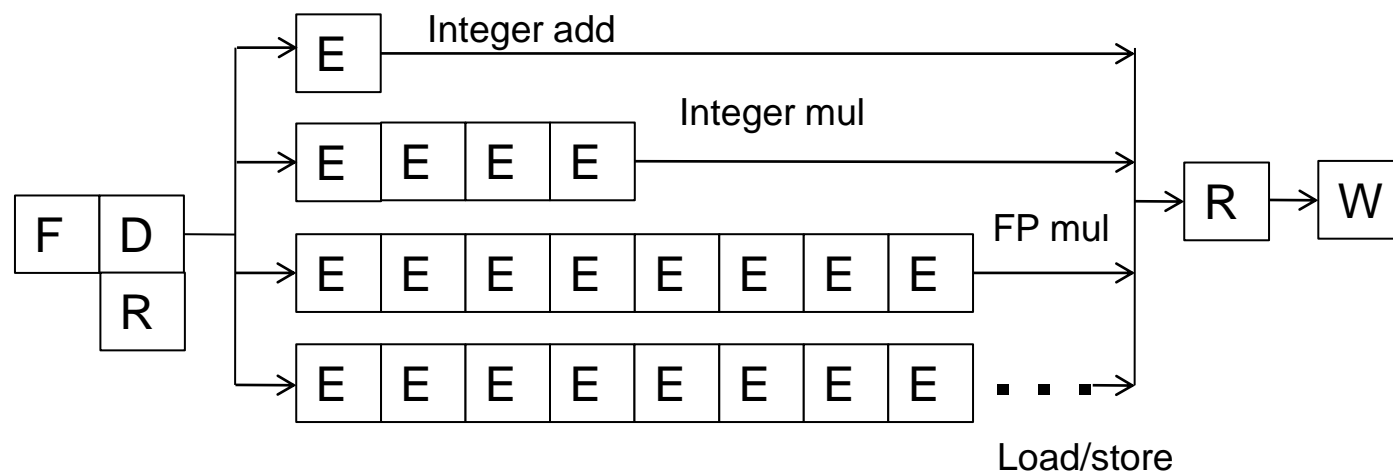


Review: In-Order Pipeline with Reorder Buffer

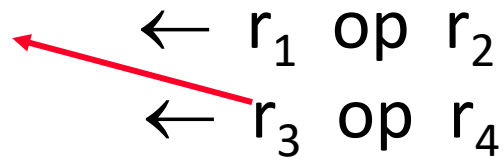
- **Decode (D)**: Access regfile/ROB, allocate entry in ROB, check if instruction can execute, if so **dispatch** instruction (i.e., send it to functional unit)
- **Execute (E)**: Instructions can complete out-of-order
- **Completion (R)**: Write result to **reorder buffer**
- **Retirement/Commit (W)**: Check oldest instruction for exceptions; if none, write result to architectural register file or memory; else, flush pipeline and start from exception handler
- **In-order dispatch/execution, out-of-order completion, in-order retirement**



Recall: Data Dependence Types

Flow dependence

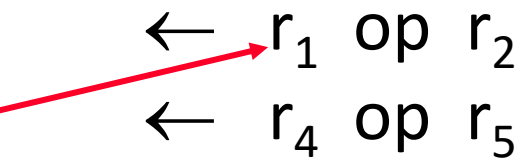
$r_3 \leftarrow r_1 \text{ op } r_2$
 $r_5 \leftarrow r_3 \text{ op } r_4$



Read-after-Write
(RAW)

Anti dependence

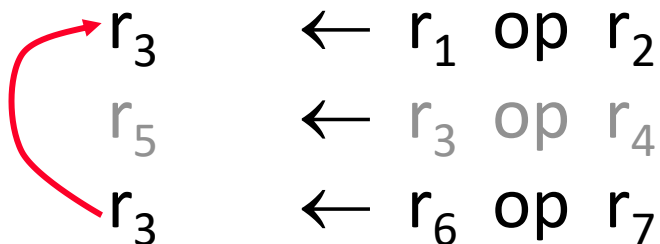
$r_3 \leftarrow r_1 \text{ op } r_2$
 $r_1 \leftarrow r_4 \text{ op } r_5$



Write-after-Read
(WAR)

Output dependence

$r_3 \leftarrow r_1 \text{ op } r_2$
 $r_5 \leftarrow r_3 \text{ op } r_4$
 $r_3 \leftarrow r_6 \text{ op } r_7$



Write-after-Write
(WAW)

Recall: Register Renaming with a Reorder Buffer

- Output and anti dependences are **not true dependences**
 - WHY? The same register refers to values that have nothing to do with each other
 - **They exist due to lack of register ID's (i.e. names) in the ISA**
- The register ID is **renamed** to the reorder buffer entry that will hold the register's value
 - Register ID → ROB entry ID
 - Architectural register ID → Physical register ID
 - After renaming, ROB entry ID used to refer to the register
- This eliminates anti and output dependences
 - Gives the illusion that there are a large number of registers

Recall: Reorder Buffer Example

Register File (RF)

R0			
R1			
R2			
R3			
R4			
R5			
R6			
R7			

Value Valid? Value Tag
 (pointer to
 ROB entry)

Initially: all registers
 are valid in RF
 & ROB is empty

Simulate:

MUL R1, R2 → R3

MUL R3, R4 → R11

ADD R5, R6 → R3

ADD R3, R8 → R12

Reorder Buffer (ROB)

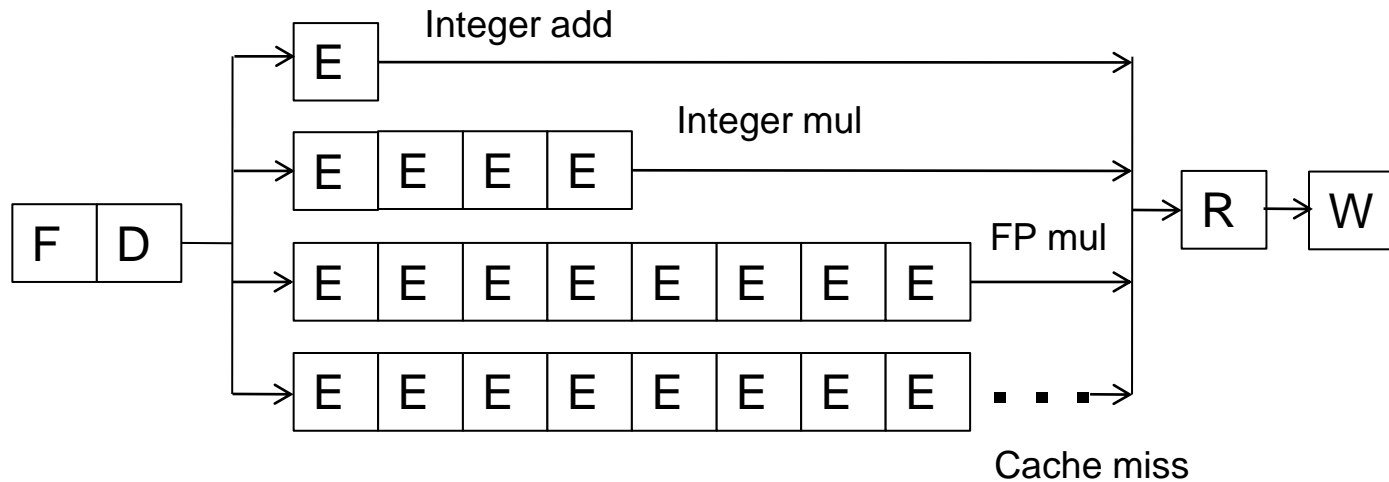
Entry 0				
Entry 1				
Entry 2				
Entry 8				
Entry 13				
Entry 14				
Entry 15				

← Oldest
 instruction

← Youngest
 instruction

Entry Valid?
 Dest reg ID
 Dest reg value
 Dest reg written?

An In-order Pipeline



- Dispatch: Act of sending an instruction to a functional unit
- Renaming with ROB eliminates stalls due to false dependences
- Problem: A non-ready instruction stalls dispatch of younger instructions into functional (execution) units

How Can We Do Better?

- What do the following two pieces of code have in common (with respect to execution in the previous design)?

```
MUL  R3 ← R1, R2
ADD  R3 ← R3, R1
ADD  R4 ← R6, R7
MUL  R5 ← R6, R8
ADD  R7 ← R9, R9
```

```
LD   R3 ← R1 (0)
ADD  R3 ← R3, R1
ADD  R4 ← R6, R7
MUL  R5 ← R6, R8
ADD  R7 ← R9, R9
```

- **Answer: First ADD stalls the whole pipeline!**
 - ADD cannot dispatch because its source register unavailable
 - **Later independent instructions cannot get dispatched**
- How are the above code portions different?
 - **Answer: Load latency is variable (unknown until runtime)**
 - What does this affect? Think compiler vs. microarchitecture

Preventing Dispatch Stalls

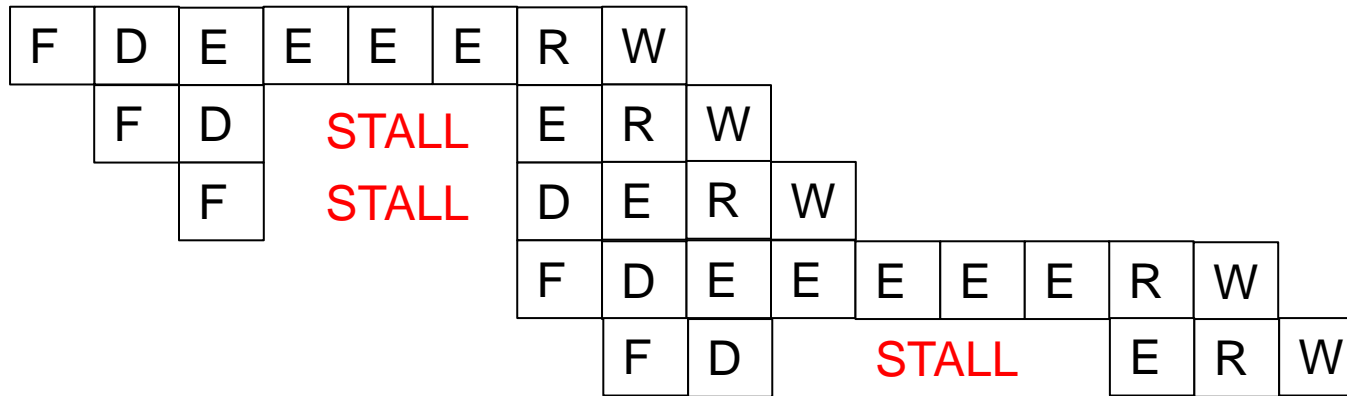
- Problem: **in-order** dispatch (scheduling, or execution)
- Solution: **out-of-order** dispatch (scheduling, or execution)
- Actually, we have seen the basic idea before:
 - **Dataflow**: “fire” an instruction only when its inputs are ready
 - We will use similar principles, but not expose it in the ISA
- Aside: Any other way to prevent dispatch stalls?
 1. Compile-time instruction scheduling/reordering
 2. Value prediction
 3. Fine-grained multithreading

Out-of-order Execution (Dynamic Scheduling)

- Idea: Move the non-ready instructions out of the way of independent ones (such that independent ones can dispatch)
 - Rest areas for non-ready instructions: Reservation stations
- Monitor the source “values” of each instruction in the resting (waiting) area
- When all source “values” of an instruction are available, “fire” (i.e., dispatch) the instruction
 - Instructions dispatched in **dataflow (not control-flow) order**
- Benefit:
 - **Latency tolerance**: Allows independent instructions to execute and complete in the presence of a long-latency operation

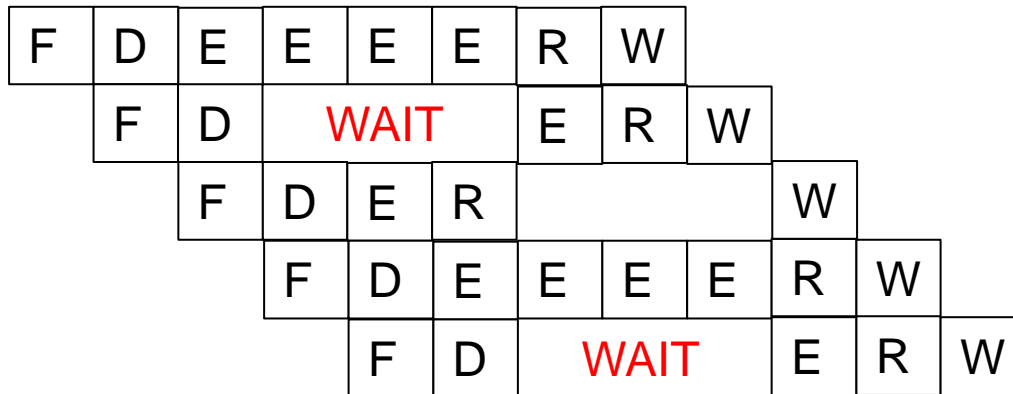
In-order vs. Out-of-order Dispatch

- In order dispatch + precise exceptions:



IMUL R3 ← R1, R2
 ADD R3 ← R3, R1
 ADD R1 ← R6, R7
 IMUL R5 ← R6, R8
 ADD R7 ← R3, R5

- Out-of-order dispatch + precise exceptions:



- 16 vs. 12 cycles

Enabling OoO Execution

1. Need to link the consumer of a value to the producer
 - ❑ Register renaming: Associate a “tag” with each data value
2. Need to buffer instructions until they are ready to execute
 - ❑ Insert instruction into reservation stations after renaming
3. Instructions need to keep track of readiness of source values
 - ❑ Broadcast the “tag” when the value is produced
 - ❑ Instructions compare their “source tags” to the broadcast tag
→ if match, source value becomes ready
4. When all source values of an instruction are ready, need to dispatch the instruction to its functional unit (FU)
 - ❑ Instruction wakes up if all sources are ready
 - ❑ If multiple instructions are awake, need to select one per FU

Tomasulo's Algorithm

- If reservation station available before renaming
 - Instruction + renamed operands (source value/tag) inserted into the reservation station
 - Only rename if reservation station is available
- Else stall
- While in reservation station, each instruction:
 - Watches common data bus (CDB) for tag of its sources
 - When tag seen, grab value for the source and keep it in the reservation station
 - When both operands available, instruction ready to be dispatched
- Dispatch instruction to the Functional Unit when instruction is ready
- After instruction finishes in the Functional Unit
 - Arbitrate for CDB
 - Put tagged value onto CDB (tag broadcast)
 - Register file is connected to the CDB
 - Register contains a tag indicating the latest writer to the register
 - If the tag in the register file matches the broadcast tag, write broadcast value into register (and set valid bit)
 - Reclaim rename tag
 - no valid copy of tag in system!

Our First OoO Machine Simulation

Program We Will Simulate

```
MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
```

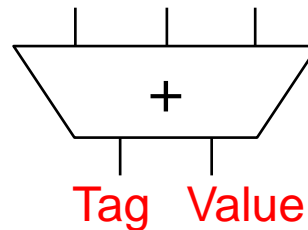
Initially:

1. Reservation Stations (RS's) are all Invalid (Empty)
2. All Registers are Valid

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

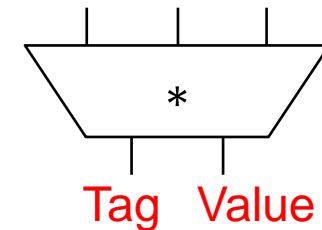
RS for ADD Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						



RS for MUL Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						



Register Alias Table

ADD and MUL Execution Units
have separate Tag & Value buses

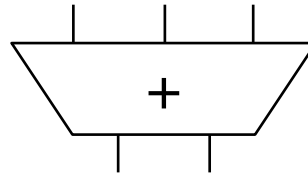
Cycle 0

Cycle

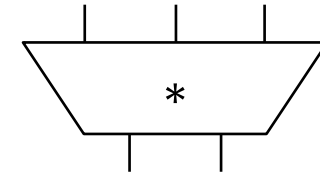
```
MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
```

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						



Cycle 1

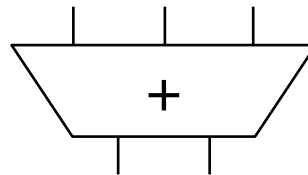
Cycle 1

F

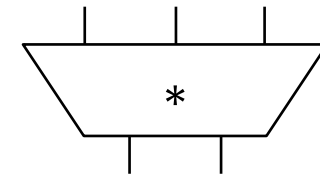
MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						



Cycle 2

MUL gets decoded and allocated into RS x

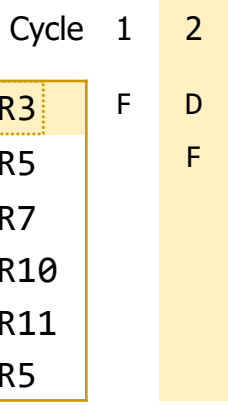
Step 1: Check if reservation station available. Yes: x

Step 2: Access the Register Alias Table

Step 3: Put source registers into reservation station x

Step 4: Rename destination register R3 → x

R3 is now renamed to x.
Its new value will be produced by the reservation station that is identified by tag x.

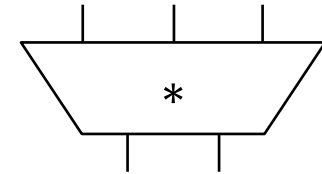
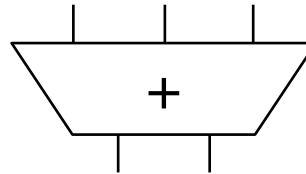


MUL	R1, R2	→	R3
ADD	R3, R4	→	R5
ADD	R2, R6	→	R7
ADD	R8, R9	→	R10
MUL	R7, R10	→	R11
ADD	R5, R11	→	R5

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x		~			~	
y						
z						
t						



MUL in RS x is ready to execute in the next cycle!

Cycle 3

- 1. MUL in RS x starts executing
- 2. ADD gets decoded and allocated into RS a

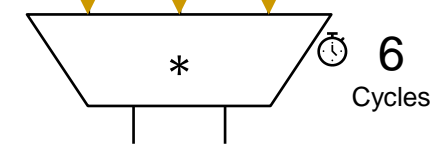
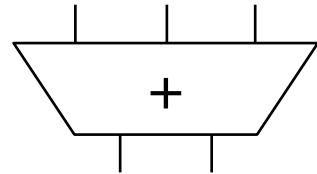
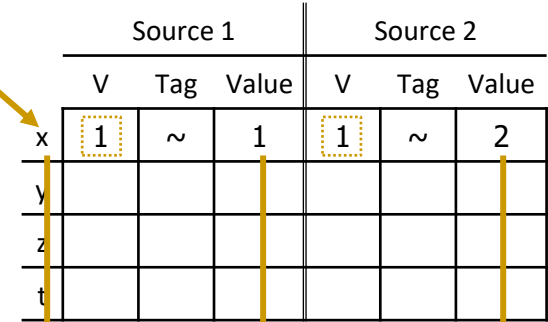
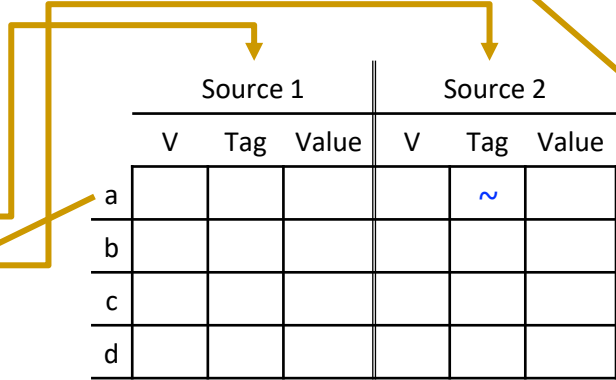
	Cycle 1	2	3
MUL R1, R2 → R3	F	D	E ₁
ADD R3, R4 → R5		F	D
ADD R2, R6 → R7			F
ADD R8, R9 → R10			
MUL R7, R10 → R11			
ADD R5, R11 → R5			

Check readiness (Both sources ready?) → Wakeup

Ready → Dispatch the instruction to the MUL unit

Same Steps 1-4 for ADD... Rename R5 → a

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11



ADD in RS a cannot execute in the next cycle: one source is not valid

Cycle 4

Cycle 1 2 3 4

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

F D E₁ E₂
 F D -
 F D
 F

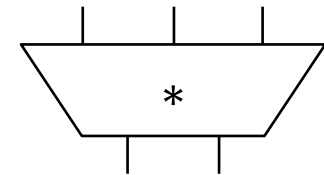
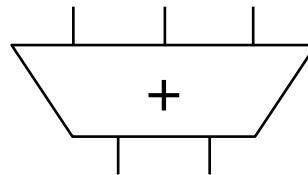
ADD in RS a waits because one source is not valid.

Rename R7 → b

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	1		10
R11	1		11

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	0	x		1	~	4
b		~			~	
c						
d						

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y						
z						
t						



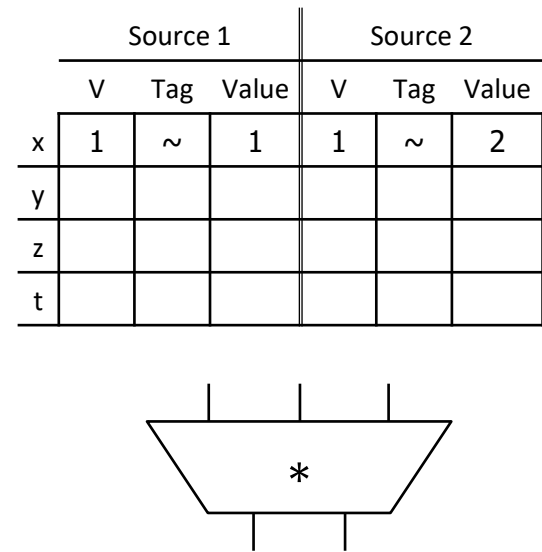
ADD in RS b is ready to execute in the next cycle!

It will be executed out of order in the next cycle.

Cycle 5

	Cycle 1	2	3	4	5
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃
ADD R3, R4 → R5		F	D	-	-
ADD R2, R6 → R7			F	D	E ₁
ADD R8, R9 → R10				F	D
MUL R7, R10 → R11					F
ADD R5, R11 → R5					

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	1		11



ADD in RS c is ready to execute in the next cycle!

Cycle 6

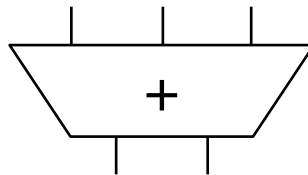
Cycle 1 2 3 4 5 6

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

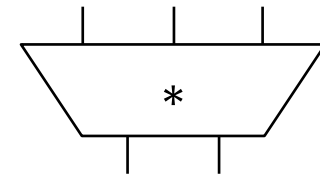
F D E₁ E₂ E₃ E₄
 F D - - -
 F D E₁ E₂
 F D E₁
 F D
 F

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	a	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	0	x		1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d						



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	0	b		0	c	
z						
t						



Cycle 7

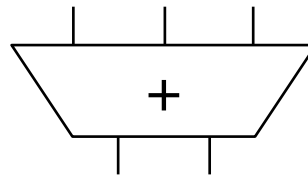
All six instructions are now decoded and renamed

Note what happened to R5: Renamed twice!

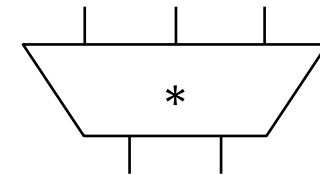
	Cycle 1	2	3	4	5	6	7
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅
ADD R3, R4 → R5		F	D	-	-	-	-
ADD R2, R6 → R7			F	D	E ₁	E ₂	E ₃
ADD R8, R9 → R10				F	D	E ₁	E ₂
MUL R7, R10 → R11					F	D	-
ADD R5, R11 → R5						F	D

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	0	x	
R4	1		4
R5	0	d	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	0	x		1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	0	b		0	c	
z						
t						



Cycle 8 (First Slide)

Cycle 1 2 3 4 5 6 7 8

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

F D E₁ E₂ E₃ E₄ E₅ E₆
 F D - - - -
 F D E₁ E₂ E₃
 F D E₁ E₂
 F D -
 F D

MUL in RS x is done

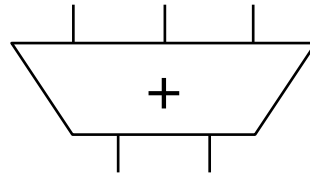
Broadcast MUL's tag (x)

- ✓ Check tag
- ✓ Check for invalidity

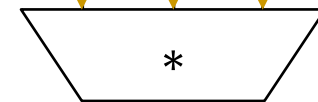
Broadcast MUL's result (2)

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	0	b	
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1		2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	0	b		0	c	
z						
t						



x 2

ADD in RS a is ready to execute in the next cycle!

Cycle 8 (Second Slide)

Cycle 1 2 3 4 5 6 7 8

MUL	R1, R2	→	R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆
ADD	R3, R4	→	R5	F	D	-	-	-	-	-	-
ADD	R2, R6	→	R7		F	D	E ₁	E ₂	E ₃	E ₄	
ADD	R8, R9	→	R10			F	D	E ₁	E ₂		
MUL	R7, R10	→	R11				F	D	-		
ADD	R5, R11	→	R5					F	D		

ADD in RS b is also done

Broadcast ADD's tag (b)

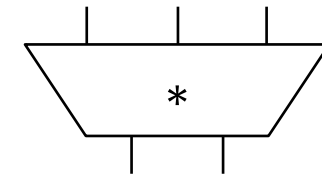
- ✓ Check tag
- ✓ Check for invalidity

Broadcast ADD's result (8)

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1		8	0	c	
z						
t						



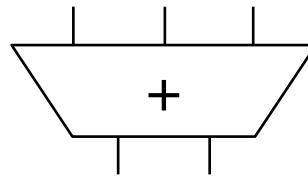
MUL in RS y is still NOT ready to execute in the next cycle!

Cycle 8 (Third Slide)

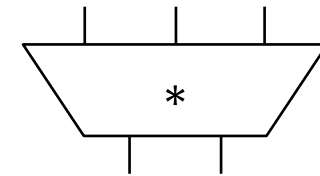
	Cycle	1	2	3	4	5	6	7	8
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	
ADD R3, R4 → R5		F	D	-	-	-	-	-	
ADD R2, R6 → R7			F	D	E ₁	E ₂	E ₃	E ₄	
ADD R8, R9 → R10				F	D	E ₁	E ₂	E ₃	
MUL R7, R10 → R11					F	D	-	-	
ADD R5, R11 → R5						F	D	-	

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	0	c	
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	0	c	
z						
t						



Cycle 9

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

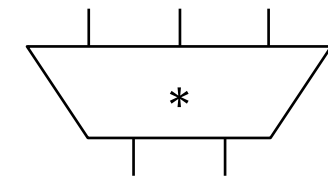
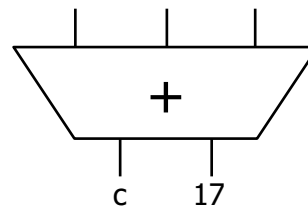
Cycle	1	2	3	4	5	6	7	8	9
F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W	
	F	D	-	-	-	-	-	E ₁	
		F	D	E ₁	E ₂	E ₃	E ₄	W	
			F	D	E ₁	E ₂	E ₃	E ₄	
				F	D	-	-	-	
					F	D	-	-	

Broadcast and Update

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



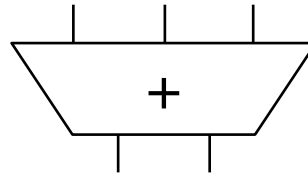
MUL in RS y is ready to execute in the next cycle!

Cycle 10

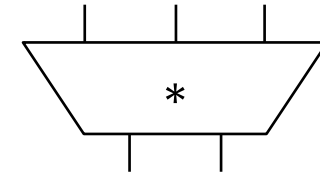
	Cycle	1	2	3	4	5	6	7	8	9	10
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W		
ADD R3, R4 → R5		F	D	-	-	-	-	-	E ₁	E ₂	
ADD R2, R6 → R7			F	D	E ₁	E ₂	E ₃	E ₄	W		
ADD R8, R9 → R10				F	D	E ₁	E ₂	E ₃	E ₄	W	
MUL R7, R10 → R11					F	D	-	-	-	E ₁	
ADD R5, R11 → R5						F	D	-	-	-	

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 11

```

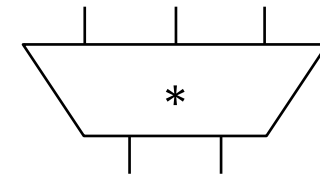
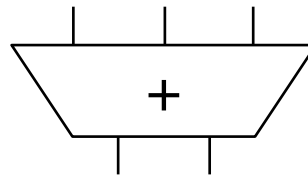
MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
  
```

Cycle	1	2	3	4	5	6	7	8	9	10	11
F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W			
	F	D	-	-	-	-	-	E ₁	E ₂	E ₃	
		F	D	E ₁	E ₂	E ₃	E ₄	W			
			F	D	E ₁	E ₂	E ₃	E ₄	W		
				F	D	-	-	-	E ₁	E ₂	
					F	D	-	-	-	-	

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	0	a		0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 12

```

MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
    
```

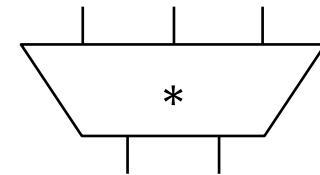
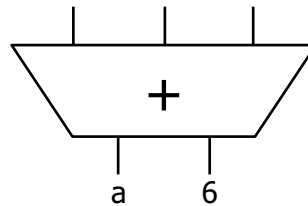
Cycle	1	2	3	4	5	6	7	8	9	10	11	12
F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W				
	F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	
		F	D	E ₁	E ₂	E ₃	E ₄	W				
			F	D	E ₁	E ₂	E ₃	E ₄	W			
				F	D	-	-	-	E ₁	E ₂	E ₃	
					F	D	-	-	-	-	-	

← Broadcast and Update

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 13

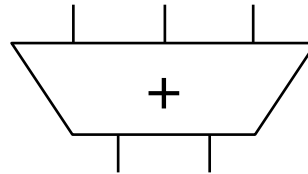
Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

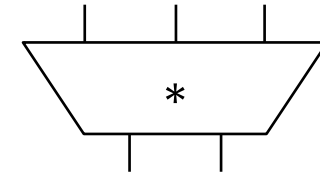
F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W					
	F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	W	
		F	D	E ₁	E ₂	E ₃	E ₄	W					
			F	D	E ₁	E ₂	E ₃	E ₄	W				
				F	D	-	-	-	E ₁	E ₂	E ₃	E ₄	
					F	D	-	-	-	-	-	-	-

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 14

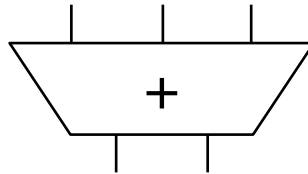
Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13 14

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

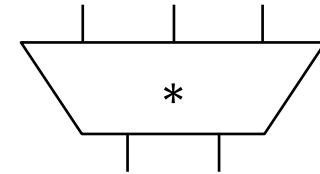
F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W						
	F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	W		
		F	D	E ₁	E ₂	E ₃	E ₄	W						
			F	D	E ₁	E ₂	E ₃	E ₄	W					
				F	D	-	-	-	E ₁	E ₂	E ₃	E ₄	E ₅	
					F	D	-	-	-	-	-	-	-	-

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	0	y	

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	0	y	



	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 15

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

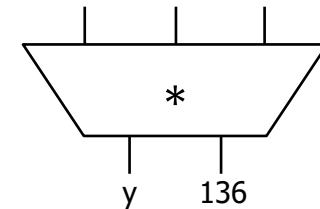
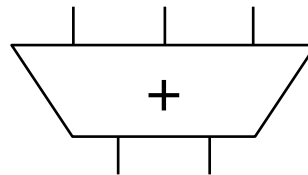
Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W						
ADD R3, R4 → R5		F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	W		
ADD R2, R6 → R7			F	D	E ₁	E ₂	E ₃	E ₄	W						
ADD R8, R9 → R10				F	D	E ₁	E ₂	E ₃	E ₄	W					
MUL R7, R10 → R11					F	D	-	-	-	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆
ADD R5, R11 → R5						F	D	-	-	-	-	-	-	-	-

Broadcast and Update

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



ADD in RS d is ready to execute in the next cycle!

Cycle 16

```

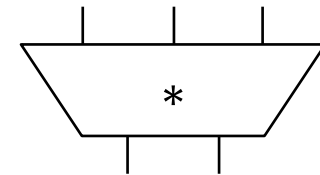
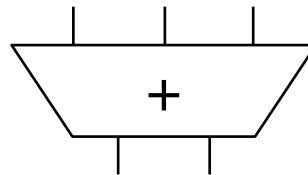
MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
    
```

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
MUL R1, R2 → R3	F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W							
ADD R3, R4 → R5		F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	W			
ADD R2, R6 → R7			F	D	E ₁	E ₂	E ₃	E ₄	W							
ADD R8, R9 → R10				F	D	E ₁	E ₂	E ₃	E ₄	W						
MUL R7, R10 → R11					F	D	-	-	-	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W
ADD R5, R11 → R5						F	D	-	-	-	-	-	-	-	-	E ₁

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 17

Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

```

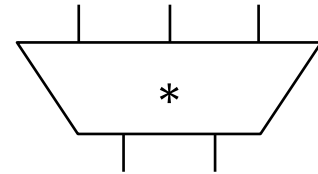
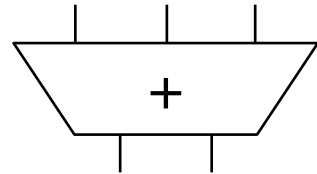
MUL R1, R2 → R3
ADD R3, R4 → R5
ADD R2, R6 → R7
ADD R8, R9 → R10
MUL R7, R10 → R11
ADD R5, R11 → R5
    
```

F	D	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W								
	F	D	-	-	-	-	-	E ₁	E ₂	E ₃	E ₄	W				
		F	D	E ₁	E ₂	E ₃	E ₄	W								
			F	D	E ₁	E ₂	E ₃	E ₄	W							
				F	D	-	-	-	E ₁	E ₂	E ₃	E ₄	E ₅	E ₆	W	
					F	D	-	-	-	-	-	-	-	-	E ₁	E ₂

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 18

Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

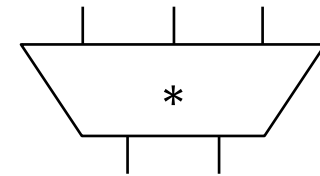
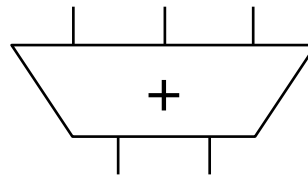
MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

F D E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D - - - E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - - E₁ E₂ E₃ E₄ E₅ E₆ W

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	0	d	
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 19

Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

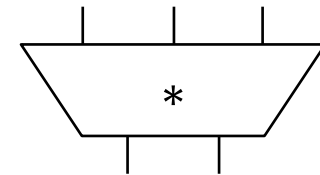
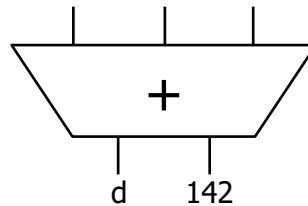
F D E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D - - - E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - - E₁ E₂ E₃ E₄

Broadcast and Update

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	1		142
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Cycle 20

Cycle 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20

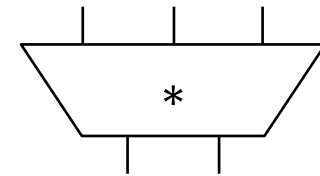
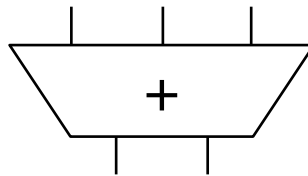
MUL R1, R2 → R3
 ADD R3, R4 → R5
 ADD R2, R6 → R7
 ADD R8, R9 → R10
 MUL R7, R10 → R11
 ADD R5, R11 → R5

F D E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D E₁ E₂ E₃ E₄ W
 F D - - - E₁ E₂ E₃ E₄ E₅ E₆ W
 F D - - - - - E₁ E₂ E₃ E₄ W

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		2
R4	1		4
R5	1		142
R6	1		6
R7	1		8
R8	1		8
R9	1		9
R10	1		17
R11	1		136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a	1	~	2	1	~	4
b	1	~	2	1	~	6
c	1	~	8	1	~	9
d	1	~	6	1	~	136

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x	1	~	1	1	~	2
y	1	~	8	1	~	17
z						
t						



Some More Questions (Design Choices)

- When is a reservation station entry deallocated?
- Should the reservation stations be dedicated to each functional unit or global across functional units?
 - Centralized vs. Distributed: What are the tradeoffs?
- Should reservation stations and ROB store data values or should there be a centralized physical register file where all data values are stored?
 - What are the tradeoffs?
- Timing: Exactly when does an instruction broadcast its tag?
- Many other design choices for OoO engines

Out-of-Order Execution with Precise Exceptions

- **Idea:** Use a reorder buffer to reorder instructions before committing them to architectural state
- An instruction updates the RAT when it completes execution
 - Also called **frontend register file**
- An instruction updates a **separate architectural register file** when it retires
 - i.e., when it is the oldest in the machine and has completed execution
 - In other words, **the architectural register file is always updated in program order**
- On an exception: flush pipeline, copy architectural register file into frontend register file

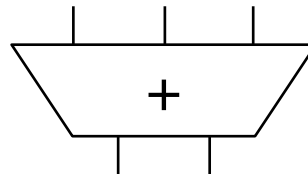
Recall: Our Initial OoO Machine

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

Register Alias Table

RS for ADD Unit

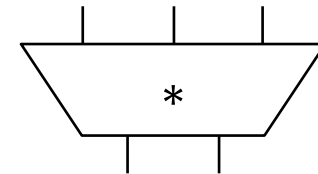
	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						



Tag Value

RS for MUL Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						



Tag Value

Add Arch Reg File & ROB for Precise Exceptions

Reorder Buffer (ROB)

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

Frontend Register File

Entry 0				
Entry 1				
Entry 2				
Entry 8				
Entry 13				
Entry 14				
Entry 15				

Register	Value
R1	1
R2	2
R3	3
R4	4
R5	5
R6	6
R7	7
R8	8
R9	9
R10	10
R11	11

Architectural Register File

OoO Machine with Precise Exceptions

RS for ADD Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

Frontend Register File

RS for MUL Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						

Register	Value
R1	1
R2	2
R3	3
R4	4
R5	5
R6	6
R7	7
R8	8
R9	9
R10	10
R11	11

Architectural Register File

Reorder Buffer (ROB)

Entry 0					
Entry 1					
Entry 2					
Entry 8					
Entry 13					
Entry 14					
Entry 15					

One Issue: Value Replication All Over the Place

RS for ADD Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
a						
b						
c						
d						

Register	Valid	Tag	Value
R1	1		1
R2	1		2
R3	1		3
R4	1		4
R5	1		5
R6	1		6
R7	1		7
R8	1		8
R9	1		9
R10	1		10
R11	1		11

Frontend Register File

RS for MUL Unit

	Source 1			Source 2		
	V	Tag	Value	V	Tag	Value
x						
y						
z						
t						

Register	Value
R1	1
R2	2
R3	3
R4	4
R5	5
R6	6
R7	7
R8	8
R9	9
R10	10
R11	11

Architectural Register File

Reorder Buffer (ROB)

Entry 0				
Entry 1				
Entry 2				
Entry 8				
Entry 13				
Entry 14				
Entry 15				

Getting Rid of Replicated Values

Pointers to PRF

Register	PR
R1	18
R2	13
R3	10
R4	22
R5	14
R6	19
R7	17
R8	20
R9	3
R10	4
R11	1

Frontend Register Map

PR	Value
PR1	1
PR2	2
PR3	3
PR4	4
PR5	5
PR6	6
PR7	7
PR8	8
PR9	9
PR10	10
PR11	11
PR12	12
PR13	13
PR14	14
PR15	15
PR16	16
PR17	17
PR18	18
PR19	19
PR20	20
PR21	21
PR22	22

Physical Register File Centralized Value Storage

Reorder Buffer (ROB)

Entry 0				
Entry 1				
Entry 2				
Entry 8				
Entry 13				
Entry 14				
Entry 15				

Pointers to PRF

Register	PR
R1	12
R2	2
R3	10
R4	22
R5	5
R6	9
R7	11
R8	20
R9	7
R10	6
R11	1

Architectural Register Map

Enabling OoO Execution, Revisited

1. **Link** the consumer of a value to the producer
 - ❑ **Register renaming**: Associate a “tag” with each data value
2. **Buffer** instructions until they are ready
 - ❑ Insert instruction into **reservation stations** after renaming
3. Keep **track** of **readiness** of source values of an instruction
 - ❑ **Broadcast the “tag”** when the value is produced
 - ❑ Instructions **compare their “source tags”** to the broadcast tag
 - if match, source value becomes ready
4. When all source values of an instruction are ready, **dispatch** the instruction to functional unit (FU)
 - ❑ **Wakeup and select/schedule** the instruction

Summary of OOO Execution Concepts

- Register renaming eliminates false dependences, enables linking of producer to consumers
- Buffering in reservation stations enables the pipeline to move for independent instructions
- Tag broadcast enables communication (of readiness of produced value) between instructions
- Wakeup and select enables out-of-order dispatch

A Modern OoO Design: Intel Pentium 4

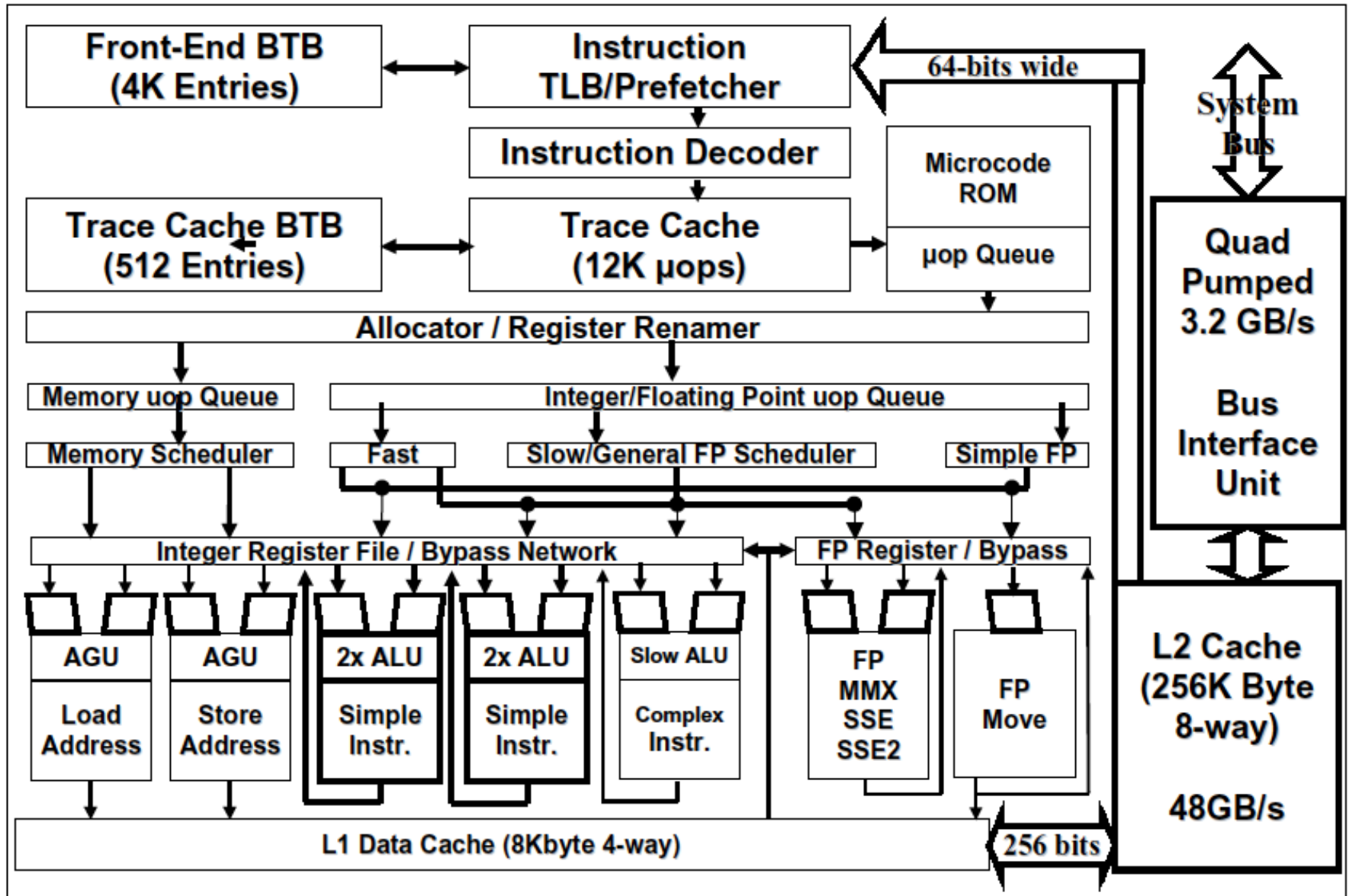
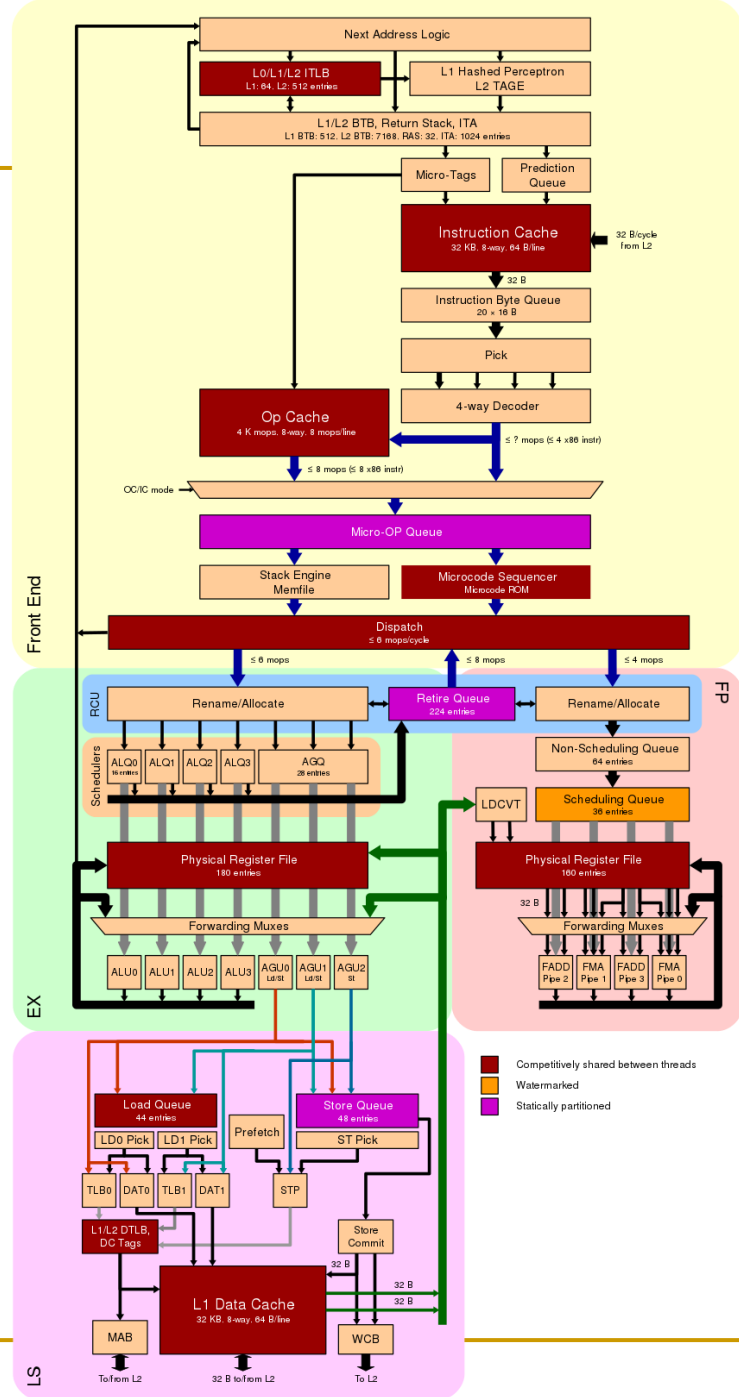


Figure 4: Pentium[®] 4 processor microarchitecture

AMD Zen2? (2019)



Apple M1 FireStorm? (2020)

