CASPER

CS230: Digital Logic Design and Computer Architecture

Lecture 22: Connecting All the Dots (O3 processor)

https://www.cse.iitb.ac.in/~biswa/courses/CS230/main.html

https://www.cse.iitb.ac.in/~biswa/

Recap

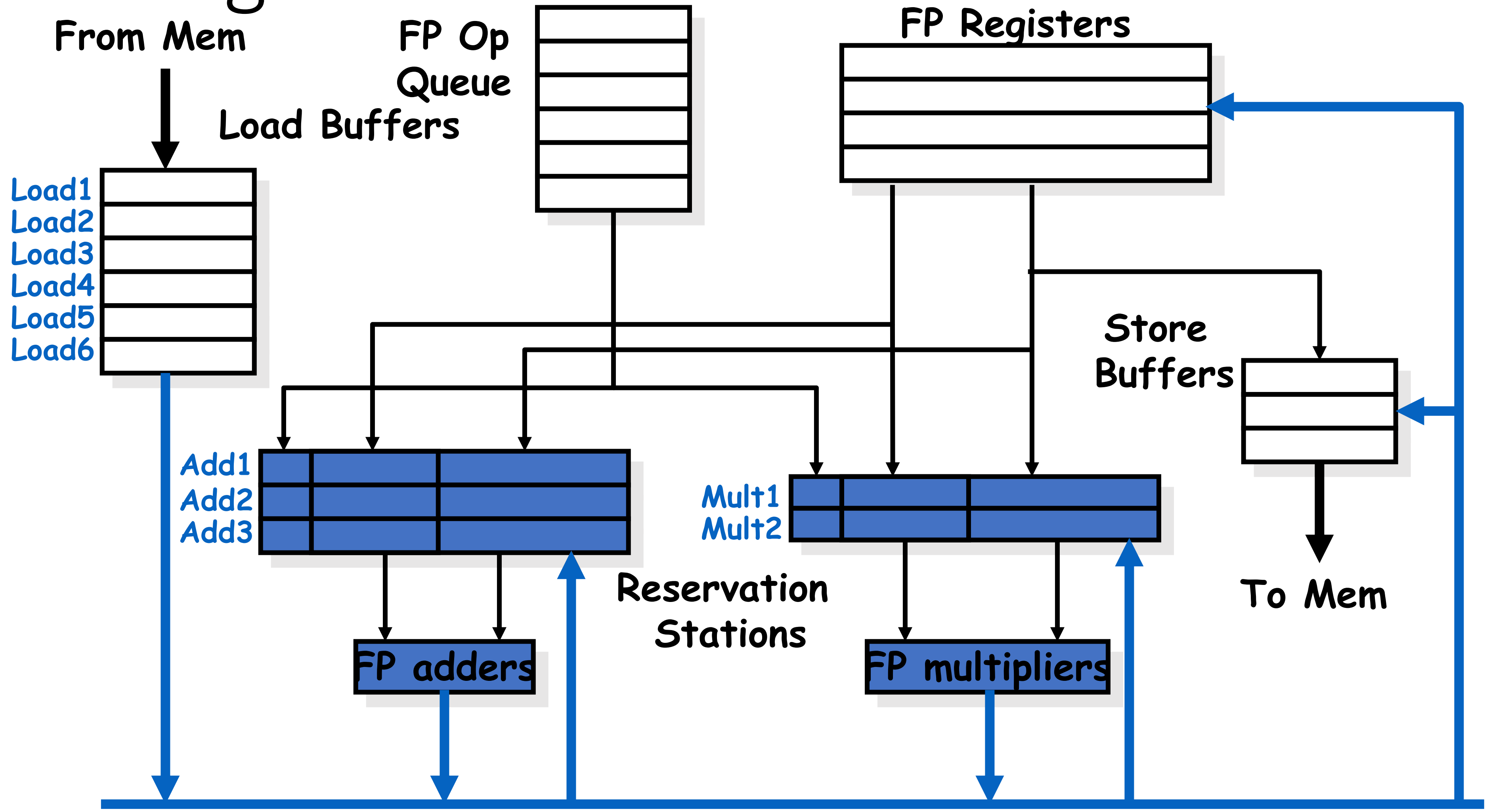
Remember out-of-order processor

Inorder fetch, out-of-order execute, inorder commit

Now, we will discuss in details

Static scheduling: Compiler can do

Tomasulo’s Organization for dynamic scheduling



Points to remember

- Control & buffers distributed with Function Units (FU)
  - FU buffers called "reservation stations"; have pending operands
- Registers in instructions replaced by values or pointers to reservation stations(RS); called register renaming ;
  - avoids WAR, WAW hazards
  - More reservation stations than registers, so can do optimizations compilers can't
- Results to FU from RS over Common Data Bus that broadcasts results to all FUs
- Load and Stores treated as FUs with RSs as wells
- Decode stage of the pipeline: becomes two stages:

Issue: Decode instructions, check structural hazards

Read operands: Wait until no data hazards, then read operands.

Some processors use the term dispatch and issue.

Reservation Station Components

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

Vj, Vk: Value of Source operands

– Store buffers has V field, result to be stored

Qj, Qk: Reservation stations producing source registers (value to be written)

– Qj,Qk=0 => ready

– Store buffers only have Qi 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.

The New Pipeline

Inorder Instruction fetch

Fetched instructions enqueued into a Q called Instruction Q (IQ).

Inorder instruction issue from the IQ

Outoforder execution

New concept of register renaming through reservation stations that eliminates WAR and WAW hazards

An Example

Instruction status:

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

	Busy	Address
Load1	No	
Load2	No	
Load3	No	

Think about memory hierarchy ☺

Reservation Stations:

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

Register result status:

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

Computer Architecture

Cycle 1

Instruction status:

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

	Busy	Address
Load1	Yes	34+R2
Load2	No	
Load3	No	

Load: 2 cycle
FP add: 2 cycles
FP multiply: 10 cycles
FP divide: 40 cycles

Reservation Stations:

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

Register result status:

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

Cycle 2

Instruction status:

Instruction		j	k	Issue	Exec Comp	Write 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:

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

Register result status:

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

can have multiple loads outstanding

Cycle 3

Instruction status:

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

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

Reservation Stations:

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

Register result status:

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

- Note: registers names are removed ("renamed") in Reservation Stations; MULT issued

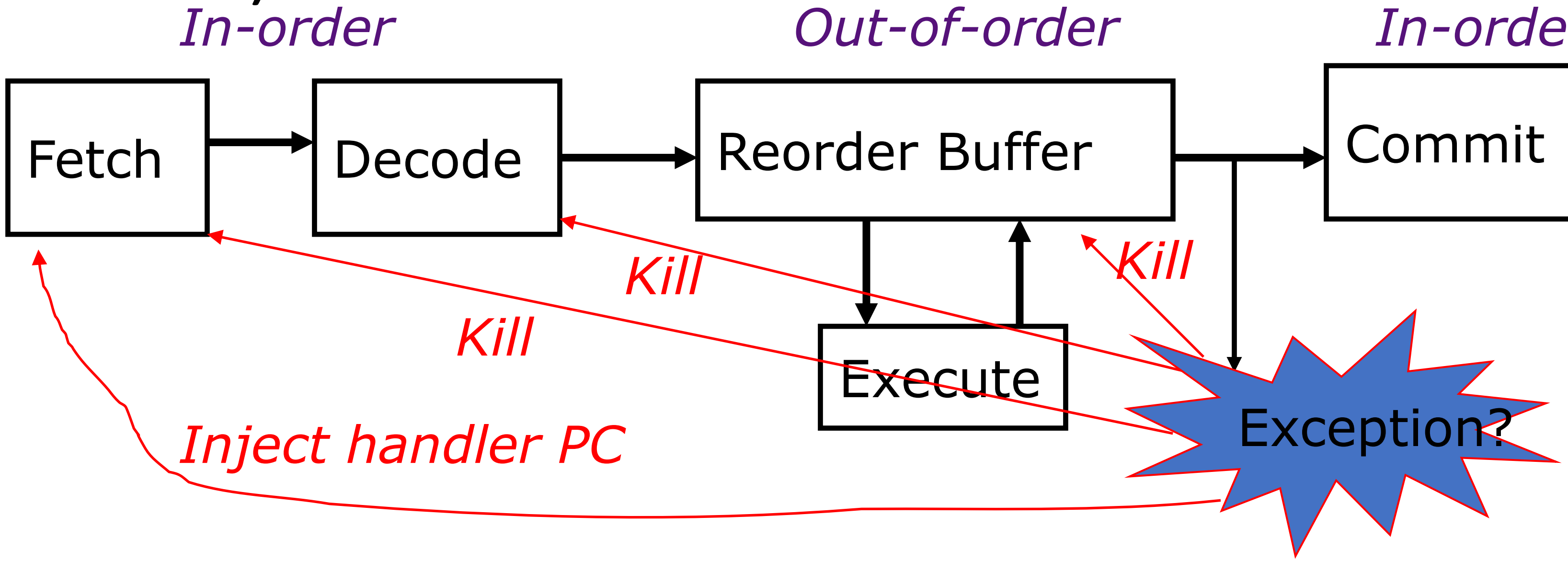
Register Renaming

- Tomasulo provides *Implicit Register Renaming*
  - User registers renamed to reservation station tags
- Explicit Register Renaming:
  - Use *physical* register file that is larger than number of registers specified by ISA
- Keep a translation table:
  - ISA register => physical register mapping
  - Physical register becomes free when not being used by any instructions in progress. More later after ROB.

Explicit Register Renaming

- Rapid access to a table of translations
- A physical register file that has more registers than specified by the ISA
- Ability to figure out which physical registers are free.
  - No free registers $\Rightarrow$ stall on issue
- Thus, register renaming doesn't require reservation stations. However:
  - Many modern architectures use explicit register renaming + Tomasulo-like reservation stations to control execution.

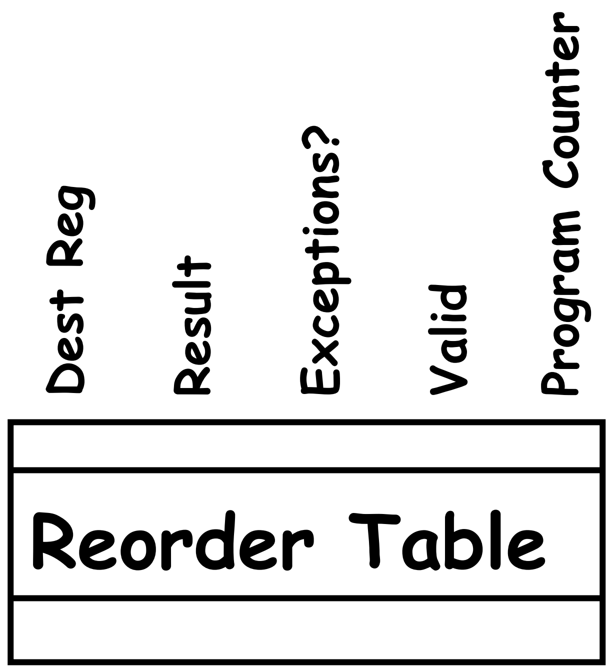
Tomasulo, O3 completion, we need inorder complete (commit)☹

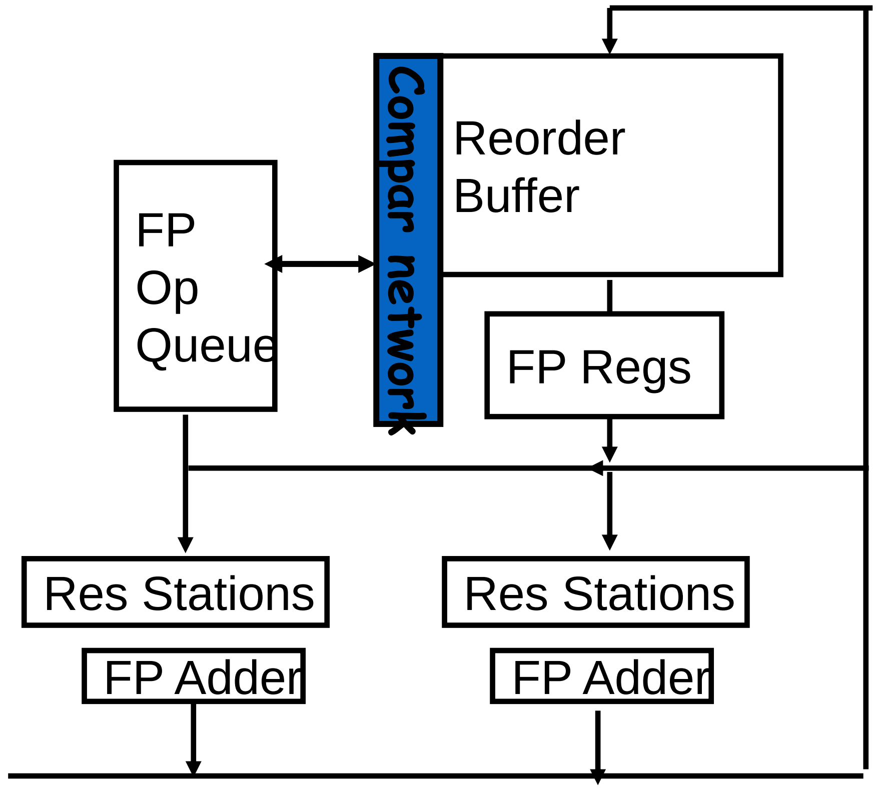


- Instructions fetched and decoded into instruction reorder buffer in-order
- Execution is out-of-order (⇒ out-of-order completion)
- *Commit* (write-back to architectural state, i.e., regfile & memory) is in-order

Temporary storage needed to hold results before commit (shadow registers and store buffers)

Dynamic scheduling with speculative execution





Need as many ports on ROB as register file

Speculative O3 with Tomasulo and ROB

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)

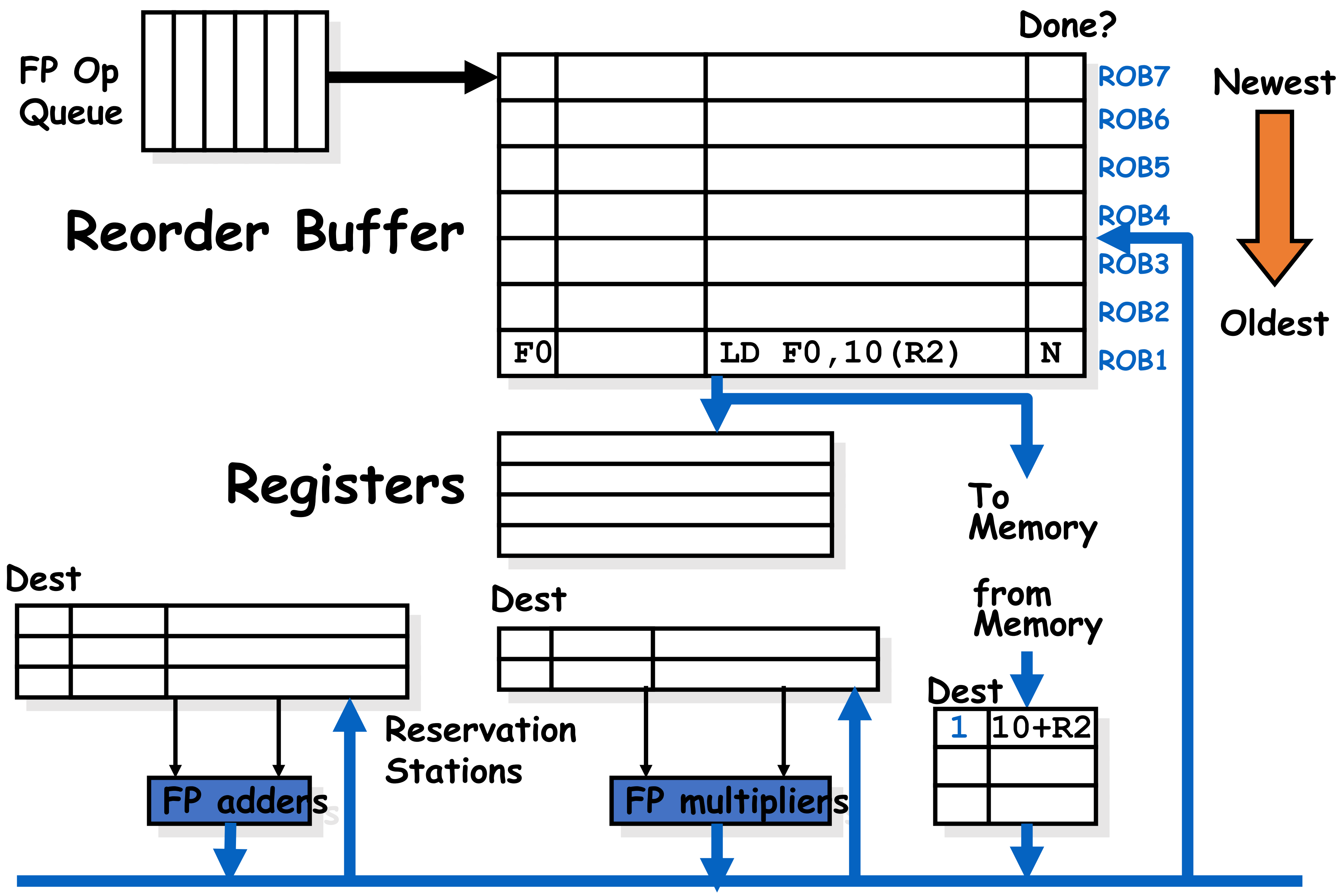
   When both operands ready then execute; if not ready, watch CDB for result; when both in reservation station, execute; checks RAW (sometimes called “issue”)

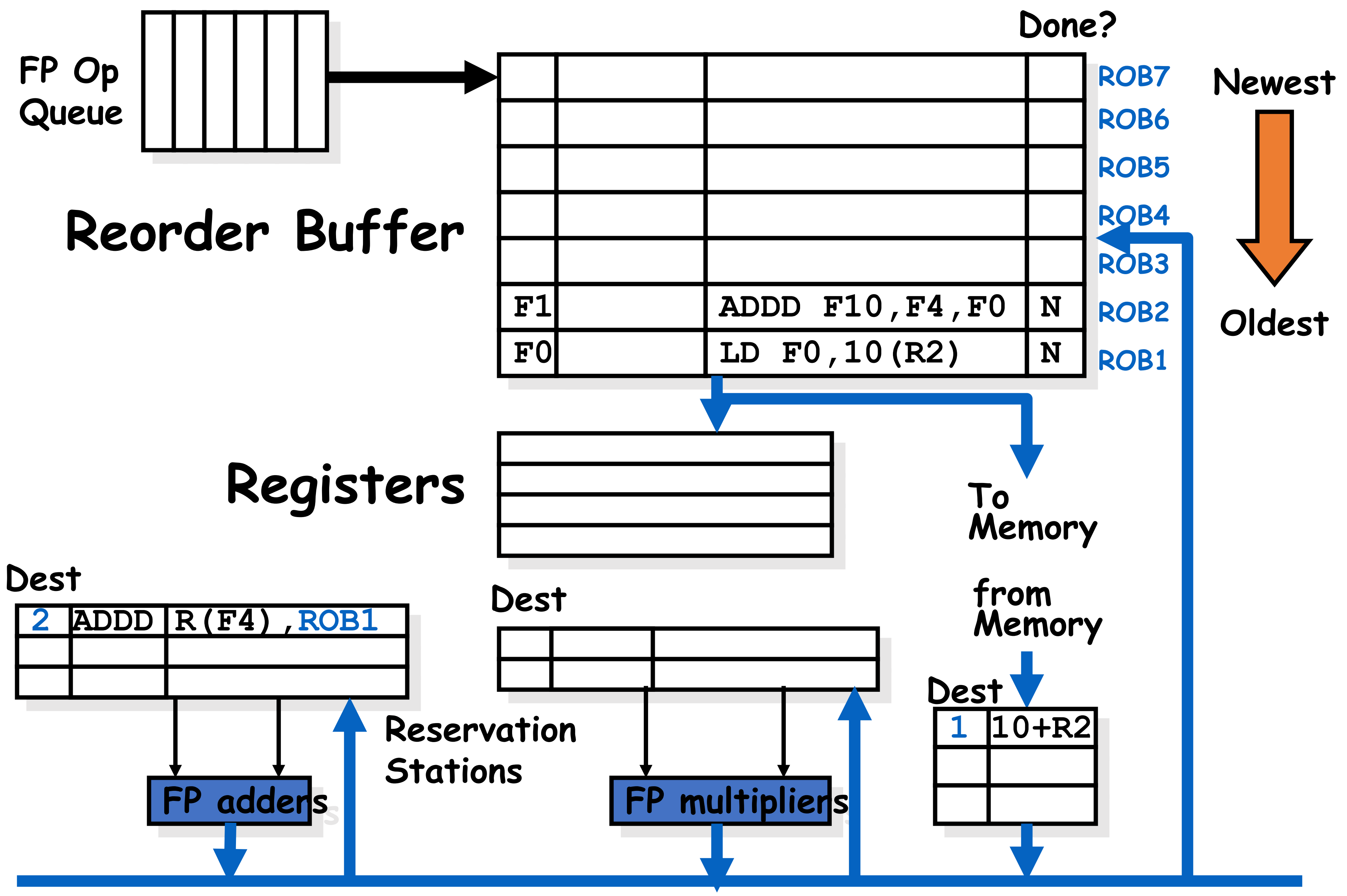
3. Write result—finish execution (WB)

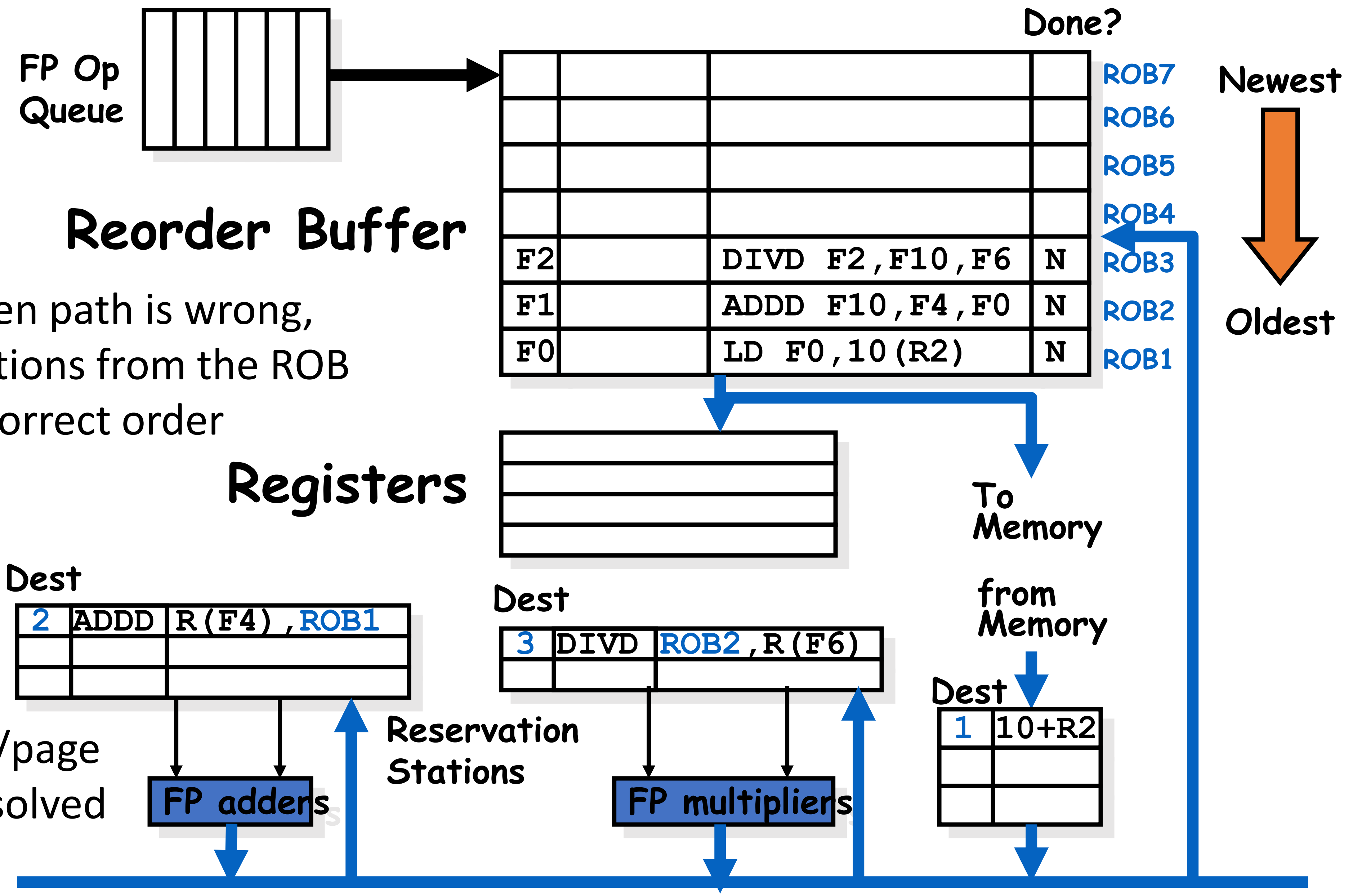
   Write on Common Data Bus to all awaiting FUs
   & reorder buffer; mark reservation station available.

4. Commit—*When instruction reaches head of the ROB,* 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”)


FP Op Queue
Reorder Buffer
Done?
F0
LD F0,10(R2)
N
ROB7
ROB6
ROB5
ROB4
ROB3
ROB2
ROB1
Newest
Oldest
Registers
To Memory
from Memory
Dest
Dest
Dest
1
10+R2
Reservation Stations
FP adders
FP multipliers


FP Op Queue
Reorder Buffer
Done?
ROB7
ROB6
ROB5
ROB4
ROB3
ROB2
ROB1
F1
ADDD F10,F4,F0
N
F0
LD F0,10(R2)
N
Newest
Oldest
Registers
To Memory
from Memory
Dest
2
ADDD
R(F4),ROB1
Dest
Dest
1
10+R2
Reservation Stations
FP adders
FP multipliers



If the predicted taken path is wrong, flush out all instructions from the ROB and reissue in the correct order

Remember exception/page fault handling gets resolved when the instruction reaches the head of the ROB

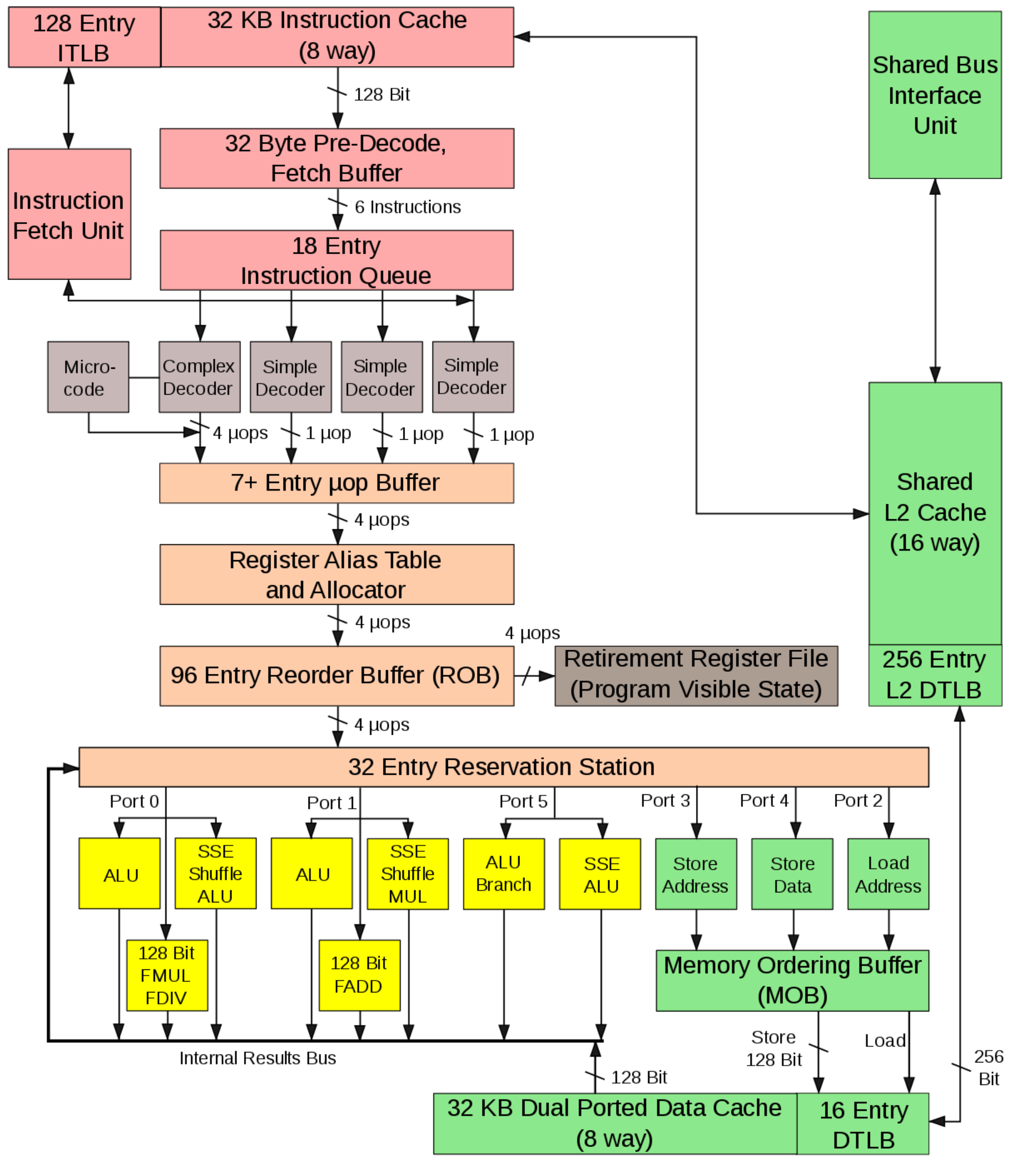
Memory Disambiguation

- 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:
```
st   0(R2),R5
ld   R6,0(R3)
```

- Can we go ahead and start the load early?
  - Answer is that we are not allowed to start load until we know that address 0(R2) $\neq$ 0(R3)

Intel Core Microarchitecture



Intel Core 2 Architecture