Out-of-Order and Advanced Topics

Computer Architecture
ECE 6913

Brandon Reagen

NYU TANDON SCHOOL OF ENGINEERING

Announcements

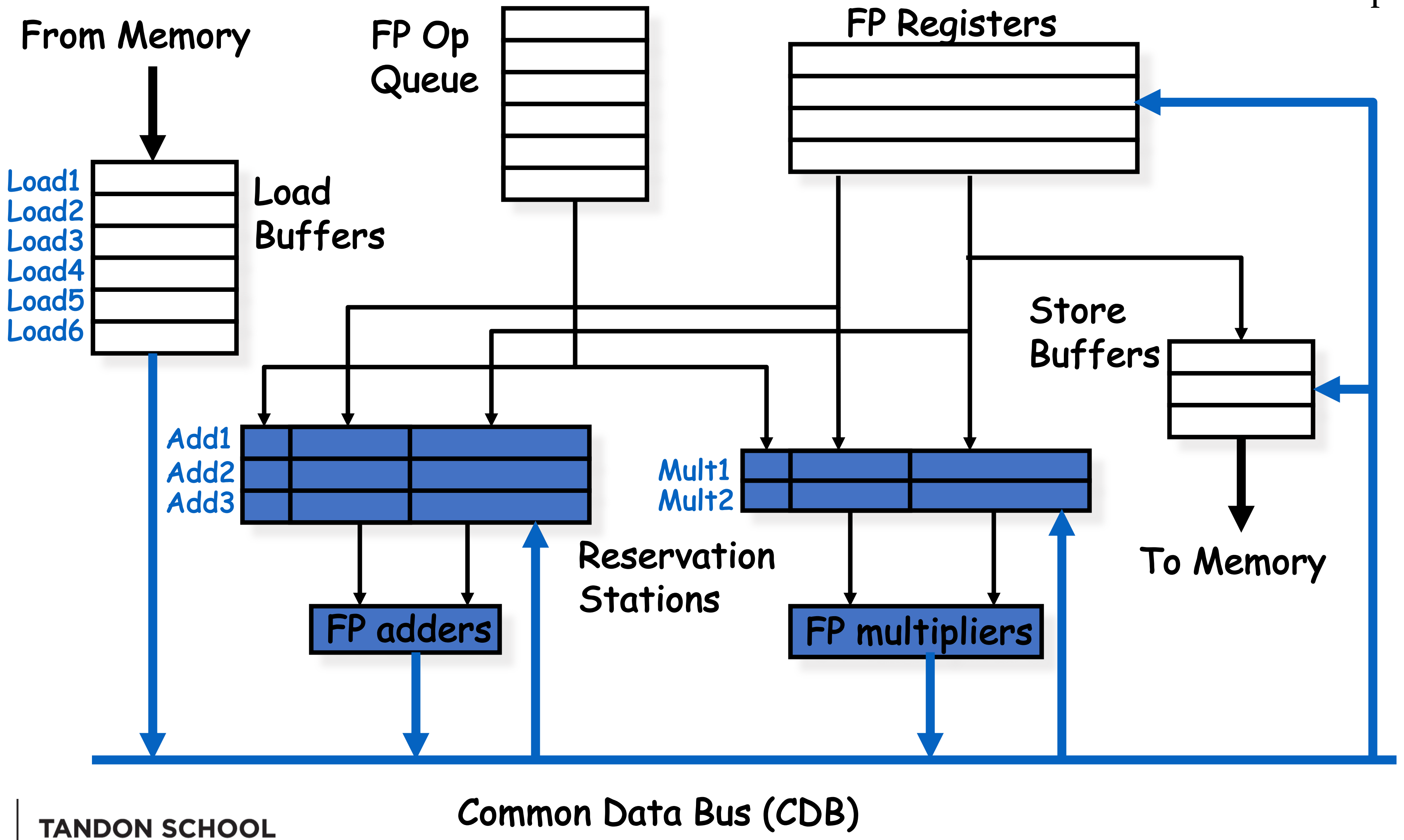
1) Labs
    1) Lab3 due Nov 24th midnight
        1) Just get it done so you can enjoy a break ☺
    2) Lab4 will go out ~11/30 and be due mid December

2) Today
    1) Finish up OoO
    2) Lightning round of advanced topics
    3) Short class!
        1) Homework: Use the time to watch Patterson-Hennessey Turing award lecture https://www.acm.org/hennessy-patterson-turing-lecture

3) Today: Dealing with exception in OoO

Tomasulo's Organization

What's the issue with out-of-order completion?



NYU TANDON SCHOOL OF ENGINEERING

Two event types

Interrupts

- Caused by external events

  Network, keyboard, Disk I/O, OS

- Asynchronous to program execution

  Can be disabled for some time

- May be handled between instructions
- Suspend or resume program once resolved

Exceptions

- Caused by internal evenets
  - Exceptional conditions (overflow, when?)

    page faults (when?)
  - Synchronous to program execution
  - Fixed by exception handler
  - Program can continue or abort

Exceptions in MIPS pipeline

Stage	*Possible exceptions*
IF	Page fault on instruction fetch; misaligned memory access; memory-protection violation
ID	Undefined or illegal opcode
EX	Arithmetic exception
MEM	Page fault on data fetch; misaligned memory access; memory-protection violation; memory error

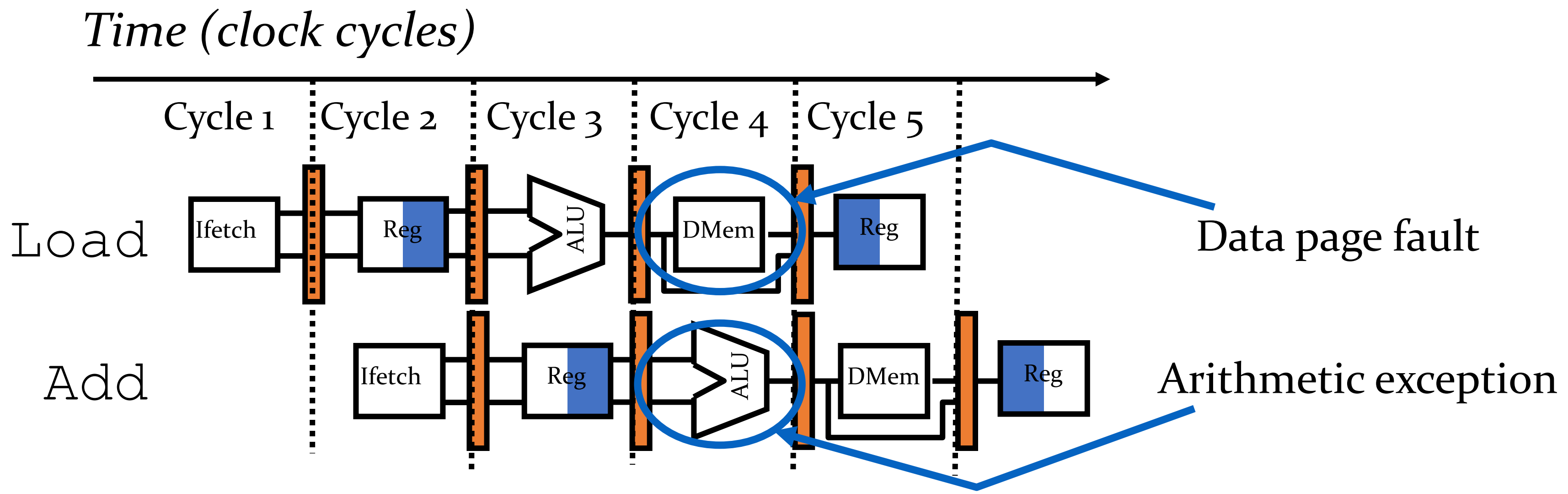
How do we stop the pipeline? How do we restart it?

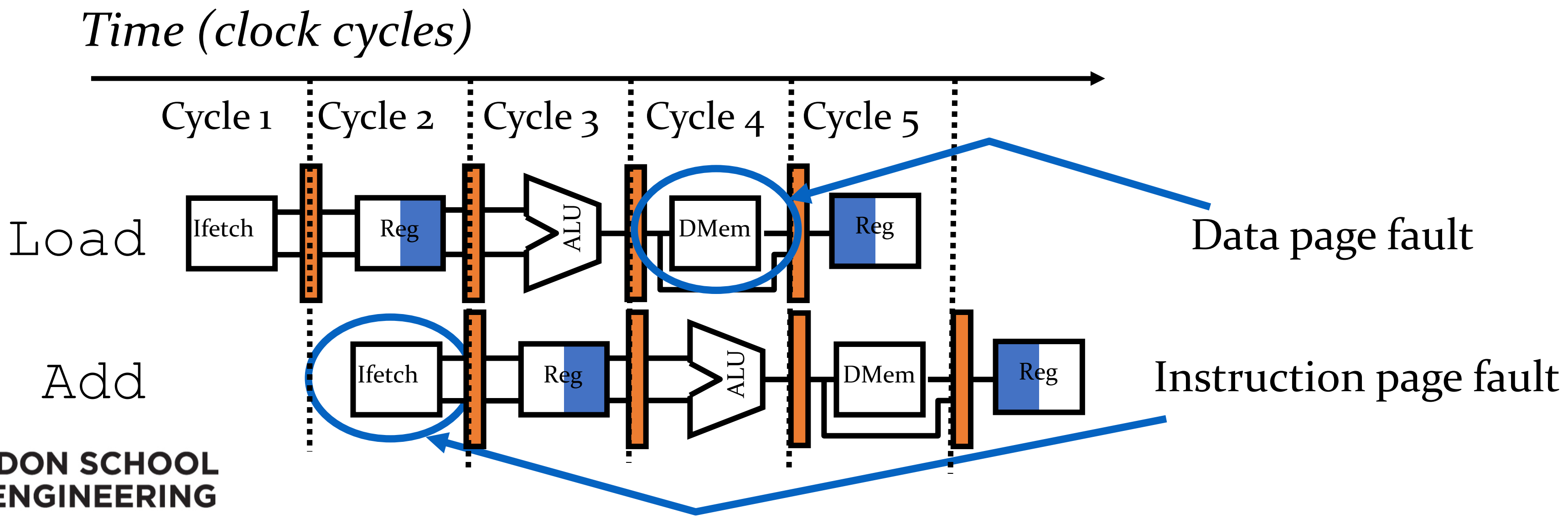
Do we interrupt immediately or wait?

5 instructions, executing in 5 different pipeline stages!

- Who caused the interrupt?

Multiple exceptions





NYU TANDON SCHOOL OF ENGINEERING

Precise Exceptions/Interrupts

Architectural state should be consistent when
exception/interrupt is ready to be handled

Precise: the outcome should be exactly the same as if instruction executed on non-pipelined (non-OOO) hardware

1. Instructions completed before exception/interrupt should be retired
2. Instructions after exception/interrupt should not retire

Retire == commit:
execution completes <u>and</u> architectural state is updated

Why is this something we need?

1) Software debugging

   You need to know what actually happened

2) Need to recover from exceptions

   Remember VM lecture, what happens after after fault resolved

3) Make it easy to restart processes

   After you come back from context switch, where to start?

Precise interrupts and speculation

Speculation: guess and check

Important for branch prediction:

- Need to "take our best shot" at predicting branch direction

If we speculate and are wrong, need to back up and
restart execution to point at which we predicted incorrectly:

- This is exactly the same as precise exceptions!

Technique for both precise interrupts/exceptions and speculation:

in-order completion or commit

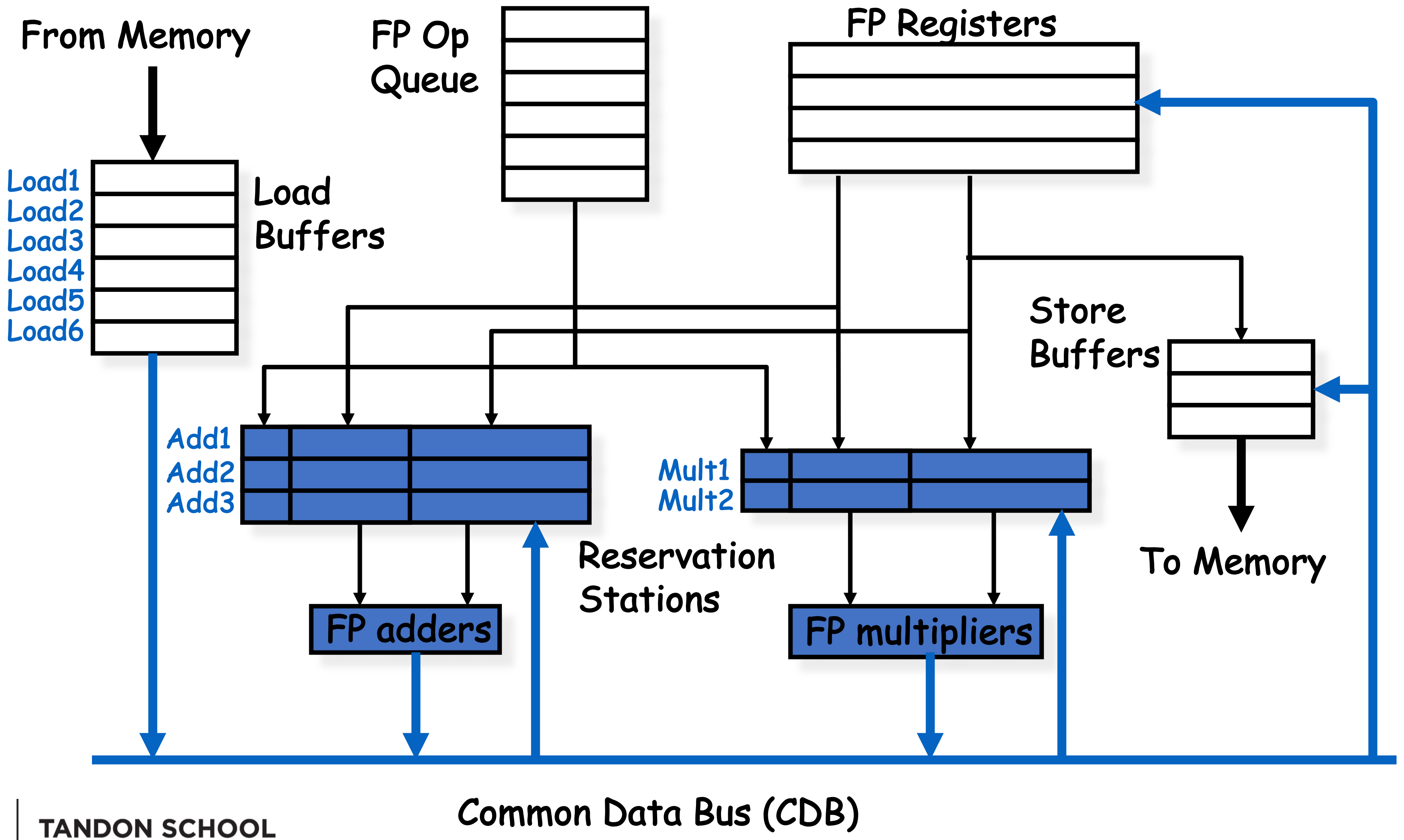
Handling exceptions

Exceptions are handled by **not recognizing** the exception until instruction that caused it is ready to commit in ROB

- If a speculated instruction raises an exception, the exception is recorded in the ROB
- This is why reorder buffers in all new processors
- Ensures all previous instructions commit, no later ones will.

Key idea: separate the completion of execution
from the update to architectural state (commit)

Tomasulo's Organization: no speculation without precise interrupts

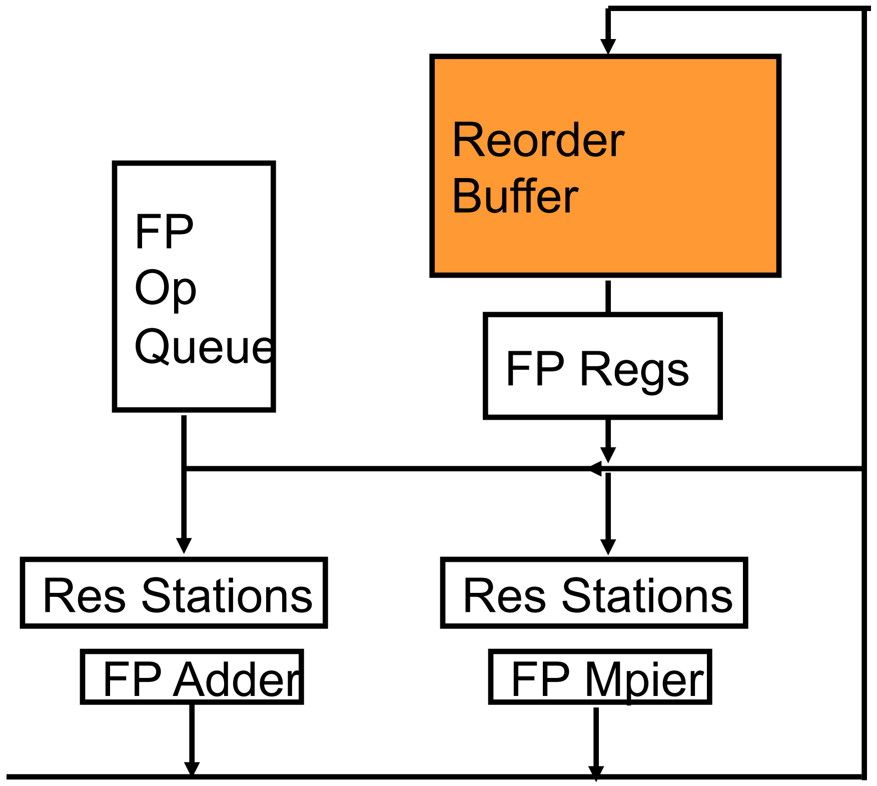


NYU TANDON SCHOOL OF ENGINEERING

ReOrder Buffer (ROB): HW support for precise interrupts

ROB= In order buffer for results of uncommitted instructions

- An instruction commits when it completes its execution and all predecessors have committed
- Once instruction commits, result is put into register
  - Therefore, easy to undo speculated instructions on mispredicted branches or exceptions
- Supplies operands between execution complete & commit



NYU TANDON SCHOOL OF ENGINEERING

In other words..

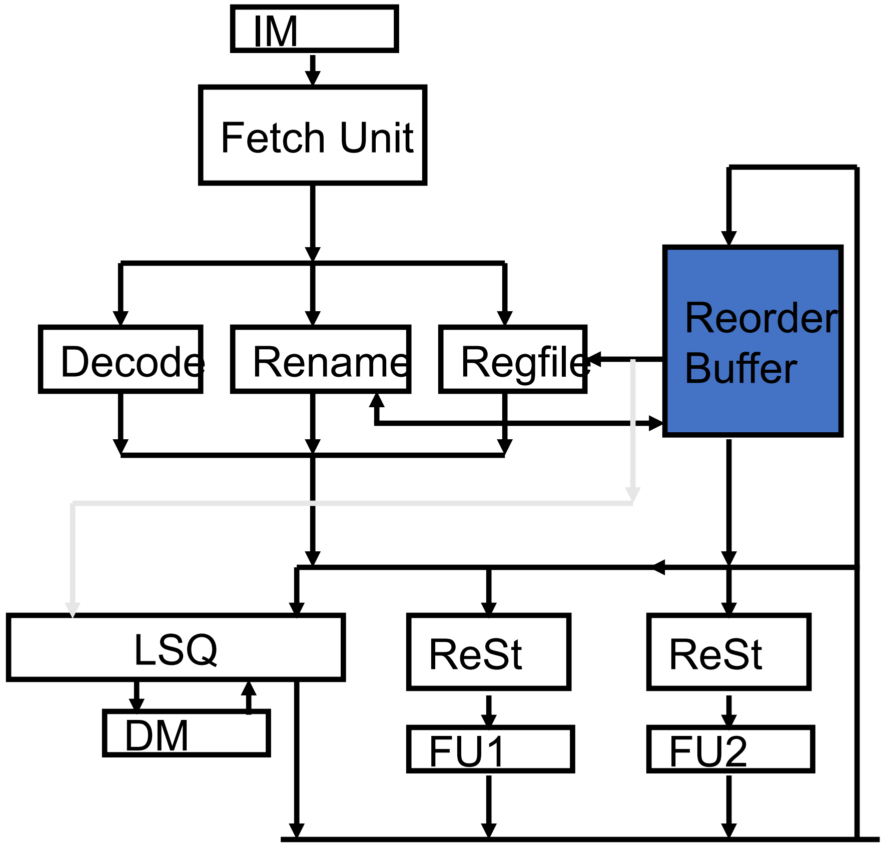
If instructions write results
in program order, reg/memory
always get the correct values

Role of ROB:

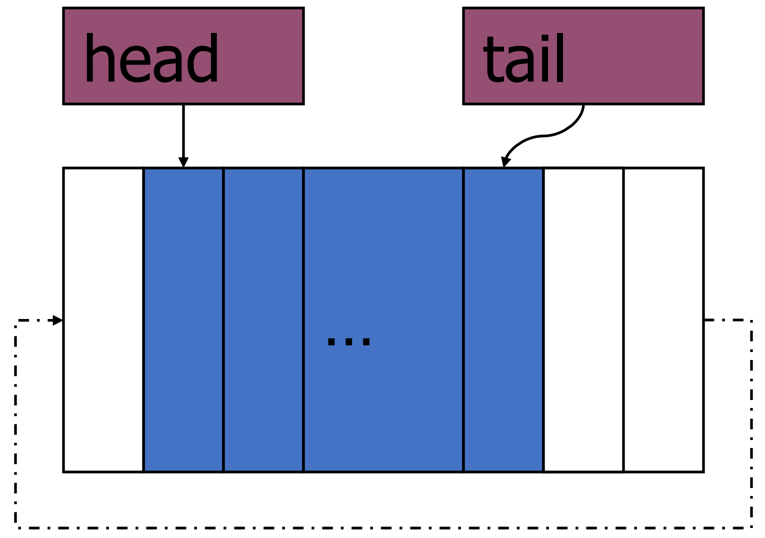
<u>reorder</u> out-of-order instruction
to program order at the time of
writing register/memory (commit)

Instruction cannot write reg/memory
immediately after execution,
so ROB also <u>buffer the results</u>

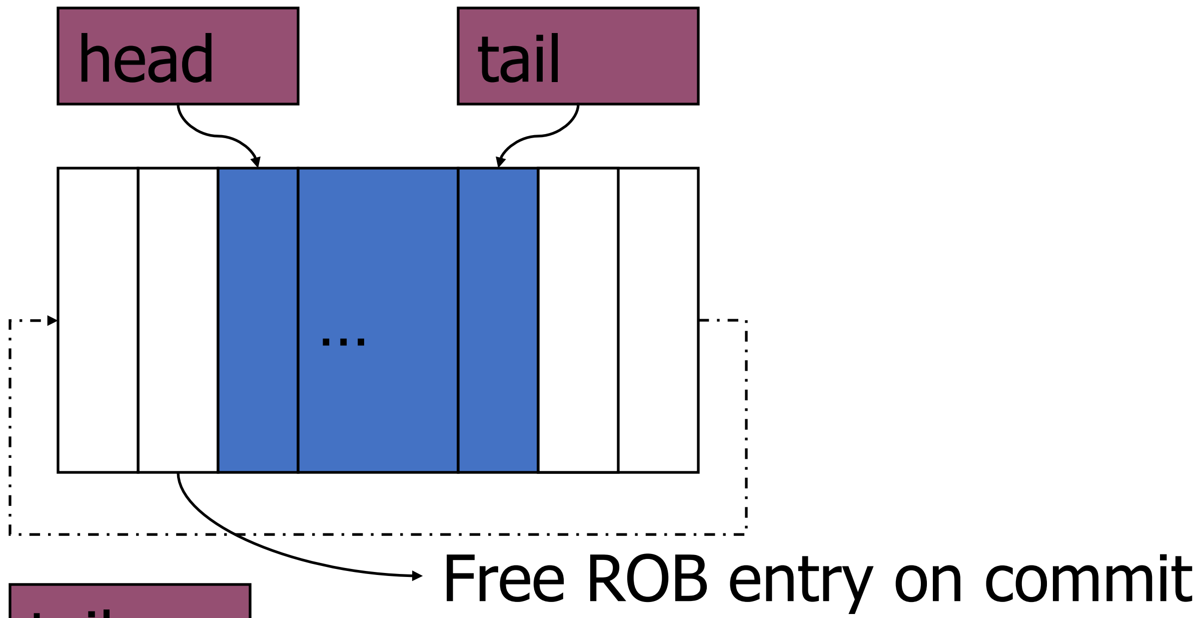
- *No such a place in original Tomasulo*



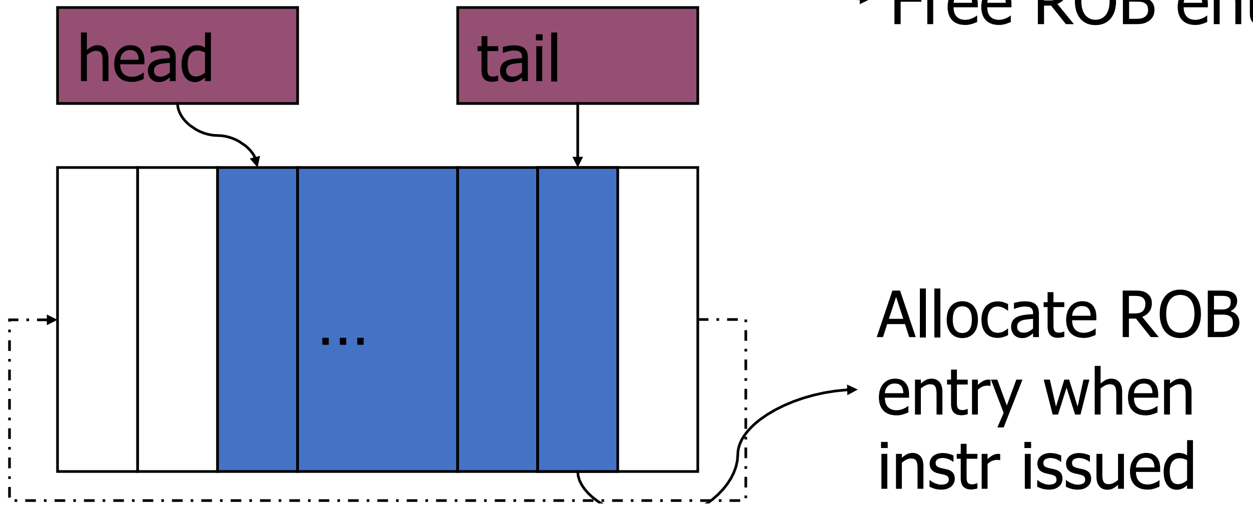
RoB: Circular buffer + Pointers



Entries between head and tail are free/valid



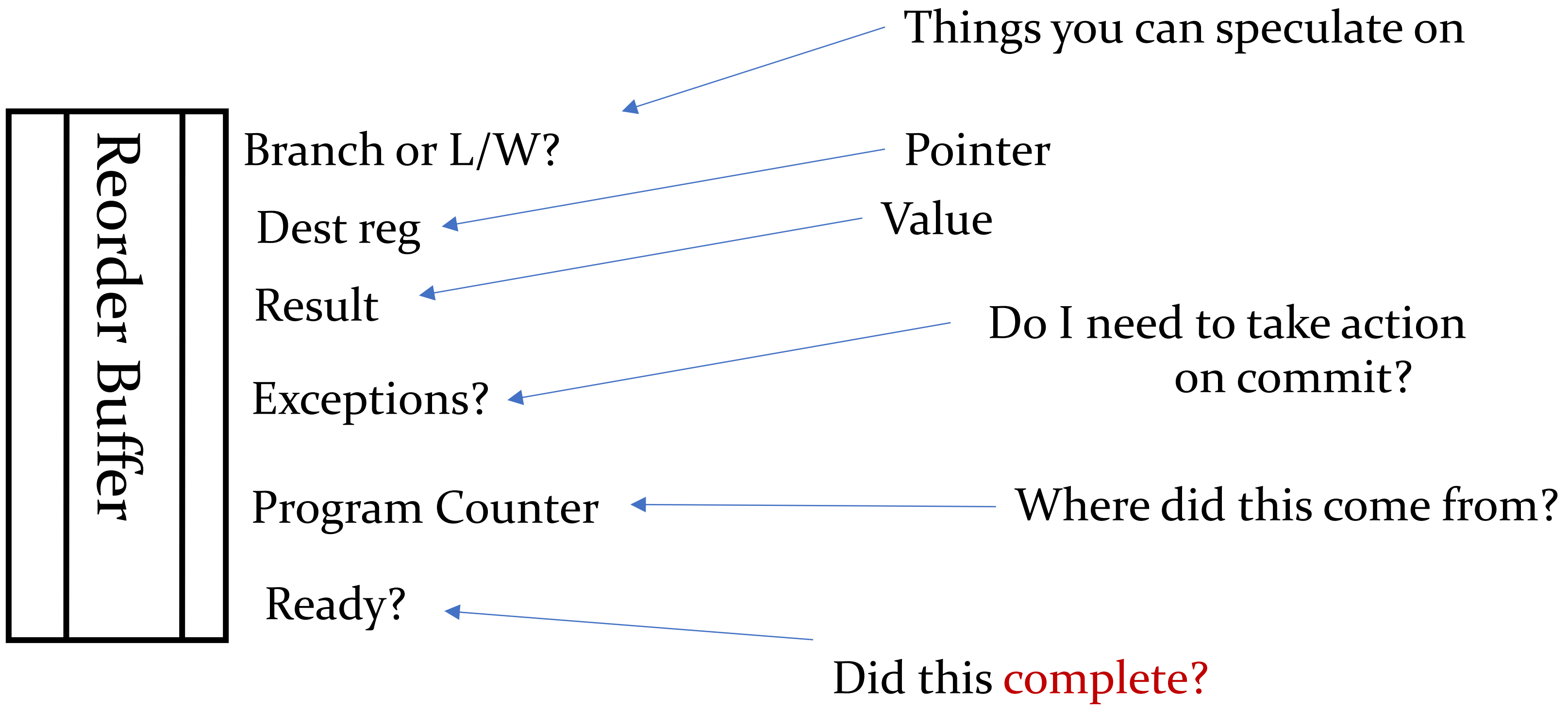
Free ROB entry on commit



Allocate ROB entry when instr issued

Quiz: What's relationship between ROB size and ILP?

RoB Entry Structure



Quiz: What's the difference between completing and committing?

Four Steps of Speculative Tomasulo Algorithm

1. Issue— get instruction from Op Queue

    If reservation station and reorder buffer slot free,
    issue instr & send operands & reorder buffer no. for destination. (this stage sometimes called "dispatch")

    Actions summary: (1) decode the instruction; (2) allocate a RS and ROB entry; (3) do source register renaming;
    (4) do dest register renaming; (5) read register file;
    (6) dispatch the decoded and renamed instruction to the RS and ROB

2. Execution— operate on operands (EX)

    Action: when both operands ready then execute;
    if not ready, watch CDB for result; when both in reservation station, execute;
    this takes care of RAW. (sometimes called "issue")

3. Write result— finish execution (WB)

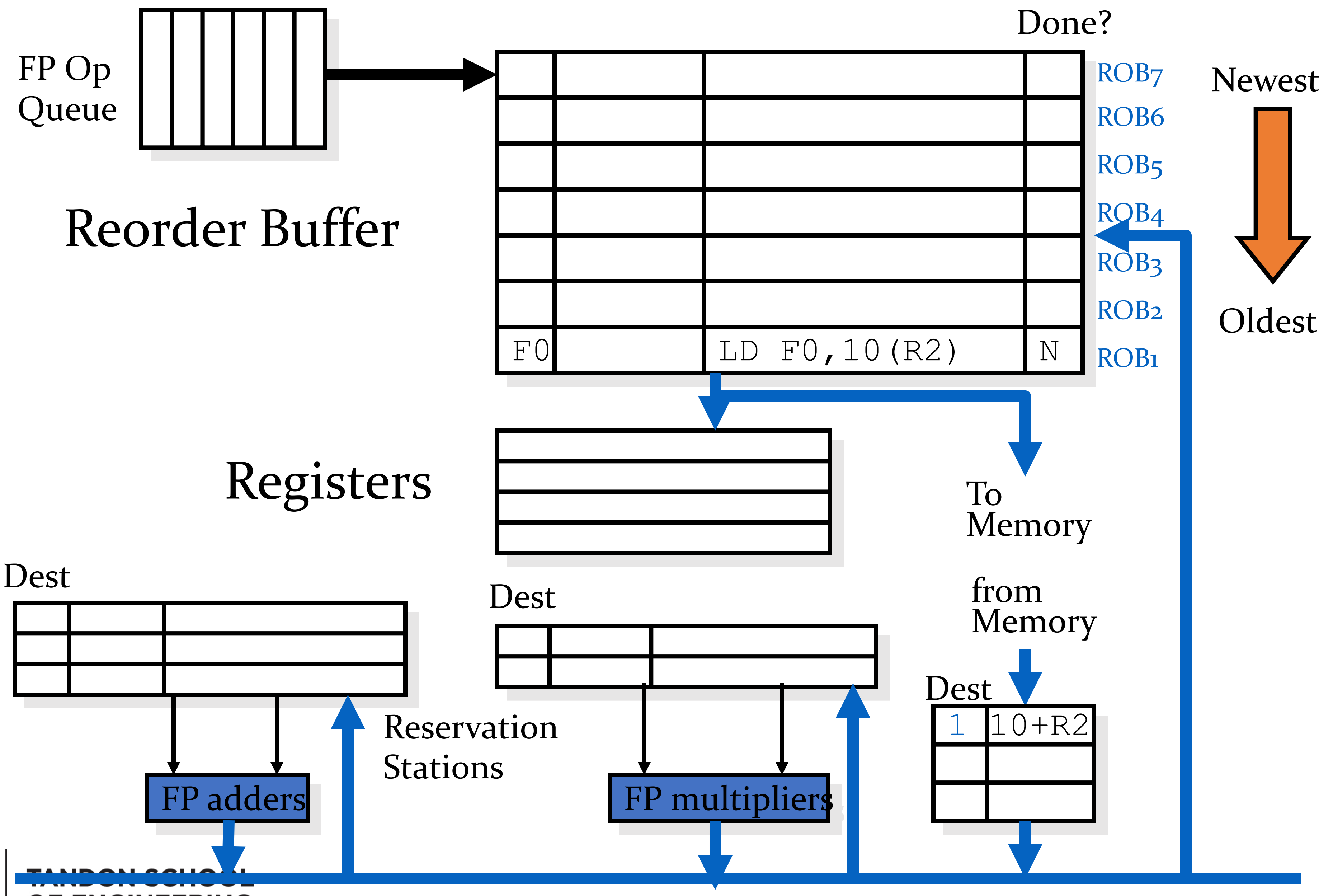
    Action: Write on Common Data Bus to all awaiting FUs & reorder buffer;
    mark reservation station available

4. Commit— update register with result from reorder buffer

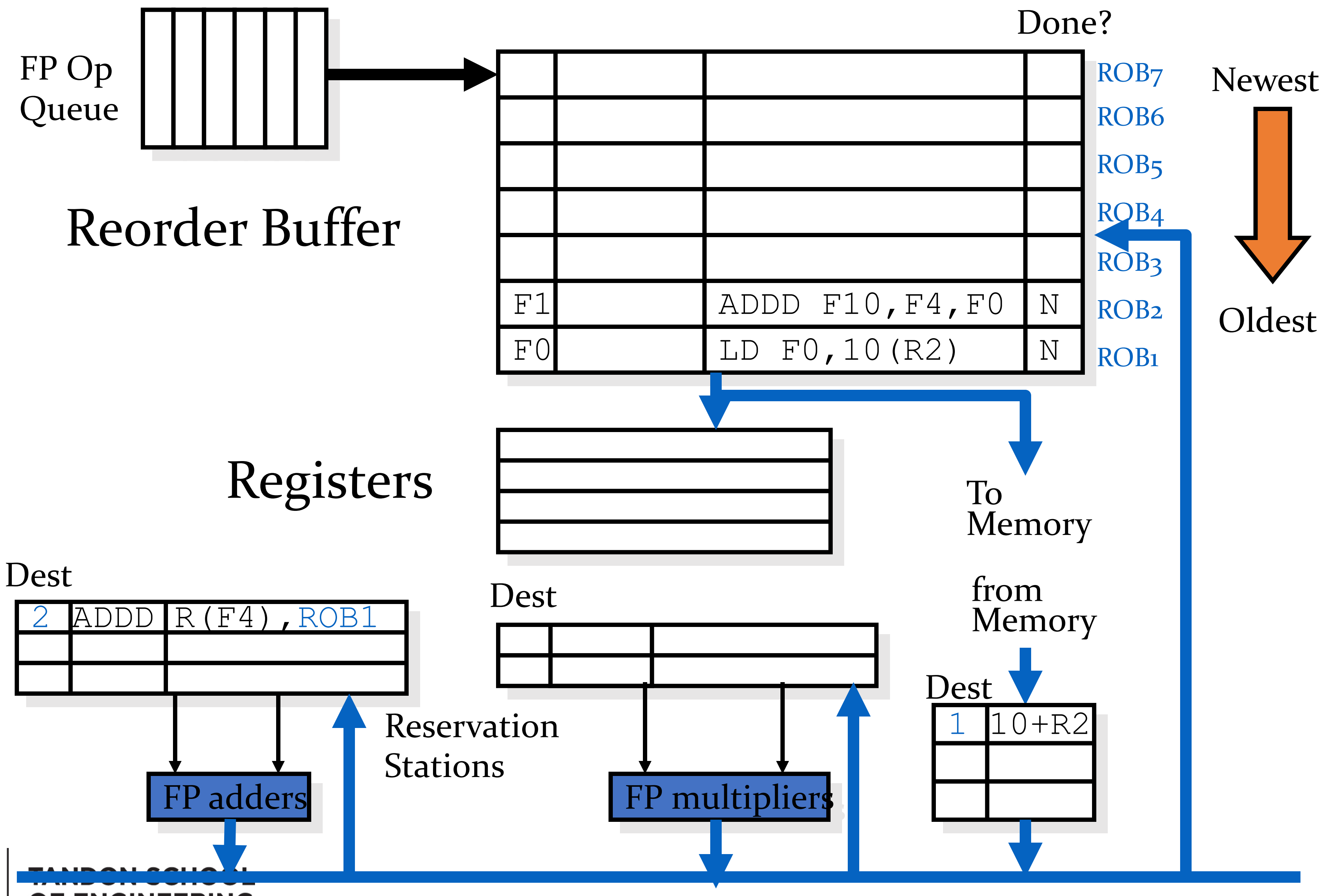
    Action: When instr. at head of ROB & result present,
    update register with result (or store to memory) and remove instr from ROB.

    Mispredicted branch flushes reorder buffer.

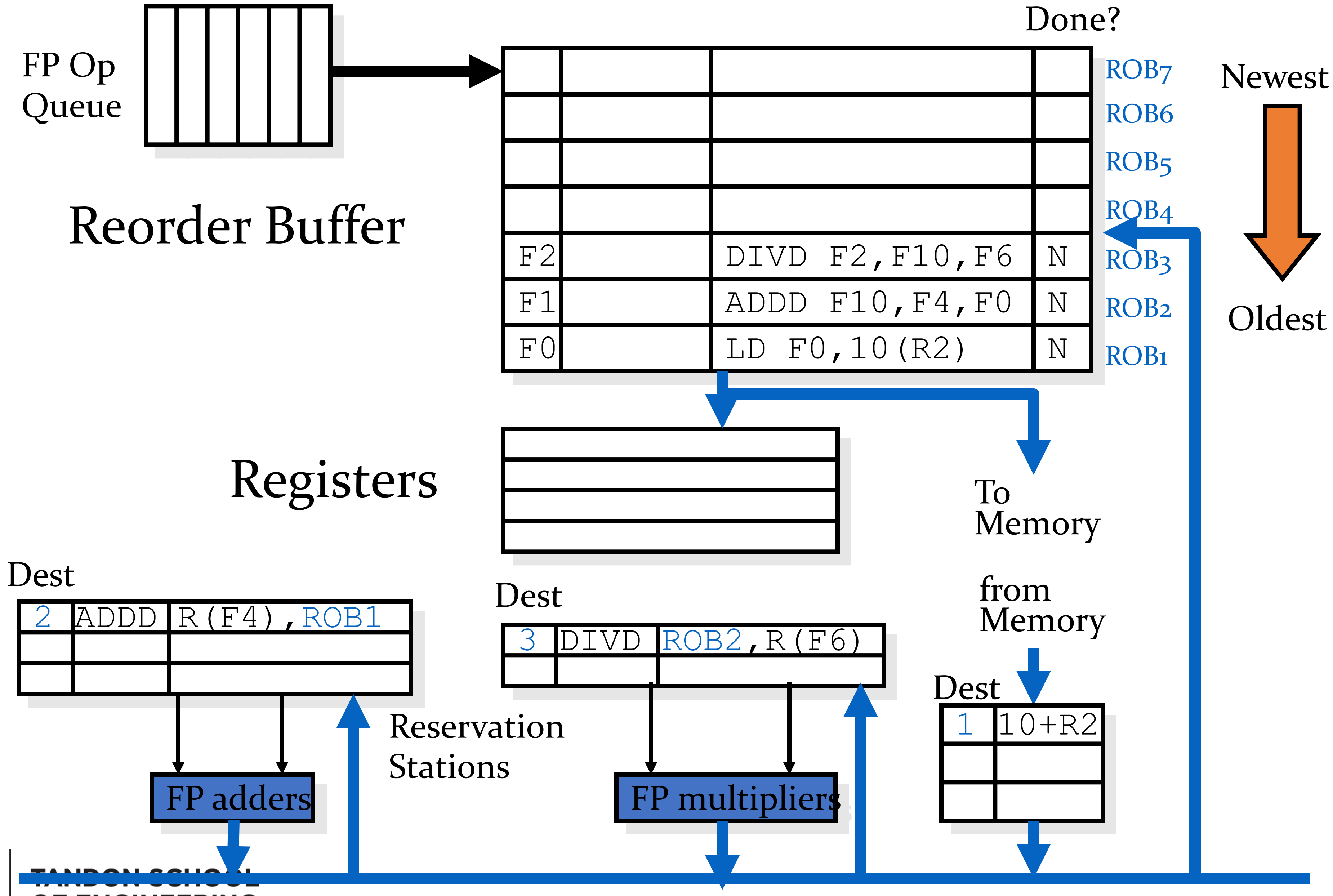
Tomasulo With Reorder buffer:



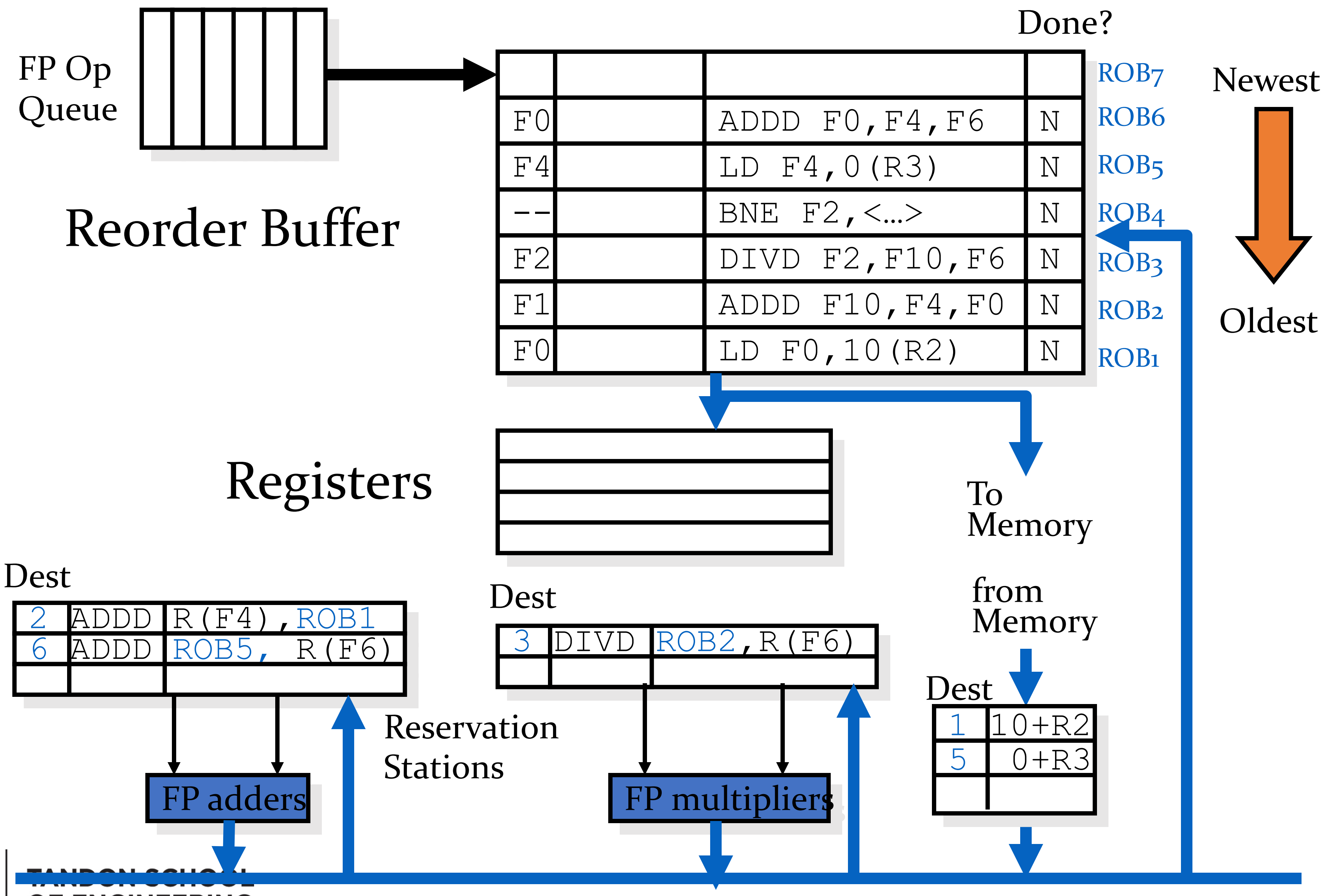
Tomasulo With Reorder buffer:



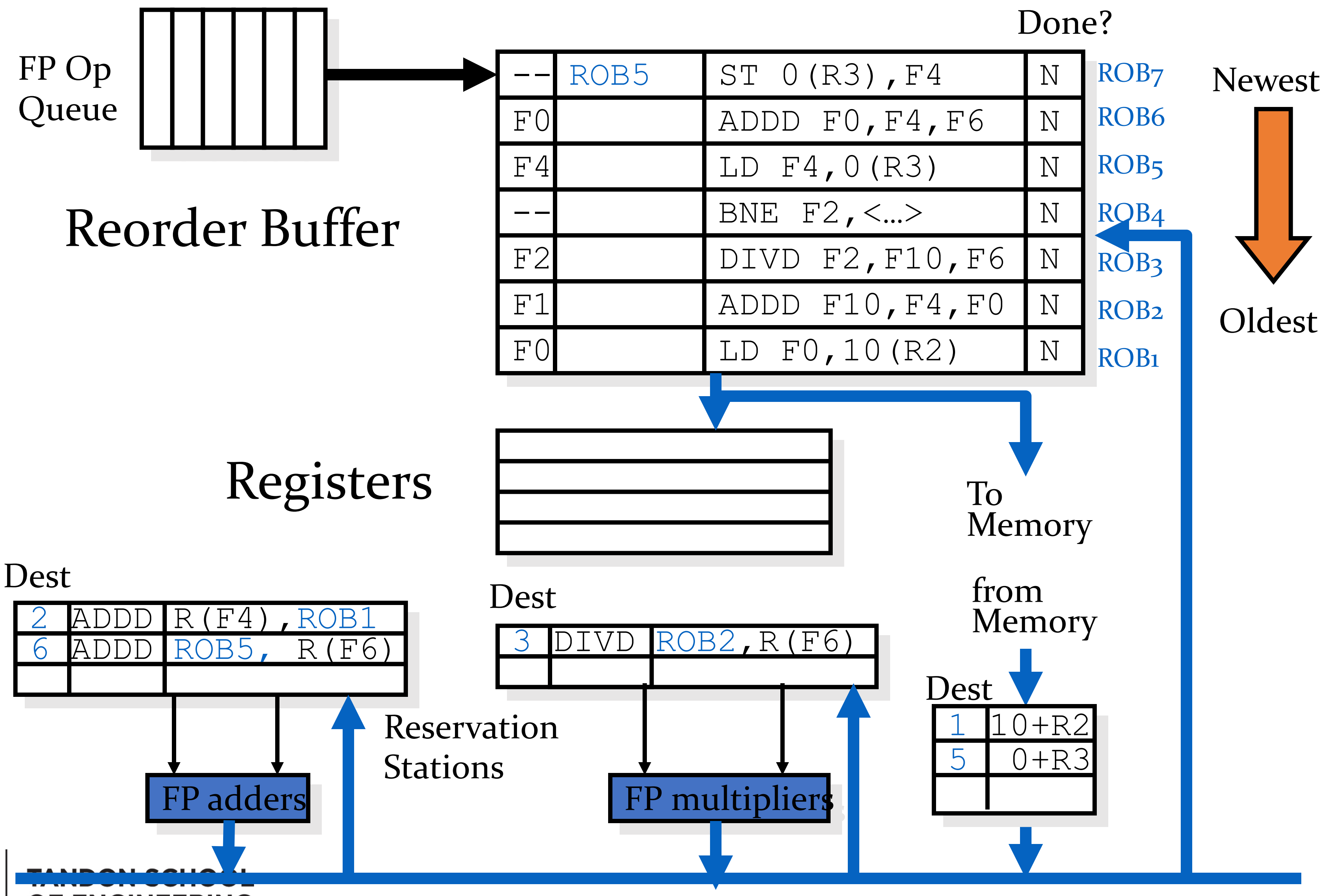
Tomasulo With Reorder buffer:



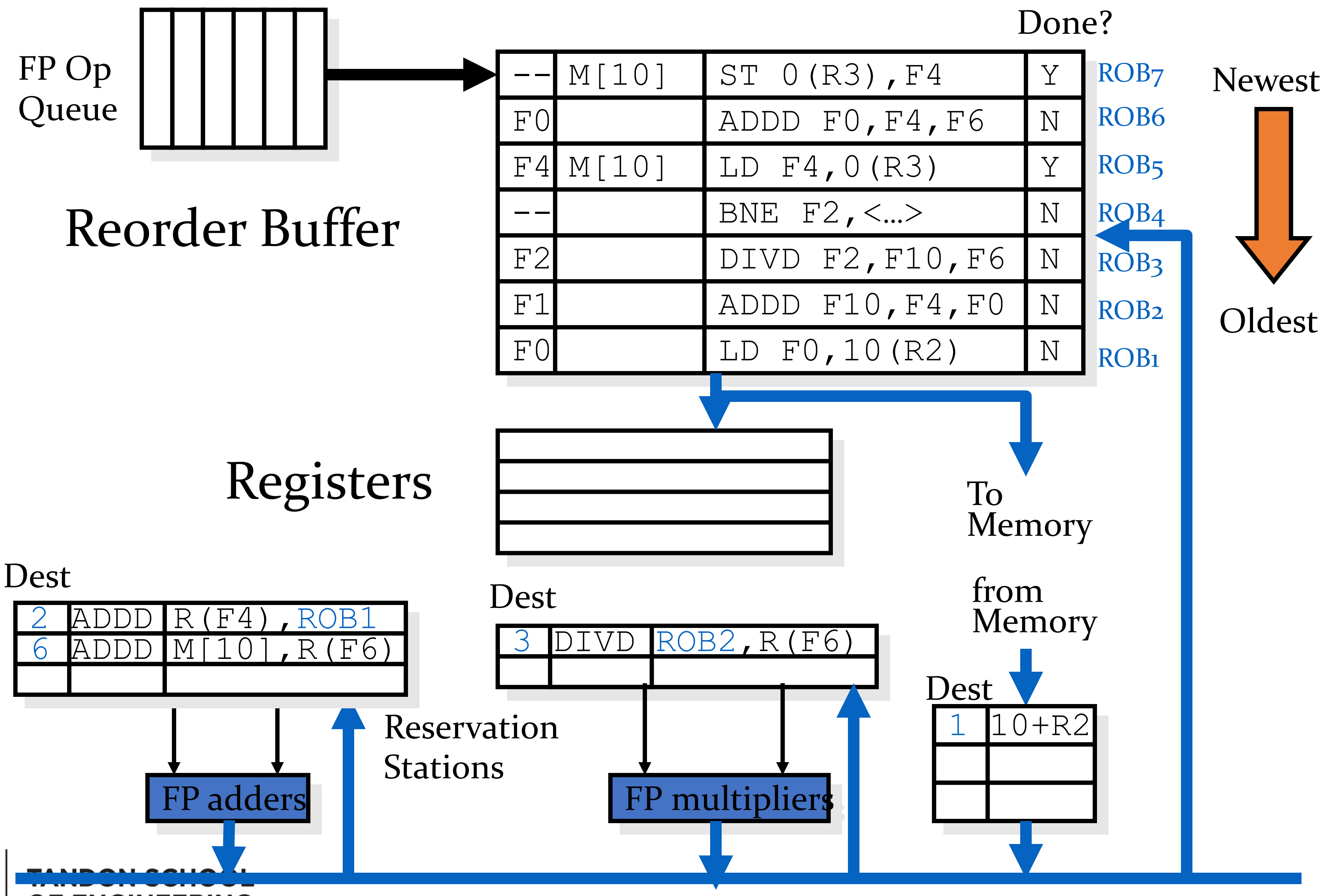
Tomasulo With Reorder buffer:



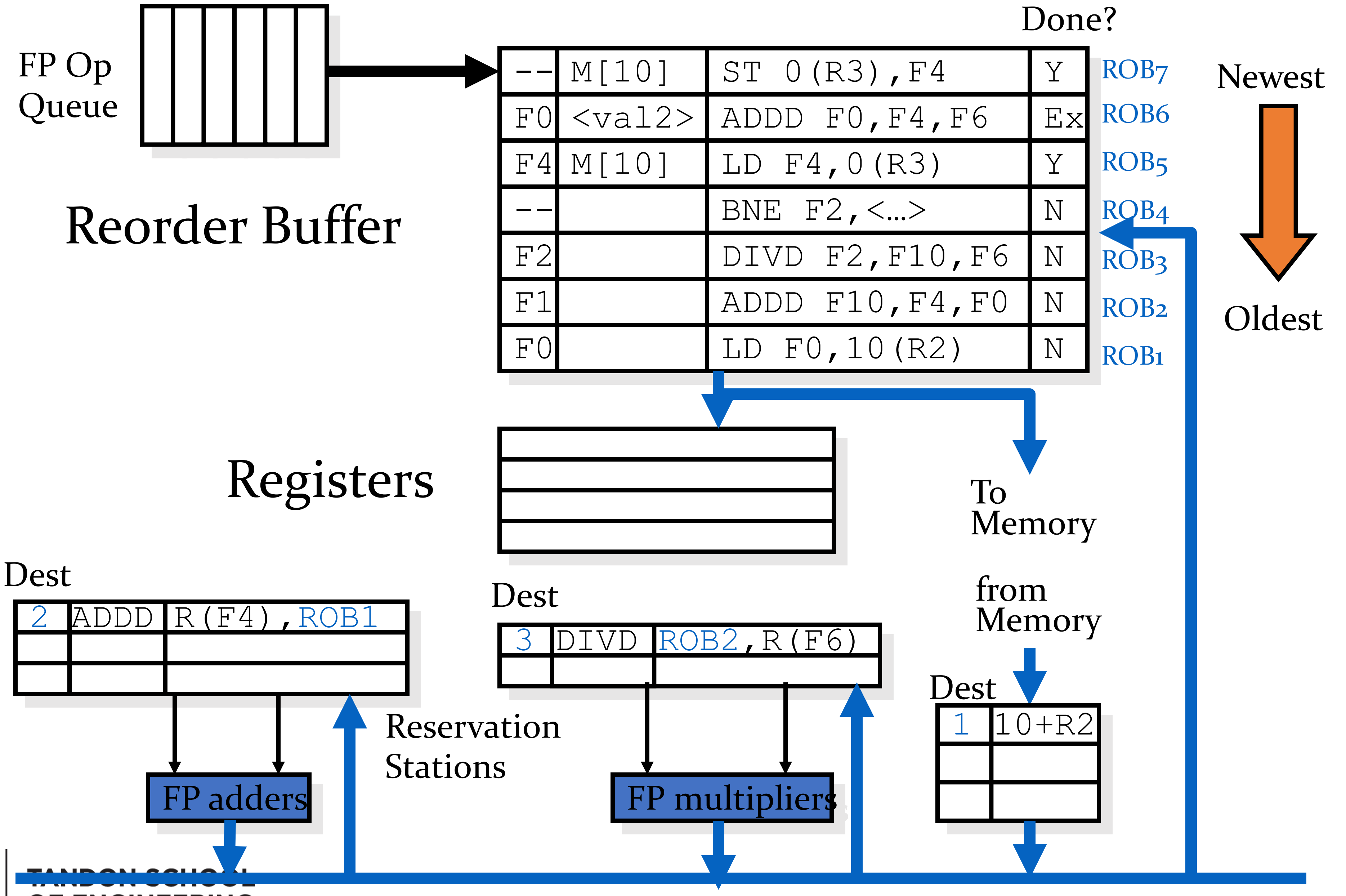
Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:

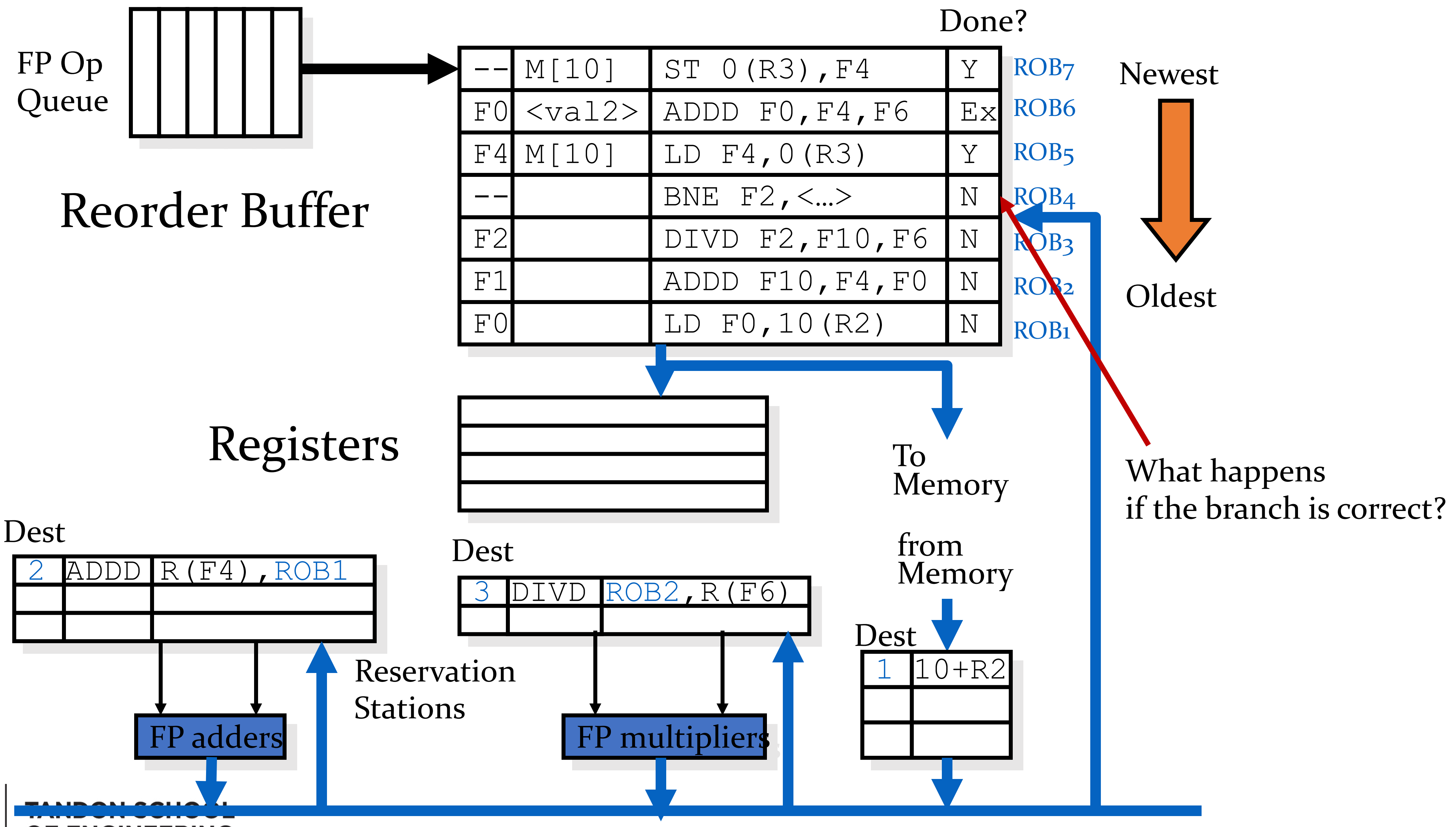

FP Op Queue
Reorder Buffer
Done?
-- | M[10] | ST 0(R3),F4 | Y | ROB7
F0 | | ADDD F0,F4,F6 | N | ROB6
F4 | M[10] | LD F4,0(R3) | Y | ROB5
-- | | BNE F2,<…> | N | ROB4
F2 | | DIVD F2,F10,F6 | N | ROB3
F1 | | ADDD F10,F4,F0 | N | ROB2
F0 | | LD F0,10(R2) | N | ROB1
Newest
Oldest
Registers
To Memory
from Memory
Dest
2 | ADDD | R(F4),ROB1
6 | ADDD | M[10],R(F6)
Dest
3 | DIVD | ROB2,R(F6)
Dest
1 | 10+R2
Reservation Stations
FP adders
FP multipliers

Tomasulo With Reorder buffer:

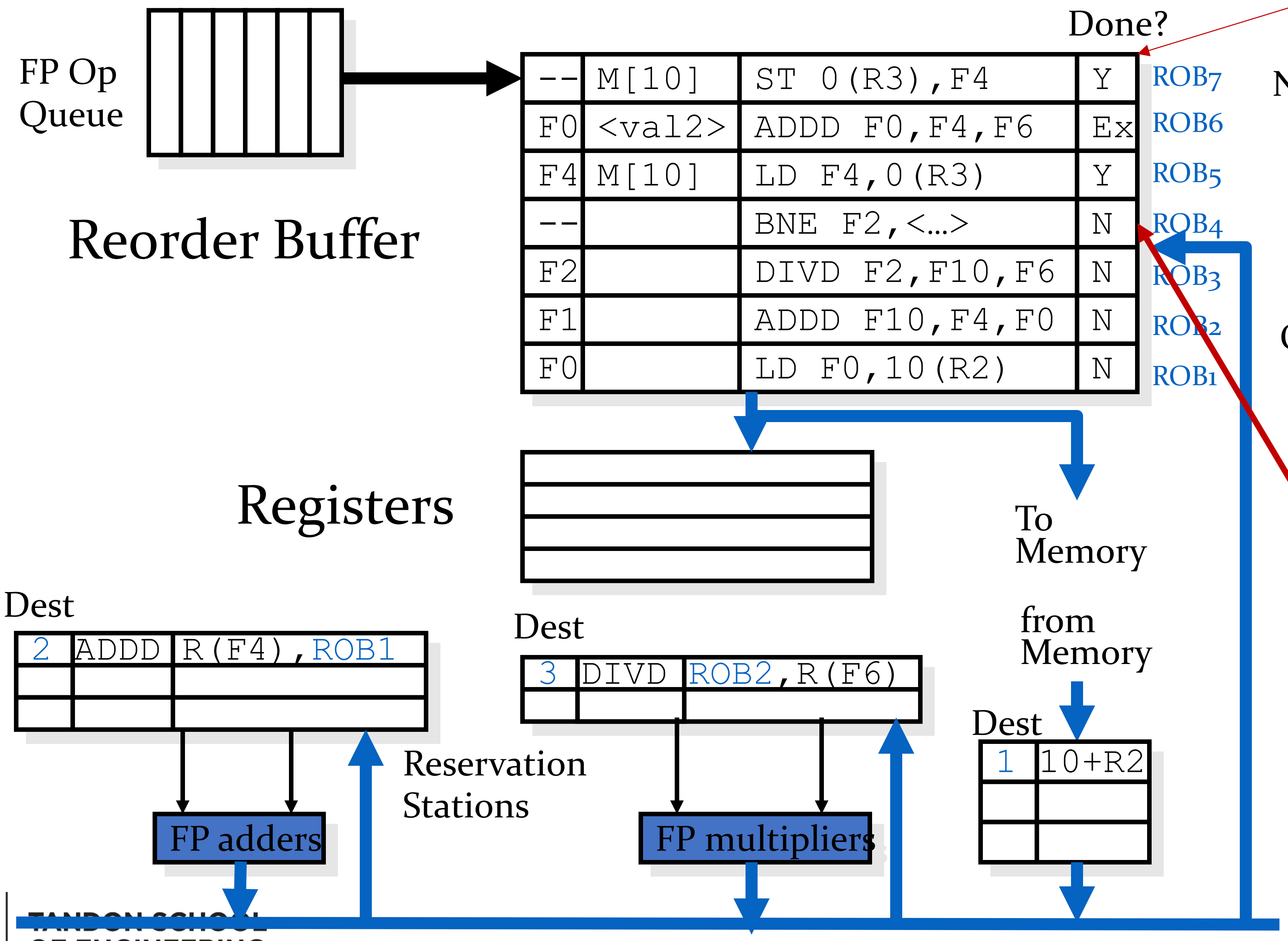


			Done?	
--	M[10]	ST 0(R3),F4	Y	ROB7
F0	<val2>	ADDD F0,F4,F6	Ex	ROB6
F4	M[10]	LD F4,0(R3)	Y	ROB5
--		BNE F2,<…>	N	ROB4
F2		DIVD F2,F10,F6	N	ROB3
F1		ADDD F10,F4,F0	N	ROB2
F0		LD F0,10(R2)	N	ROB1

Tomasulo With Reorder buffer:



Tomasulo With Reorder buffer:


FP Op Queue
Reorder Buffer
Done?
-- | 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
F1 | | 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

Newest

Oldest

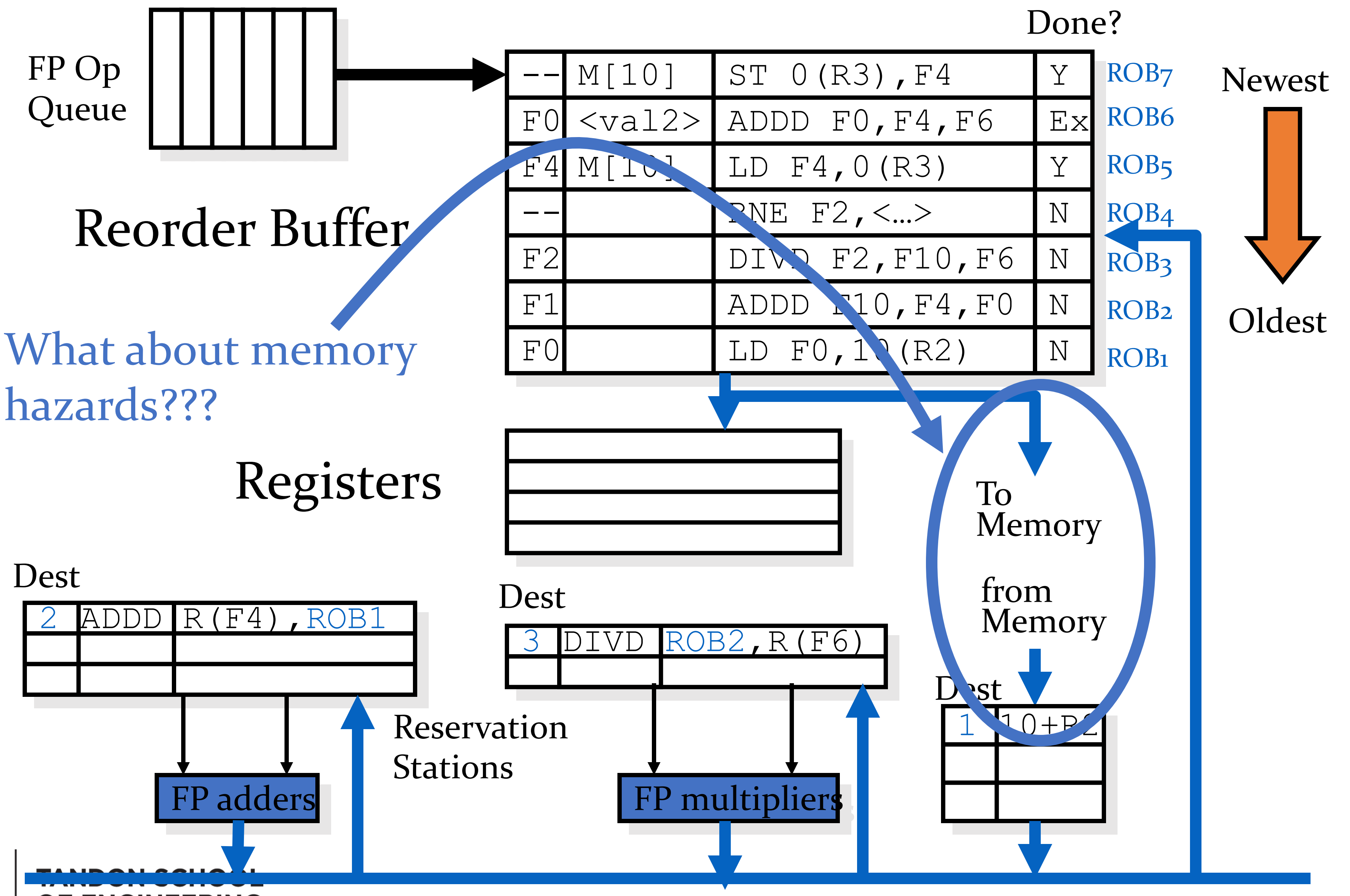
When does store happen?

What happens if the branch is incorrect?

Just erase!
Flush ld, add, st state

What would happen if let ST complete OOO?

Tomasulo With Reorder buffer:



			Done?	
--	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
F1		ADDD F10,F4,F0	N	ROB2
F0		LD F0,10(R2)	N	ROB1

Dest		
2	ADDD	R(F4),ROB1

Dest		
3	DIVD	ROB2,R(F6)

Dest	
1	10+R2

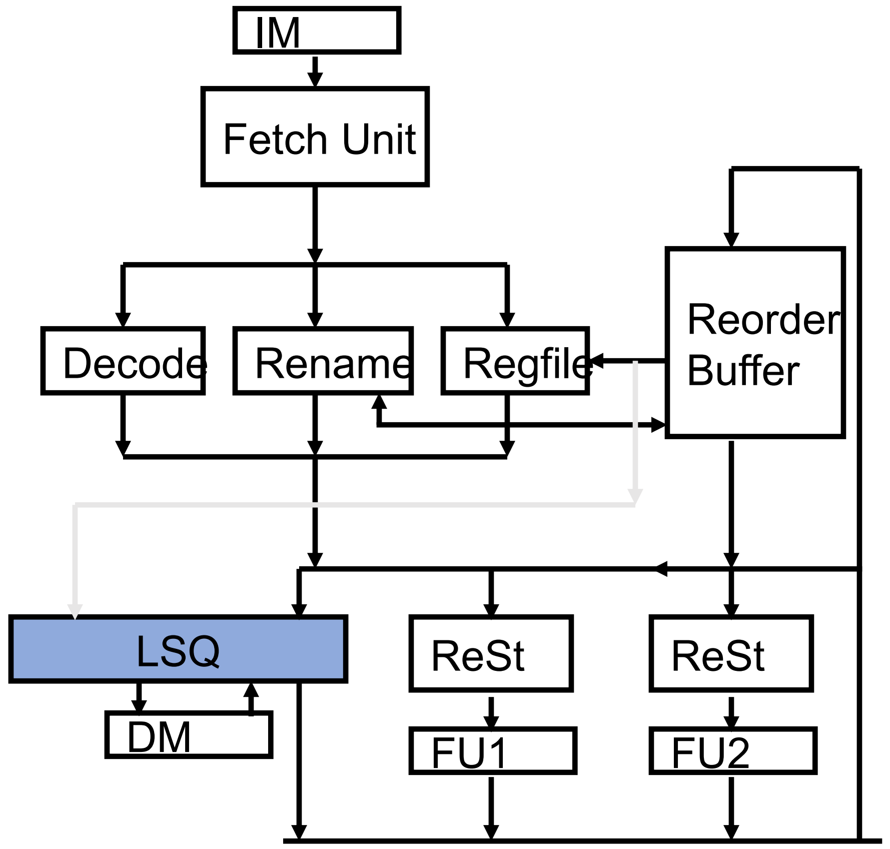
Load-Store Queue (LSQ)

Like a RoB for memory

All Loads and Stores send to
load-store Q <u>in-order</u>

Does dynamic memory
disambiguation to determine
if we have memory hazards

What's the difference between
register and memory hazard?



The Load-Store Queue

Program:

```
LW R1, 0(R1)
SW R2, 0(R3)
SW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?

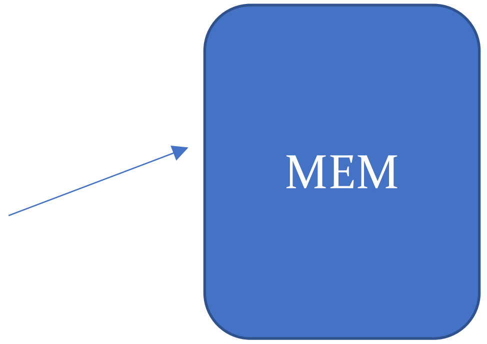
MEM

The Load-Store Queue

Program:

```
LW R1, 0(R1)
SW R2, 0(R3)
SW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?
L	104		

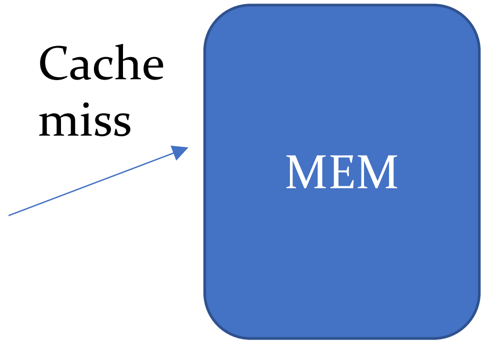


The Load-Store Queue

Program:

```
LW R1, 0(R1)
SW R2, 0(R3)
SW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?
L	104		
S	204	15	Y



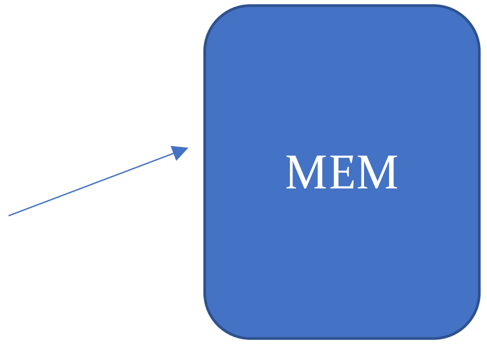
Can commit?

The Load-Store Queue

Program:

```
LW R1, 0(R1)
SW R2, 0(R3)
SW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?
L	104		
S	204	15	Y
L	204		



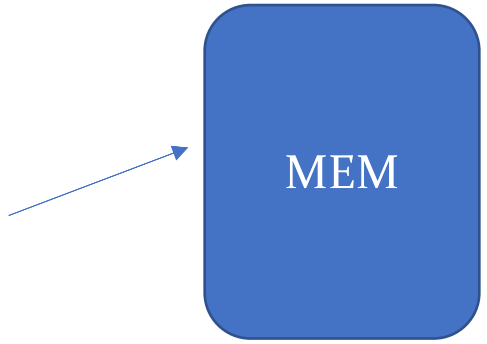
Can commit?

The Load-Store Queue

Program:

```
LW R1, 0(R1)
SW R2, 0(R3)
SW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?
L	104		
S	204	15	Y
L	204	15	



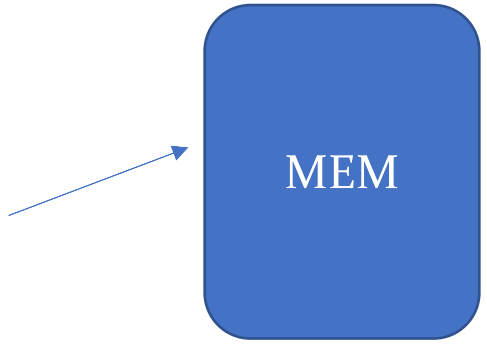
Can commit?

Store-to-Load forwarding

The Load-Store Queue

Program:
```
LW R1, 0(R1)
SW R2, 0(R3)
LW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)
```

Ld/St	Addr	Value	Comp?
L	104		
S	204	15	Y
L	204	15	
S	?	?	N



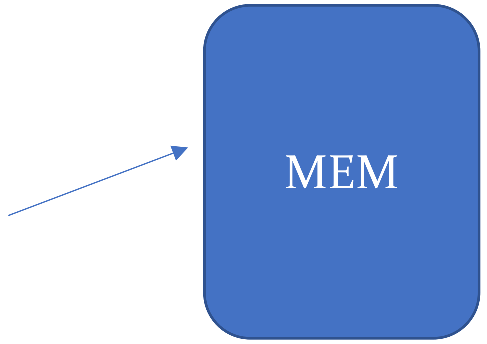
Can commit?

Store-to-Load forwarding

The Load-Store Queue

Program:
LW R1, 0(R1)
SW R2, 0(R3)
LW R4, 0(R4)
SW R5, 0(R0)
LW R5, 0(R8)

Ld/St	Addr	Value	Comp?
L	104		
S	204	15	Y
L	204	15	
S	?	?	N
L	174		



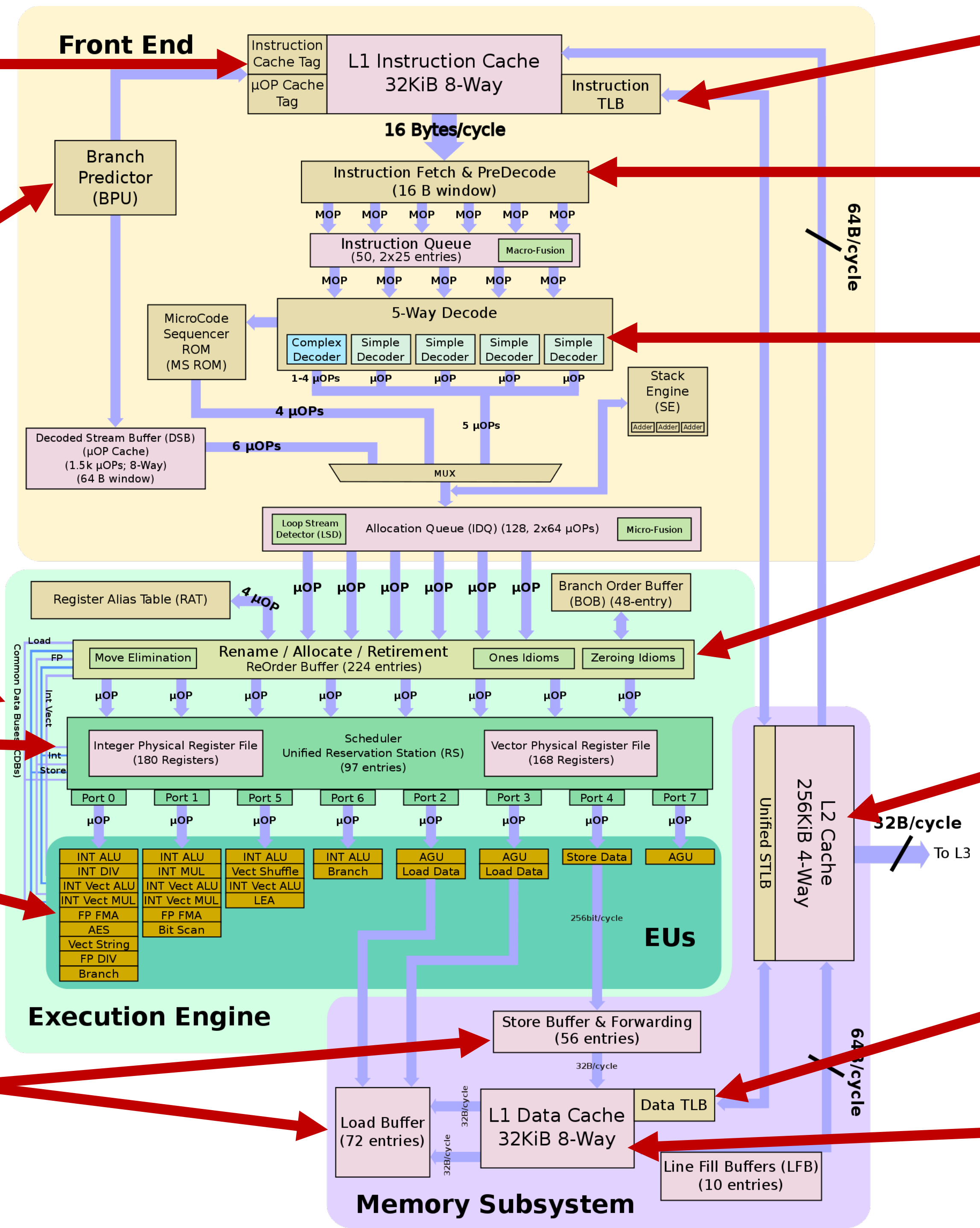
Can commit?

Load 174 options:
1) Strict In-order:
   wait for everything
2) Store address resolve:
   will know what to do
3) Speculate!
   Guess not hazard,
   fix if there was

Steps for load:
  Check all previous stores.
  if addr match:
    Val=> St-Ld fwd
    !Val => wait
  else:
    Speculative read

When St addr resolves,
make sure no hazard.

All material for the course!



L1 Inst cache

iTLB

Inst Fetch

Branch Predictor

Inst Decode

ReOrder buffer (224 entries)

Common data bus

Unified L2 cache

Reservation station

Execution unit pipes

dTLB

~LSQ

L1 Data Cache

NYU | TANDON SCHOOL OF ENGINEERING

Advanced Topics

In the core:

0) Lots more speculation and caches

- Alias predictors, trace caches, etc...

1) Superscaler

- Issue more than one instruction per cycle

2) Simultaneous multi-thread

- To better utilize HW (throughput) when ILP is limited, issue instruction from different threads/processes.

- This is Intel's "Hyperthread"

Multi-core

Today's computers have more than one "core"

1) How do you get speedup?

2) Where's the data?
    1) Cache Coherence
    2) Memory Consistency

Think about LSQ, now many core and shared caches..
How do you know which store was first?

Memory systems

1) Prefetching: avoid long latencies by guessing what to bring in

2) How to use eNVMs?

3) "Unified" memory spaces
   If I have accelerators, like GPU or a TPU,
      should they see the same memory or not?
   Datacenter-scale memory space

Improving parallelism

SIMT: Single Instruction, Multiple Threads

GPU style. Have a bunch of simple cores that all do the same thing and have access to local variables

SIMD: Single instruction, multiple data

SSE/AVX style. 1 register (word) packed with many data.

One instruction executes on each data at the same time.

Vector: 1 vector instruction launches many of the same

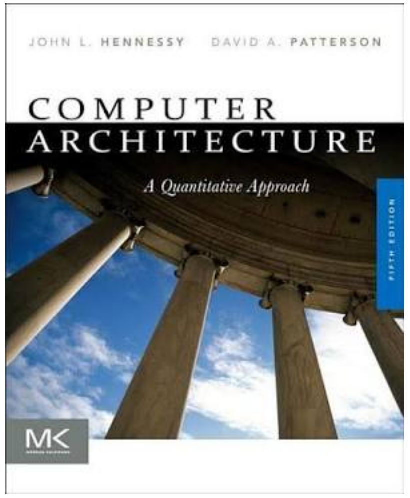
computations, but not limited to word size.

SIMD Instructions Considered Harmful

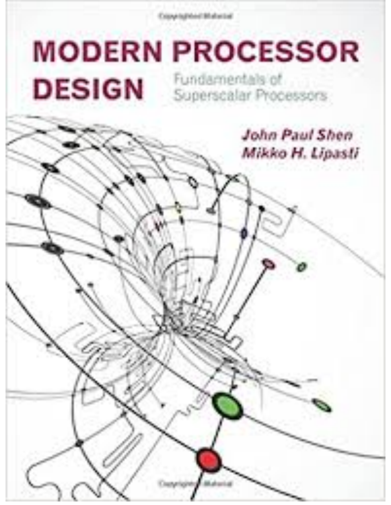
VLIW: Very Long Instruction Word. Many different (maybe the same) instructions into one with SW. Execute each in parallel.
Great idea, but really hard.

NYU TANDON SCHOOL OF ENGINEERING

Resources

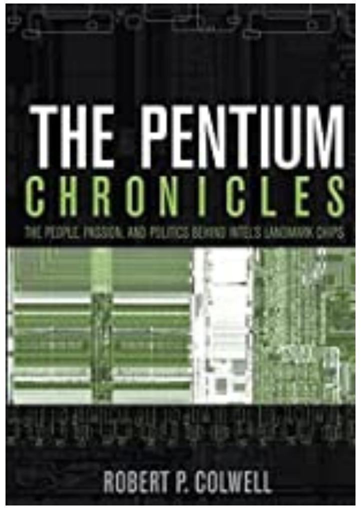


What most people "use"



<u>Very</u> detailed description of uArchitecture. Would recommend if you really want to learn more

Collection of freely available papers


THE PENTIUM CHRONICLES
THE PEOPLE, PASSION, AND POLITICS BEHIND INTEL'S LANDMARK CHIPS
ROBERT P. COLWELL

Fun read

100-200 page "books" on single topics on comp arch.
Free access with NYU login!
(At least I could this morning.)
Everything from caches, multi-core, ML accelerators

<u>Check out conferences:</u>
ISCA, HPCA, ASPLOS, MICRO
Find something you like.
Looks up references and work backwards

NYU TANDON SCHOOL OF ENGINEERING

Remaining lectures

Paper reading.
What's happening in Comp Arch Today?

Will assign 2-3 papers / class.
Spend ~45 minutes actively discussing each.

Expectations:

1) Read papers before class.

2) Participate in discussion if possible.

This is what next semester will be like.

NYU TANDON SCHOOL OF ENGINEERING

ECE 9413: Parallel and Customized Computer Architecture

W1: Limits of ILP

W2: Parallel Architectures

W3: The Power Wall

W4: The Need for Customization

W5: Application Specific Architectures

W6: Dataflow machines

W6: HW for ML I: CNNs

W7: HW for ML II: Real chips

W8: HW for ML III: Sparse accelerators

W9: Enabling private computing: SGX + FHE

Project: Implement FHE in CUDA

What is Fully Homomorphic Encryption?

- A new type of encryption that enables computation directly on encrypted data!
- Sounds amazing, but ~1Mx too slow..

What's CUDA?

- The language NVIDIA GPUs speak
- We want to see how much of that 1Mx we can get by implementing key kernels in CUDA.
- Big course project, at the end we'll collectively submit a paper.