EEL 5764 Computer Architecture

Lecture 27: Hardware speculation

Sandip Ray
Department of Electrical and Computer Engineering
University of Florida

Hardware-Based Speculation

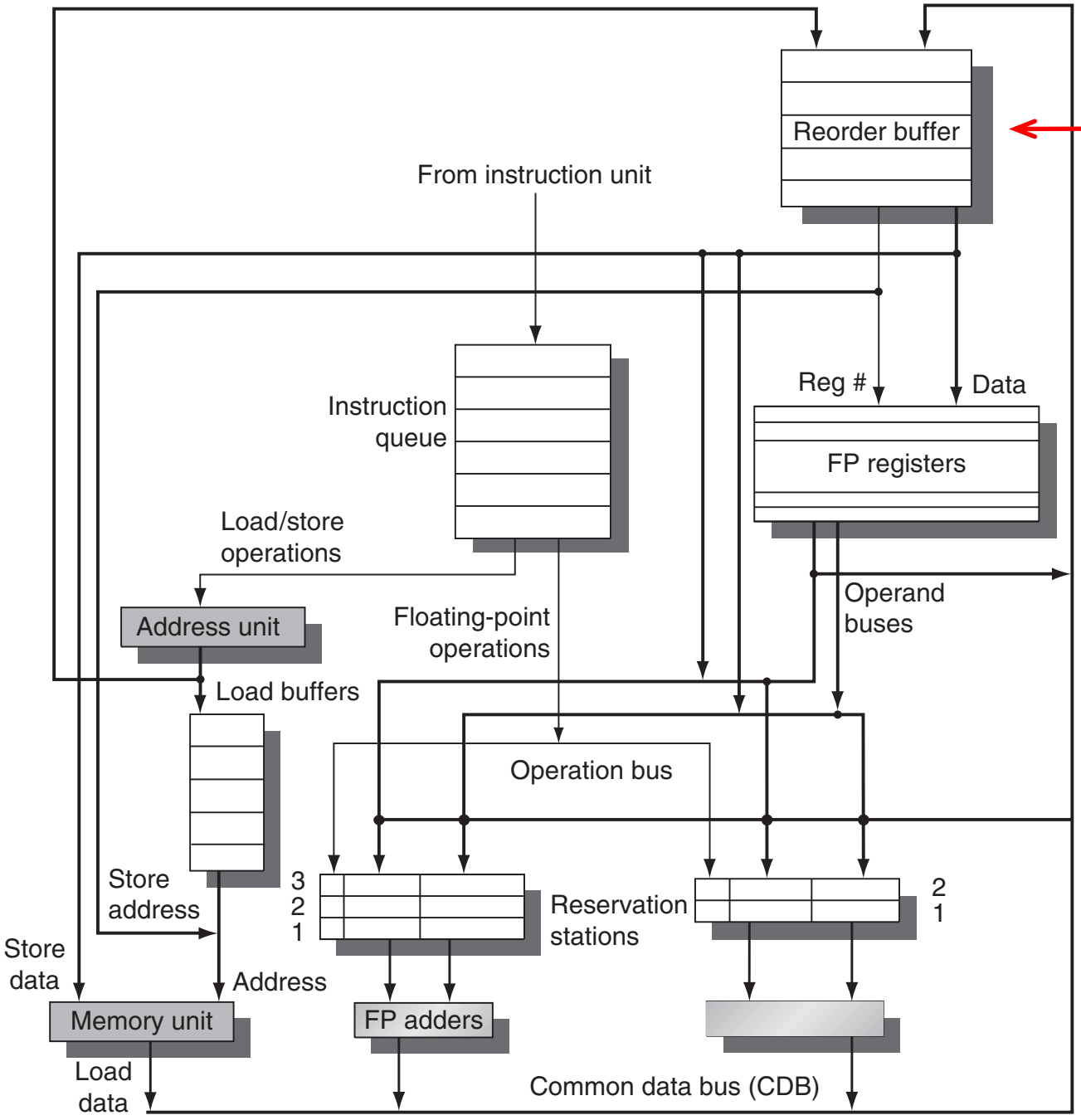
- Combine dynamic scheduling and branch prediction
- Execute instructions along predicted execution paths but only commit the results if prediction was correct
- Instruction commit: allowing an instruction to update the register/mem when instruction is no longer speculative
  →i.e. branch prediction is correct.
- Instruction execute out-of-order, commit in-order
- 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

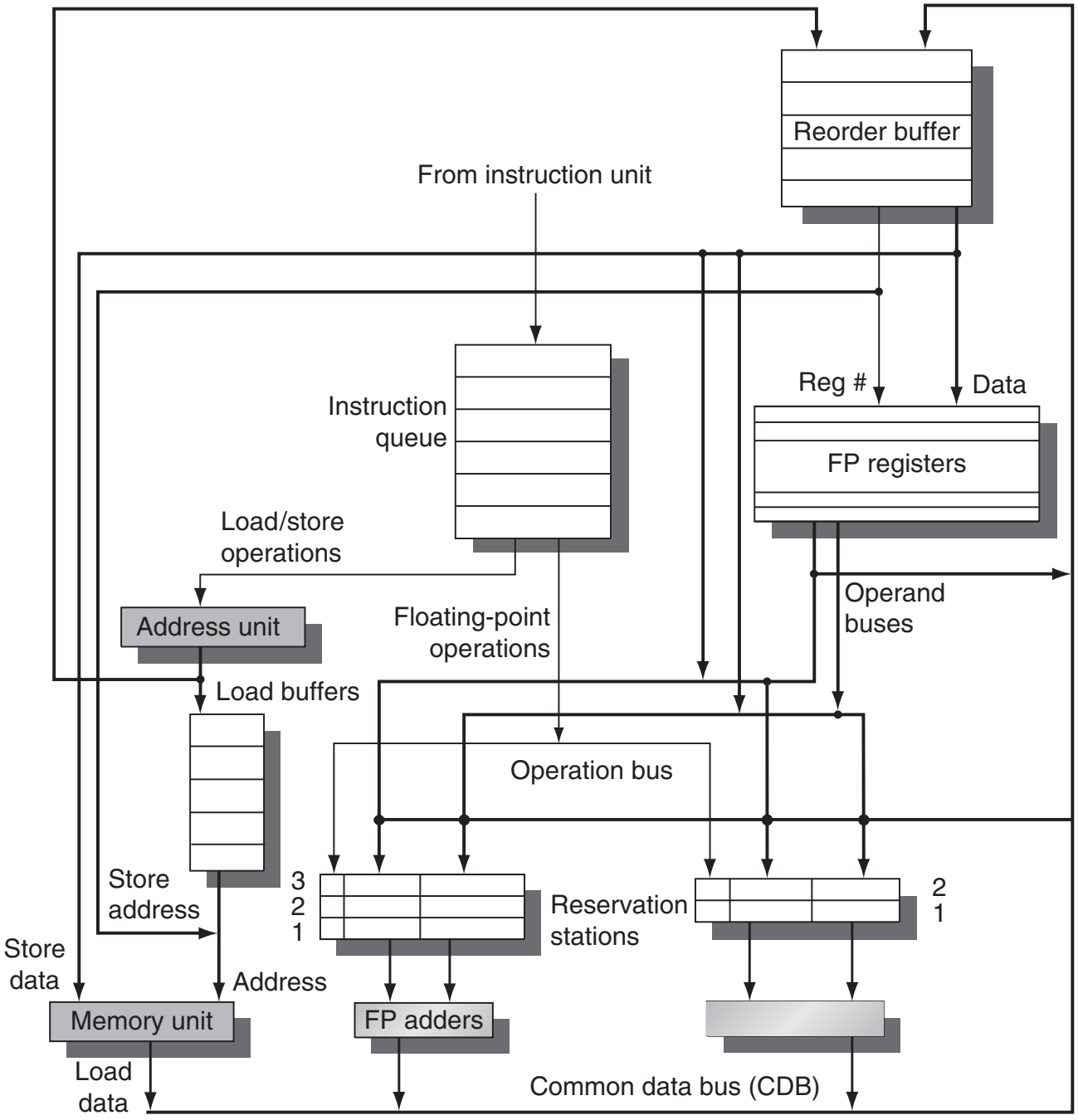
- Reorder buffer (ROB) – holds the result of instruction between completion and commit
- Four fields:
  - → Instruction type: branch/store/register
  - → Destination field: register number/memory address
  - → Value field: output value
  - → Ready field: completed execution?
- Source operands are in
  - → Registers <– committed instructions
  - → ROB <– instructions that complete execution before commit
- Modify reservation stations:
  - → Operand sources (Vj or Vk) are now ROB entries instead of RS#

Reorder Buffer

- Values to registers/memory are not written until an instruction commits
- On mis-prediction:
  → Speculated entries in ROB are cleared
- Exceptions:
  → Not recognized until it is ready to commit

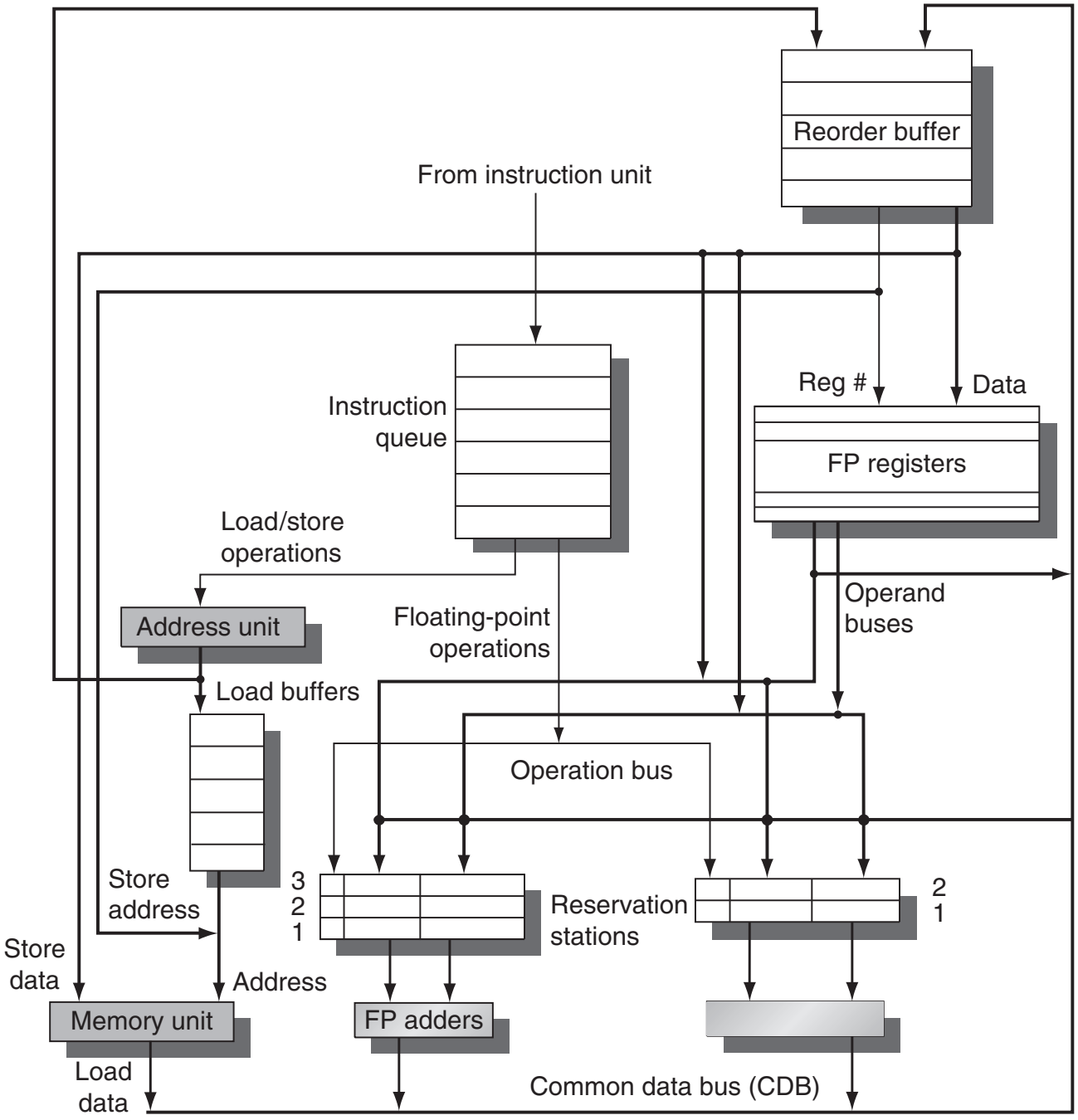


Combined w store buffers – A queue



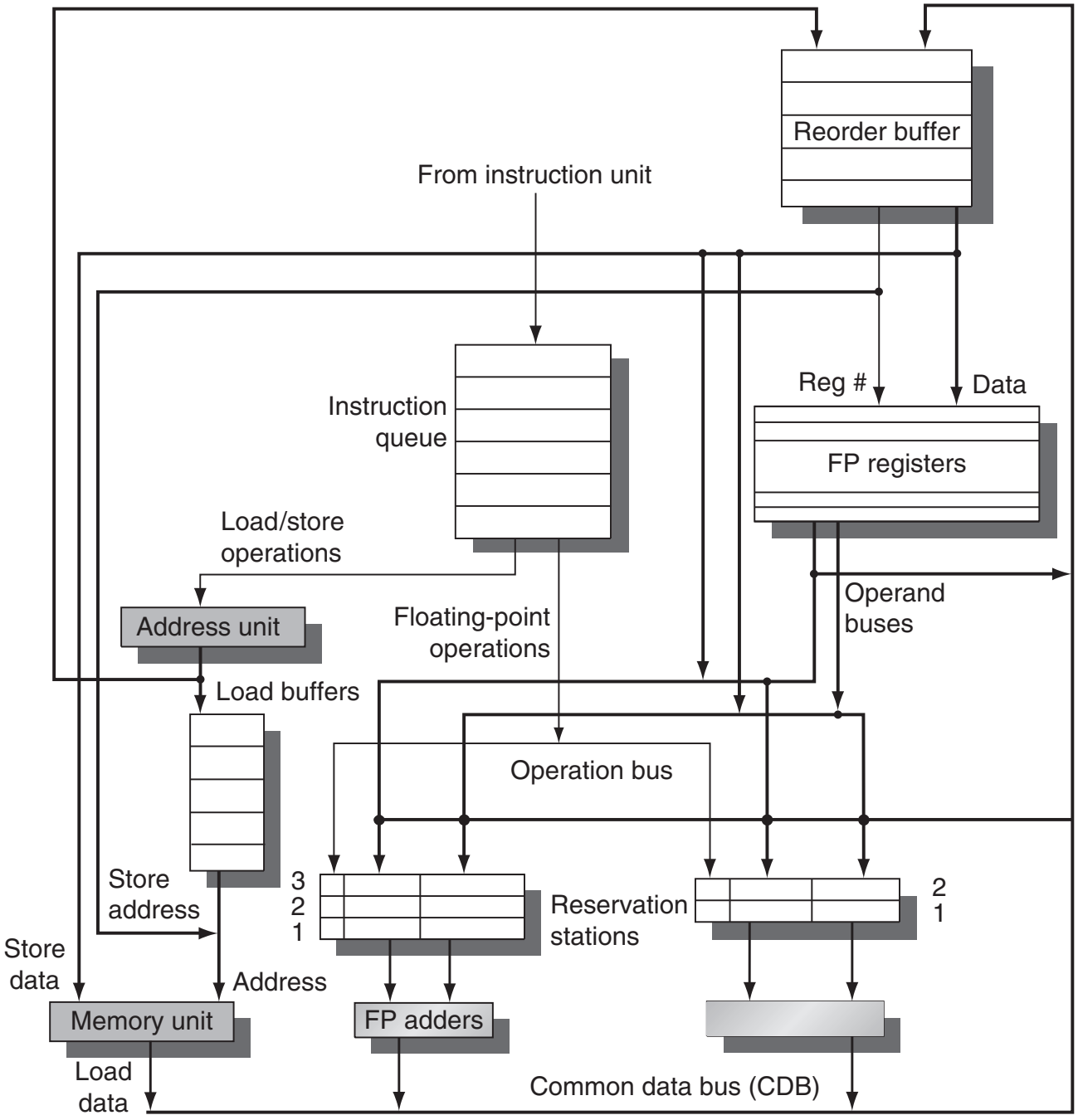
- **Issue** an instruction if there is an empty RS and ROB. Stall otherwise.
- Send the operands to RS if they are available or ROB# if not.
- Tag the instruction result with the ROB# for the destination register

<ROB#, value>



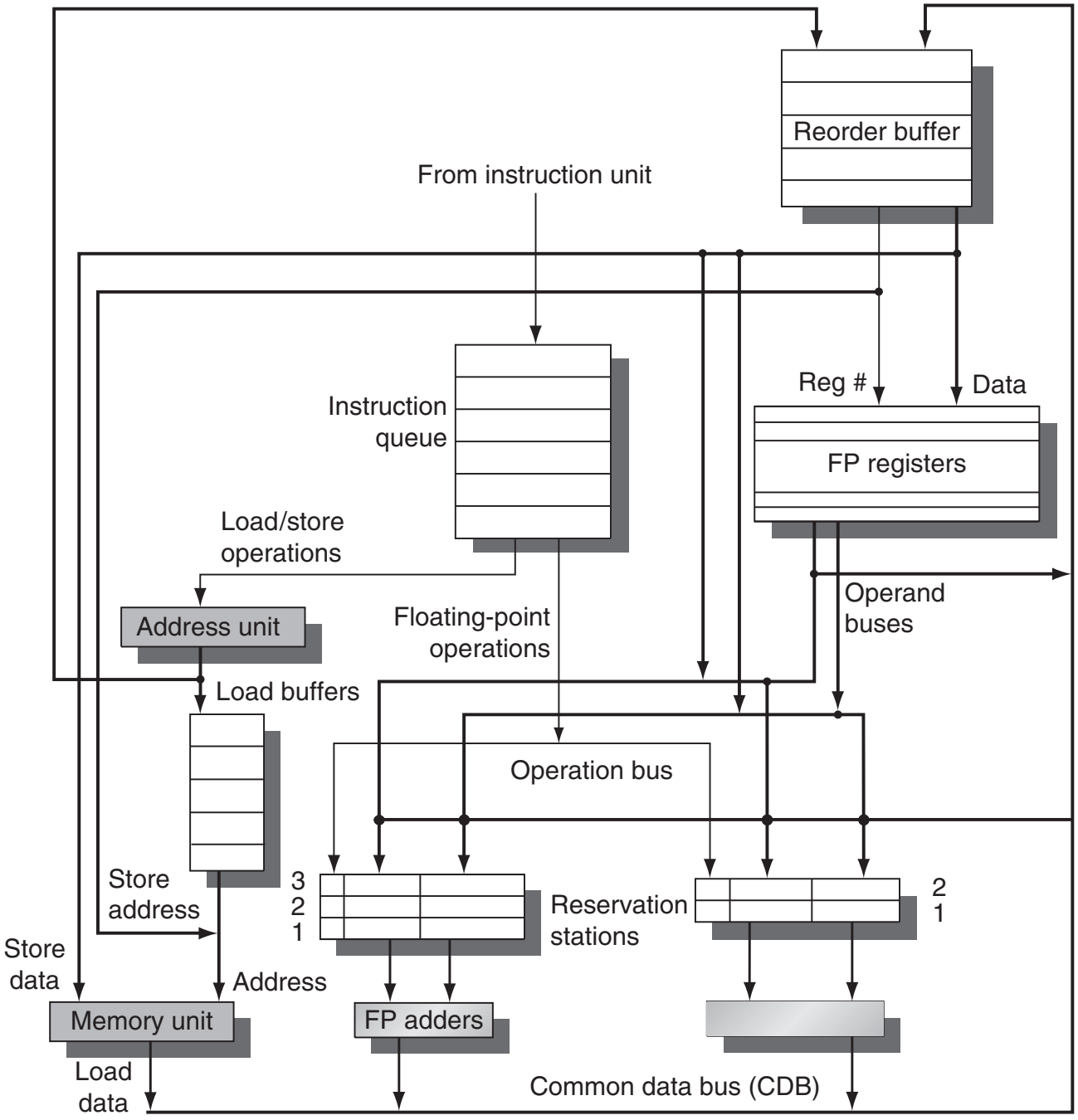
- **Issue**
- **Execute –** start after both operands are available.
- Results are tagged with ROB#

<ROB#, value>



- **Issue**
- **Execute –** start after both operands are available.
- **Write Result –** result put on CDB and goes to ROB and RS
- Any RS or ROB entry expecting a result with ROB# will grab value when <ROB#, value> is put on CDB

<ROB#, value>



- **Issue**
- **Execute**
- **Write Result**
- **Commit –** write result of instruction at the head of ROB
- If the instruction is NOT branch, update M/Reg with the value in ROB
- If the head of ROB is a mis-predicted branch, flush ROB, and restart from the branch target.
- Otherwise, commit as normal.

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	L.D	F6,32(R2)	Commit	F6	Mem[32 + Regs[R2]]
2	No	L.D	F2,44(R3)	Commit	F2	Mem[44 + Regs[R3]]
3	Yes	MUL.D	F0,F2,F4	Write result	F0	#2 × Regs[F4]
4	Yes	SUB.D	F8,F2,F6	Write result	F8	#2 – #1
5	Yes	DIV.D	F10,F0,F6	Execute	F10	
6	Yes	ADD.D	F6,F8,F2	Write result	F6	#4 + #2

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	Dest	A
Load1	No							
Load2	No							
Add1	No							
Add2	No							
Add3	No							
Mult1	No	MUL.D	Mem[44 + Regs[R3]]	Regs[F4]			#3	
Mult2	Yes	DIV.D		Mem[32 + Regs[R2]]	#3		#5	

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F10
Reorder #	3						6		4	5
Busy	Yes	No	No	No	No	No	Yes	. . .	Yes	Yes

Our favorite loop

```
Loop: L.D      F0, 0(R1)
      MUL.D    F4, F0, F2
      S.D      F3, 0(R1)
      DADDIU   R1, R1, -8
      BNE      R1, R2, Loop
```

A Loop Example

Reorder buffer

Entry	Busy	Instruction		State	Destination	Value
1	No	`L.D`	`F0,0(R1)`	Commit	`F0`	`Mem[0 + Regs[R1]]`
2	No	`MUL.D`	`F4,F0,F2`	Commit	`F4`	#1 × `Regs[F2]`
3	Yes	`S.D`	`F4,0(R1)`	Write result	`0 + Regs[R1]`	#2
4	Yes	`DADDIU`	`R1,R1,#-8`	Write result	`R1`	`Regs[R1] – 8`
5	Yes	`BNE`	`R1,R2,Loop`	Write result		
6	Yes	`L.D`	`F0,0(R1)`	Write result	`F0`	`Mem[#4]`
7	Yes	`MUL.D`	`F4,F0,F2`	Write result	`F4`	#6 × `Regs[F2]`
8	Yes	`S.D`	`F4,0(R1)`	Write result	0 + #4	#7
9	Yes	`DADDIU`	`R1,R1,#-8`	Write result	`R1`	#4 – 8
10	Yes	`BNE`	`R1,R2,Loop`	Write result		

FP register status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8
Reorder #	6				7				
Busy	Yes	No	No	No	Yes	No	No	...	No

Hazards Through Memory

- No WAW/WAR hazards by
  - → In-order commit to update reg/memory
- *Store updates MEM in commit, while load reads MEM in execute*
- Avoid RAW hazards through MEM by
  - → Not allowing a load to read Mem if it's a field matches the destination field of any active ROB entry for a store, and
  - → Maintaining the program order of effective address computation of a load *w.r.t.* all earlier stores.

Hardware Speculation – Summary

- Out-of-order execution, in-order commit
- On branch mis-prediction, flush ROB, and processor restart from the branch target.
  → Instructions before the branch are completed as normal
- Branch prediction has higher importance.
  → Impact of mis-prediction is higher
- Instruction exception is delayed until instruction commit
  → Allow precise exception
- WAW/WAR hazards are eliminated by in-order commit
- No RAW hazards with additional control

Multiple Issue

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 Coretex 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

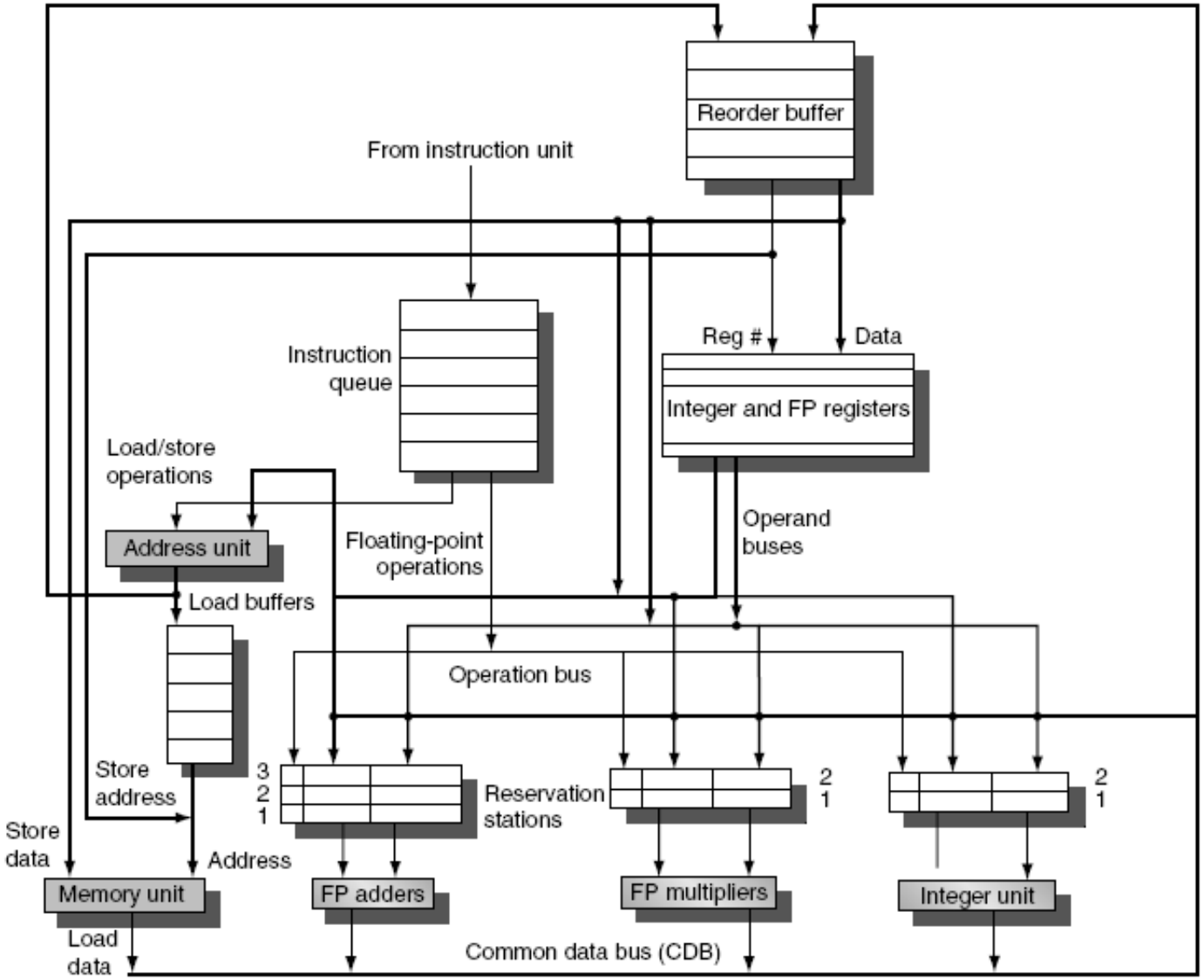
VLIW Processors

- Disadvantages:
  - → Statically finding parallelism – loop unrolling
  - → 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”, and pre-allocate RS and ROB
  - → 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

```
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
```

What happens when issue?

```
        LD      R2, 0(R1)
        DADDIU  R2, R2, #1
```

Example – Multi-Issue, 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 – Multi-Issue w Speculation

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