MORE HARDWARE DESIGNS ON PARALLEL PROCESSING

Lecturer: Hui Annie Guo

h.guo@unsw.edu.au

K17-501F

Lecture overview

- **Topics**
  - **Deep pipeline**
  - **VLIW architecture**
  - **Superscalar architecture**

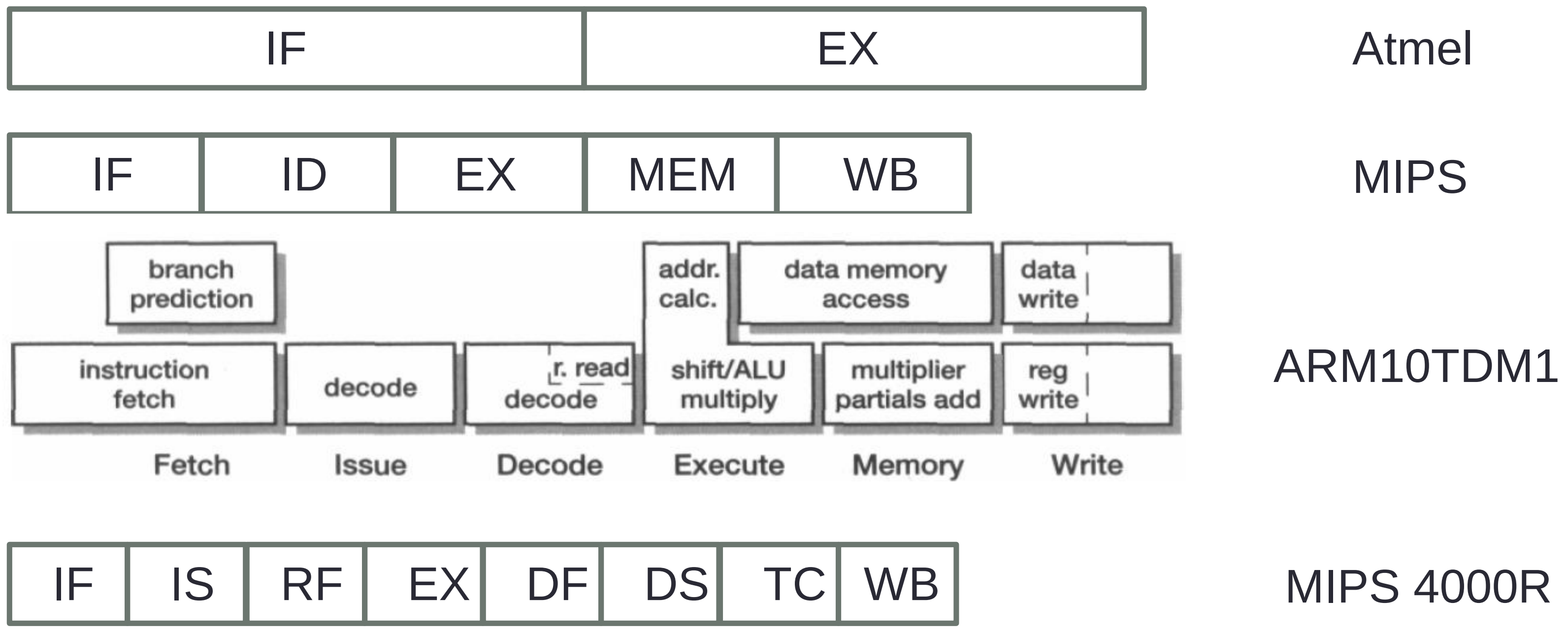
- **Suggested reading**
  - **H&P Chapter 4.10. 4.11**

Improving performance

- Performance can be improved by exploiting parallelism at different design levels
- Data level parallelism
  - Widening the basic word length of the machine
    - 8 bit → 16 bit → 32 bit → 64 bit →…
  - Vector execution
    - Single instruction on multiple data
- Instruction level parallelism
- Thread level
- System level

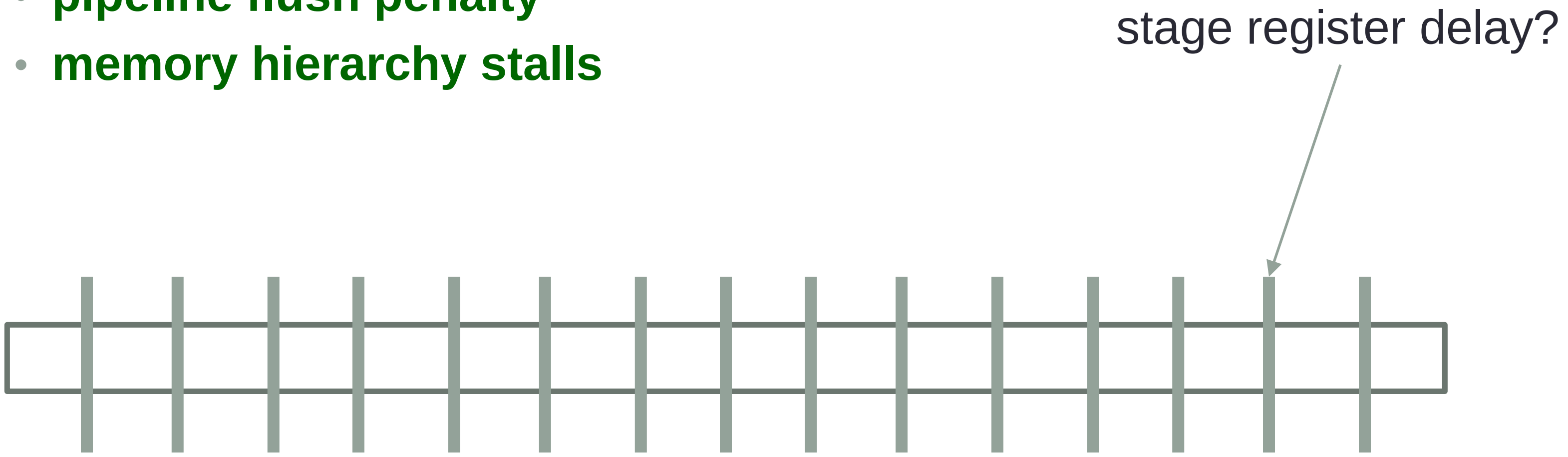
Deep pipeline

- **The depth of the pipeline is often increased to achieve higher clock frequencies.**



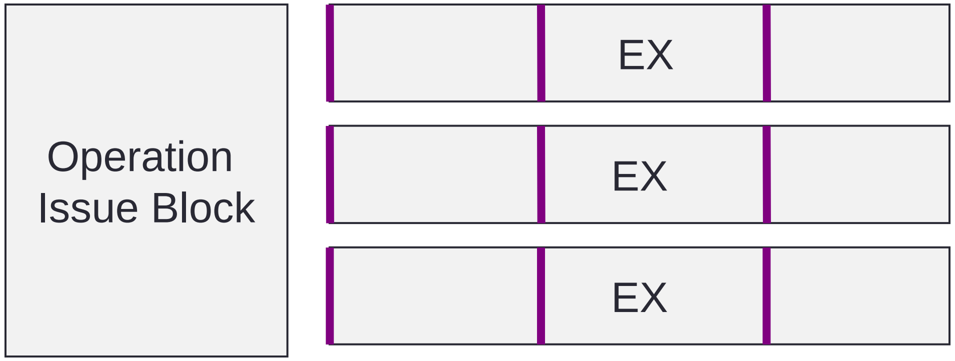
Deep pipeline (cont.)

- Limitations
  - Stage delay cannot be arbitrarily reduced
  - CPI may increase due to
    - pipeline flush penalty
    - memory hierarchy stalls



CPU with parallel processing structure

- **Multiple execution components that can perform simultaneously.**



- **The operation issue block is responsible for supplying instructions to each execution component.**

CPU with parallel processing structure (cont.)

- **The issue block can have different types of implementations**
  - **Software – VLIW architecture**
  - **Hardware – superscalar architecture**

Example-VLIW

$$\sum_{i=1}^{n}(a_i x^i + b_i)$$

```
cnt ←0
sum ←0
ld          ; x

ld          ; $a_i$
ld          ; $b_i$
mul         ; $x^i = x^{i-1}*x$
mul         ; $x^i *a_i$
add         ; $x^i *a_i + b_i$
add         ; sum = sum + $x^i *a_i + b_i$
inc         ; cnt++
beq         ; cnt == n?
```

type of instructions used

Operations involved

1st loop iteration:

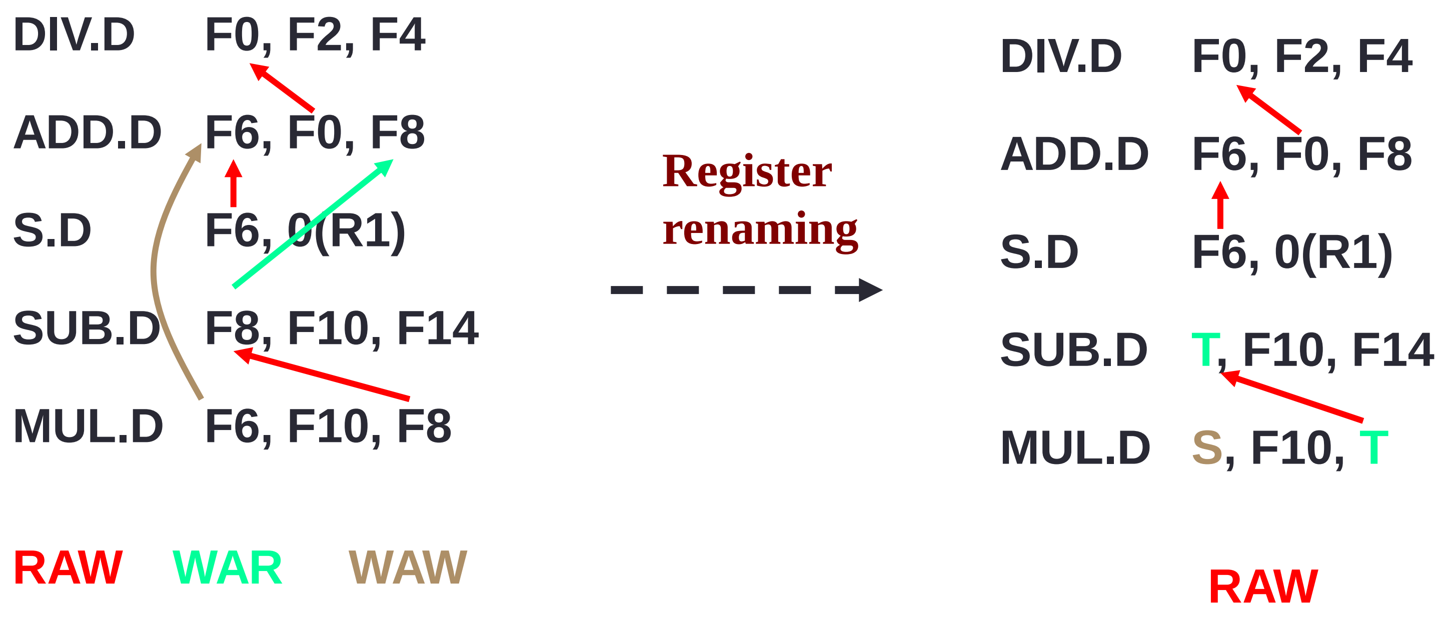
ALU	ALU	MU
nop	nop	ld x
nop	$x^{i-1}*x$	ld a_i
nop	$x^i *a_i$	ld b_i
cnt++	$x^i *a_i + b_i$	nop
beq	sum	nop

Challenges in multiple issue machines

- More instructions executing in parallel
  - May create more data hazards
  - Forwarding in the pipelined datapath becomes hard
  - Identifying parallel instructions is not easy
- More aggressive scheduling required

Example

- Convert the following sequence of instructions into parallel processing
  - Data dependency should be maintained



```
DIV.D   F0, F2, F4
ADD.D   F6, F0, F8
S.D     F6, 0(R1)
SUB.D   F8, F10, F14
MUL.D   F6, F10, F8
```

RAW WAR WAW

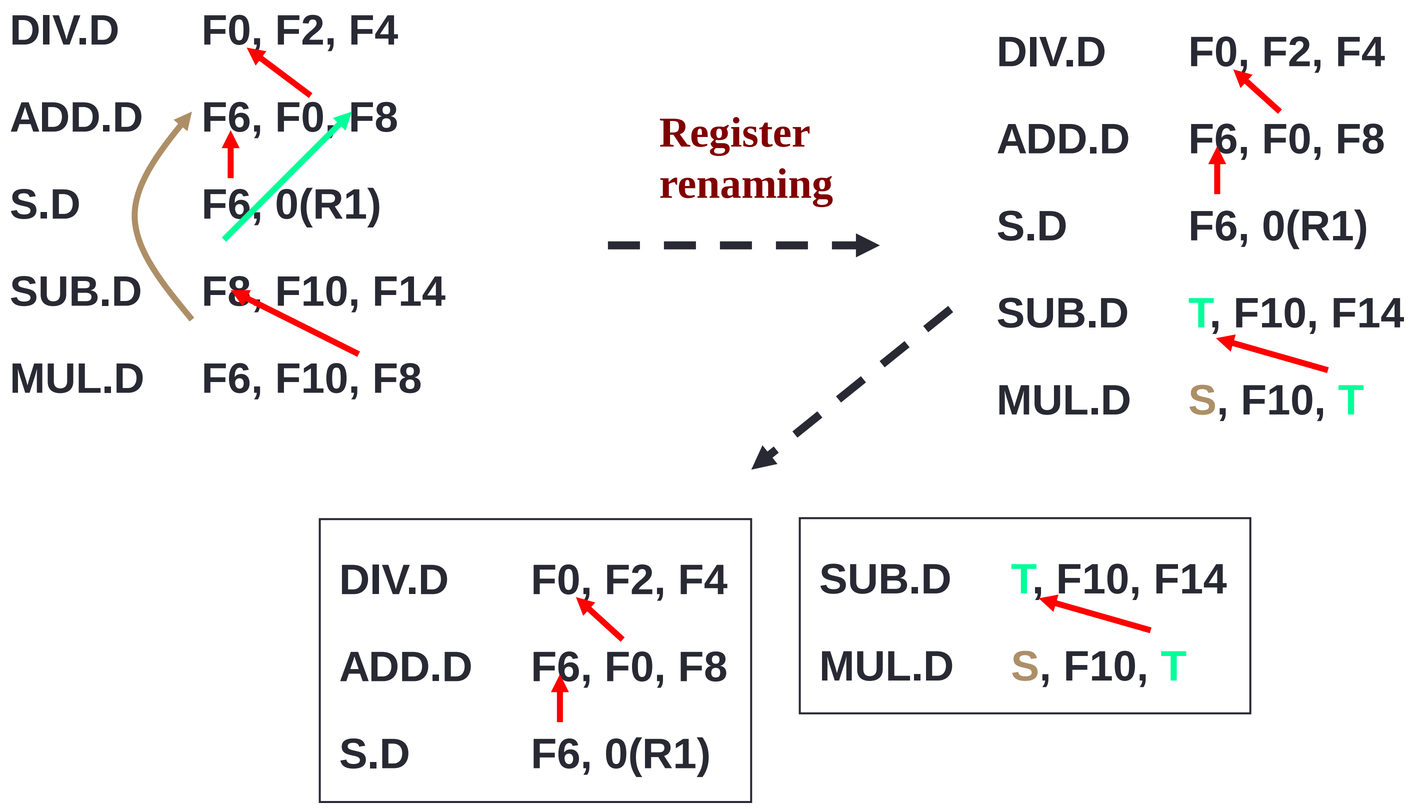
Register renaming

```
DIV.D   F0, F2, F4
ADD.D   F6, F0, F8
S.D     F6, 0(R1)
SUB.D   T, F10, F14
MUL.D   S, F10, T
```

RAW

Example (cont.)

- **After register renaming, a single sequence of instructions can be transformed into two parallel sequences. Instructions from different sequences can be executed in parallel.**



```
DIV.D    F0, F2, F4
ADD.D    F6, F0, F8
S.D      F6, 0(R1)
SUB.D    F8, F10, F14
MUL.D    F6, F10, F8
```

```
DIV.D    F0, F2, F4
ADD.D    F6, F0, F8
S.D      F6, 0(R1)
SUB.D    T, F10, F14
MUL.D    S, F10, T
```

```
DIV.D    F0, F2, F4
ADD.D    F6, F0, F8
S.D      F6, 0(R1)
```

```
SUB.D    T, F10, F14
MUL.D    S, F10, T
```

Dynamic scheduling

- During execution, the hardware issue component in the processor schedules instructions to different parallel execution units
- Basic idea:
  - Tracking instruction dependencies to allow instruction execution as soon as the operands are available
  - Renaming registers to avoid WAR and WAW hazards

Three steps in dynamic scheduling

- **Issue**
- **Execute**
- **Write result**

Issue

- Get next instruction from instruction queue
- Issue the instruction and related available operands from the register file to a matching reservation station entry if it is available; otherwise stall the instruction
  - in order issue

Execute

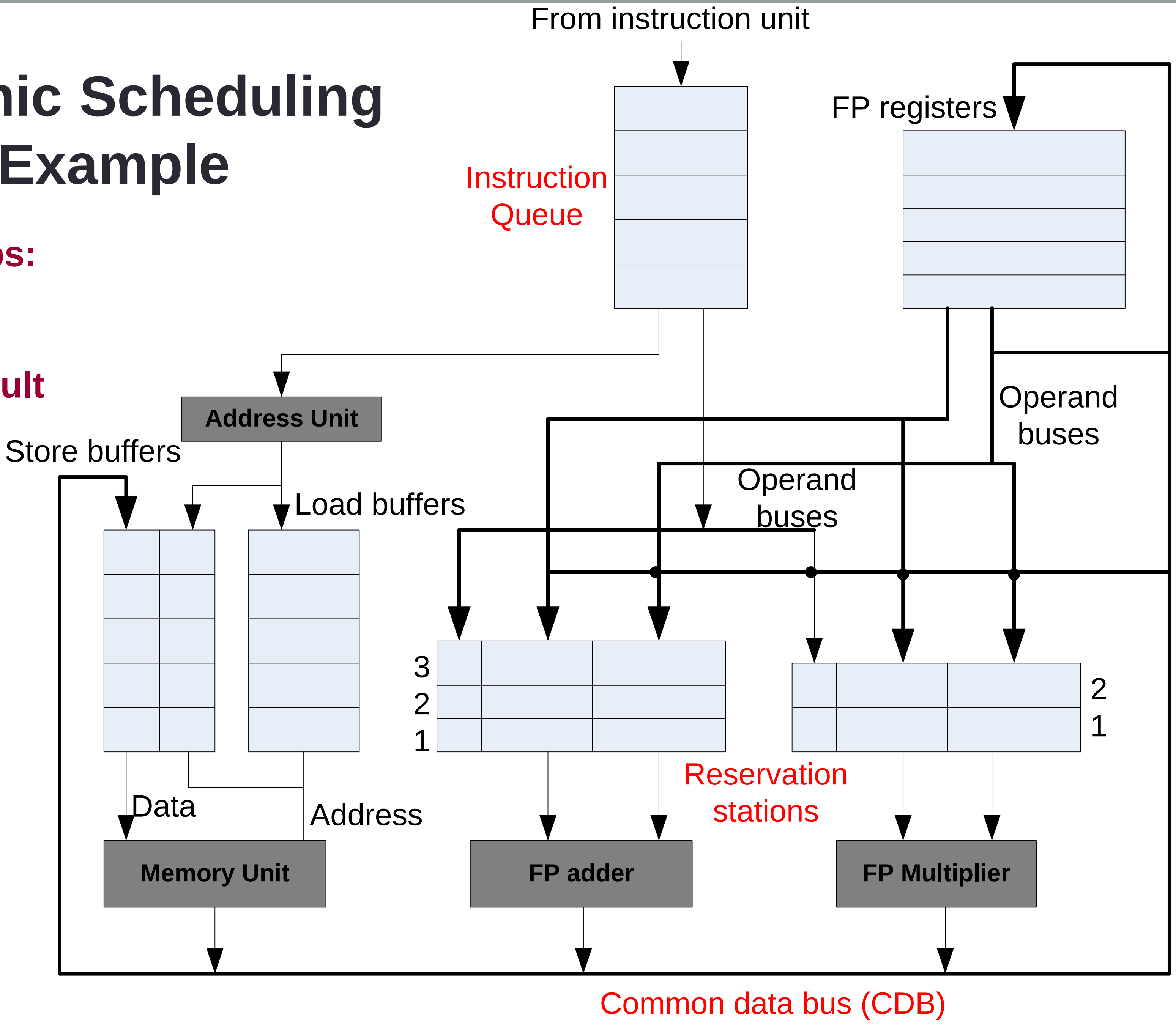
- **Execute ready instructions in the reservation stations**
- **Monitor the common data bus (CDB) for the operands of not-ready instructions**
- **If no ready instruction for an execution unit, the execution unit is idle**

Write result

- Results from execution units are sent through CDB (common data bus) to destinations
  - Reservation station
  - Memory load buffers
  - Register file
- The write operations to the destination should be controlled to avoid hazards

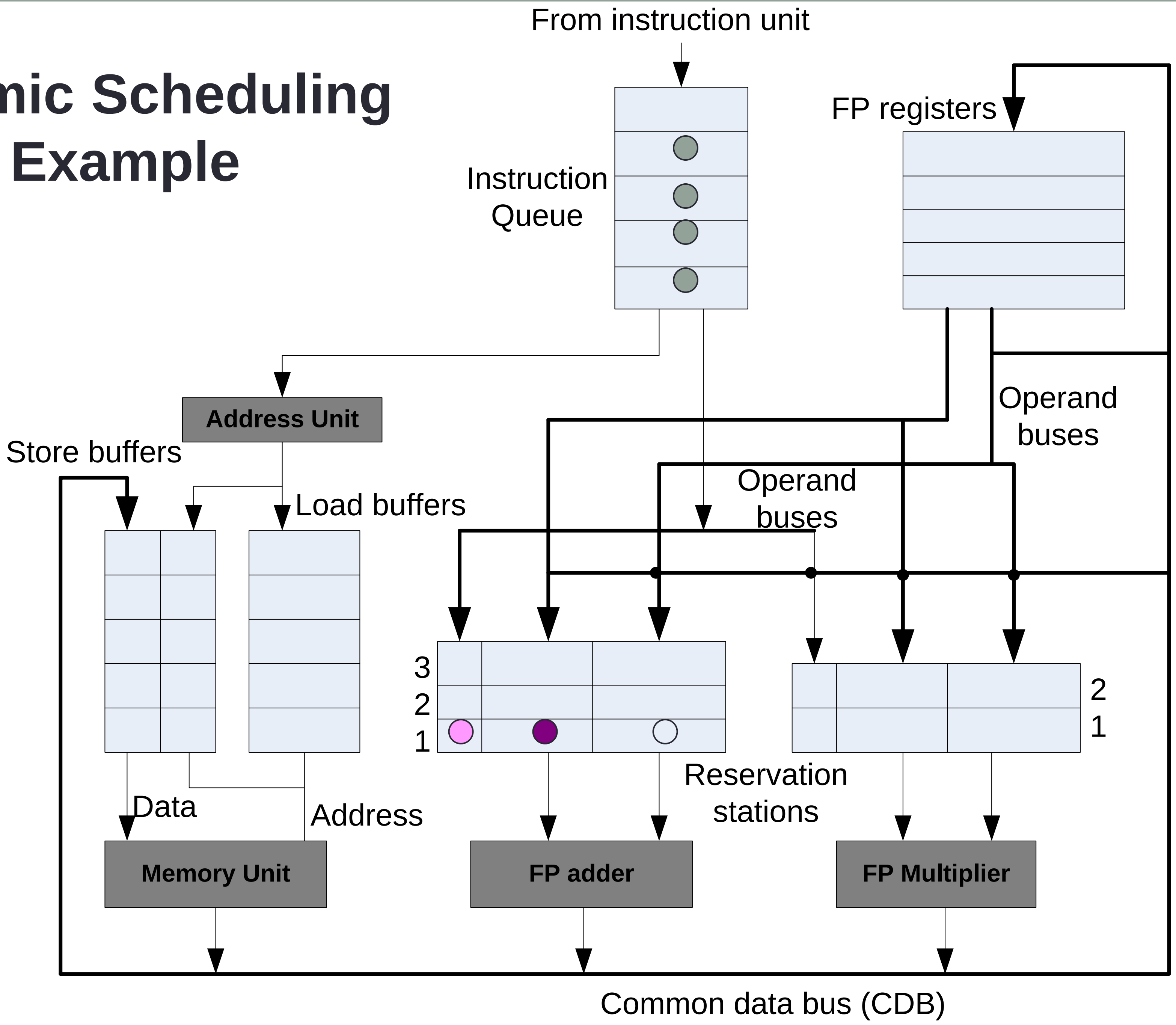
Dynamic Scheduling Example

Three steps:
- **Issue**
- **Execute**
- **Write Result**



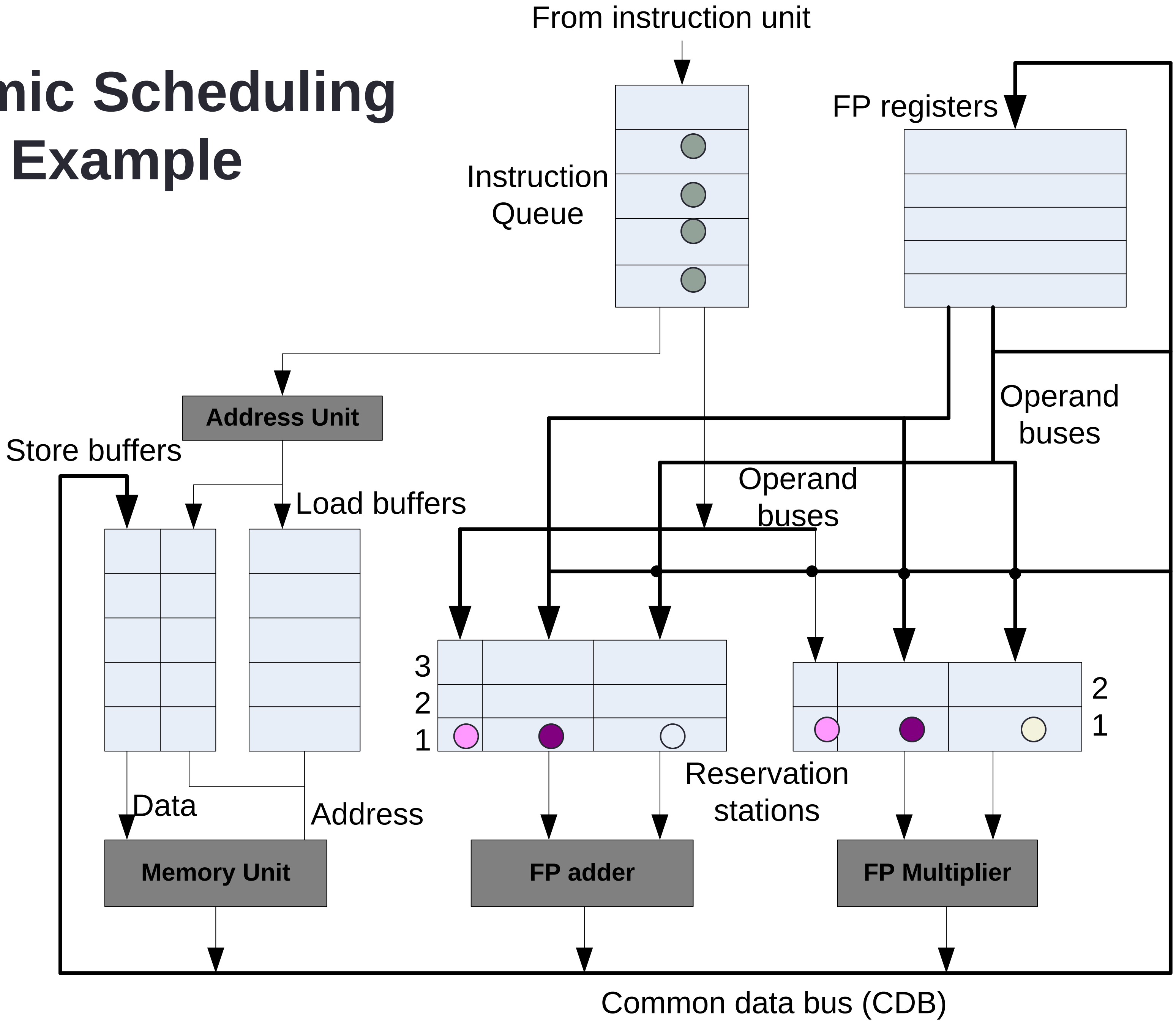
Dynamic Scheduling Example

Issue



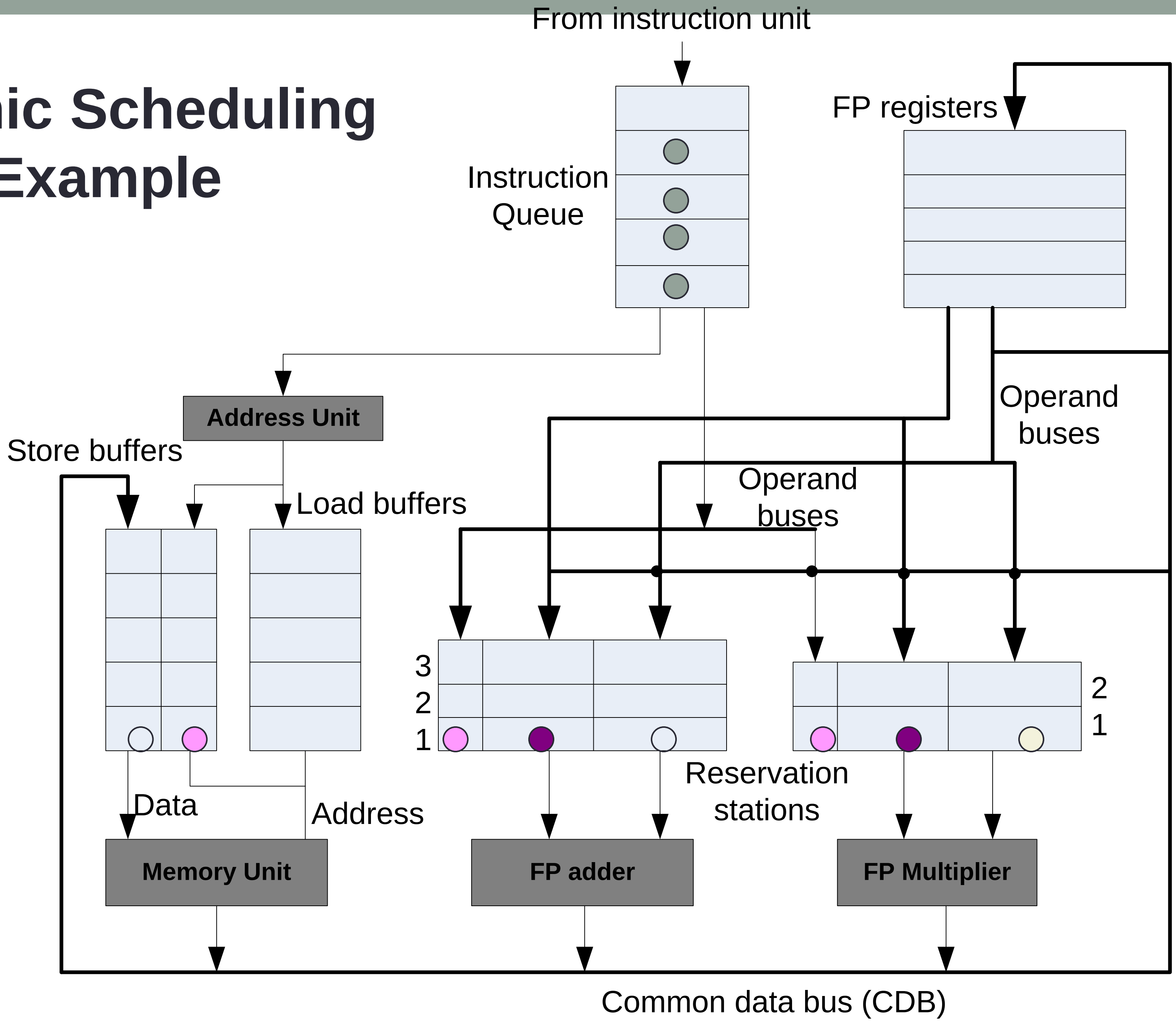
Dynamic Scheduling Example

Issue



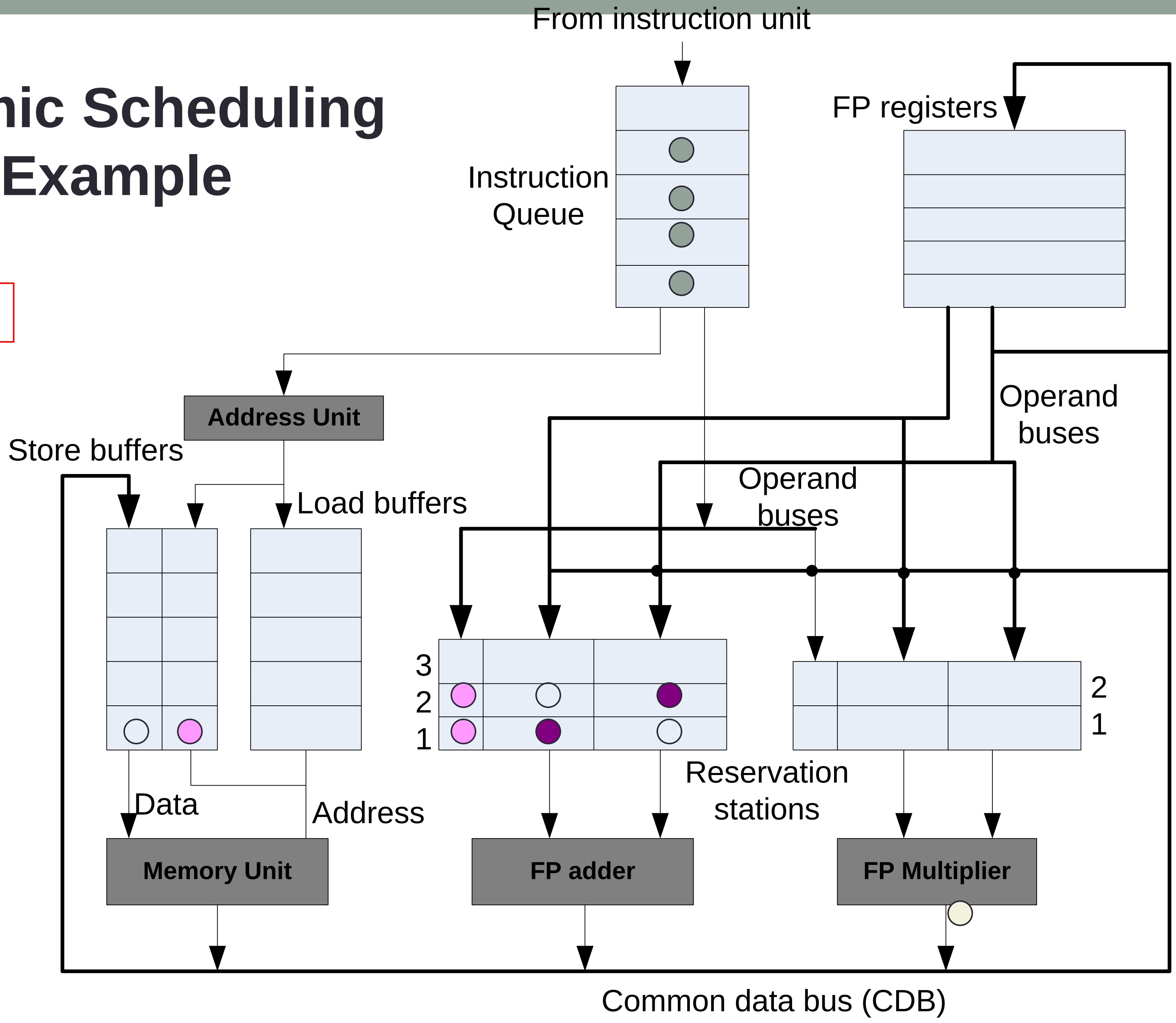
Dynamic Scheduling Example

Issue



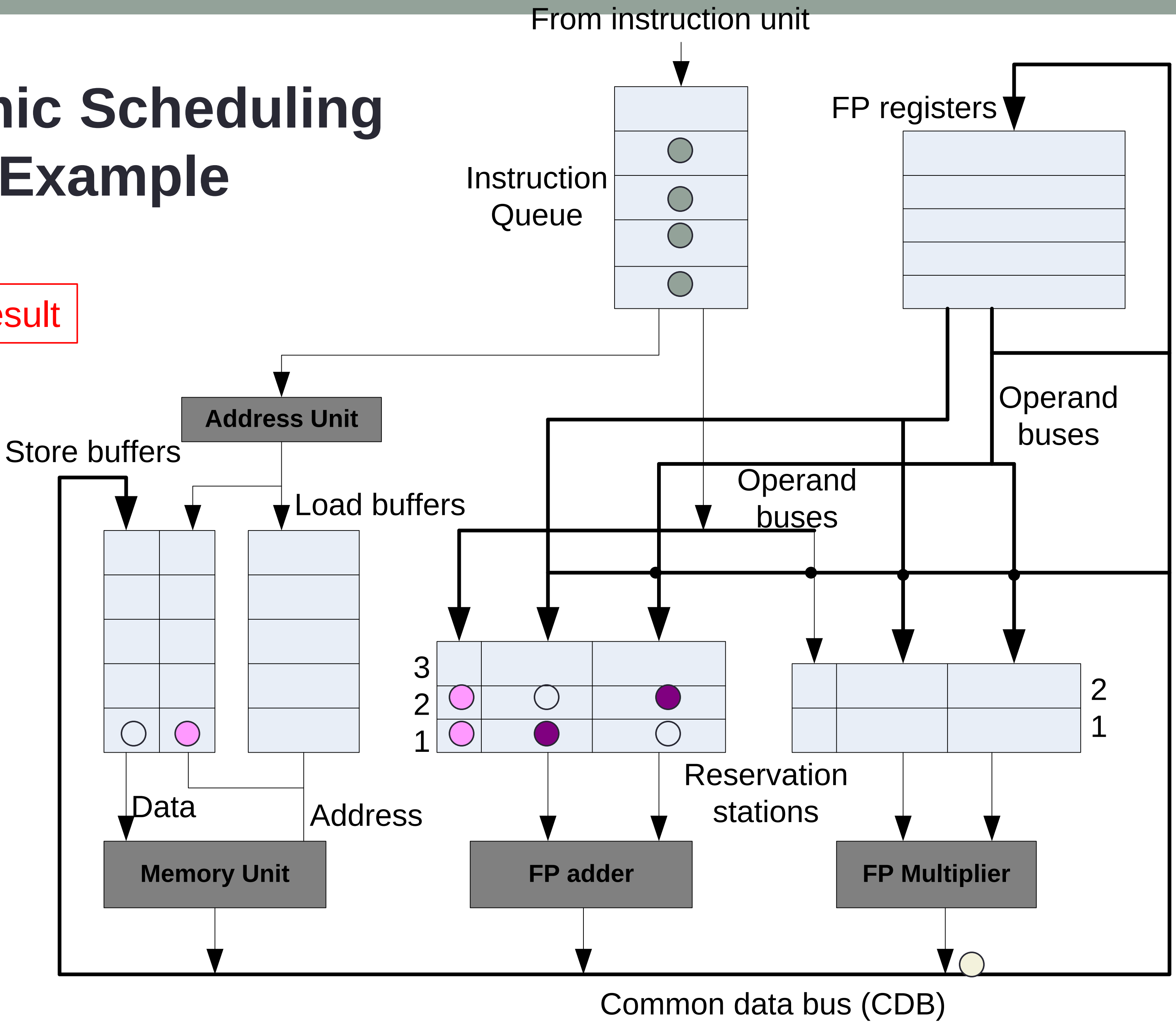
Dynamic Scheduling Example

Execute



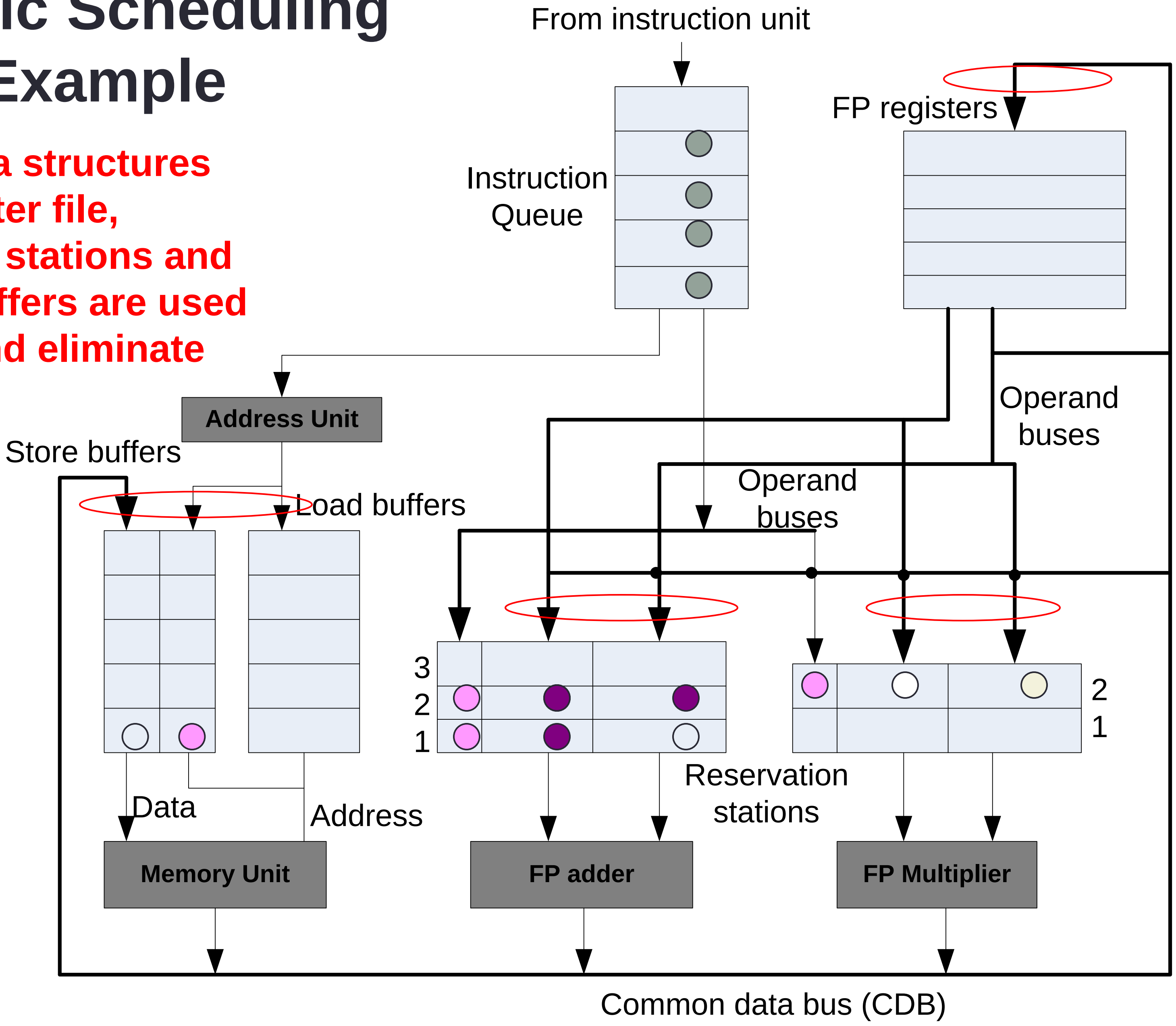
Dynamic Scheduling Example

Write Result



Dynamic Scheduling Example

Special data structures in the register file, reservation stations and memory buffers are used to detect and eliminate hazards



Structure of reservation station

- **A reservation station holds information for an issued instruction**
  - **Each entry in the reservation table contains a number of fields for an instruction to be executed**
    - **Busy: the availability of the entry**
    - **Op: the type of operation to be performed**
    - **Vj and Vk: to hold real operand values**
    - **Qj and Qk: to hold the location of the instruction generating the required operand**
    - **A: address for memory access**
  - **Each entry has an index to identify an instruction issued.**

index	busy	Op	Vj	Vk	Qj	Qk	A

Example: 34+?

1000	yes	add	34	null	null	0011	null

add1

To be determined by the instruction in the reservation station with index 0011

State table for register file

- **Each register in the register file has a state:**

	Register Status												
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11 …**	**F31**
Qi													

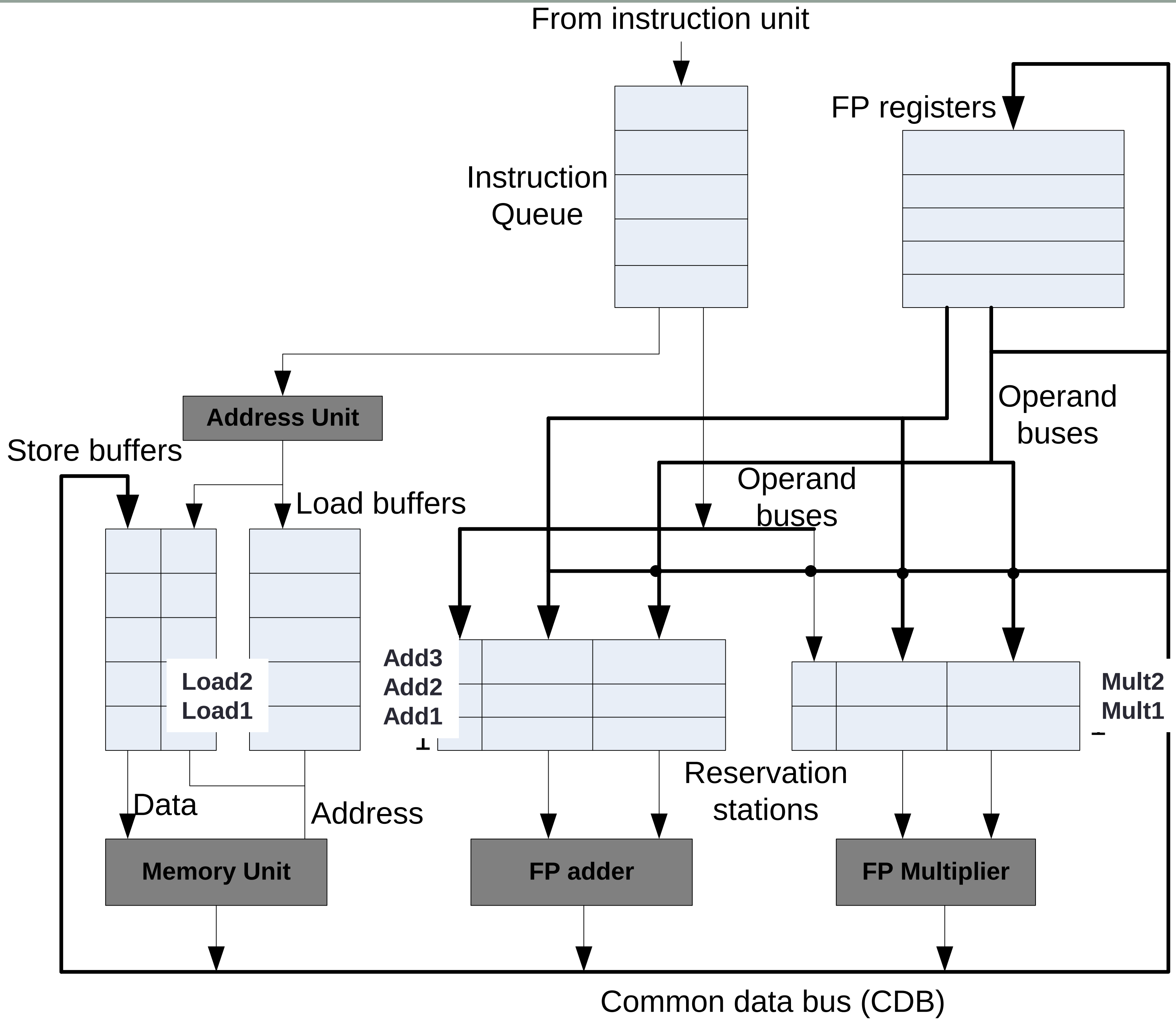
Example

	Register Status												
Field	**F0**	**F1**	**F2**	**F3**	**F4**	**F5**	**F6**	**F7**	**F8**	**F9**	**F10**	**F11 …**	**F31**
Qi				add1				load2					

Dynamic scheduling example

- **Given the superscalar datapath shown in the next slide, show how the station tables are updated for the following code sequence after the first load has completed and its result has just been saved, namely**
  - **Complete the reservation stations and the register state table for the issues of rest instructions**

```
L.D      F6, 34(R2)
L.D      F2, 45(R3)
MUL.D    F0, F2, F4
SUB.D    F8, F2, F6
DIV.D    F10, F0, F6
ADD.D    F6, F8, F2
```


From instruction unit
Instruction Queue
FP registers
Address Unit
Store buffers
Load buffers
Operand buses
Operand buses
Load2
Load1
Add3
Add2
Add1
Mult2
Mult1
Data
Address
Reservation stations
Memory Unit
FP adder
FP Multiplier
Common data bus (CDB)

Instruction		Status issue	execute	write-result
L.D	F6, 34(R2)	X	X	X
L.D	F2, 45(R3)	X	X	
MUL.D	F0, F2, F4	X		
SUB.D	F8, F2, F6	X		
DIV.D	F10, F0, F6	X		
ADD.D	F6, F8, F2	X		

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	**yes**	**Load**					**45+R3**
Add1	**yes**	**SUB**		**F6**	**Load2**		
Add2	**yes**	**ADD**			**Add1**	**Load2**	
Add3							
Mult1	**yes**	**MUL**		**F4**	**Load2**		
Mult2	**yes**	**DIV**		**F6**	**Mult1**		

Register Status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11 …	F31
Qi	**Mult1**		**Load2**				**Add2**		**Add1**		**Mult2**		

Instruction		Status issue	Status execute	Status write-result
L.D	F6, 34(R2)	X	X	X
L.D	F2, 45(R3)	X	X	X
MUL.D	F0, F2, F4	X		
SUB.D	F8, F2, F6	X		
DIV.D	F10, F0, F6	X		
ADD.D	F6, F8, F2	X		

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	**yes**	**Load**					**45+R3**
Add1	**yes**	**SUB**		**F6**	**Load2**		
Add2	**yes**	**ADD**			**Add1**	**Load2**	
Add3							
Mult1	**yes**	**MUL**		**F4**	**Load2**		
Mult2	**yes**	**DIV**		**F6**	**Mult1**		

Register Status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11 …	F31
Qi	**Mult1**		**Load2**				**Add2**		**Add1**		**Mult2**		

Instruction		issue	execute	write-result
L.D	F6, 34(R2)	X	X	X
L.D	F2, 45(R3)	X	X	X
MUL.D	F0, F2, F4	X	X	
SUB.D	F8, F2, F6	X	X	X
DIV.D	F10, F0, F6	X		
ADD.D	F6, F8, F2	X	X	

In order issue, out of order execution!

Reservation stations

Name	Busy	Op	Vj	Vk	Qj	Qk	A
Load1	no						
Load2	no	Load					45+R3
Add1	no	SUB		F6	Load2		
Add2	yes	ADD			Add1	Load2	
Add3							
Mult1	yes	MUL		F4	Load2		
Mult2	yes	DIV		F6	Mult1		

Register Status

Field	F0	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11 …	F31
Qi	Mult1		Load2				Add2		Add1		Mult2		

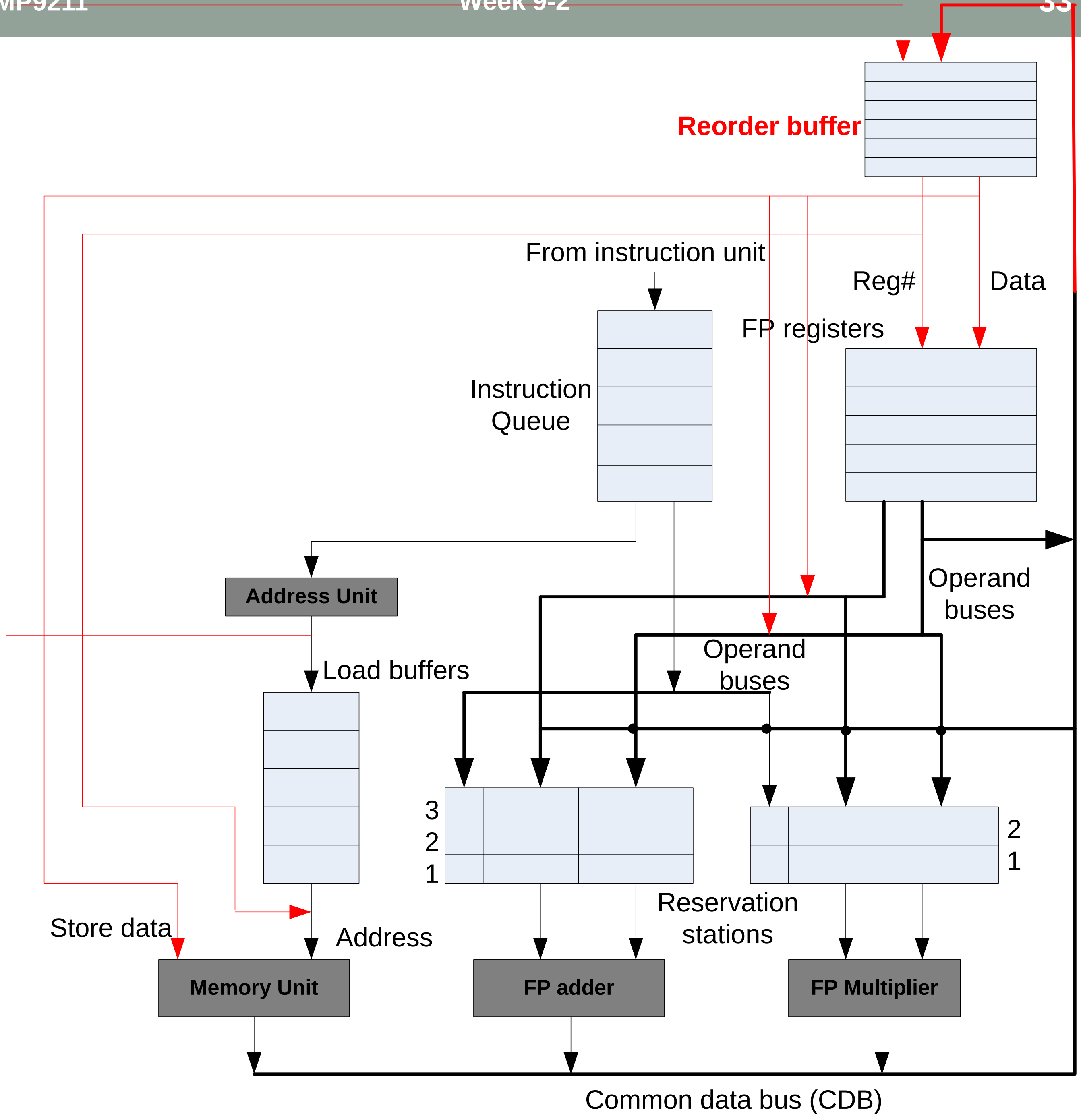
What will happen for the following code?

Instruction		Status		
		issue	execute	write-result
L.D	F6, 34(R2)	X	X	X
L.D	F2, 45(R3)	X	X	
L.D	F0, 20(R1)	X	X	
BEQ	F0, F2, D1	X		
SUB.D	F8, F4, F6	X	X	X
DIV.D	F10, F0, F6	X		
ADD.D	F6, F8, F2	X		
…				
D1: …		X		

Dynamic execution with speculation

- **Four steps**
  - **Issue**
  - **Execute**
  - **Write result**
  - **Commit**

- **Key idea**
  - **To allow instructions to execute out of order**
  - **But force them to commit in correct execution order**
  - **Prevent any irrevocable actions**

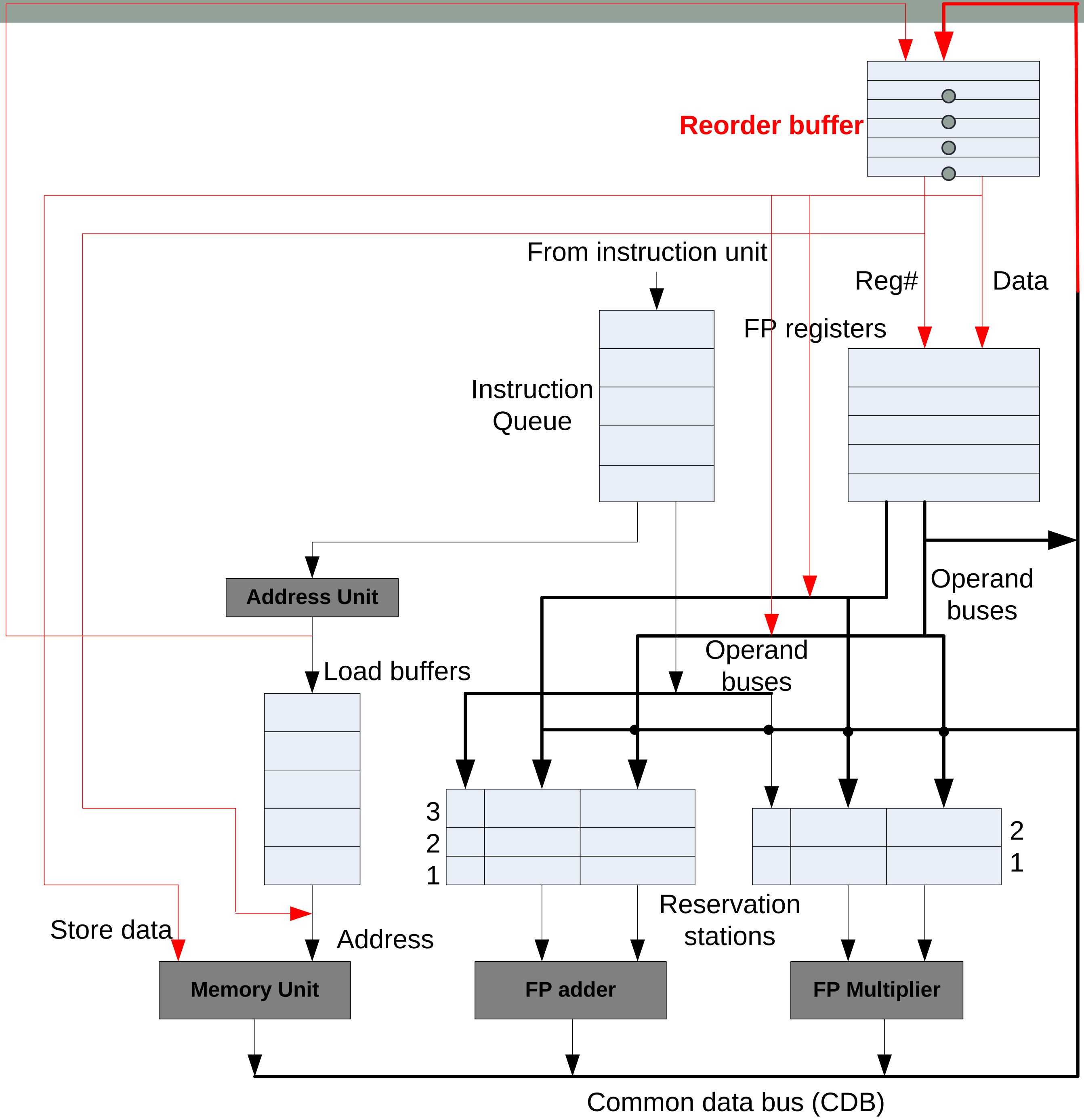

Reorder buffer
From instruction unit
Reg#
Data
FP registers
Instruction Queue
Address Unit
Operand buses
Operand buses
Load buffers
3
2
1
2
1
Reservation stations
Store data
Address
Memory Unit
FP adder
FP Multiplier
Common data bus (CDB)

Reorder Buffer (ROB)

- Reorder Buffer contains four fields
  - Instruction type
    - Branch (has no destination result)
    - Store (with memory address)
    - ALU Op
  - Destination
    - Register (for load and ALU operations)
  - Value
  - Ready status
    - The instruction has completed execution and the value is ready
- Demonstration of the commit operation is given in the next slides

<u>Commit</u>
Case: 1/4
ALU Instr.

✓

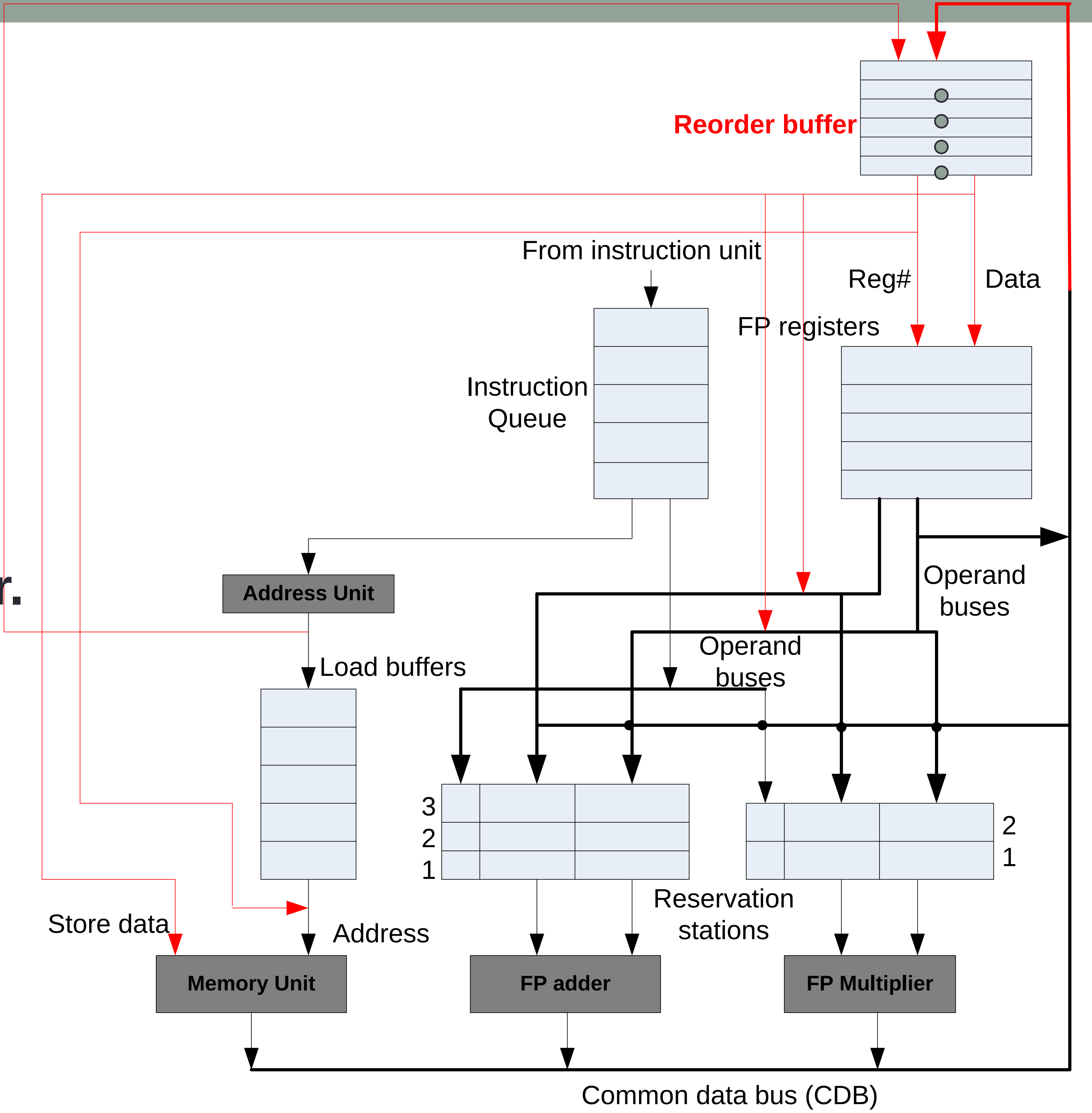


Commit

Case: 2/4

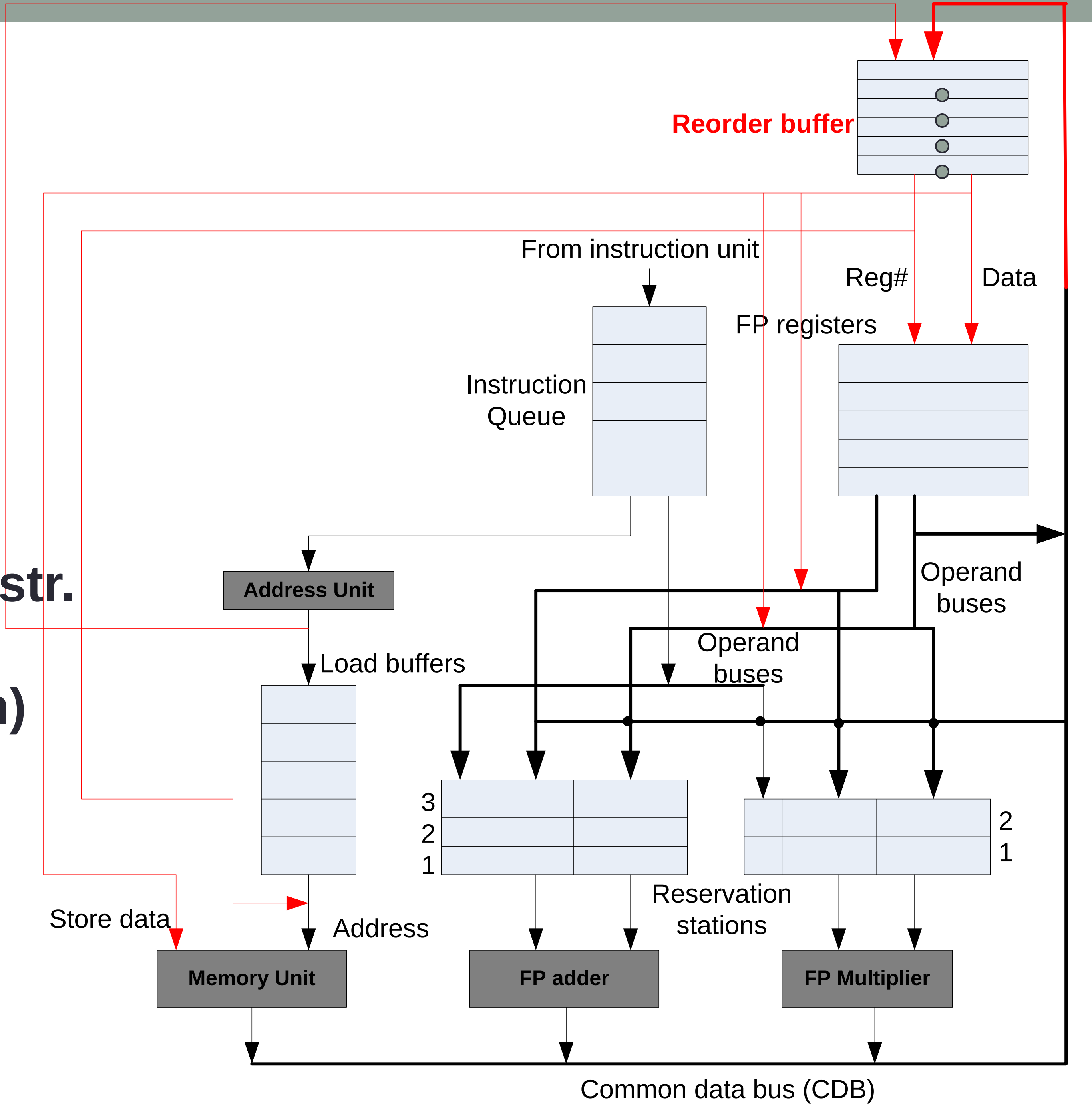
Store instr.

✓

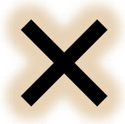


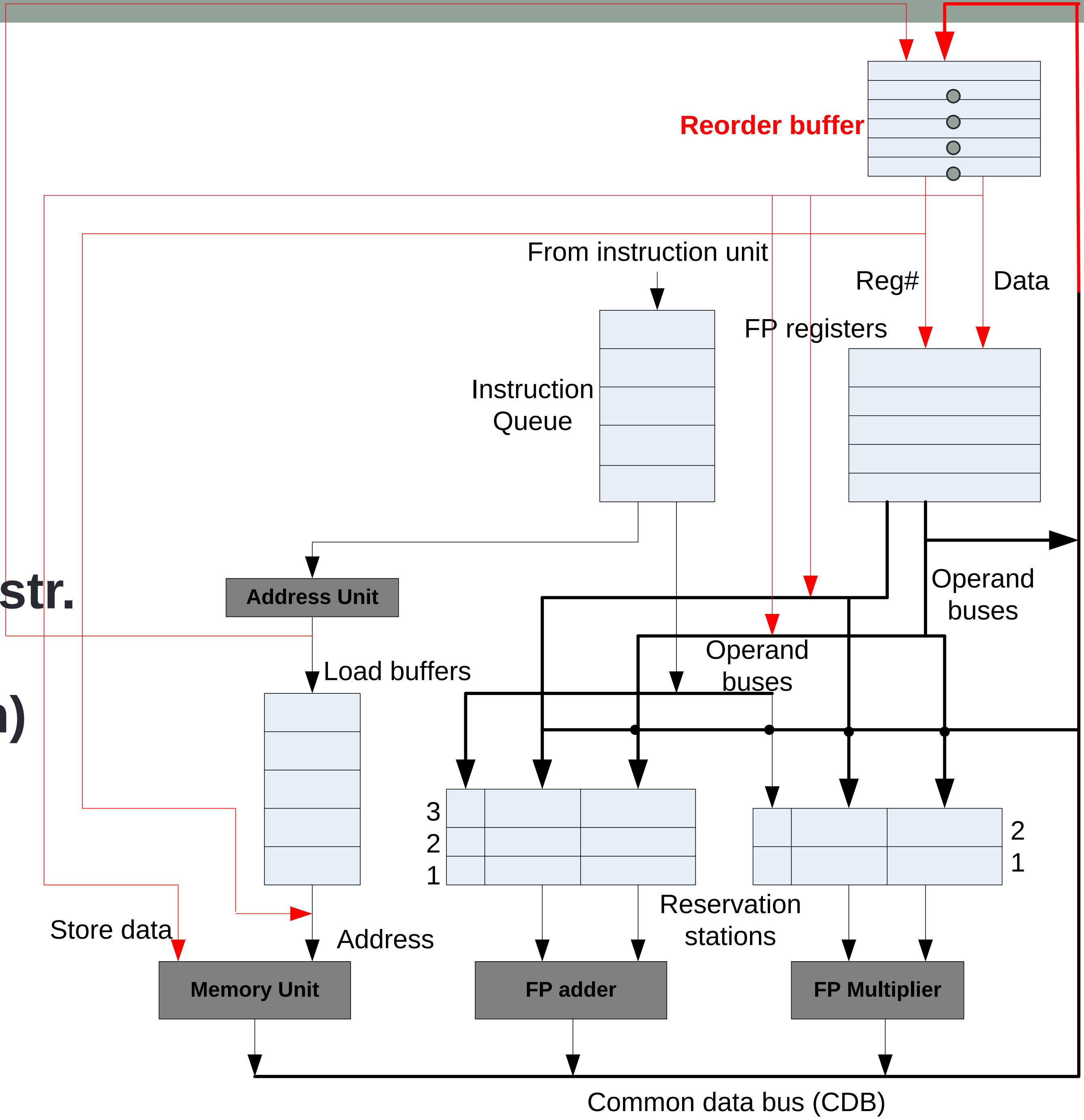
Commit
Case: 3/4
Branch instr. (correct prediction)

✓



Commit Case: 4/4 Branch instr. (wrong prediction)




Reorder buffer
From instruction unit
Reg#
Data
FP registers
Instruction Queue
Address Unit
Operand buses
Operand buses
Load buffers
3
2
1
2
1
Reservation stations
Store data
Address
Memory Unit
FP adder
FP Multiplier
Common data bus (CDB)