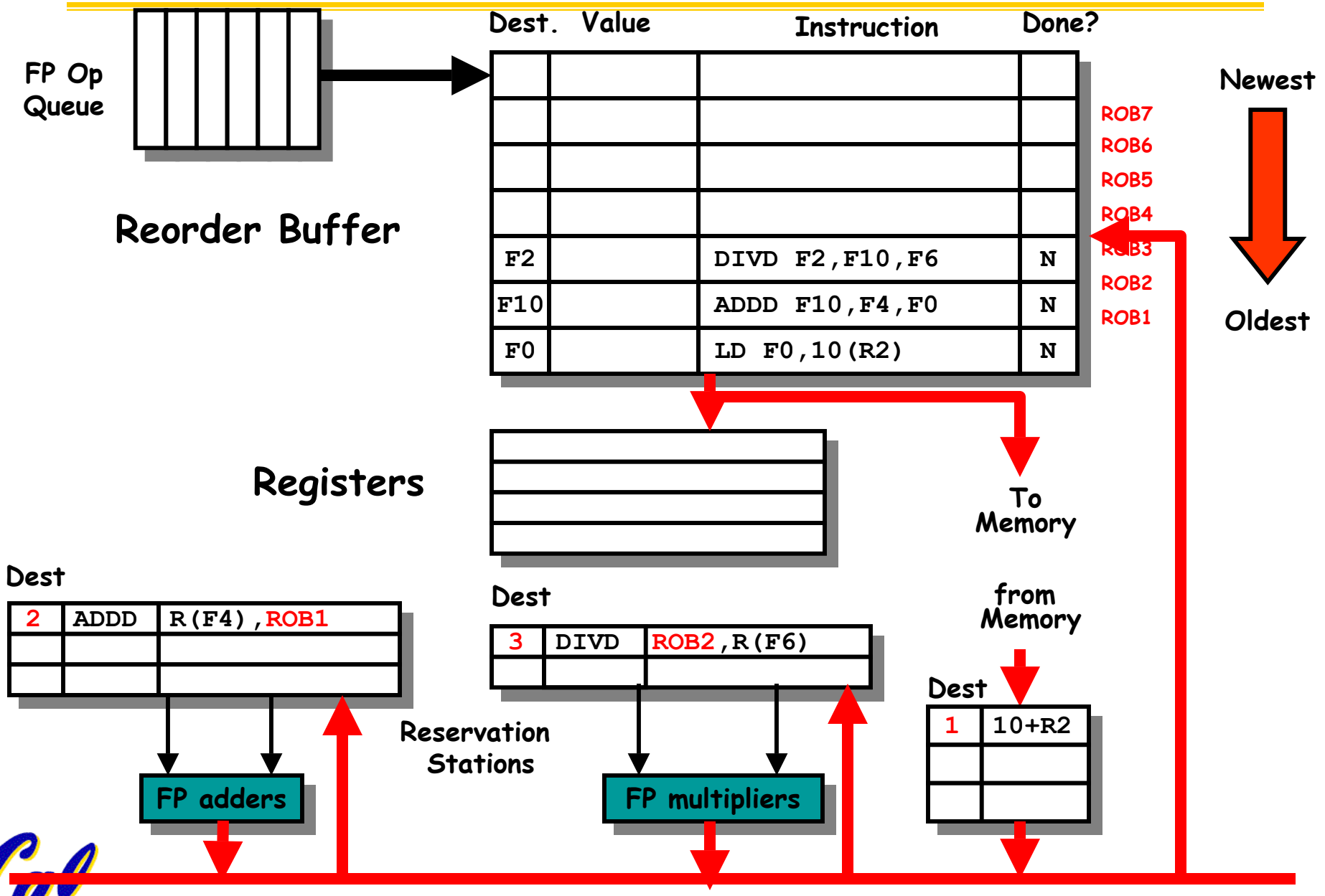
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:

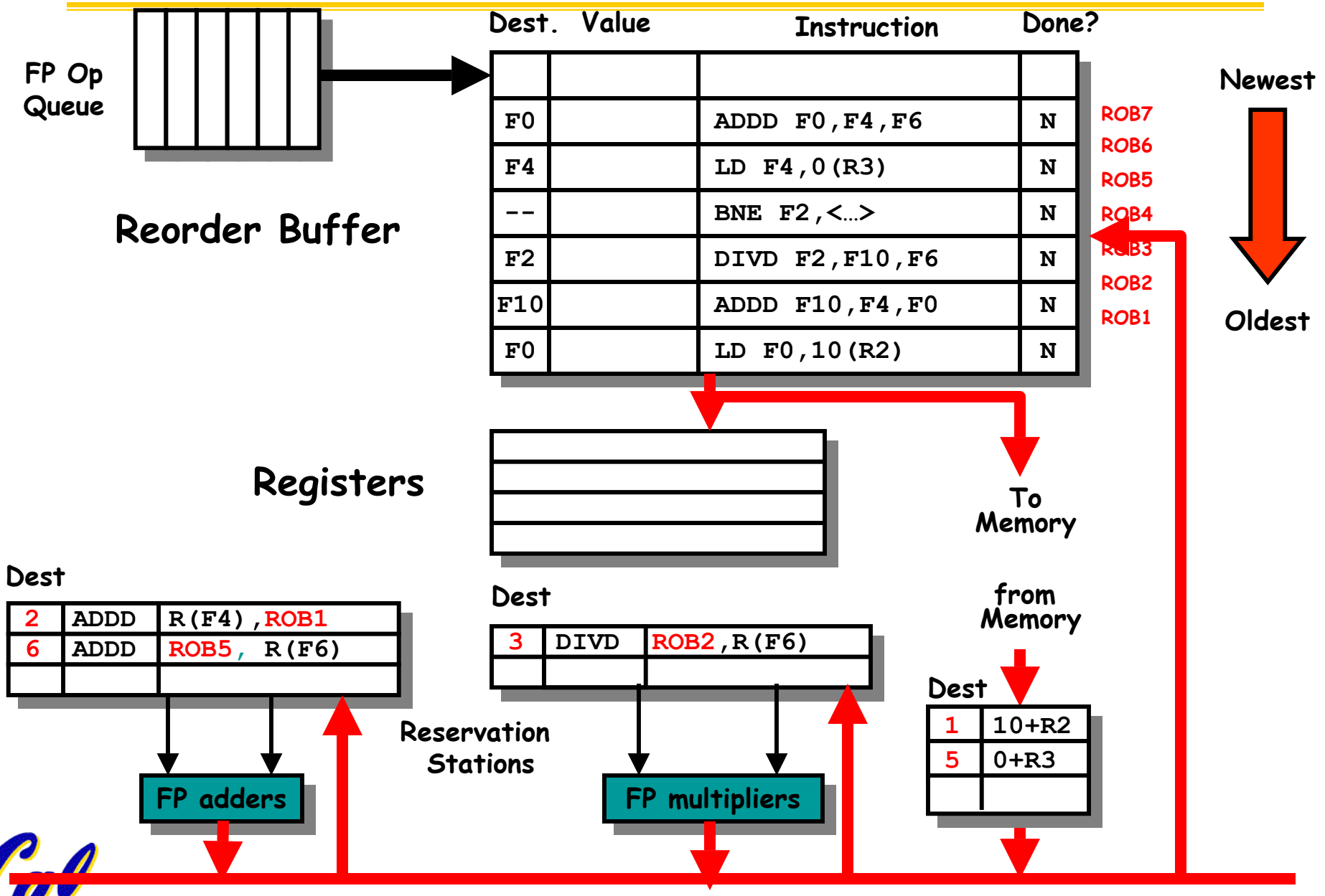


Tomasulo With Reorder buffer:

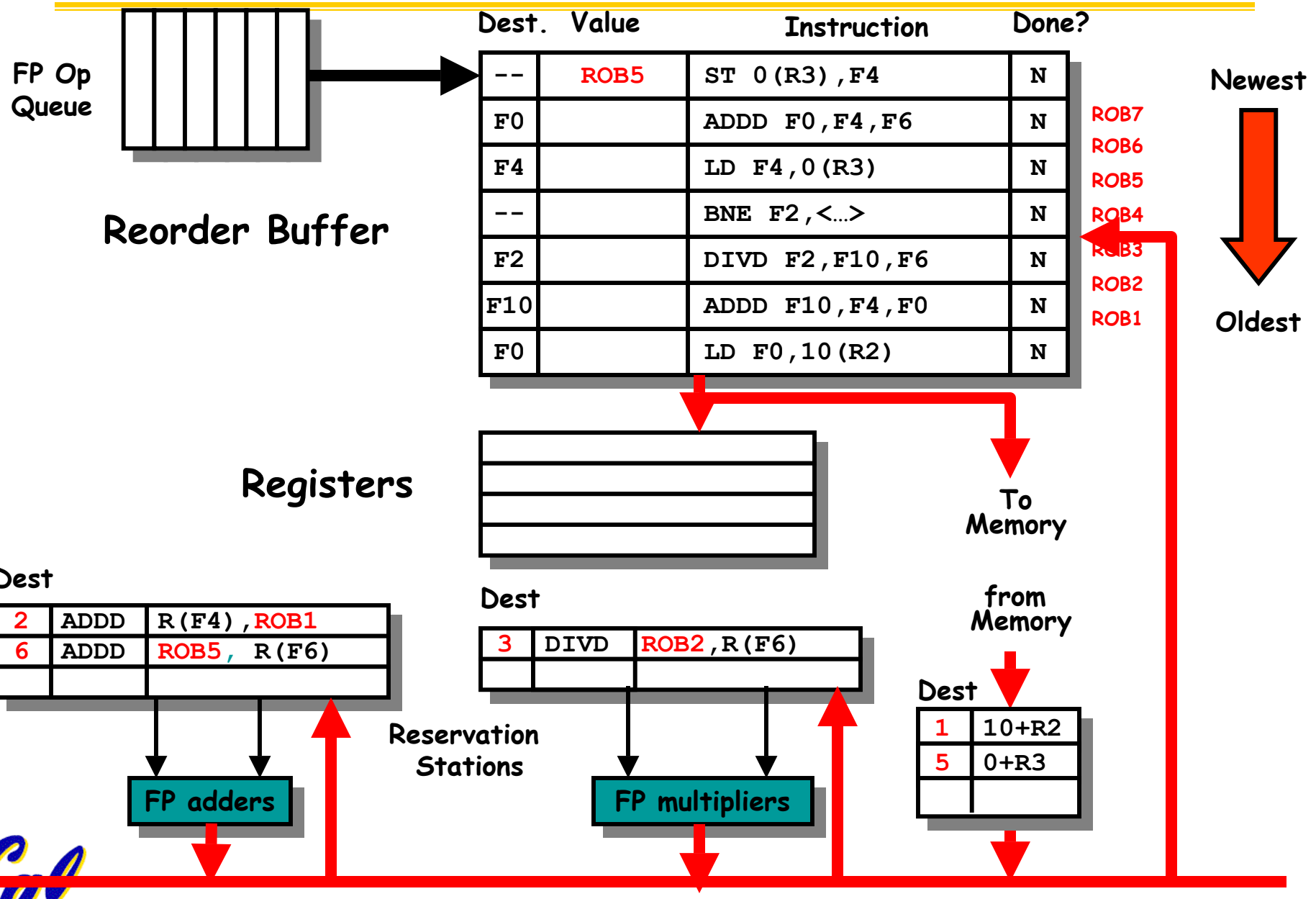


- Skip some cycles

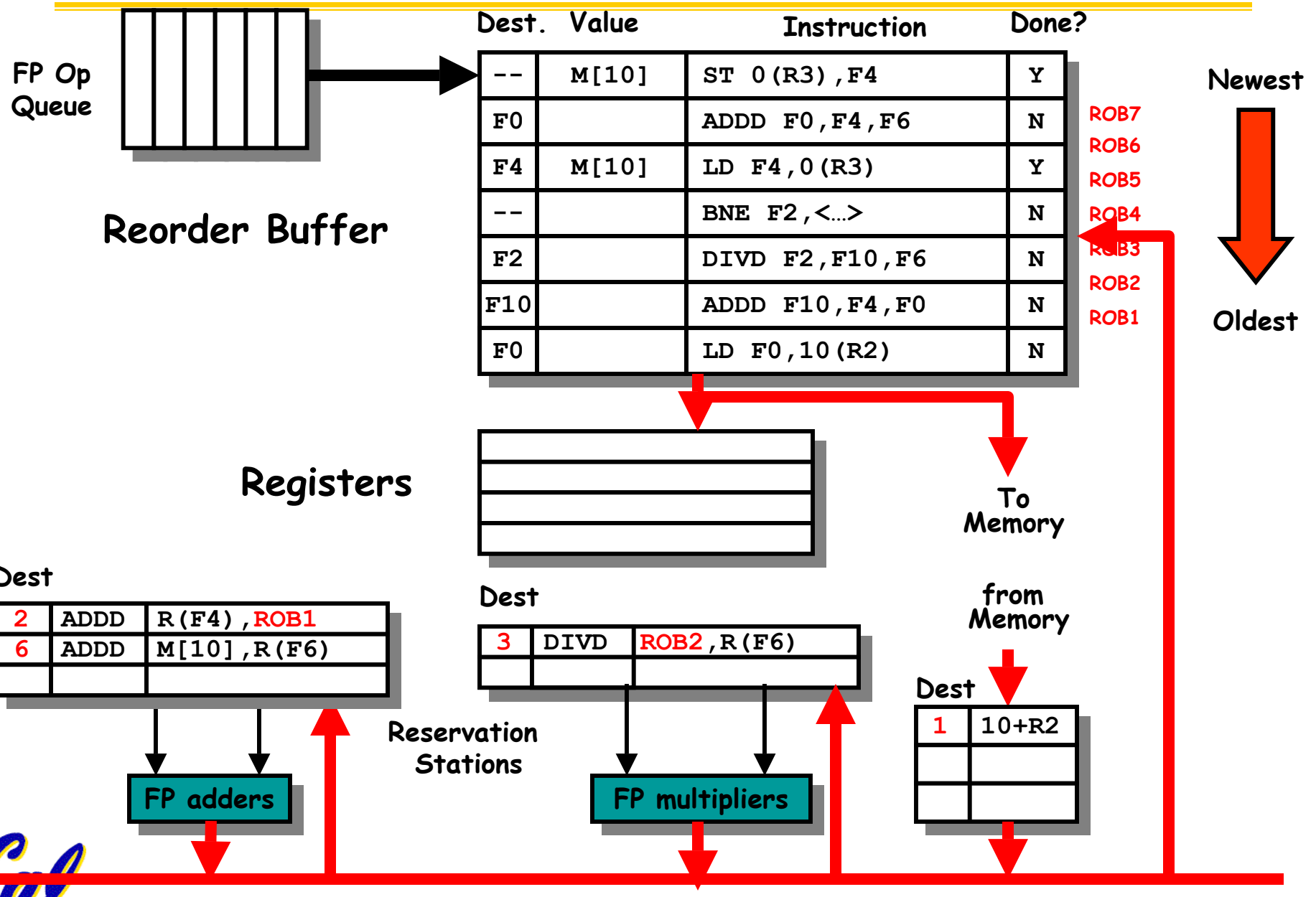
Tomasulo With Reorder buffer:



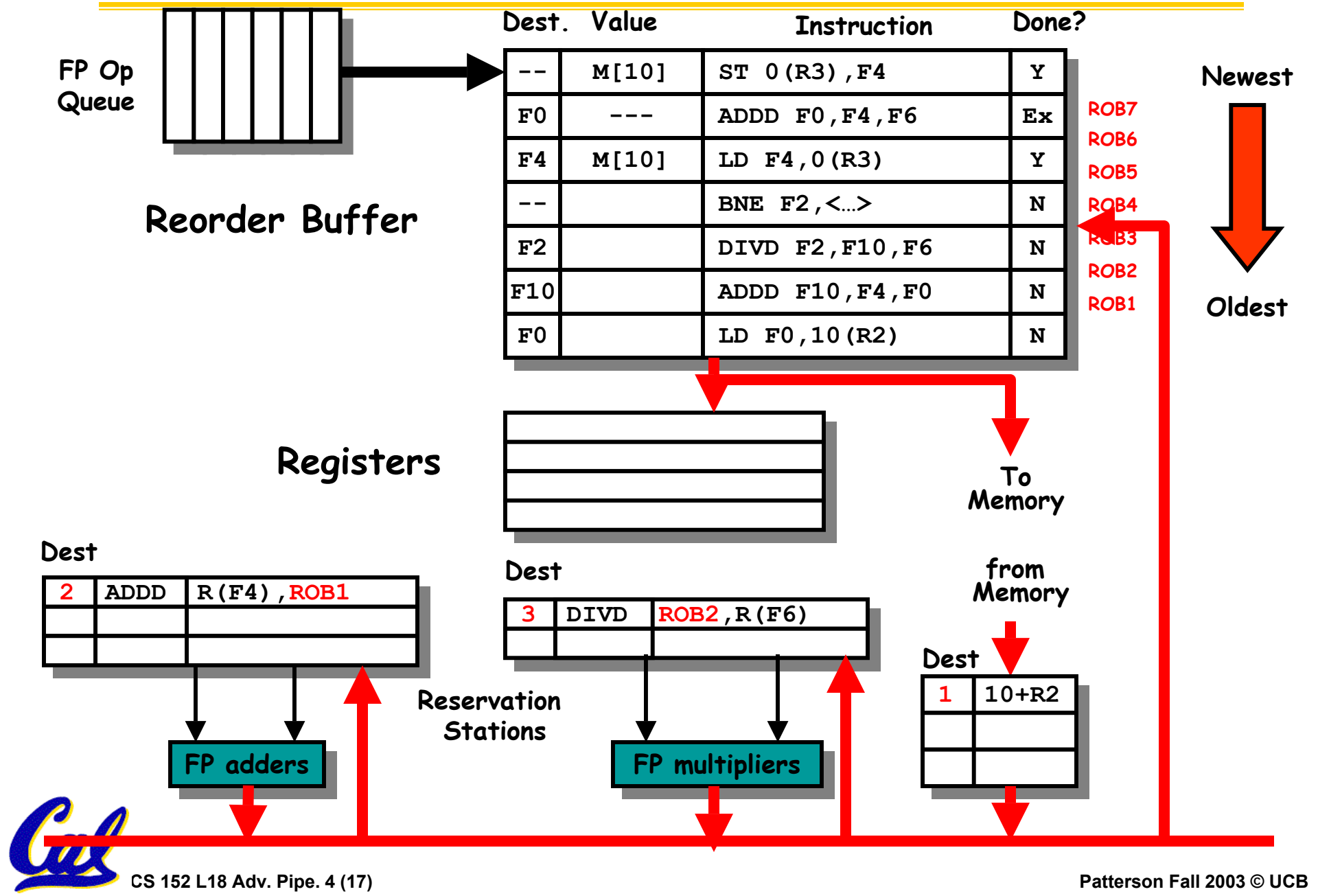
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



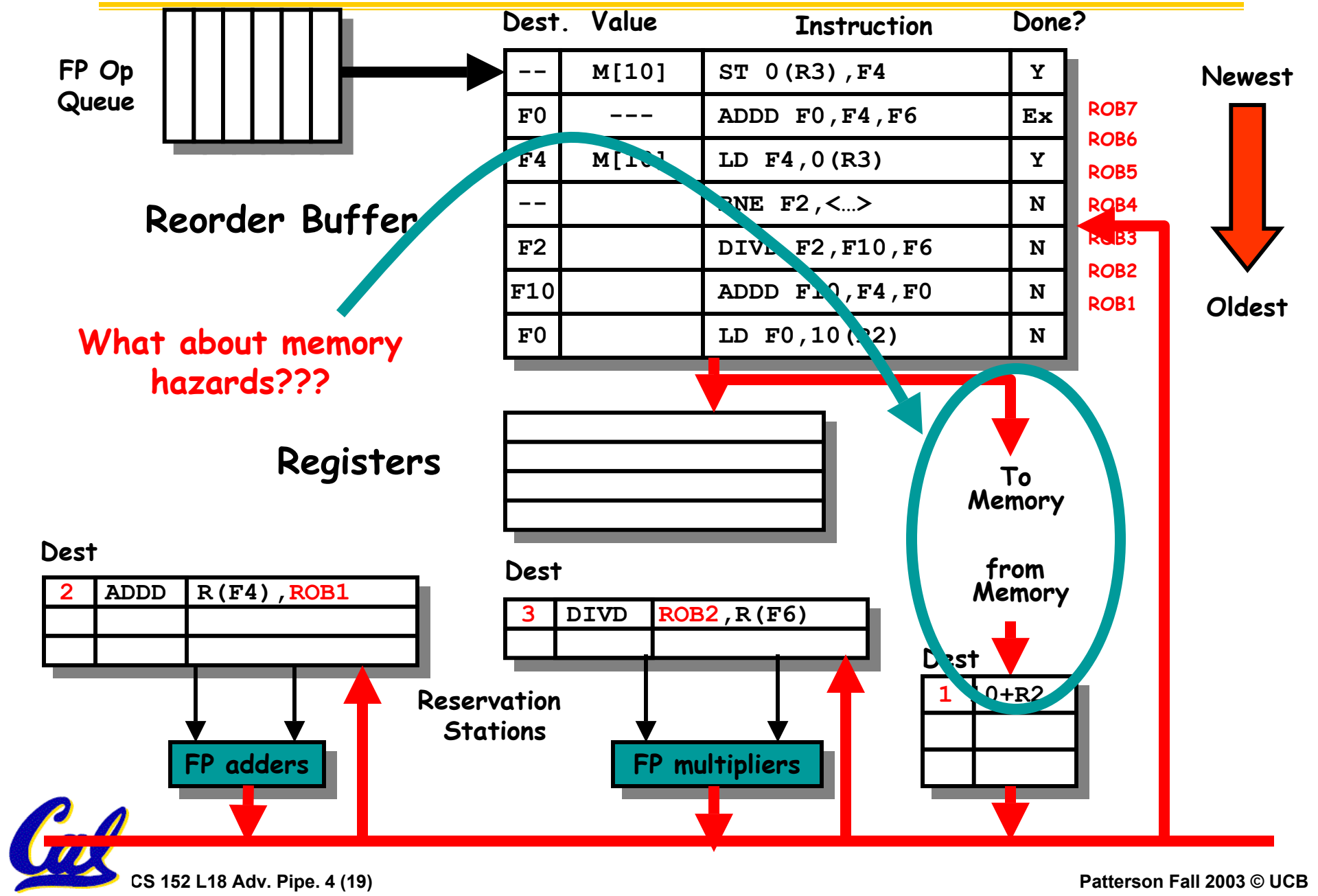
Tomasulo With Reorder buffer:



Administrivia

- Full cache demo board Friday 10/31
  - 8 more PCs in 125 Cory this week; more boards?
- Thur 11/6: Design Doc for Final Project due
  - Deep pipeline? Superscalar? Out-of-order?
- Tue 11/11: Veteran's Day (no lecture)
- Fri 11/14: Demo Project modules
- Wed 11/19: 5:30 PM Midterm 2 in 1 LeConte
- Tues 11/22: Field trip to Xilinx
- CS 152 Project Week: 12/1 to 12/5
  - Mon: TA Project demo, Tue: 30 min Presentation, Wed: Processor racing, Fri: Written report

Tomasulo With Reorder buffer:



Memory Disambiguation: Handling RAW Hazards in memory

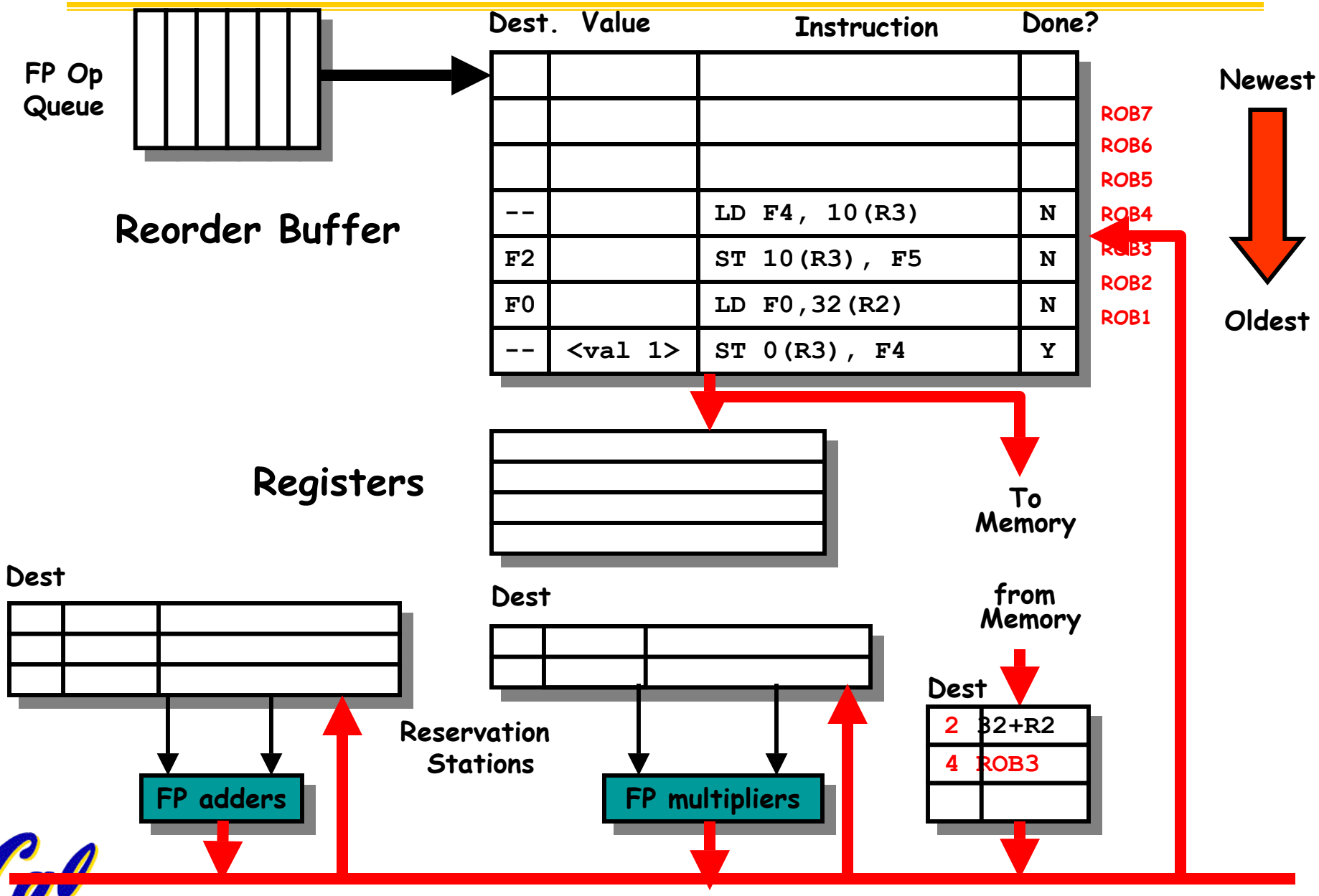
- Question: Given a load that follows a store in program order, are the two related?
  - (Alternatively: is there a RAW hazard between the store and the load)?

    Eg:
    ```
    SW      0(R2),R5
    LW      R6,0(R3)
    ```
- Can we go ahead and start the load early?
  - Store address could be delayed for a long time by some calculation that leads to R2 (divide?).
  - We might want to issue/begin execution of both operations in same cycle.
- Two techiques:
  - No Speculation: we are not allowed to start load until we know *for sure* that address 0(R2) ≠ 0(R3)
  - Speculation: We might guess at whether or not they are dependent (called "dependence speculation") and use reorder buffer to fixup if we are wrong.

Hardware Support for Memory Disambiguation

- Need buffer to keep track of all outstanding stores to memory, in program order.
  - Keep track of address (when becomes available) and value (when becomes available)
  - FIFO ordering: will retire stores from this buffer in program order
- When issuing a load, record current head of store queue (know which stores are ahead of you).
- When have address for load, check store queue:
  - If *any* store prior to load is waiting for its address, stall load.
  - If load address matches earlier store address (associative lookup), then we have a *memory-induced RAW hazard*:
    - store value available ⇒ return value
    - store value not available ⇒ return ROB number of source
  - Otherwise, send out request to memory
- Actual stores commit in order, so no worry about WAR/WAW hazards through memory.

Memory Disambiguation:



PRS: Out-of-order (OOO) execution

- Which are true about OOO execution?

1. Since integer execution is fast compared to floating point, floating point programs are the primary beneficiary of OOO
2. Reservation stations remove WAR, WAW hazards of Scoreboard
3. Reorder Buffers enable In-Order Issue, Out-Of-Order execution, and In-Order completion

1. ABC: FFF
2. ABC: FFT
3. ABC: FTF
4. ABC: FTT
5. ABC: TFF
6. ABC: TFT
7. ABC: TTF
8. ABC: TTT

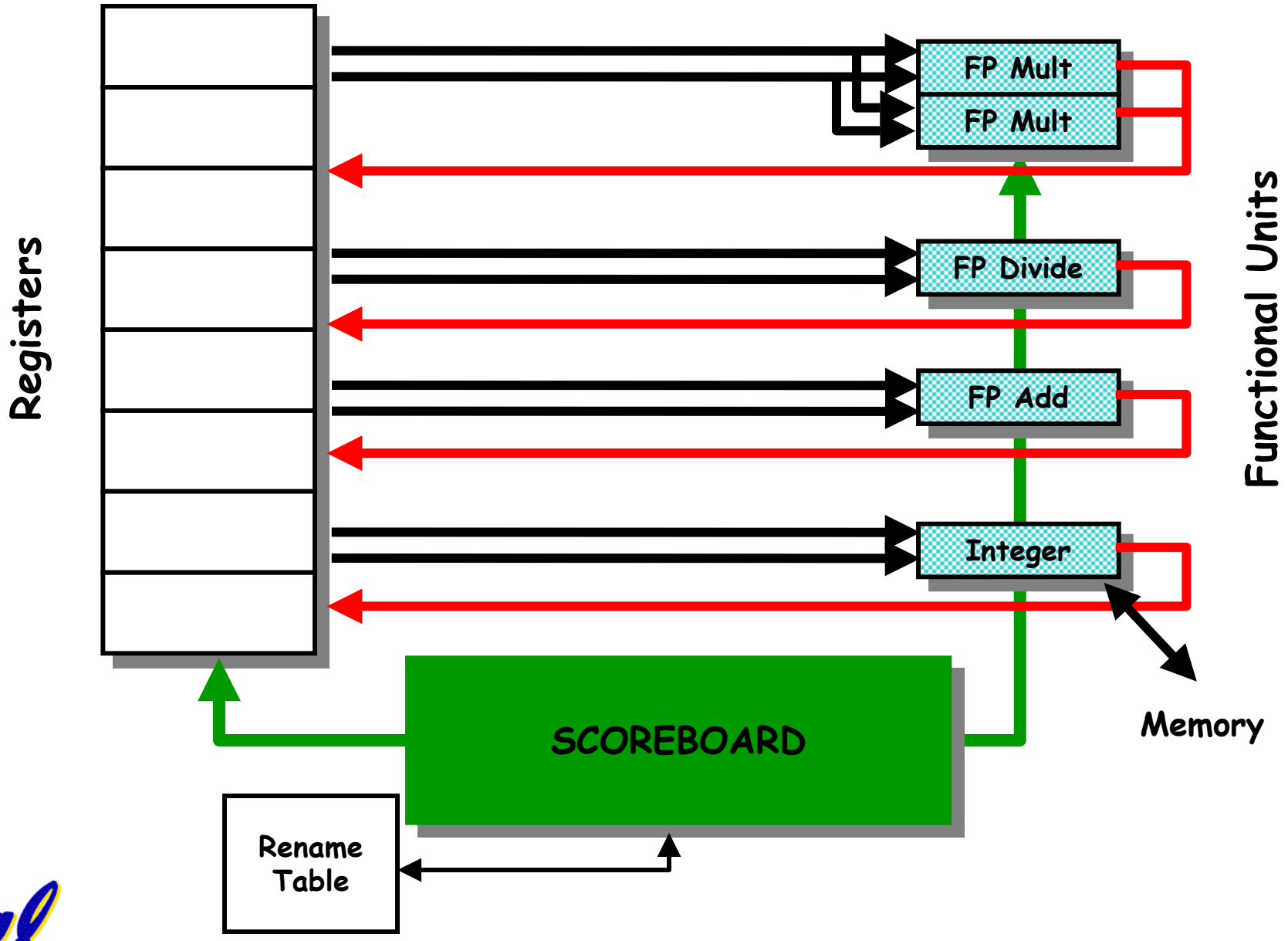
Explicit Register Renaming

- Make use of a *physical* register file that is larger than number of registers specified by ISA
- Key insight: Allocate a new physical destination register for every instruction that writes
  - Removes all chance of WAR or WAW hazards
  - Like Tomasulo, good for allowing full out-of-order completion
  - Like hardware-based dynamic compilation?
- Mechanism? Keep a translation table:
  - ISA register $\Rightarrow$ physical register mapping
  - When register written, replace entry with new register from free list
  - Physical register becomes free when not used by any active instructions
- Large register file combines Reservation Station with Reorder Buffer
- Extra 20 to 80 registers: used by all modern MPUs

Advantages of Explicit Renaming

- Decouples *renaming* from *scheduling:*
  - Pipeline can be exactly like "standard" MIPS pipeline (perhaps with multiple operations issued per cycle)
  - Or, pipeline could be tomasulo-like or a scoreboard, etc.
  - Standard forwarding or bypassing could be used
- Allows data to be fetched from single register file
  - No need to bypass values from reorder buffer
  - This can be important for balancing pipeline
- Many processors use a variant of this technique:
  - R10000, Alpha 21264, HP PA8000
- Another way to get precise interrupt points:
  - All that needs to be "undone" for precise break point is to undo the table mappings
  - Provides an interesting mix between reorder buffer and future file
    - Results are written immediately back to register file
    - Registers *names* are "freed" in program order (by ROB)

Can we use explicit register renaming with scoreboard?



Stages of Scoreboard Control With Explicit Renaming

- 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
- Read operands—wait until no hazards, read operands
  - All real dependencies (RAW hazards) resolved in this stage, since we wait for instructions to write back data.
- Execution—operate on operands
  - The functional unit begins execution upon receiving operands. When the result is ready, it notifies the scoreboard
- Write result—finish execution

- Note: No checks for WAR or WAW hazards!

Scoreboard With Explicit Renaming

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

- **Initialized Rename Table**

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	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								
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

PRS: Out-of-order (OOO) execution

- Which are true about OOO execution?

1. Pro. Register renaming vs. ROB is simplified instruction commit: only record mapping is no longer speculative & free old physical regs
2. Pro. RR vs. ROB: need only check register file vs. check ROB and register file
3. Con. RR means architecture registers never fixed, making debugging of design complex

1. ABC: FFF
2. ABC: FFT
3. ABC: FTF
4. ABC: FTT
5. ABC: TFF
6. ABC: TFT
7. ABC: TTF
8. ABC: TTT

Why issue in-order?

- In-order issue permits us to analyze data flow of program
  - Know which results flow to which subsequent instructions
  - If we issued out-of-order, we would confuse RAW and WAR hazards!

- This idea works perfectly well "in principle" with multiple instructions issued per clock:
  - Need to multi-port "rename table" and be able to rename a sequence of instructions together
  - Need to be able to issue to multiple reservation stations in a single cycle.
  - Need to have 2x number of read ports and 1x number of write ports in register file.

- In-order issue can be serious bottleneck when issuing multiple instructions per clock-cycle

Limits to Multi-Issue Machines

- Multi-issue: simple matter of accounting
  - Must do dataflow analysis across multiple instructions simultaneously
  - Rename table updated as if instructions happened serially!
- Inherent limitations of ILP
  - 1 branch in 5: How to keep a 5-way superscalar busy?
  - Latencies of units: many operations must be scheduled
  - Need about Pipeline Depth x No. Functional Units of independent instructions to keep fully busy
  - Increase ports to Register File
    - VLIW example needs 7 read and 3 write for Int. Reg. & 5 read and 3 write for FP reg
  - Increase ports to memory
  - Current state of the art: Many hardware structures (such as issue/rename logic) has delay proportional to square of number of instructions issued/cycle

Limits to ILP

Conflicting studies of amount

- Benchmarks (vectorized Fortran FP vs. integer C programs)
- Hardware sophistication
- Compiler sophistication

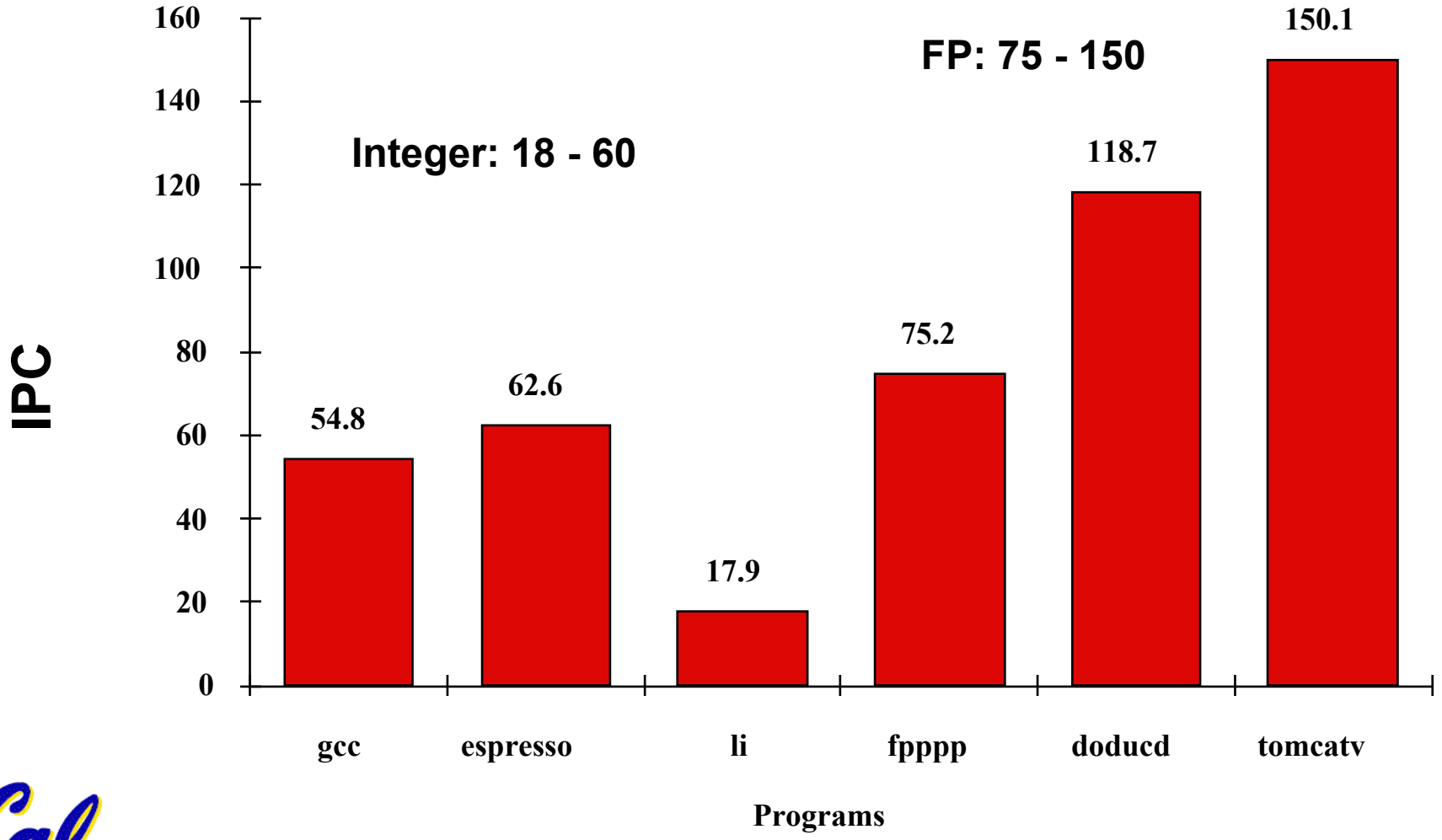
Initial HW Model here; MIPS compilers.

Assumptions for ideal/perfect machine to start:

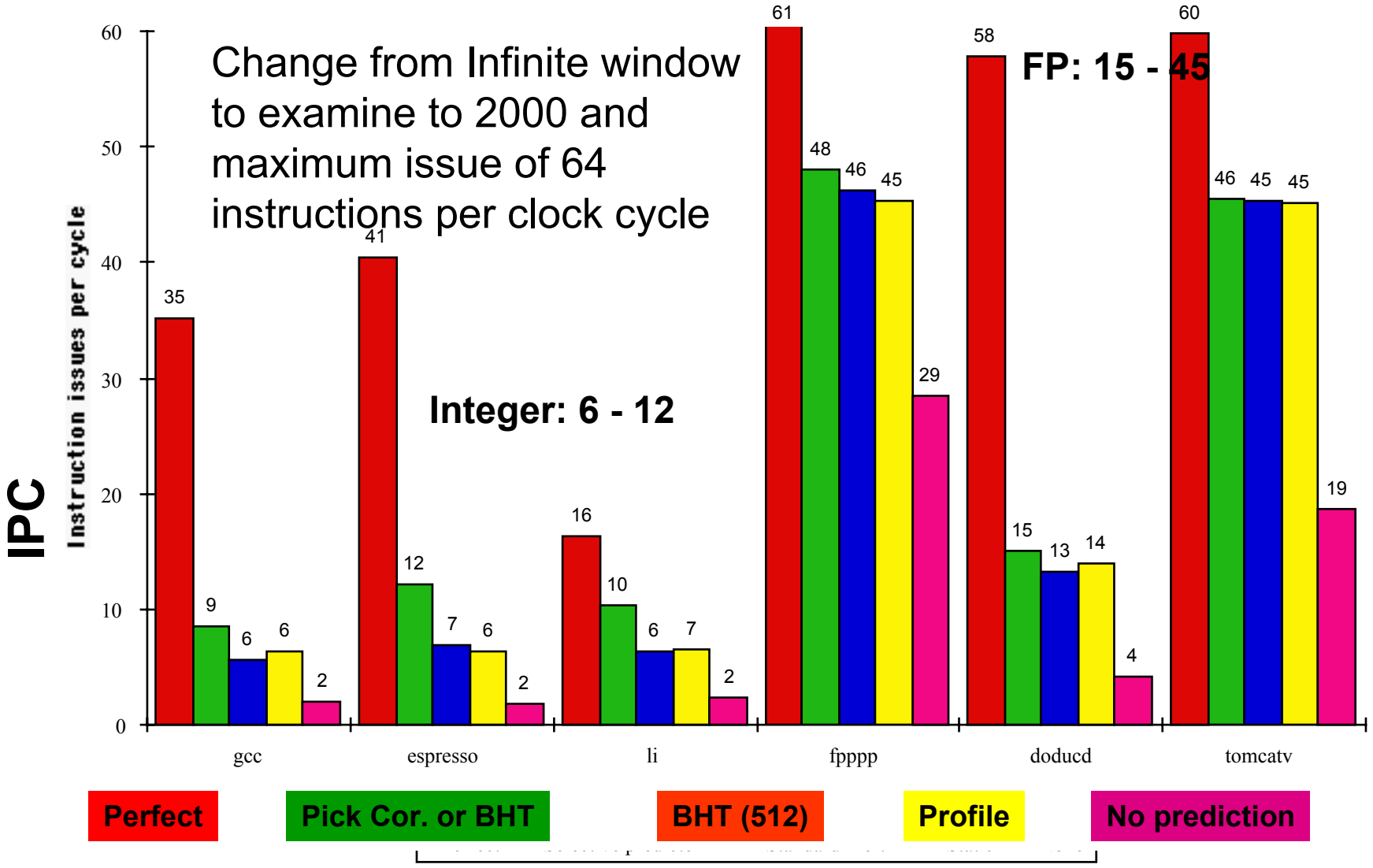
1. *Register renaming*–infinite virtual registers and all WAW & WAR hazards are avoided
2. *Branch prediction*–perfect; no mispredictions
3. *Jump prediction*–all jumps perfectly predicted => machine with perfect speculation & an unbounded buffer of instructions available
4. *Memory-address alias analysis*–addresses are known & a store can be moved before a load provided addresses not equal

One cycle latency for all instructions; unlimited number of instructions issued per clock cycle

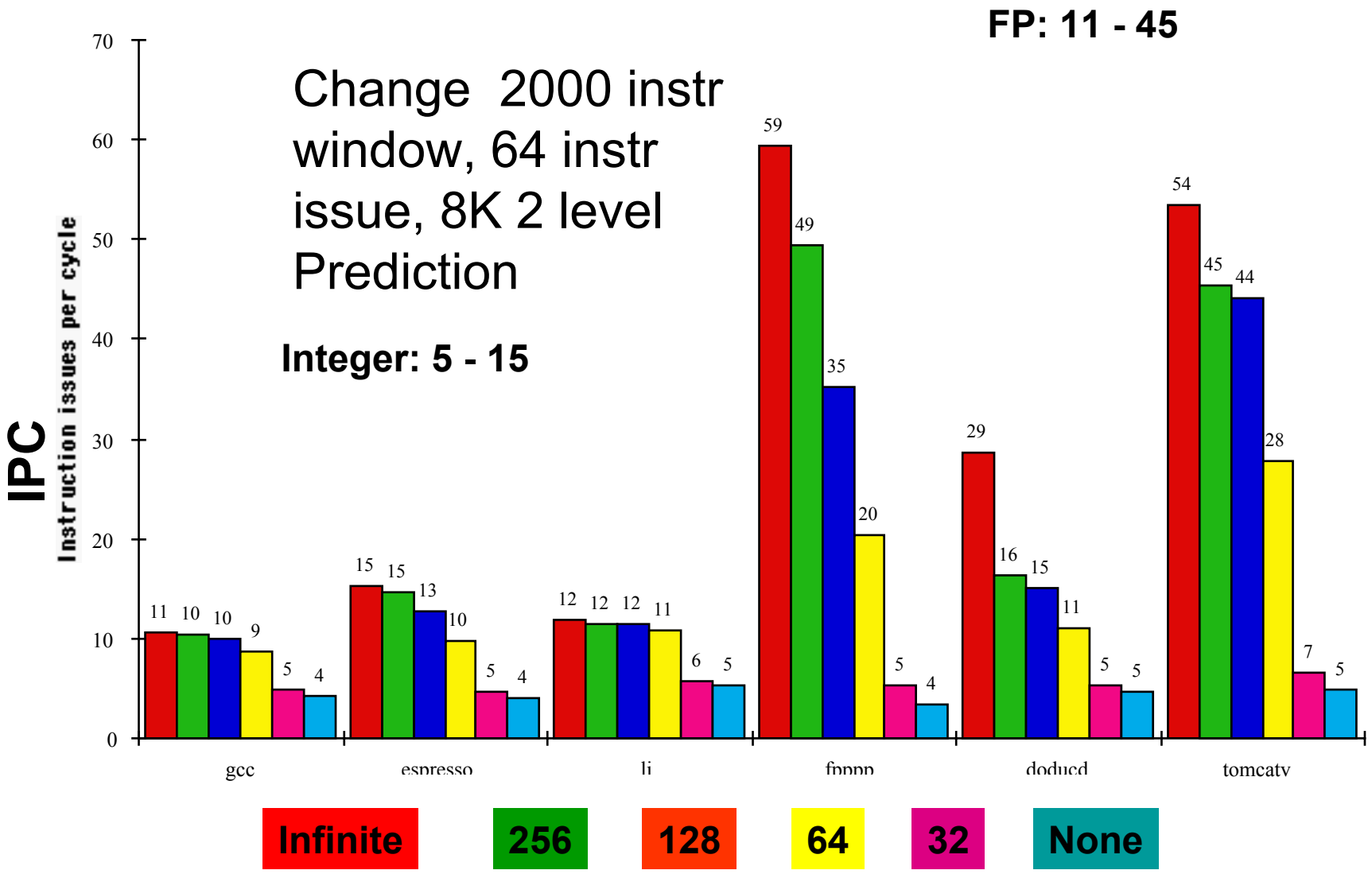
Upper Limit to ILP: Ideal Machine



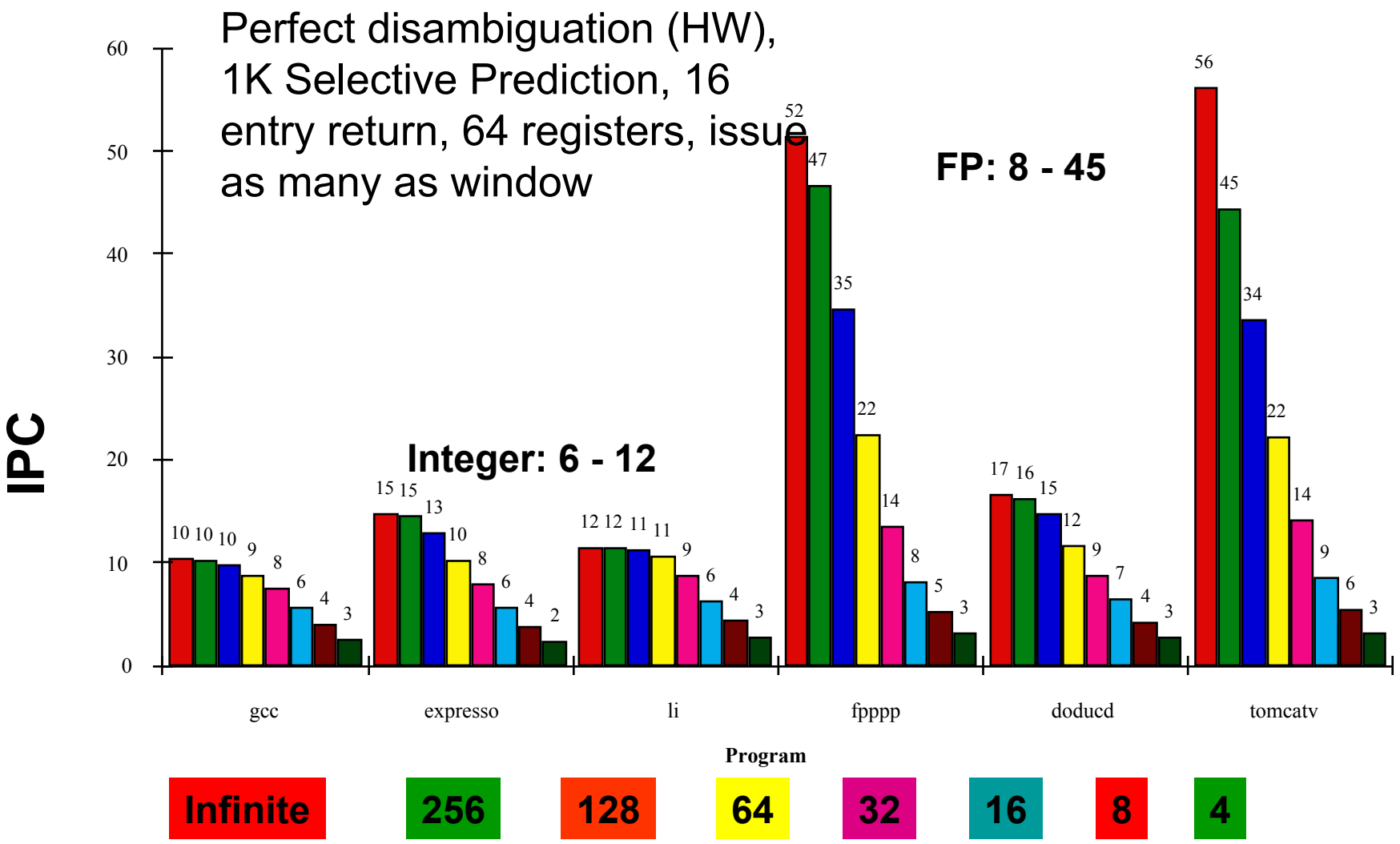
More Realistic HW: Branch Impact



More Realistic HW: Register Impact (rename regs)



Realistic HW for '9X: Window Impact



Summary #1/2

- Reservations stations: renaming to larger set of registers + buffering source operands
  - Prevents registers as bottleneck
  - Avoids WAR, WAW hazards of Scoreboard
  - Allows loop unrolling in HW
- Not limited to basic blocks (integer units gets ahead, beyond branches)
  - Dynamic hardware schemes can unroll loops dynamically in hardware
  - Dependent on renaming mechanism to remove WAR and WAW hazards
- Helps cache misses as well

Summary #2/2

- Reorder Buffer:
  - Provides generic mechanism for “undoing” computation
  - Instructions placed into Reorder buffer in *issue order*
  - Instructions exit in same order – providing *in-order-commit*
  - Trick: Don’t want to be canceling computation too often!
- Branch prediction important to good performance
  - Depends on ability to cancel computation (Reorder Buffer)
- Explicit Renaming: more physical registers than ISA.
  - Separates *renaming* from *scheduling*
    - Opens up lots of options for resolving RAW hazards
  - Rename table: tracks current association between architectural registers and physical registers
  - Potentially complicated rename table management
- Parallelism hard to get from real hardware beyond today