Adding Speculation to Tomasulo

- Must separate execution from allowing instruction to finish or "commit"
- This additional step called instruction commit
- When an instruction is no longer speculative, allow it to update the register file or memory
- Requires additional set of buffers to hold results of instructions that have finished execution but have not committed
- This reorder buffer (ROB) is also used to pass results among instructions that may be speculated

Reorder Buffer (ROB)

- In Tomasulo's algorithm, once an instruction writes its result, any subsequently issued instructions will find result in the register file
- With speculation, the register file is not updated until the instruction commits
  - (we know definitively that the instruction should execute)
- Thus, the ROB supplies operands in interval between completion of instruction execution and instruction commit
  - ROB is a source of operands for instructions, just as reservation stations (RS) provide operands in Tomasulo's algorithm
  - ROB extends architectured registers like RS

Reorder Buffer Entry

Each entry in the ROB contains four fields:

1. Instruction type
   - a branch (has no destination result), a store (has a memory address destination), or a register operation (ALU operation or load, which has register destinations)
2. Destination
   - Register number (for loads and ALU operations) or memory address (for stores)
     where the instruction result should be written
3. Value
   - Value of instruction result until the instruction commits
4. Ready
   - Indicates that instruction has completed execution, and the value is ready

Reorder Buffer operation

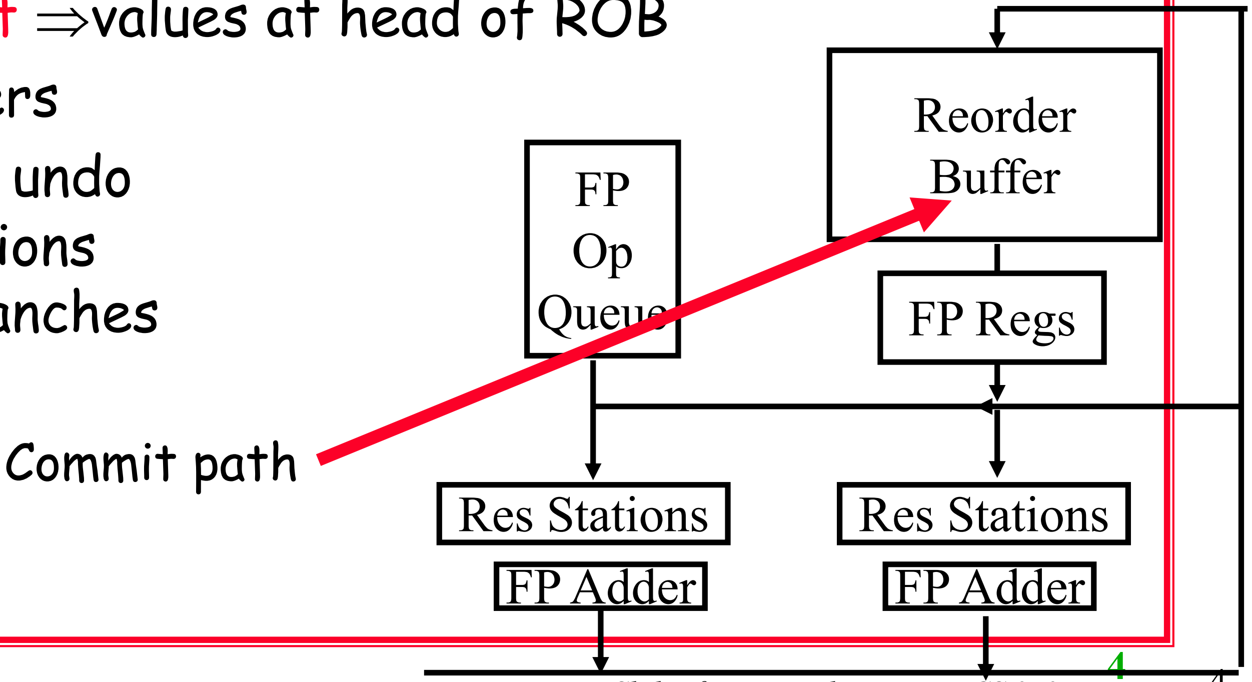
Holds instructions in FIFO order, exactly as issued

When instructions complete, results placed into ROB

- Supplies operands to other instruction between execution complete & commit ⇒ more registers like RS
- Tag results with ROB buffer number instead of reservation station

Instructions commit ⇒values at head of ROB placed in registers

As a result, easy to undo speculated instructions on mispredicted branches or on exceptions



Recall: 4 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 (this stage sometimes called "dispatch")

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 (sometimes called "issue")

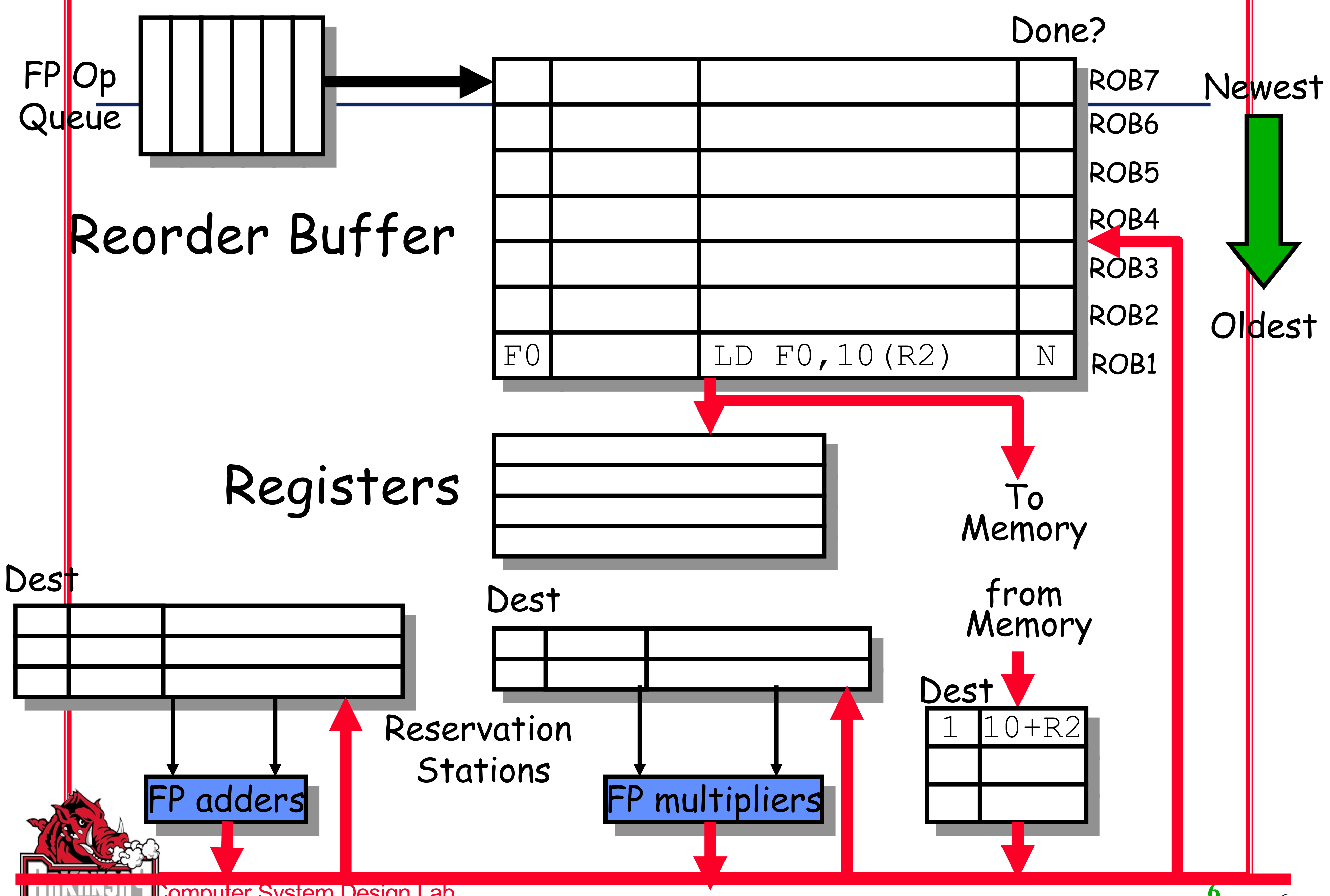
3. **Write result**—finish execution (WB)

   Write on Common Data Bus to all awaiting FUs & reorder buffer; mark reservation station available.

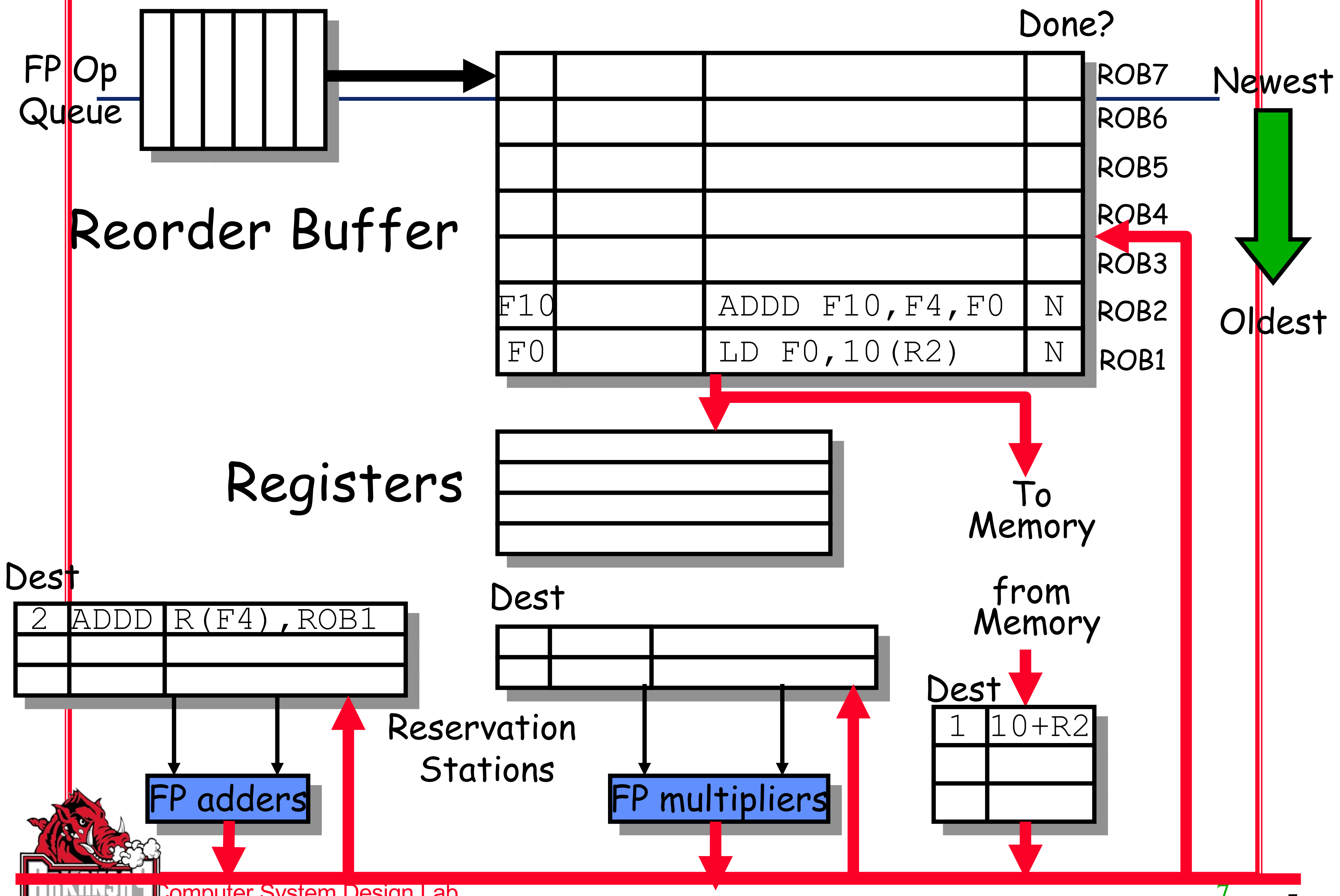
4. **Commit**—update register with reorder result

   When instr. at head of reorder buffer & result present, update register with result (or store to memory) and remove instr from reorder buffer. Mispredicted branch flushes reorder buffer (sometimes called "graduation")

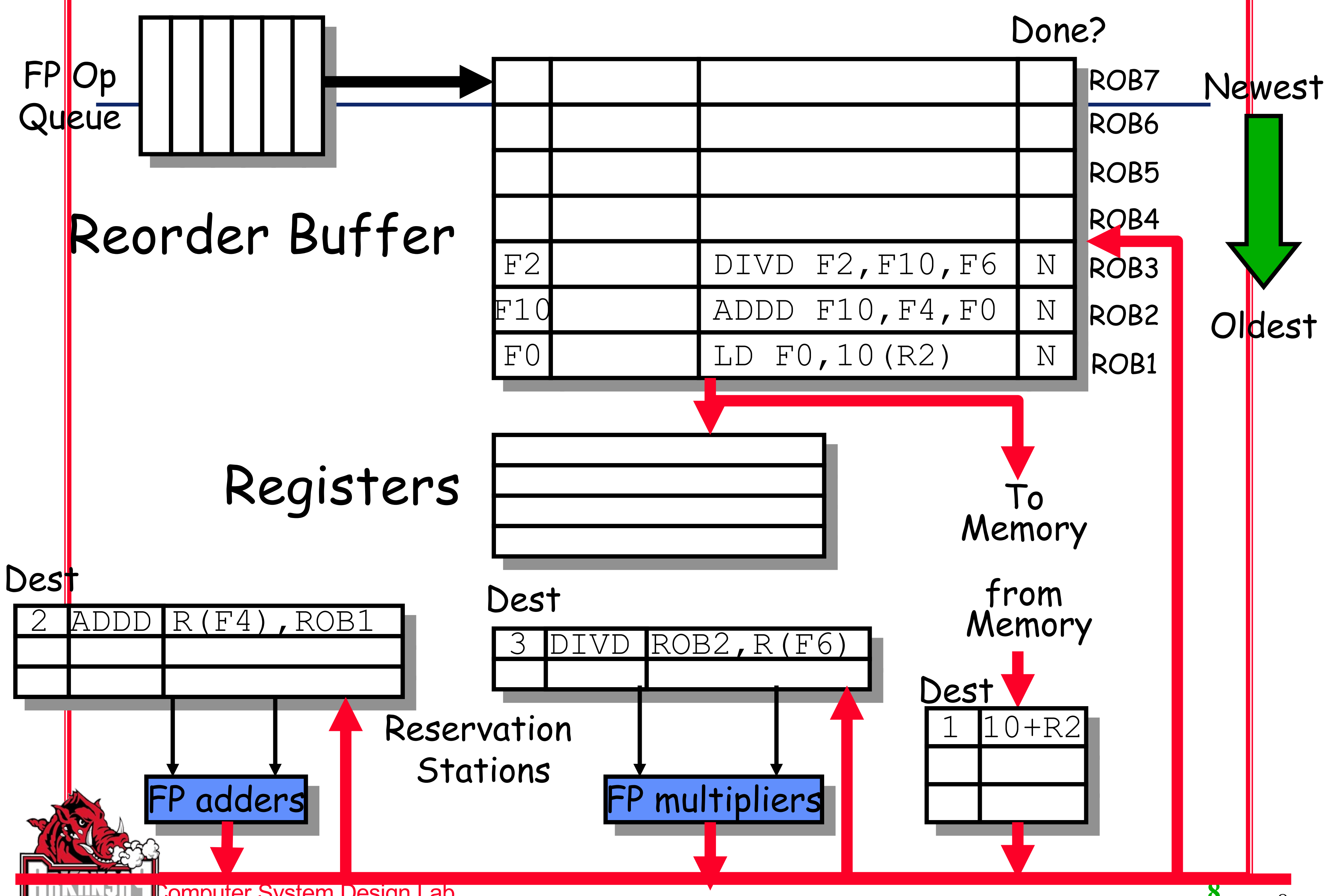
Tomasulo With Reorder buffer:



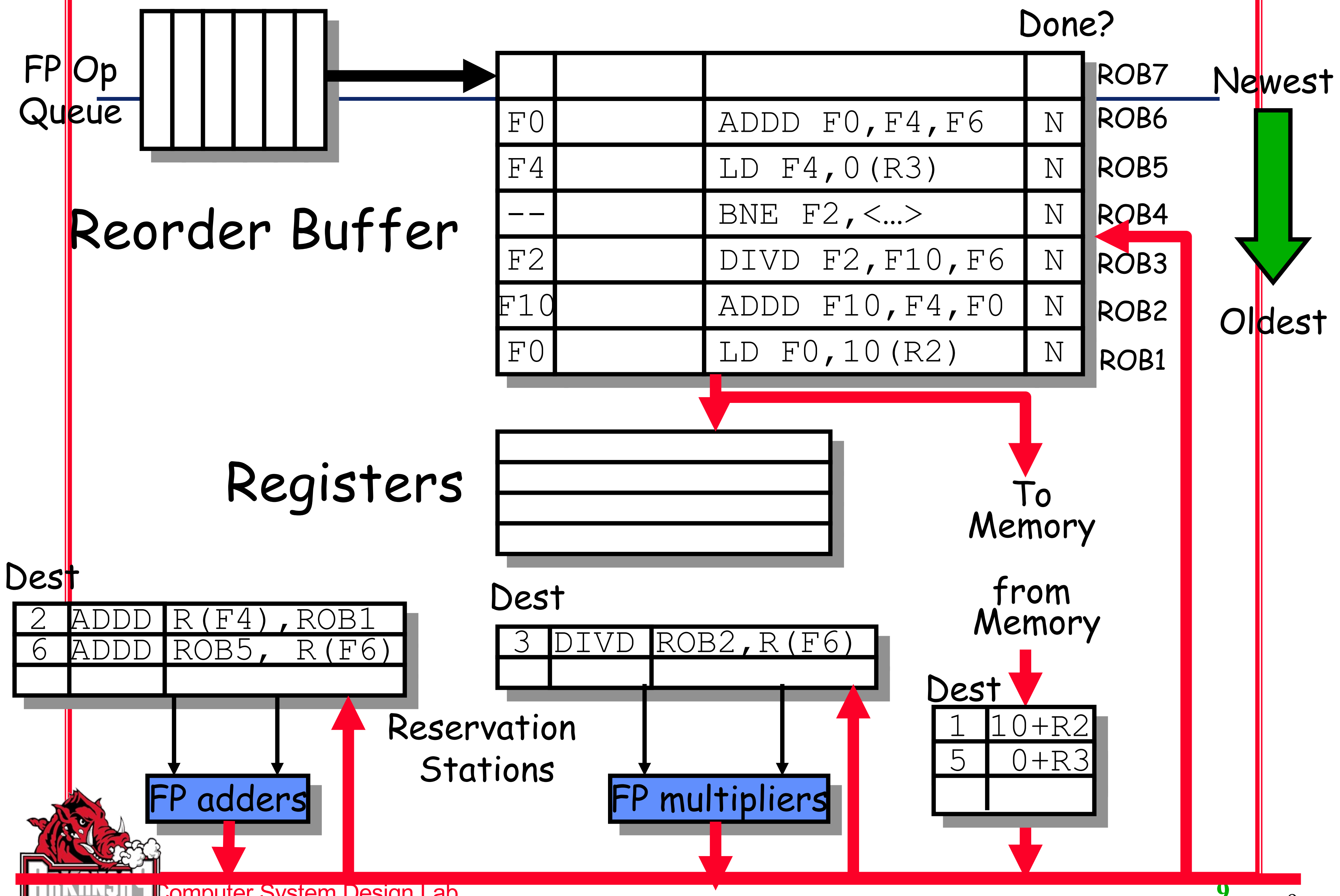
Tomasulo With Reorder buffer:



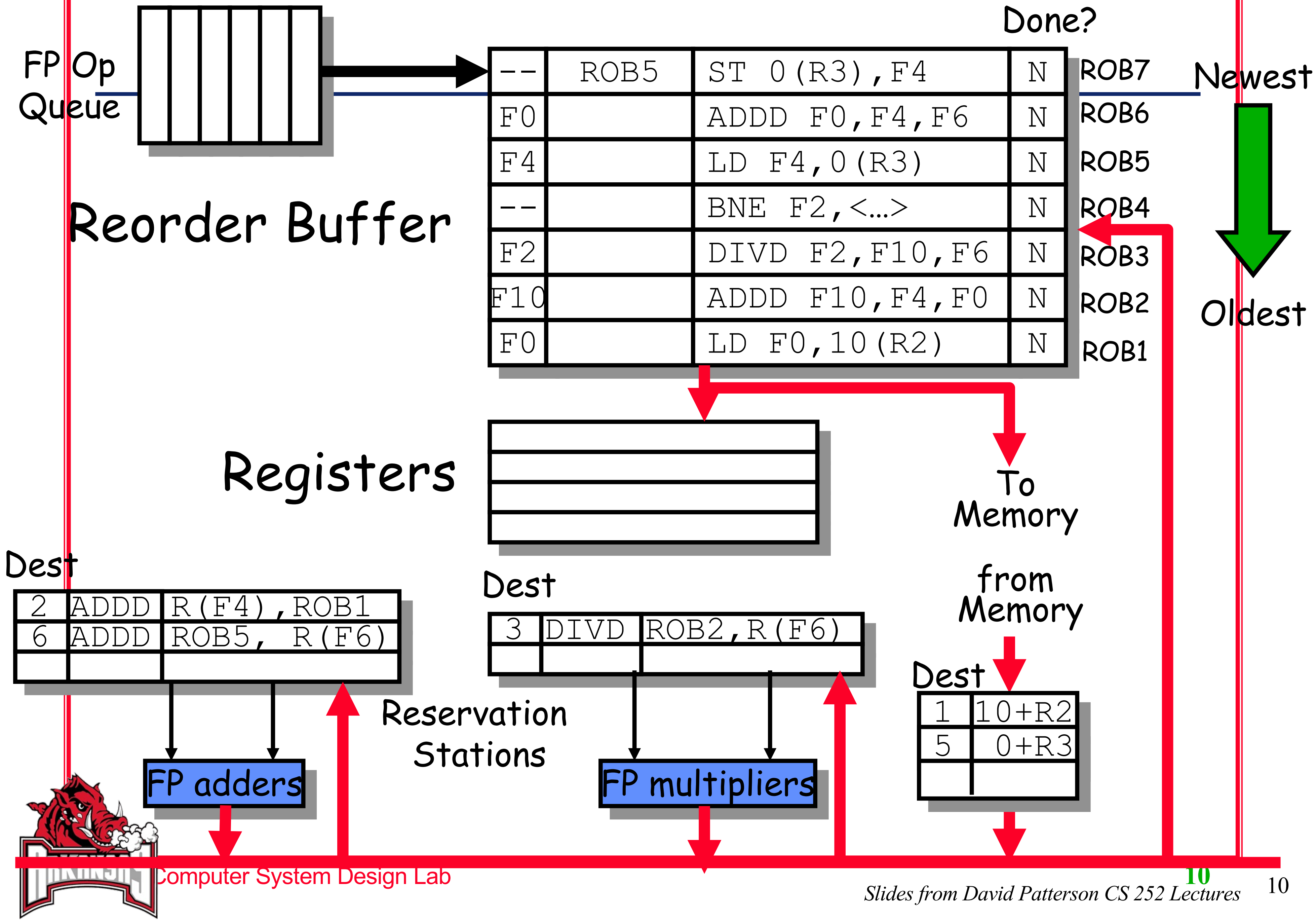
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:



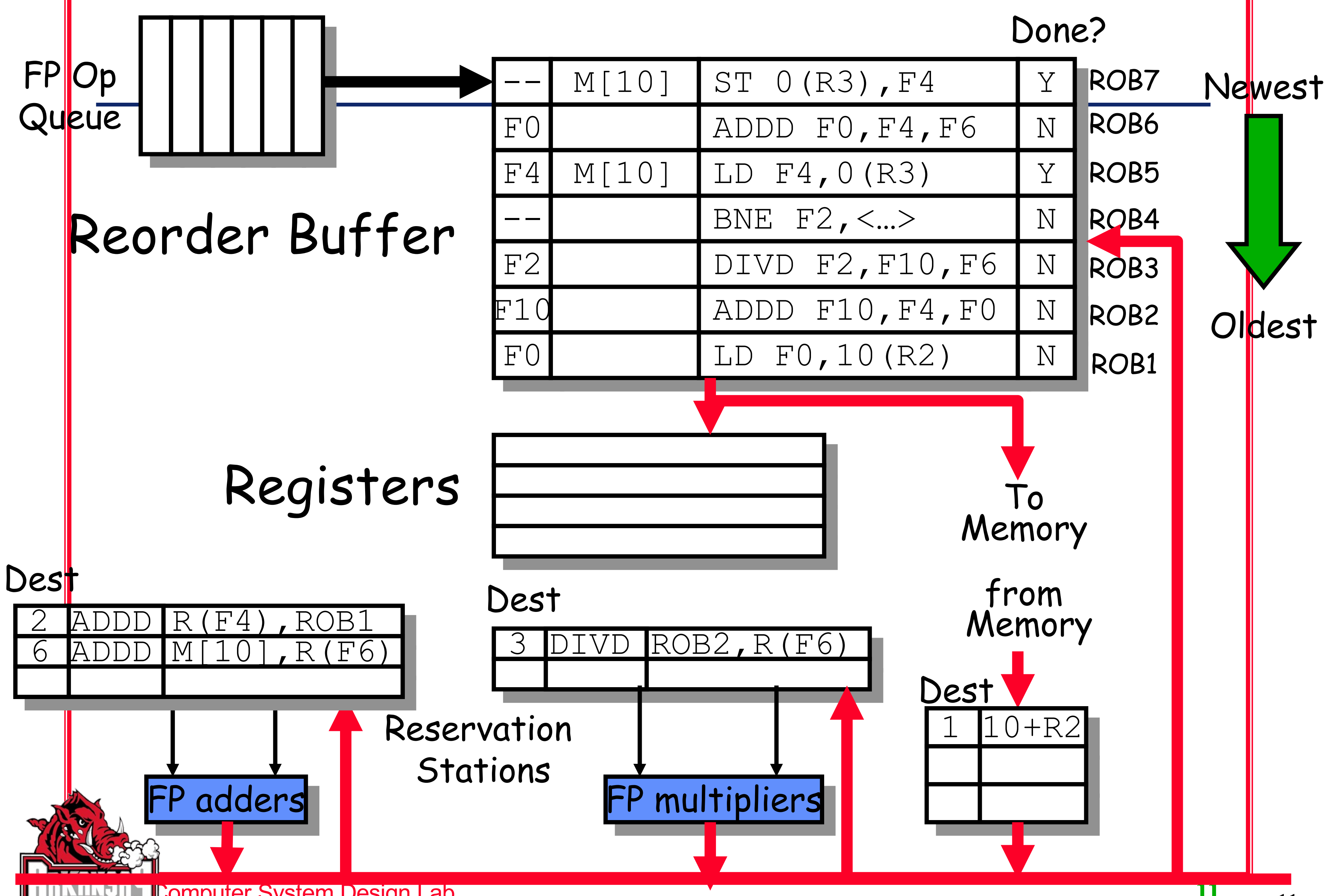
			Done?	
--	ROB5	ST 0(R3),F4	N	ROB7
F0		ADDD F0,F4,F6	N	ROB6
F4		LD F4,0(R3)	N	ROB5
--		BNE F2,<…>	N	ROB4
F2		DIVD F2,F10,F6	N	ROB3
F10		ADDD F10,F4,F0	N	ROB2
F0		LD F0,10(R2)	N	ROB1

Dest		
2	ADDD	R(F4),ROB1
6	ADDD	ROB5, R(F6)

Dest		
3	DIVD	ROB2,R(F6)

Dest	
1	10+R2
5	0+R3

Tomasulo With Reorder buffer:



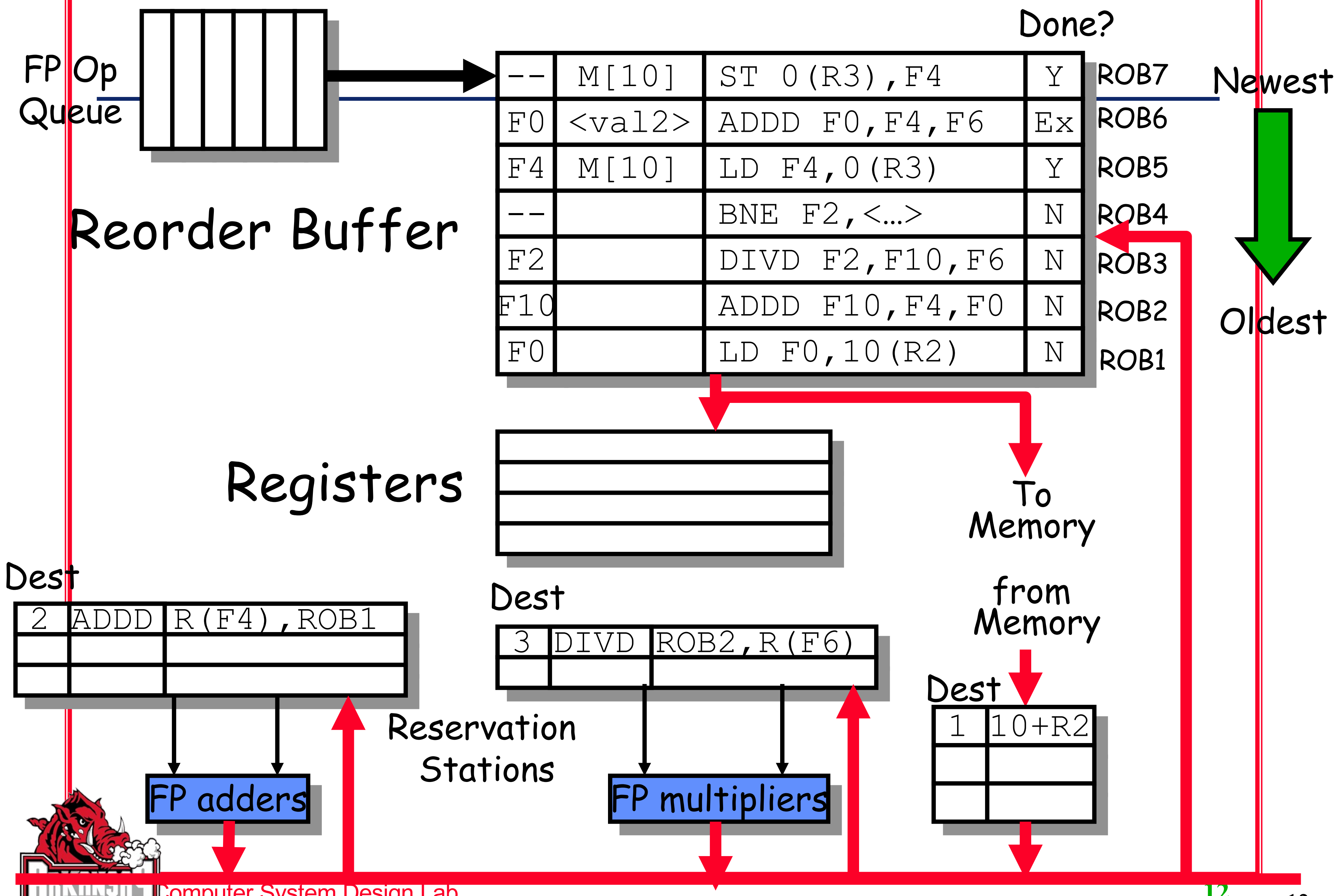
			Done?	
--	M[10]	ST 0(R3),F4	Y	ROB7
F0		ADDD F0,F4,F6	N	ROB6
F4	M[10]	LD F4,0(R3)	Y	ROB5
--		BNE F2,<…>	N	ROB4
F2		DIVD F2,F10,F6	N	ROB3
F10		ADDD F10,F4,F0	N	ROB2
F0		LD F0,10(R2)	N	ROB1

Dest		
2	ADDD	R(F4),ROB1
6	ADDD	M[10],R(F6)

Dest		
3	DIVD	ROB2,R(F6)

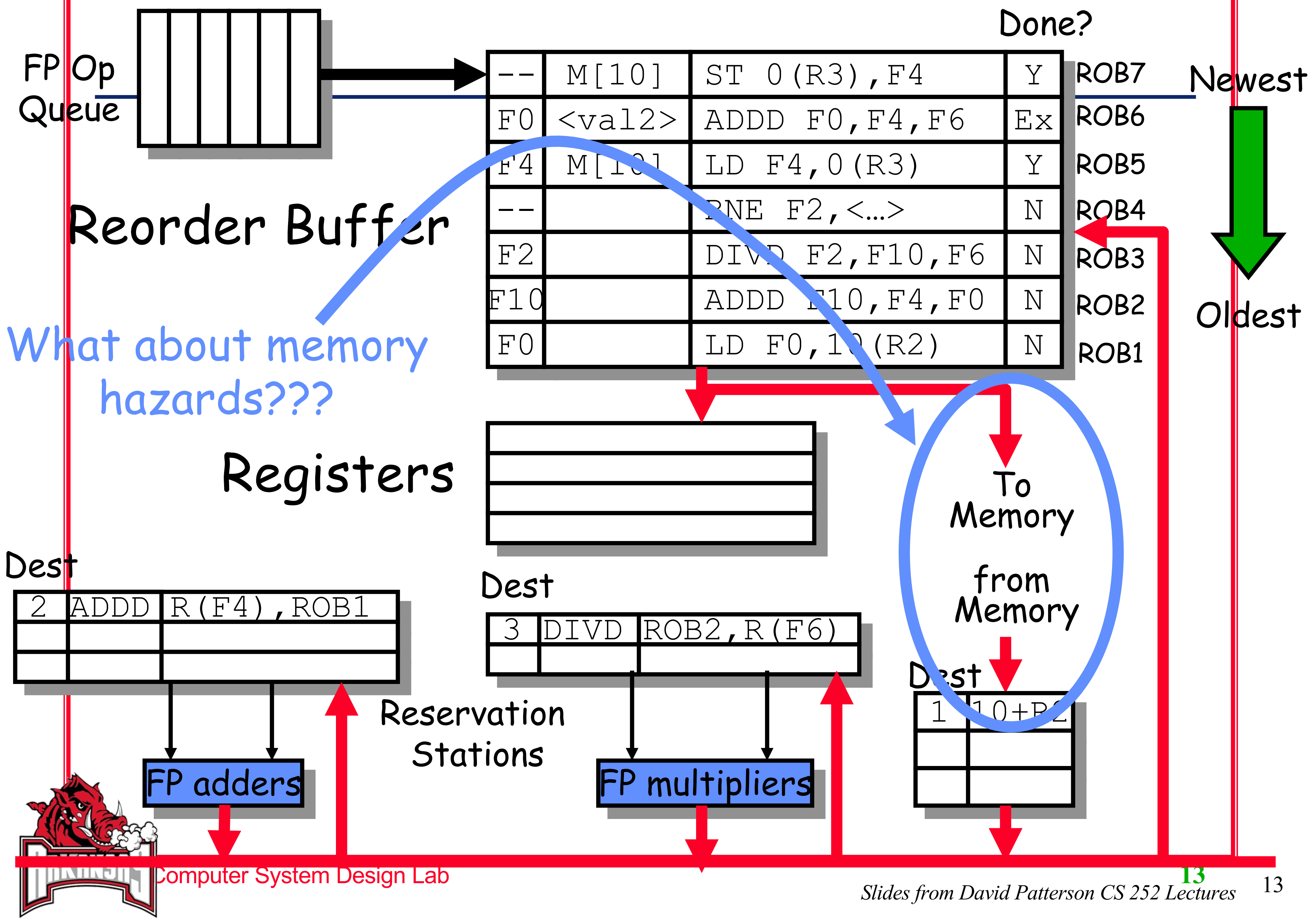
Dest	
1	10+R2

Tomasulo With Reorder buffer:



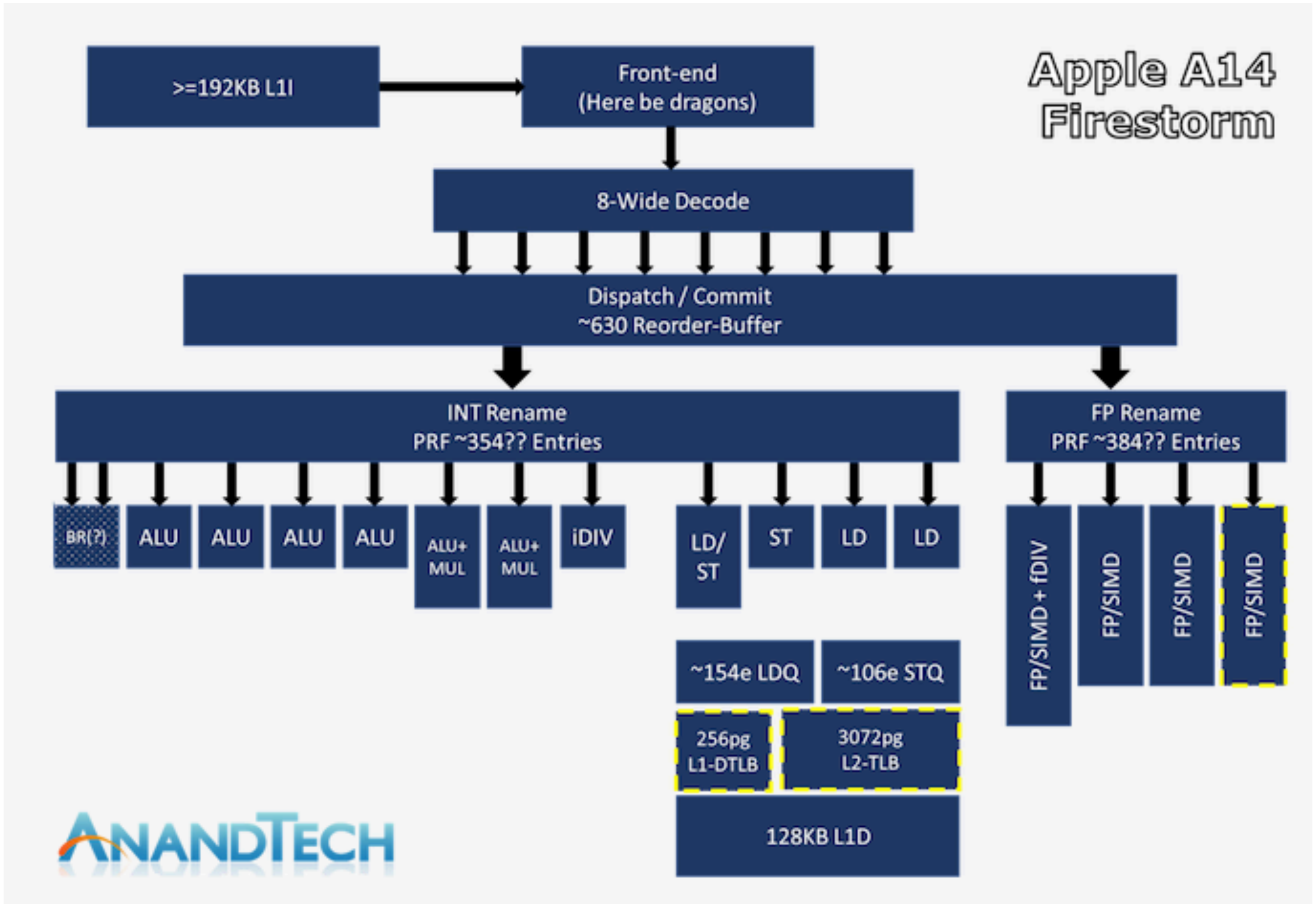
FP Op Queue
Done?
Newest
Oldest
Reorder Buffer
-- | M[10] | ST 0(R3),F4 | Y | ROB7
F0 | <val2> | ADDD F0,F4,F6 | Ex | ROB6
F4 | M[10] | LD F4,0(R3) | Y | ROB5
-- | | BNE F2,<…> | N | ROB4
F2 | | DIVD F2,F10,F6 | N | ROB3
F10 | | ADDD F10,F4,F0 | N | ROB2
F0 | | LD F0,10(R2) | N | ROB1
Registers
To Memory
from Memory
Dest
2 ADDD R(F4),ROB1
Dest
3 DIVD ROB2,R(F6)
Dest
1 10+R2
Reservation Stations
FP adders
FP multipliers

Tomasulo With Reorder buffer:

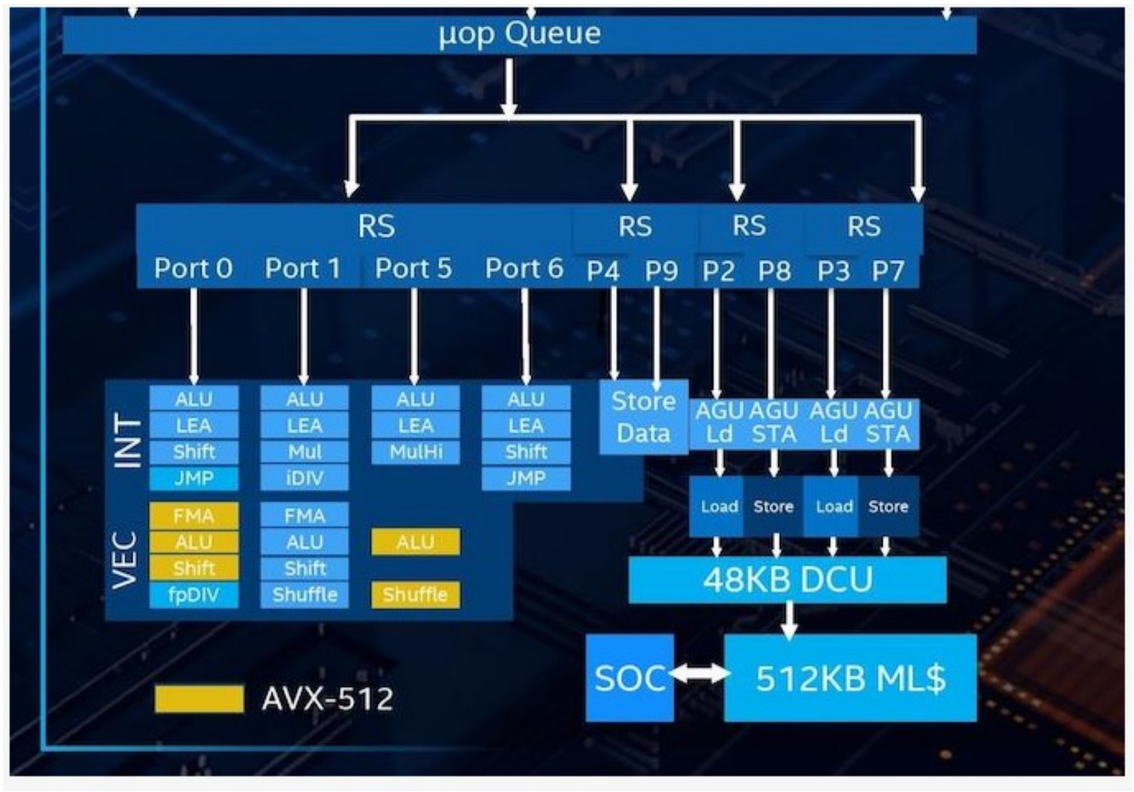


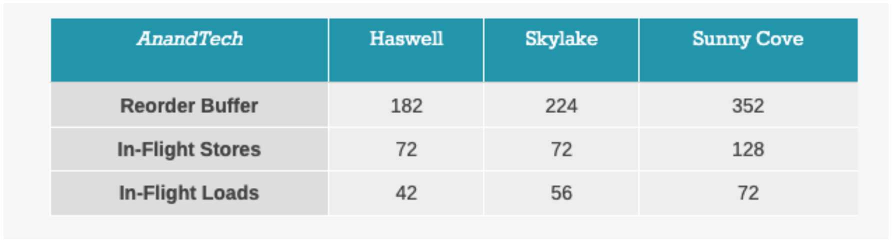
FP Op Queue
Done?
Reorder Buffer
-- | M[10] | ST 0(R3),F4 | Y | ROB7
F0 | <val2> | ADDD F0,F4,F6 | Ex | ROB6
F4 | M[10] | LD F4,0(R3) | Y | ROB5
-- | | BNE F2,<…> | N | ROB4
F2 | | DIVD F2,F10,F6 | N | ROB3
F10 | | ADDD F10,F4,F0 | N | ROB2
F0 | | LD F0,10(R2) | N | ROB1
Newest
Oldest
What about memory hazards???
Registers
To Memory
from Memory
Dest
2 ADDD R(F4),ROB1
Dest
3 DIVD ROB2,R(F6)
Dest
1 10+R2
Reservation Stations
FP adders
FP multipliers

How Big Is a Real Reorder Buffer?



Intel Sunny Cove





AnandTech	Haswell	Skylake	Sunny Cove
Reorder Buffer	182	224	352
In-Flight Stores	72	72	128
In-Flight Loads	42	56	72

AMD Ryzen “Zen 3”

“ZEN 3” OVERVIEW

2 THREADS PER CORE (SMT)

STATE-OF-THE-ART BRANCH PREDICTOR

CACHES
- I-cache 32k, 8-way
- Op-cache, 4K instructions
- D-cache 32k, 8-way
- L2 cache 512k, 8-way

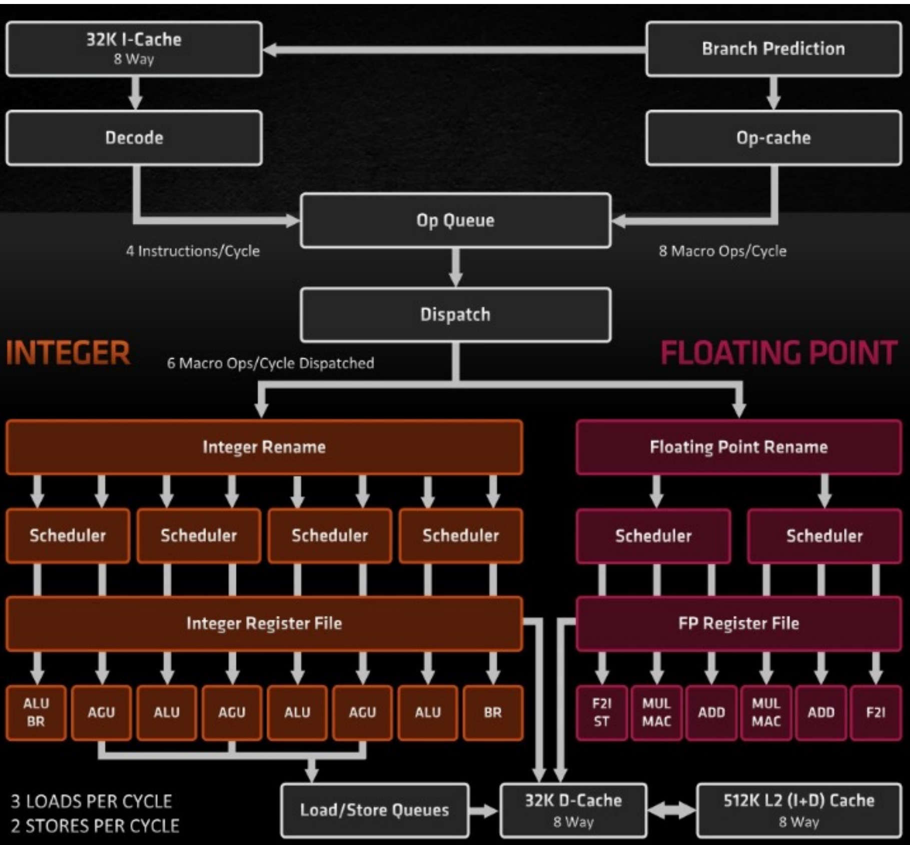
DECODE
- 4 instructions / cycle from decode or 8 ops from Op-cache
- 6 ops / cycle dispatched to Integer or Floating Point

EXECUTION CAPABILITIES
- 4 integer units
- Dedicated branch and store data units
- 3 address generations per cycle

3 MEMORY OPS PER CYCLE
- Max 2 can be stores

TLBs
- L1 64 entries I & D, all page sizes
- L2 512 I, 2K D, everything but 1G

TWO 256-BIT FP MULTIPLY ACCUMULATE / CYCLE



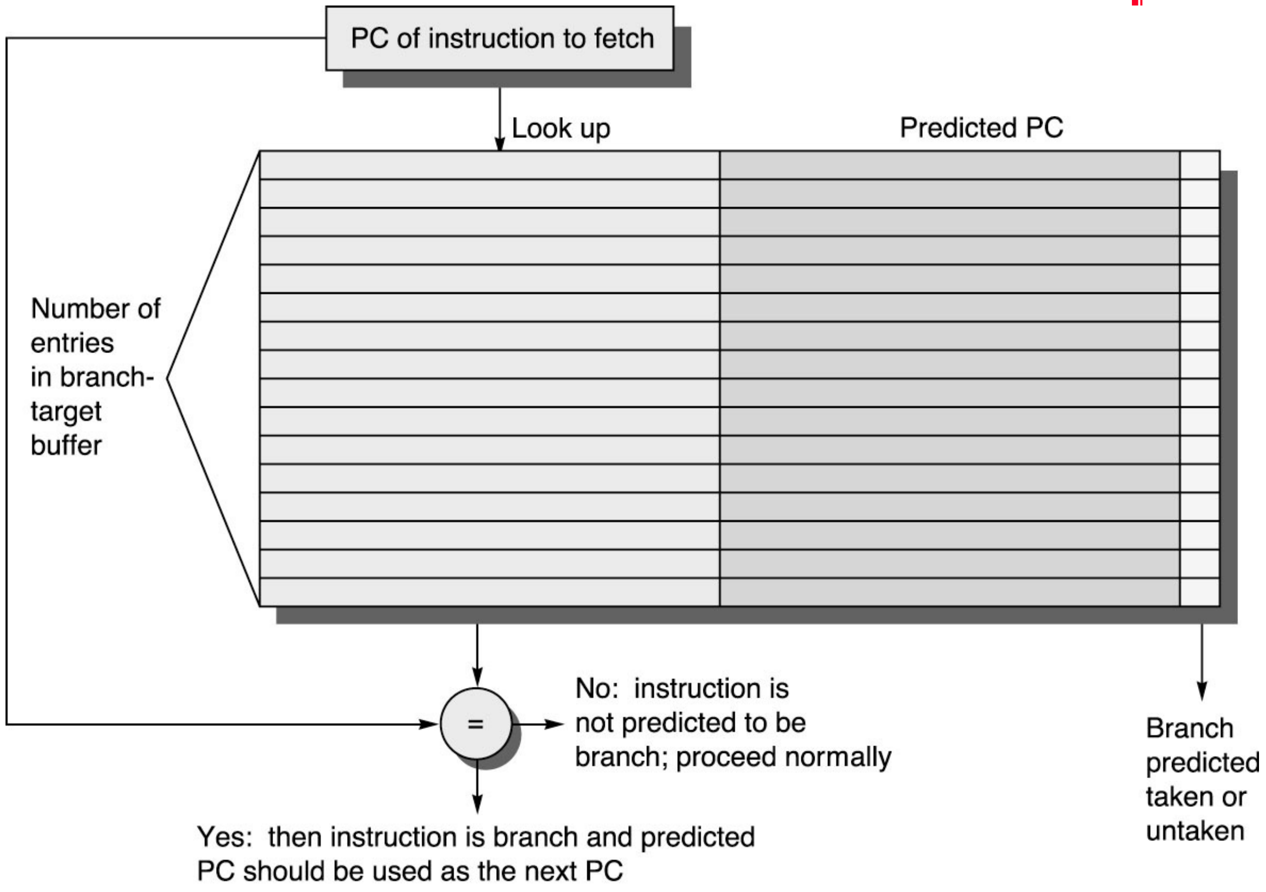
5 WHERE GAMING BEGINS: AMD RYZEN | AMD CONFIDENTIAL

AMD

Increasing Instruction Fetch Bandwidth

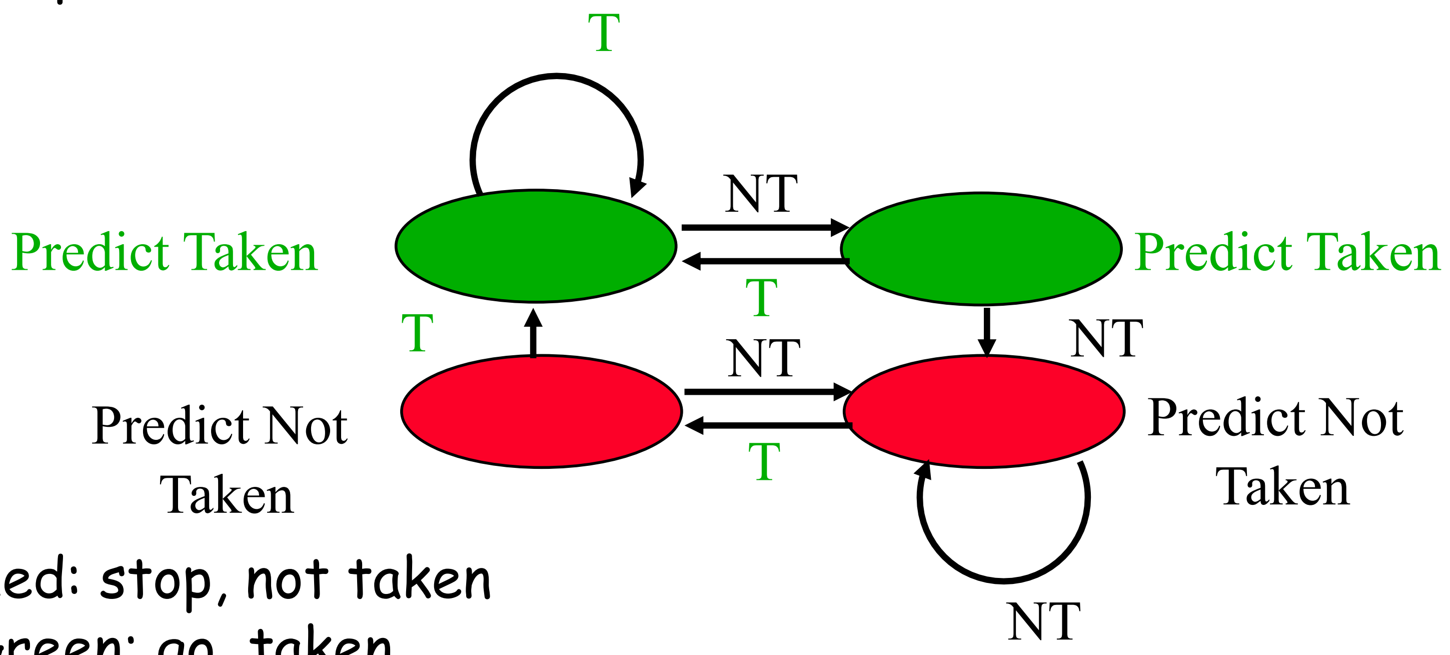
- Predicts next instruct address, sends it out *before* decoding instruction
- PC of branch sent to BTB
- When match is found, Predicted PC is returned
- If branch predicted taken, instruction fetch continues at Predicted PC

Branch Target Buffer (BTB)


PC of instruction to fetch
Look up
Predicted PC
Number of entries in branch-target buffer
=
No: instruction is not predicted to be branch; proceed normally
Yes: then instruction is branch and predicted PC should be used as the next PC
Branch predicted taken or untaken

2-bit Predictor "Smooths Out Edges" For Nested Loops

- Solution: 2-bit scheme where change prediction only if get misprediction *twice*



- Red: stop, not taken
- Green: go, taken
- Adds *hysteresis* to decision making process

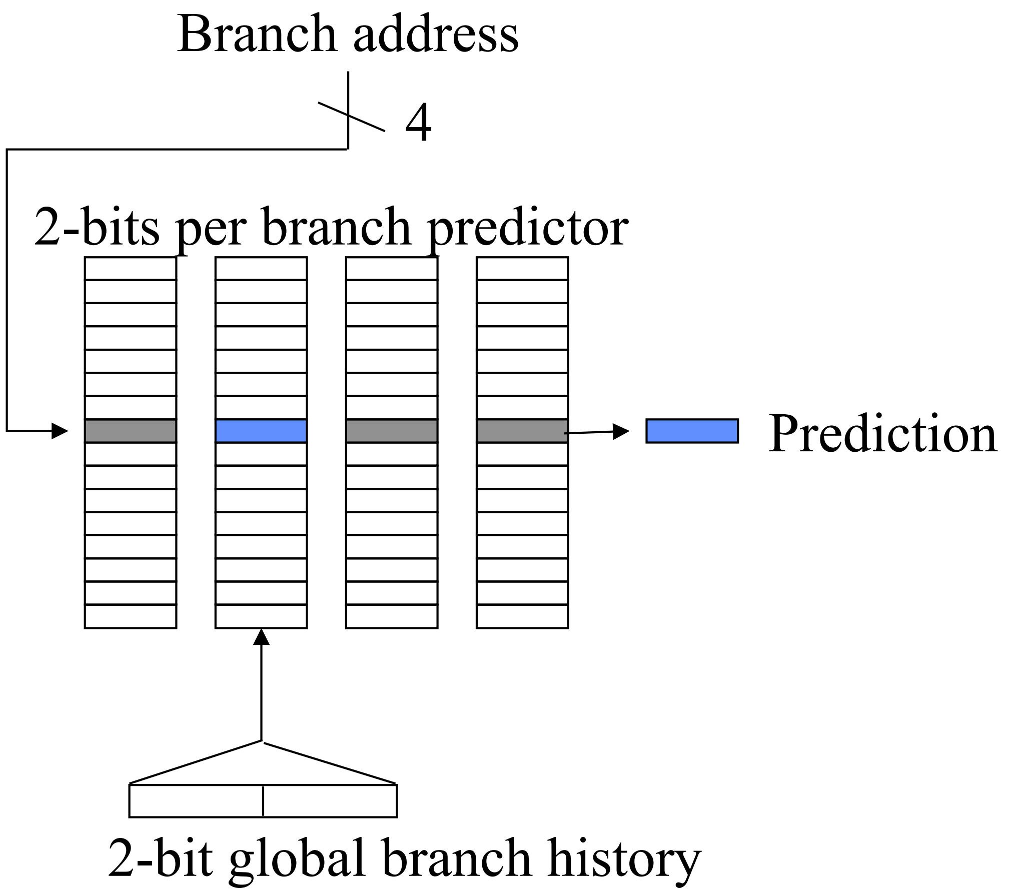
Correlated Branch Prediction

- Idea: record *m* most recently executed branches as taken or not taken, and use that pattern to select the proper *n*-bit branch history table
- In general, (*m*,*n*) predictor means record last *m* branches to select between 2^m history tables, each with *n*-bit counters
  - Thus, old 2-bit BHT is a (0,2) predictor
- Global Branch History: *m*-bit shift register keeping T/NT status of last *m* branches.
- Each entry in table has *m* *n*-bit predictors.

Correlating Branches

(2,2) predictor

- Behavior of recent branches selects between four predictions of next branch, updating just that prediction


Branch address
4
2-bits per branch predictor
Prediction
2-bit global branch history