Lecture 11:
CPU – OoO Execution

Hunjun Lee
hunjunlee@hanyang.ac.kr

Problem of the existing pipeline …

- No identical operations **e.g., R-type, I-type & J-type**

  ⇒ Unify instruction types

  - Combine instruction types to flow through “multi-function” pipe

- No uniform sub-operations **e.g., RF read VS. Memory write?**

  ⇒ Balance pipeline stages

  - Stage-latency calculation to make balanced stages

- No independent operations **e.g., Waiting data to be produced?**

  ⇒ Remove dependency and/or busy resources

  - Inter-instruction dependency detection and resolution

Problem of the existing pipeline …

명령어 종류에 따라 Cycle 수가 다르다.

- No identical operations **e.g., R-type, I-type & J-type**

  ⇒ Unify instruction types

  - Combine instructi… …pipe

Each instruction takes different # of cycles to finish!

- No u… …g., RF read VS. Memory write?**

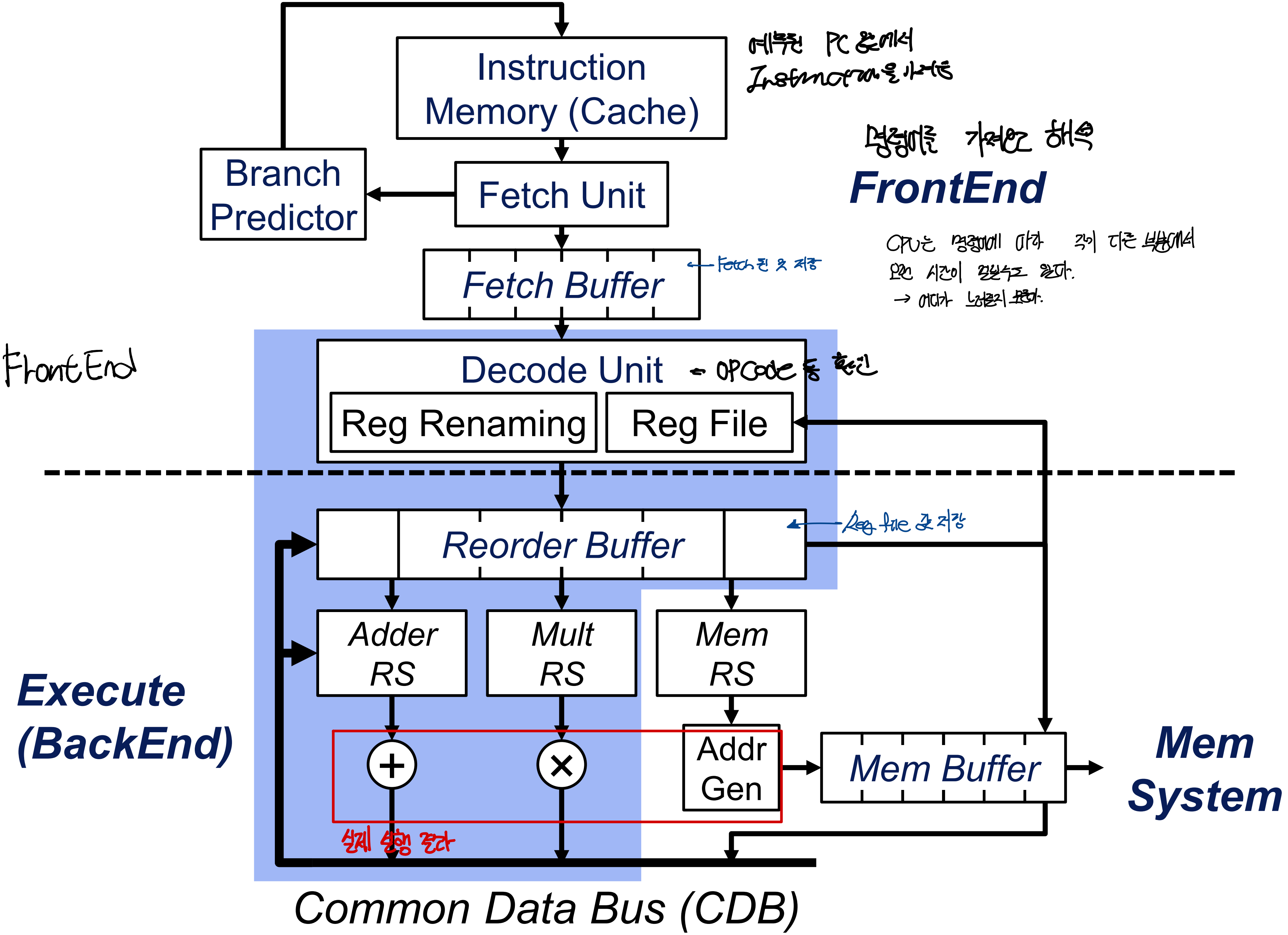
  ⇒ Balance pipeline stages

  - Stage-latency calculation to make balanced stages

- No independent operations **e.g., Waiting data to be produced?**

  ⇒ Remove dependency and/or busy resources

  - Inter-instruction dependency detection and resolution



Instruction Memory (Cache)

예측된 PC 값에서 Instruction을 가져옴

Branch Predictor

Fetch Unit

명령어를 가져오고 해석

FrontEnd

CPU는 명령어에 따라 각기 다른 부분에서 오랜 시간이 걸릴수도 있다.
→ 어디가 느려질지 모른다.

Fetch Buffer ← Fetch된 것 저장

FrontEnd

Decode Unit ← OPCode 등 확인

Reg Renaming

Reg File

Reorder Buffer ← Reg file 로 저장

Adder RS

Mult RS

Mem RS

Execute (BackEnd)

+

×

Addr Gen

Mem Buffer

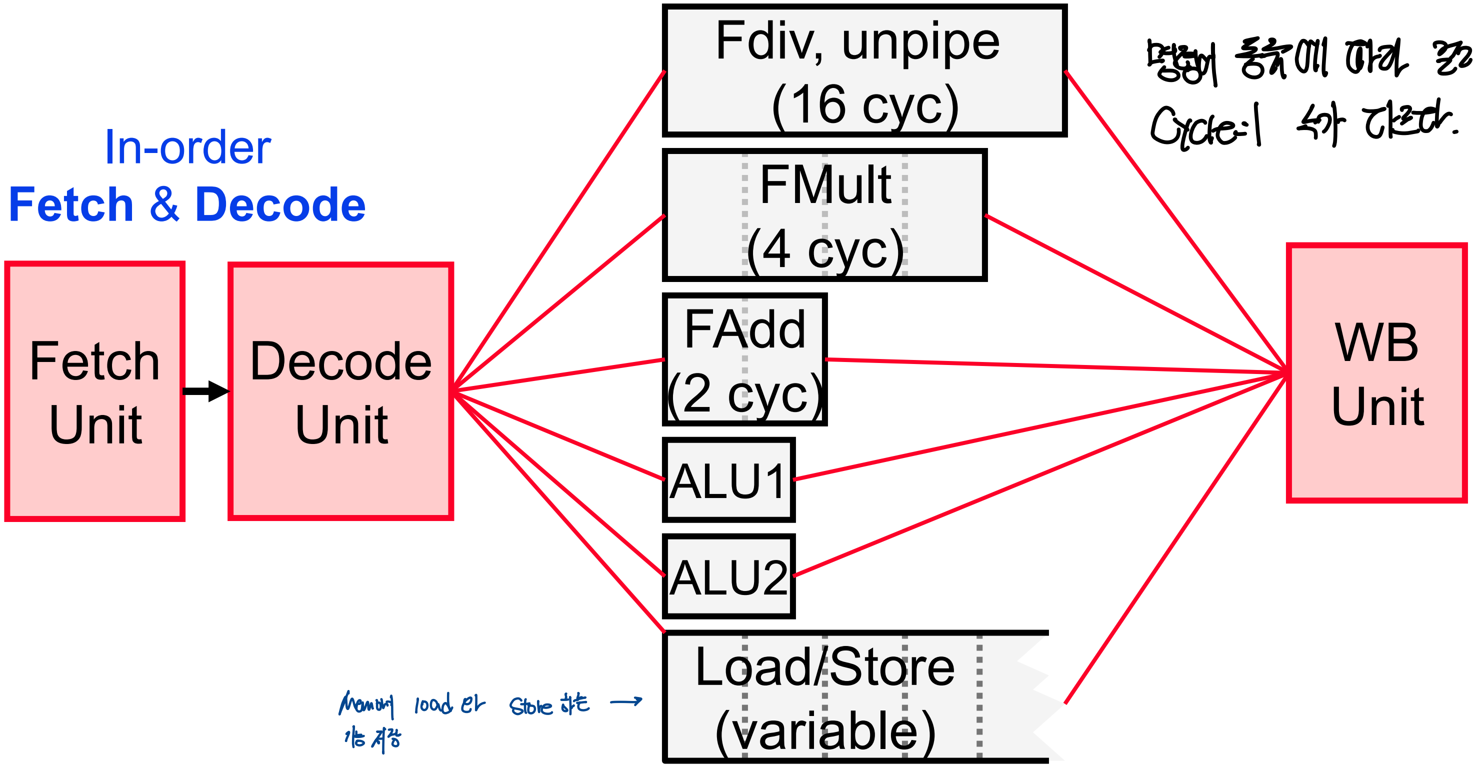
Mem System

실제 실행 결과

Common Data Bus (CDB)

Multipath Execution

- There are multiple execution stages in reality (separate execution paths, and memory unit)
- Some instructions take longer to finish than others

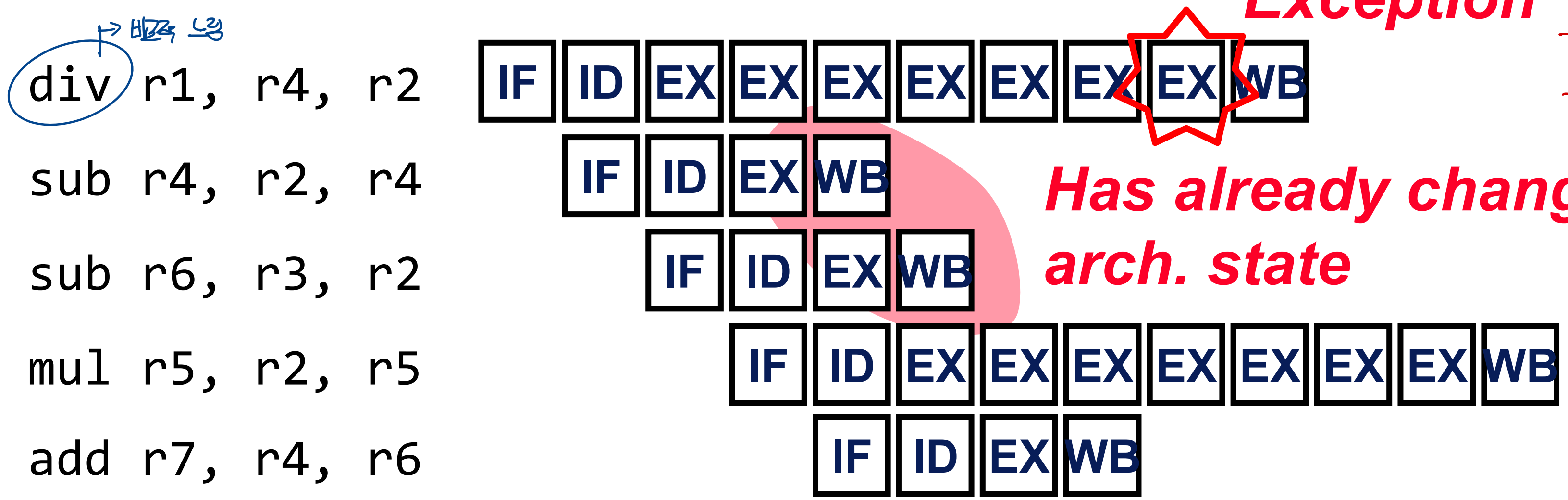


명령어 종류에 따라 걸리는
Cycle의 수가 다르다.

Memory에 load or Store 하는 →
기능 저장

Exceptions in multi-cycle execution: Option #1

- Using a multi-path execution, the instructions may terminate out-of-order!



→ 비교적 느림

`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	WB			
`sub r4, r2, r4`		IF	ID	EX	WB								
`sub r6, r3, r2`			IF	ID	EX	WB							
`mul r5, r2, r5`				IF	ID	EX	EX	EX	EX	EX	EX	EX	WB
`add r7, r4, r6`					IF	ID	EX	WB					

Div by 0 Exception

← div 이후 명령어는 Flush 되어야함
→ 근데 이미 Write Back 해버림
→ Functionality 위반

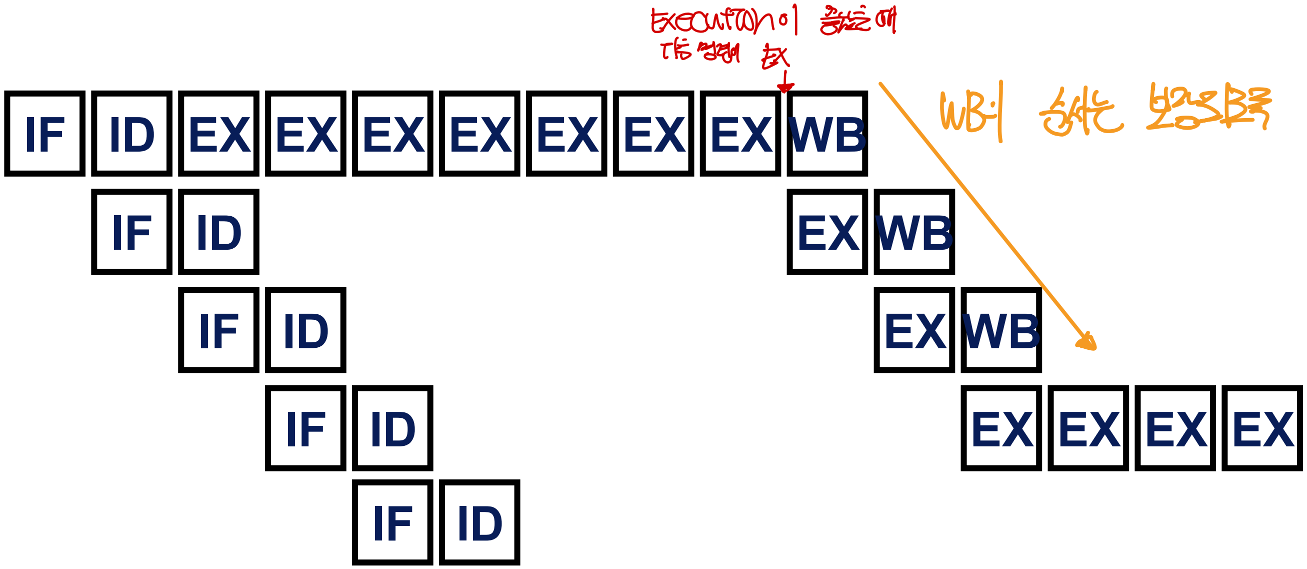
Has already changed arch. state

We should not write the result before the prior instruction has completed!

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Exceptions in multi-cycle execution: Option #2

- A single slow-running instruction may delay the younger instructions



`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	WB					
`sub r4, r2, r4`		IF	ID							EX	WB				
`sub r6, r3, r2`			IF	ID							EX	WB			
`mul r5, r2, r5`				IF	ID							EX	EX	EX	EX
`add r7, r4, r6`					IF	ID									

execution이 끝났을 때
다음 명령어 EX

WB의 순서는 보장불가

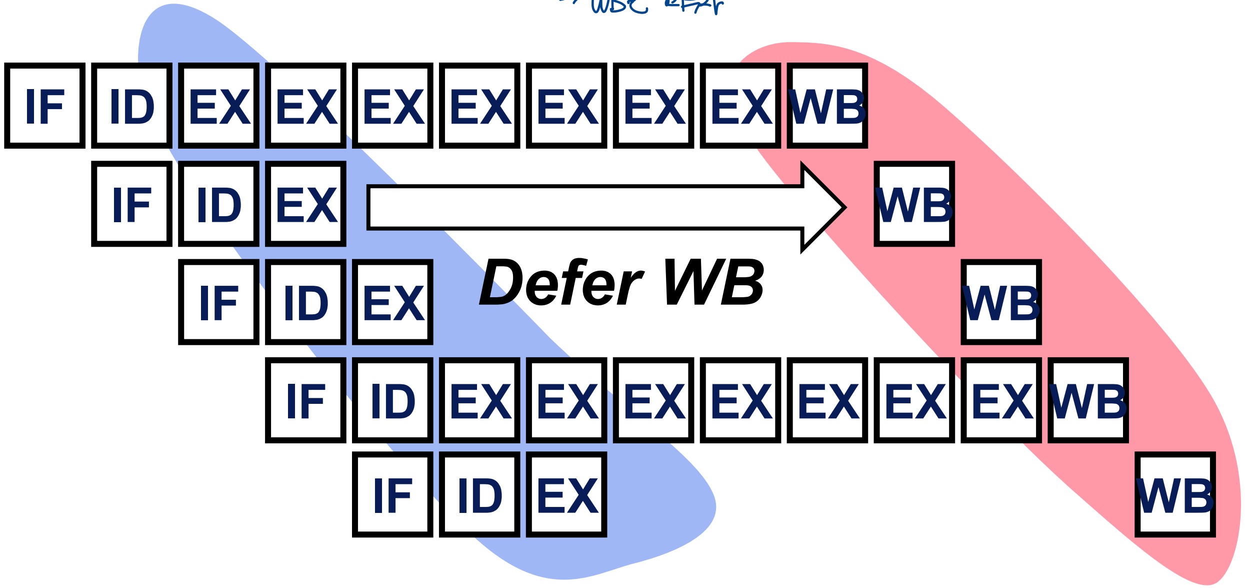
→ 비효율적이다.

Out-of-order execution: What we want!

- The younger instructions are executed early, but defer WB

OoO Execution → younger instructions finish EX stage earlier

→ WB만 미루자



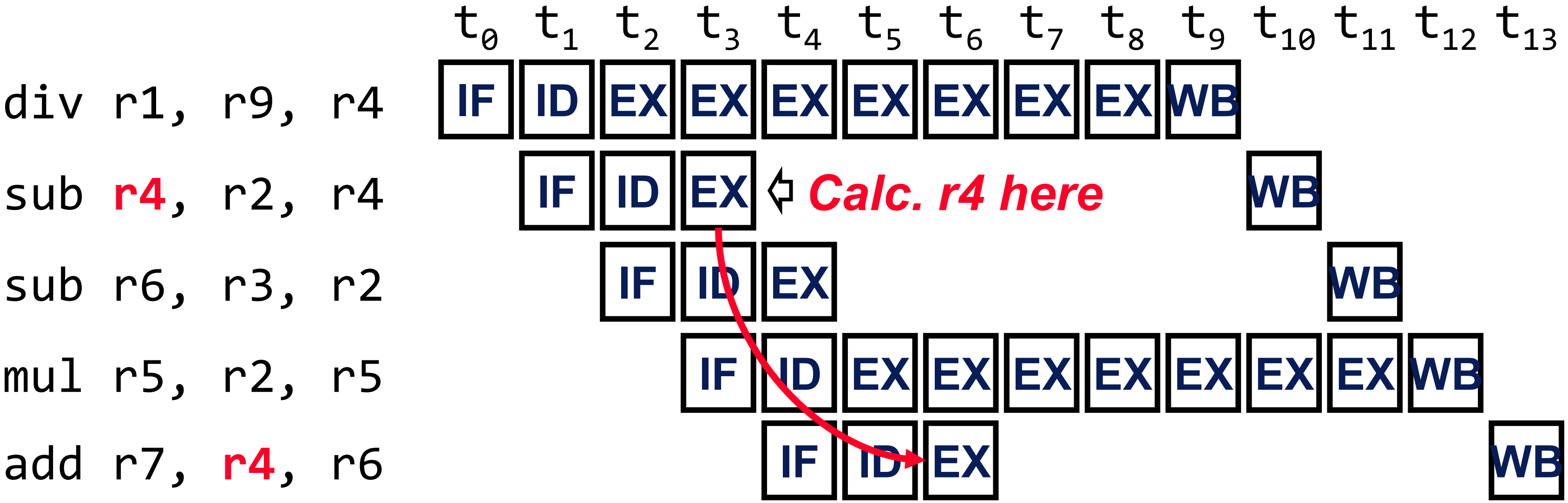
`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	WB				
`sub r4, r2, r4`		IF	ID	EX							WB			
`sub r6, r3, r2`			IF	ID	EX							WB		
`mul r5, r2, r5`				IF	ID	EX	EX	EX	EX	EX	EX	EX	WB	
`add r7, r4, r6`					IF	ID	EX							WB

WB in-order

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Problem of the OoO Execution

- The younger instructions are executed early, but defer WB



	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}
div r1, r9, r4	IF	ID	EX	EX	EX	EX	EX	EX	EX	WB				
sub **r4**, r2, r4		IF	ID	EX							WB			
sub r6, r3, r2			IF	ID	EX							WB		
mul r5, r2, r5				IF	ID	EX	EX	EX	EX	EX	EX	EX	WB	
add r7, **r4**, r6					IF	ID	EX							WB

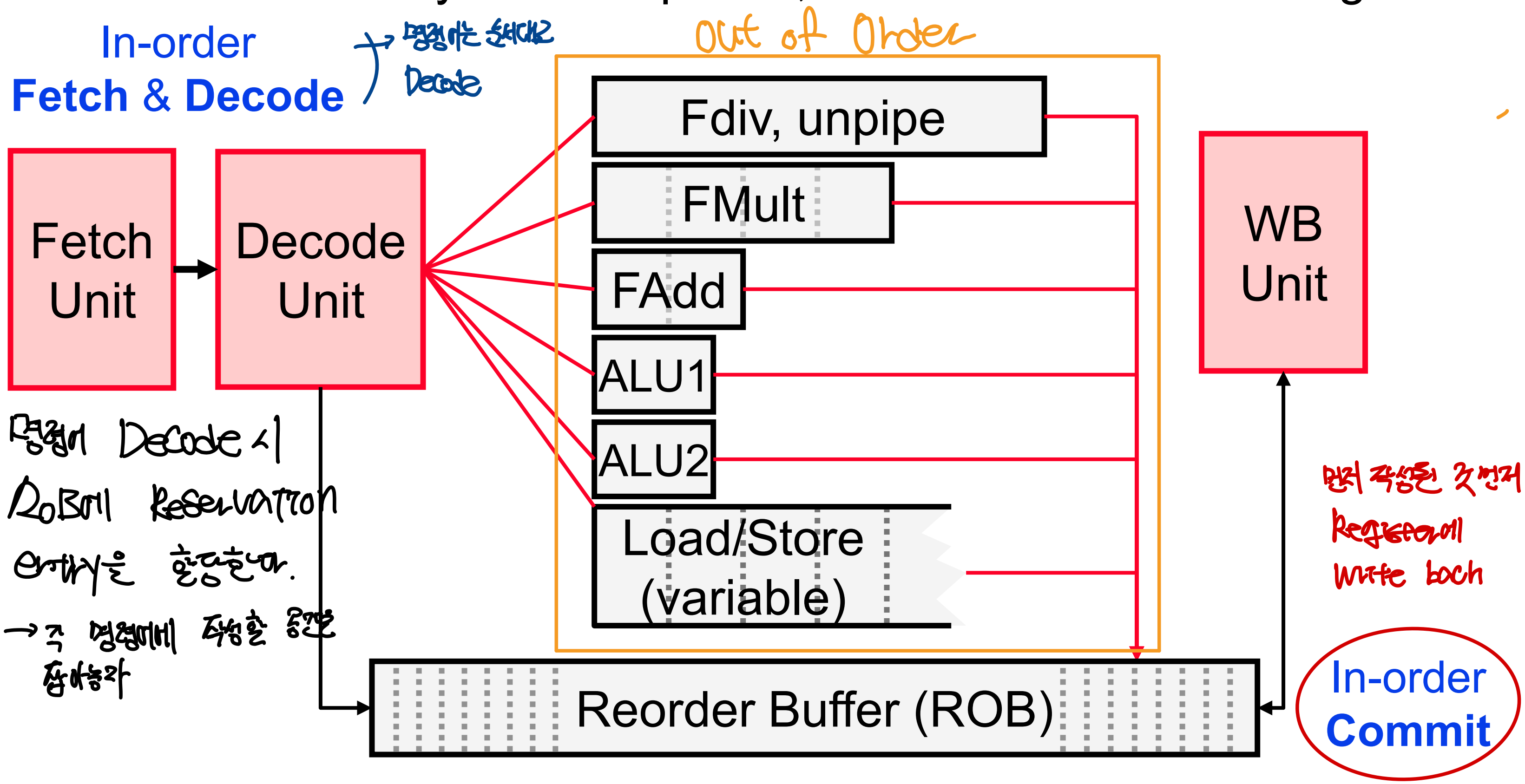
Calc. r4 here

We need a **temporary storage** to keep the calculated data and forward the data

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Reorder buffer concept

- The instructions are completed out-of-order, but reorder them before changing the architectural state
  - Reserve an entry of the decoded instructions in order
  - Write the results to the ROB upon completion
  - If the oldest entry has completed, write the result to the register file



→ 명령어는 순서대로 Decode

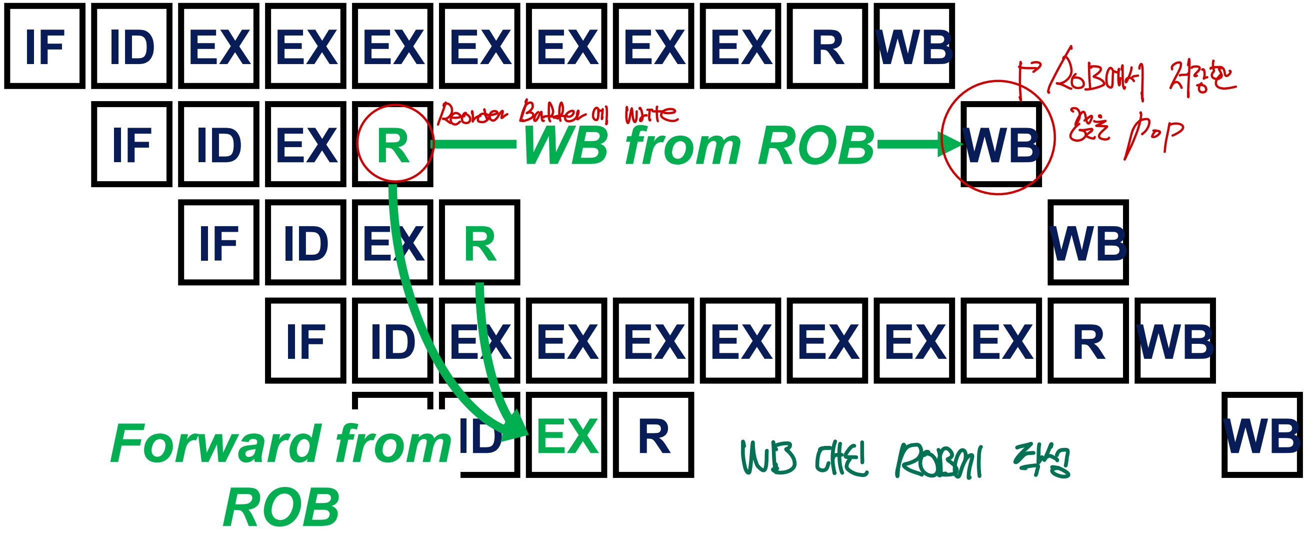
명령어 Decode 시 ROB에 Reservation entry를 할당한다.
→ 즉 명령어에 작성할 공간을 잡아놓자

먼저 작성된 것부터 Register에 Write back

The effects of ROB

- An instruction completes in an in-order manner while the execution completes in an out-of-order manner
- But, sometimes we want to use the data in the ROB!

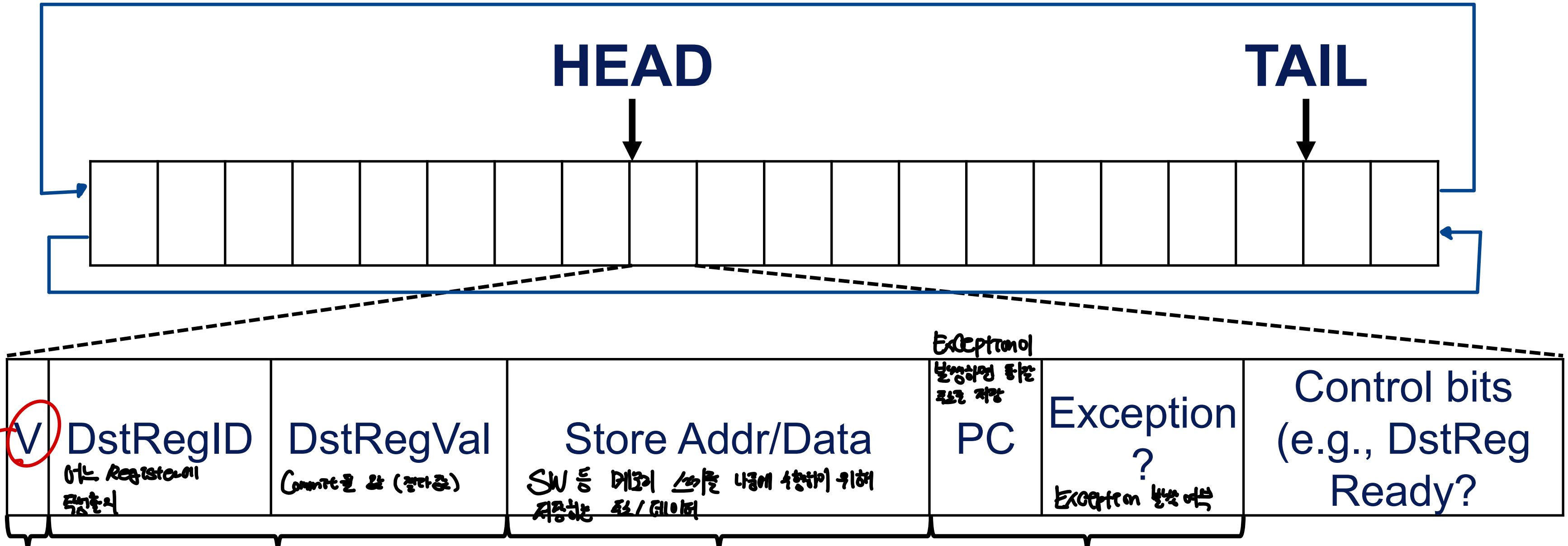
```
div r1, r4, r2
sub r4', r2, r4
sub r6', r3, r2
mul r5, r2, r5
add r7, r4', r6'
```



Reorder buffer implementation

ROB : Circular Queue 구조

- It is essentially a circular queue to store temporary values and control precise writebacks



Commit이 가능한지 여부
V=1 → 가능
V=0 → 불가능

어느 Register에 덮어쓸지

Commit될 값 (결과값)

SW 등 메모리 쓰기를 나중에 수행하기 위해 저장하는 주소/데이터

Exception이 발생하면 돌아갈 주소 저장

Exception 발생 여부

For branch misprediction

→ Beq 등에서 조건이 안맞아서 Not taken인지 여부를 'V=1'일 때 확인 가능하다.

Reg Addr & Value to Write

Memory Store (Not covered)

To handle exceptions

DstReg = 1
→ DstRegVal : Write

How to use the data in ROB?

```
div r1, r4, r2     IF
sub r4, r2, r4
sub r6, r3, r2
mul r5, r2, r5
add r7, r4, r6
```

Register File (RF)

Valid	Name	Value
1	r0	100
1	r1	200
1	r2	300
1	r3	400
1	r4	500
1	r5	600
1	r6	700
1	r7	800

ROB Entry가 예약, 할당되어있는지

Architectual state

값이 WB를 통해 되었는지

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD/TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

div r1, r4, r2	IF	ID
sub r4, r2, r4		IF
sub r6, r3, r2		
mul r5, r2, r5		
add r7, r4, r6		

ID에서 Destination Reg가 정해질시 Entry 하나 추가

Register File (RF)

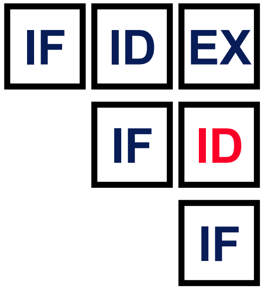
Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
1	r4	500
1	r5	600
1	r6	700
1	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

```
div r1, r4, r2
sub r4, r2, r4
sub r6, r3, r2
mul r5, r2, r5
add r7, r4, r6
```



Register File (RF)

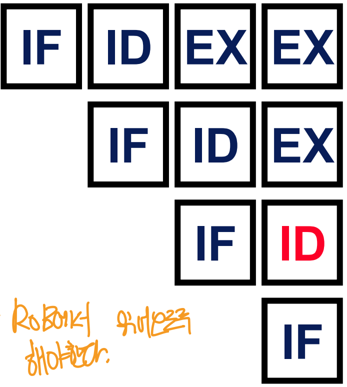
Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
1	r5	600
1	r6	700
1	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
	1	r4	-	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

```
div r1, r4, r2
sub r4, r2, r4
sub r6, r3, r2
mul r5, r2, r5
add r7, r4, r6
```



ROB에서 읽어오게 해야한다.

Register File (RF)

Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
1	r5	600
0	r6	700
1	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
	1	r4	-	0
	1	r6	-	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

`div r1, r4, r2`	IF	ID	EX	EX	EX
`sub r4, r2, r4`		IF	ID	EX	R
`sub r6, r3, r2`			IF	ID	EX
`mul r5, r2, r5`				IF	ID
`add r7, r4, r6`					IF

Register File (RF)

Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
0	r5	600
0	r6	700
1	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
	1	r4	-200	1
	1	r6	-	0
	1	r5	-	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

`div r1, r4, r2`	IF	ID	EX	EX	EX	EX
`sub r4, r2, r4`		IF	ID	EX	R	
`sub r6, r3, r2`			IF	ID	EX	R
`mul r5, r2, r5`				IF	ID	EX
`add r7, r4, r6`					IF	ID

add의 ID 단계

Register File (RF)

Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
0	r5	600
0	r6	700
0	r7	800

ROB에서 읽어와야함

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
	1	r4	-200	1
	1	r6	100	1
	1	r5	-	0
	1	r7	-	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

div r1, r4, r2	IF	ID	EX	EX	EX	EX	EX	EX
sub r4, r2, r4		IF	ID	EX	R			
sub r6, r3, r2			IF	ID	EX	R		
mul r5, r2, r5				IF	ID	EX	EX	EX
add r7, r4, r6					IF	ID	EX	R

Register File (RF)

Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
0	r5	600
0	r6	700
0	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	-	0
	1	r4	-200	1
	1	r6	100	1
	1	r5	-	0
	1	r7	-100	1
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	R
`sub r4, r2, r4`		IF	ID	EX	R					
`sub r6, r3, r2`			IF	ID	EX	R				
`mul r5, r2, r5`				IF	ID	EX	EX	EX	EX	EX
`add r7, r4, r6`					IF	ID	EX	R		

Register File (RF)

Valid	Name	Value
1	r0	100
0	r1	200
1	r2	300
1	r3	400
0	r4	500
0	r5	600
0	r6	700
0	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
HEAD	1	r1	2	1
	1	r4	-200	1
	1	r6	100	1
	1	r5	-	0
	1	r7	-100	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	R	WB
`sub r4, r2, r4`		IF	ID	EX	R						
`sub r6, r3, r2`			IF	ID	EX	R					
`mul r5, r2, r5`				IF	ID	EX	EX	EX	EX	EX	EX
`add r7, r4, r6`					IF	ID	EX	R			

Register File (RF)

Valid	Name	Value
1	r0	100
1	r1	2
1	r2	300
1	r3	400
0	r4	500
0	r5	600
0	r6	700
0	r7	800

Reorder Buffer (ROB)

HEAD 이동

	Valid	Dst Name	Value	Ready
	0	-	-	0
HEAD	1	r4	-200	1
	1	r6	100	1
	1	r5	-	0
	1	r7	-100	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

div r1, r4, r2	IF	ID	EX	EX	EX	EX	EX	EX	EX	R	WB	
sub r4, r2, r4		IF	ID	EX	R							WB
sub r6, r3, r2			IF	ID	EX	R						
mul r5, r2, r5				IF	ID	EX	EX	EX	EX	EX	EX	EX
add r7, r4, r6					IF	ID	EX	R				

Register File (RF)

Valid	Name	Value
1	r0	100
1	r1	2
1	r2	300
1	r3	400
1	r4	-200
0	r5	600
0	r6	700
0	r7	800

Reorder Buffer (ROB)

	Valid	Dst Name	Value	Ready
	0	-	-	0
	0	-	-	0
HEAD	1	r6	100	1
	1	r5	-	0
	1	r7	-100	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to use the data in ROB?

`div r1, r4, r2`	IF	ID	EX	EX	EX	EX	EX	EX	EX	R	WB		
`sub r4, r2, r4`		IF	ID	EX	R							WB	
`sub r6, r3, r2`			IF	ID	EX	R							WB
`mul r5, r2, r5`				IF	ID	EX	EX	EX	EX	EX	EX	EX	R
`add r7, r4, r6`					IF	ID	EX	R					

Register File (RF)

Valid	Name	Value
1	r0	100
1	r1	2
1	r2	300
1	r3	400
1	r4	-200
0	r5	600
1	r6	100
0	r7	800

Reorder Buffer (ROB)

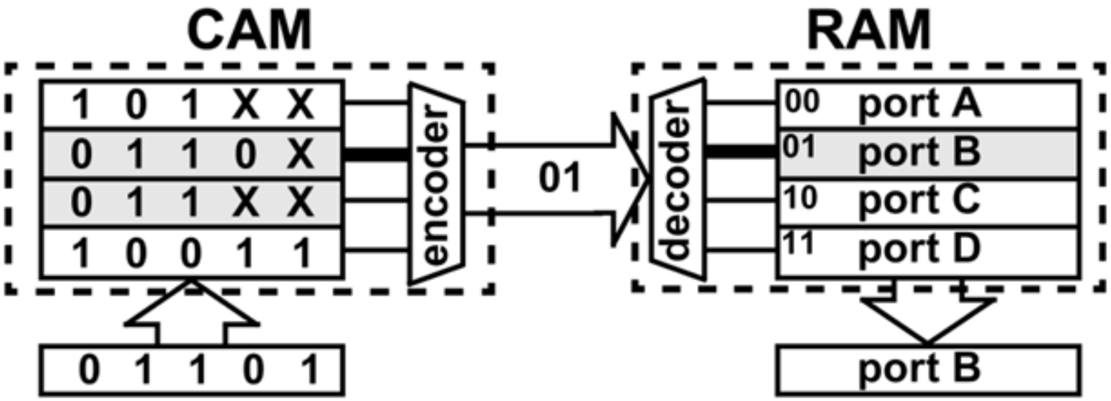
	Valid	Dst Name	Value	Ready
	0	-	-	0
	0	-	-	0
	0	-	-	0
HEAD	1	r5	180000	1
	1	r7	-100	0
TAIL	0	-	-	0
	0	-	-	0
	0	-	-	0

How to implement this?
Option #1: Using CAM

Value를 가지고
Address를 찾는것

- ROB utilizes a content addressable memory to search for entry (with the target dst reg)
  - You do not need to iterate over the buffers to check if the ROB entry keeps the operand!

CAM 사용 X → RoB 전체를 돌면서 찾음

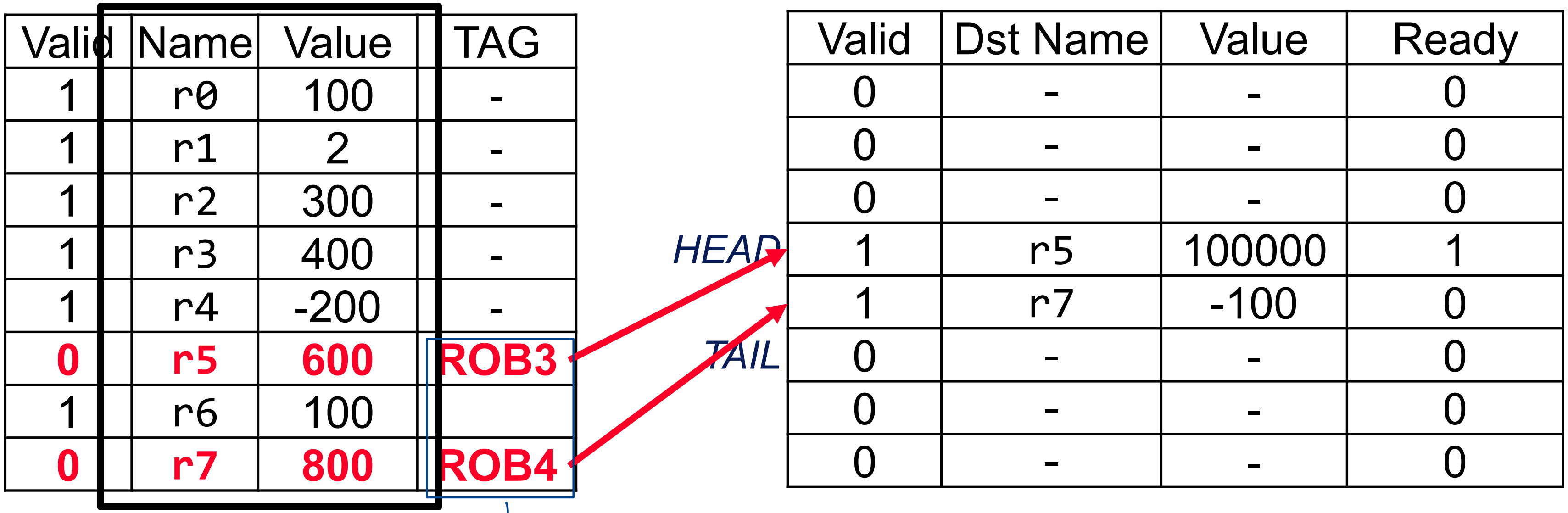


How would you handle multiple matches?

How to implement this?
Option #2: Using Indirection

- You can write down the entry address in the register file!

Register File (RF) ***Reorder Buffer (ROB)***



Valid	Name	Value	TAG
1	r0	100	-
1	r1	2	-
1	r2	300	-
1	r3	400	-
1	r4	-200	-
0	r5	600	ROB3
1	r6	100	
0	r7	800	ROB4

Valid	Dst Name	Value	Ready
0	-	-	0
0	-	-	0
0	-	-	0
1	r5	100000	1
1	r7	-100	0
0	-	-	0
0	-	-	0
0	-	-	0

↳ TAG 이용

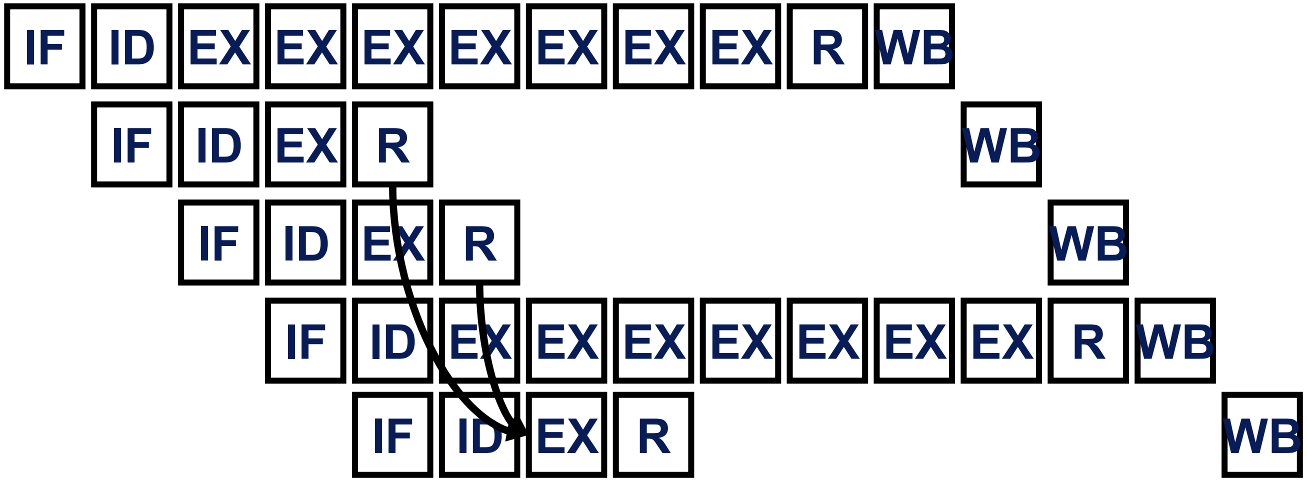
Architectural RF

You can think of this as renaming:

the source operand can either be in (1) RF or (2) ROB

Renaming Result!

Renaming → ROB에서 읽도록

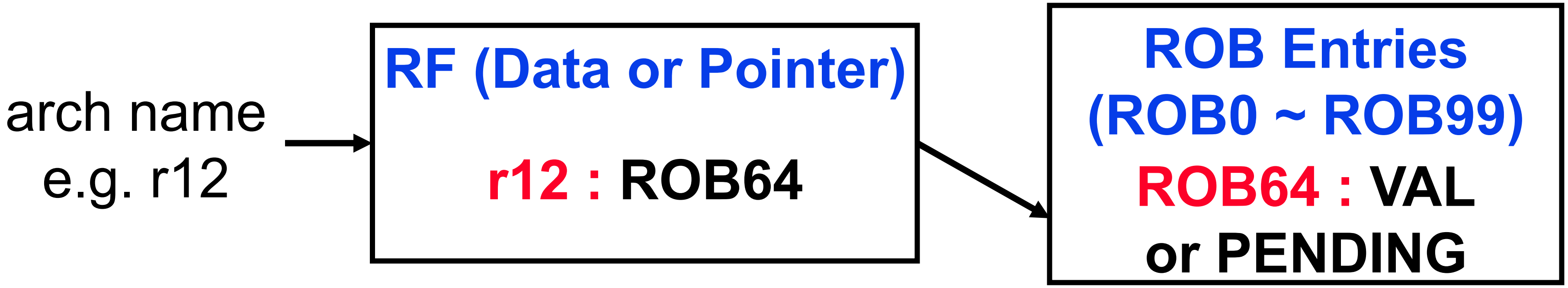


div r1, r4, r2	IF	ID	EX	EX	EX	EX	EX	EX	EX	R	WB				
sub **r4’**, r2, r4		IF	ID	EX	R							WB			
sub **r6’**, r3, r2			IF	ID	EX	R							WB		
mul r5, r2, r5				IF	ID	EX	EX	EX	EX	EX	EX	EX	R	WB	
add r7, **r4’**, **r6’**					IF	ID	EX	R							WB

We do not read from the register file, but rather from the renamed entries @ ROB

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Hardware register renaming



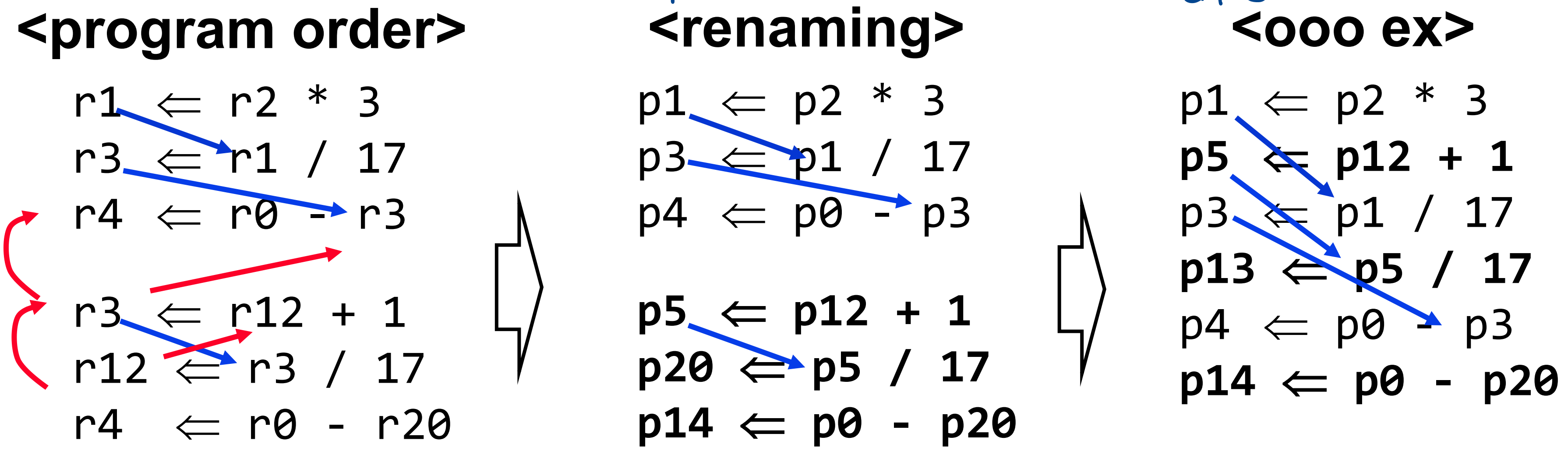
- Renaming maintains bindings from **arch reg names** to **uarch reg names** RoB
  - Compiler does not know about uarch rename registers.
- When issuing an instruction that updates 'architecture' register 'rd':
  - Allocate an unused rename 'physical' register 'px'
  - Record binding from 'rd' to 'px'
- When to remove a binding? When to de-allocate a rename register?

Out-of-order execution (runtime scheduling!)

- After renaming, the WAW and WAR dependencies are almost eliminated
- If there is a true dependency (RAW) for a decoded instruction, the instruction cannot be executed (stalled)
- While a previous instruction is stalled, the later instructions can be dispatched to the EX/MEM units (unless there is a true dependency)

→ WAW, WAR 사라짐

Data dependency가 없는 명령어는 먼저 실행하도록!



<program order>

```
r1  ⇐ r2 * 3
r3  ⇐ r1 / 17
r4  ⇐ r0 - r3

r3  ⇐ r12 + 1
r12 ⇐ r3 / 17
r4  ⇐ r0 - r20
```

<renaming>

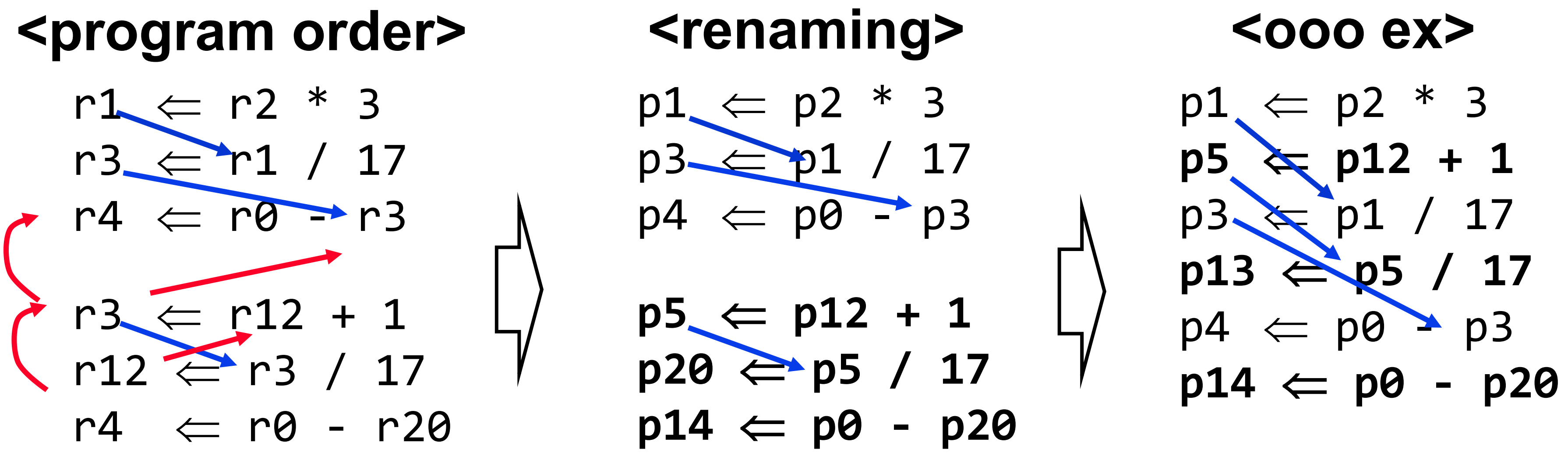
```
p1  ⇐ p2 * 3
p3  ⇐ p1 / 17
p4  ⇐ p0 - p3

p5  ⇐ p12 + 1
p20 ⇐ p5 / 17
p14 ⇐ p0 - p20
```

<ooo ex>

```
p1  ⇐ p2 * 3
p5  ⇐ p12 + 1
p3  ⇐ p1 / 17
p13 ⇐ p5 / 17
p4  ⇐ p0 - p3
p14 ⇐ p0 - p20
```

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

Out-of-order execution (runtime scheduling!)



<program order>

```
r1  ⇐ r2 * 3
r3  ⇐ r1 / 17
r4  ⇐ r0 - r3

r3  ⇐ r12 + 1
r12  ⇐ r3 / 17
r4   ⇐ r0 - r20
```

<renaming>

```
p1  ⇐ p2 * 3
p3  ⇐ p1 / 17
p4  ⇐ p0 - p3

p5  ⇐ p12 + 1
p20 ⇐ p5 / 17
p14 ⇐ p0 - p20
```

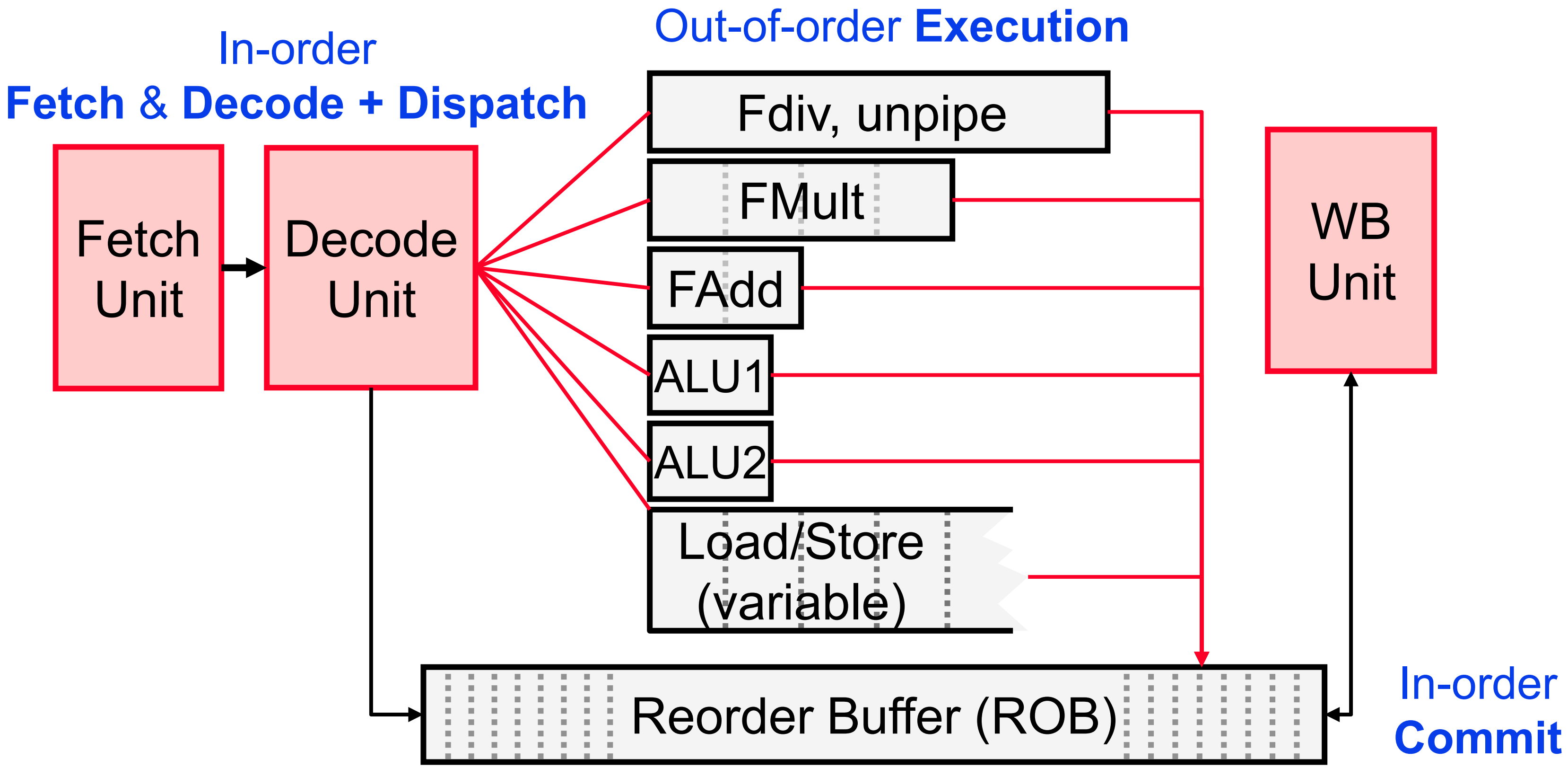
<ooo ex>

```
p1  ⇐ p2 * 3
p5  ⇐ p12 + 1
p3  ⇐ p1 / 17
p13 ⇐ p5 / 17
p4  ⇐ p0 - p3
p14 ⇐ p0 - p20
```

- If dep. distance > issue distance, even RAW is eliminated
- With OoO, superscalar gets much better!
- But OoO is an microarchitectural feature **(not exposed to the programmer and compiler)**

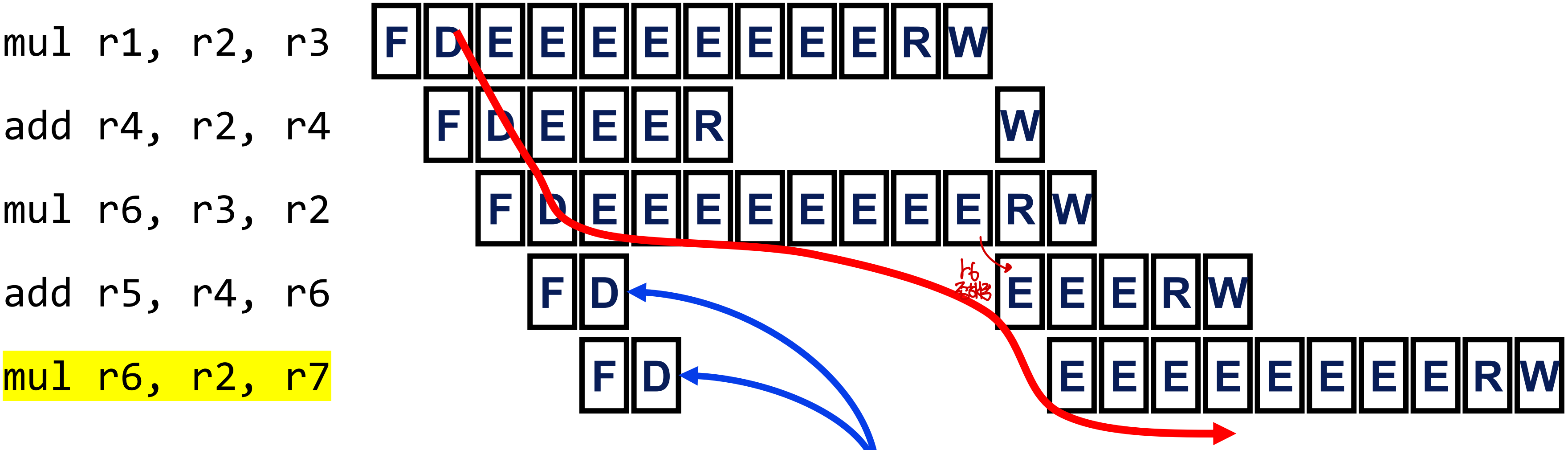
Reorder buffer concept

- The instructions are
  - (1) fetched / decoded + dispatched to the execution units **in order**
  - (2) executed **out-of-order** (a newer instruction may complete execution earlier)
  - (3) written back to the register file **in-order**



Potential improvement of reordering

- add/sub utilizes the same unit and takes three cycles
- mul utilizes a different unit and takes eight cycles



`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W											
`add r4, r2, r4`		F	D	E	E	E	R						W										
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R	W									
`add r5, r4, r6`				F	D								E	E	E	R	W						
`mul r6, r2, r7`					F	D								E	E	E	E	E	E	E	E	R	W

In-order dispatch
Decoded data stored in **ID_EX_LATCH**

In-order dispatch

It seems that `mul r6, r2, r7` becomes the bottleneck… (Can we do better?)

Out-of-order dispatch

→ 순서대로 dispath 하지 않음

- We can dispatch `mul r6, r2, r7` before dispatching `add r5, r6, r5`
  - Can hide the executing latency!

`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W						
`add r4, r2, r4`		F	D	E	E	E	R						W					
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R	W				
`add r5, r4, r6`				F	D								E	E	E	R	W	
`mul r6, r2, r7`					F	D	E	E	E	E	E	E	E	E	R			W

← Decode한 결과는 기록해야할 공간이 필요하다.

← 다른 명령어에서 이 명령어가 사용할 reg의 값이 정해졌다고 알림을 받는 Buffer가 필요

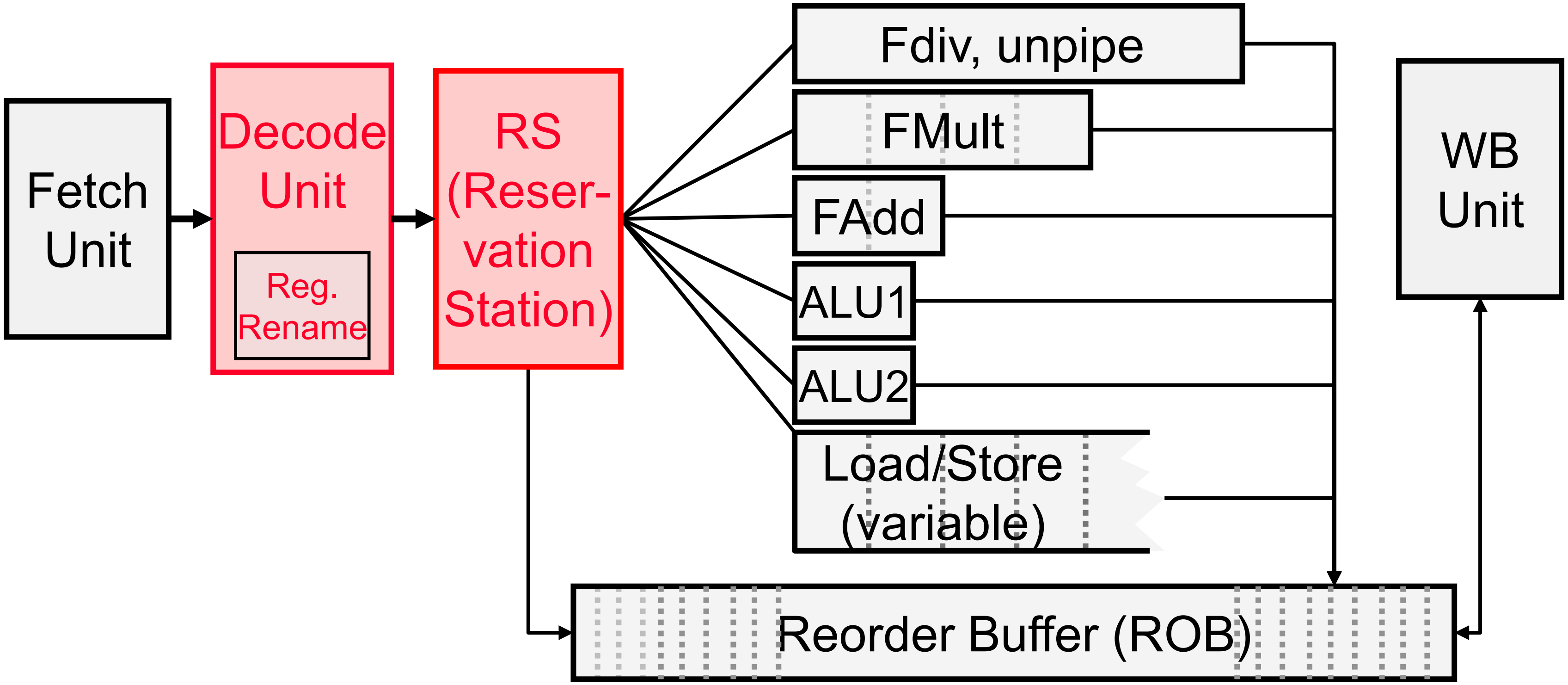
→ r2, r7은 다른 명령에 기다릴 필요가 없다.

Overwrites ID_EX_LATCH

The decoded data @ add r5, r4, r6 are overwritten @ mul r6, r2, r7

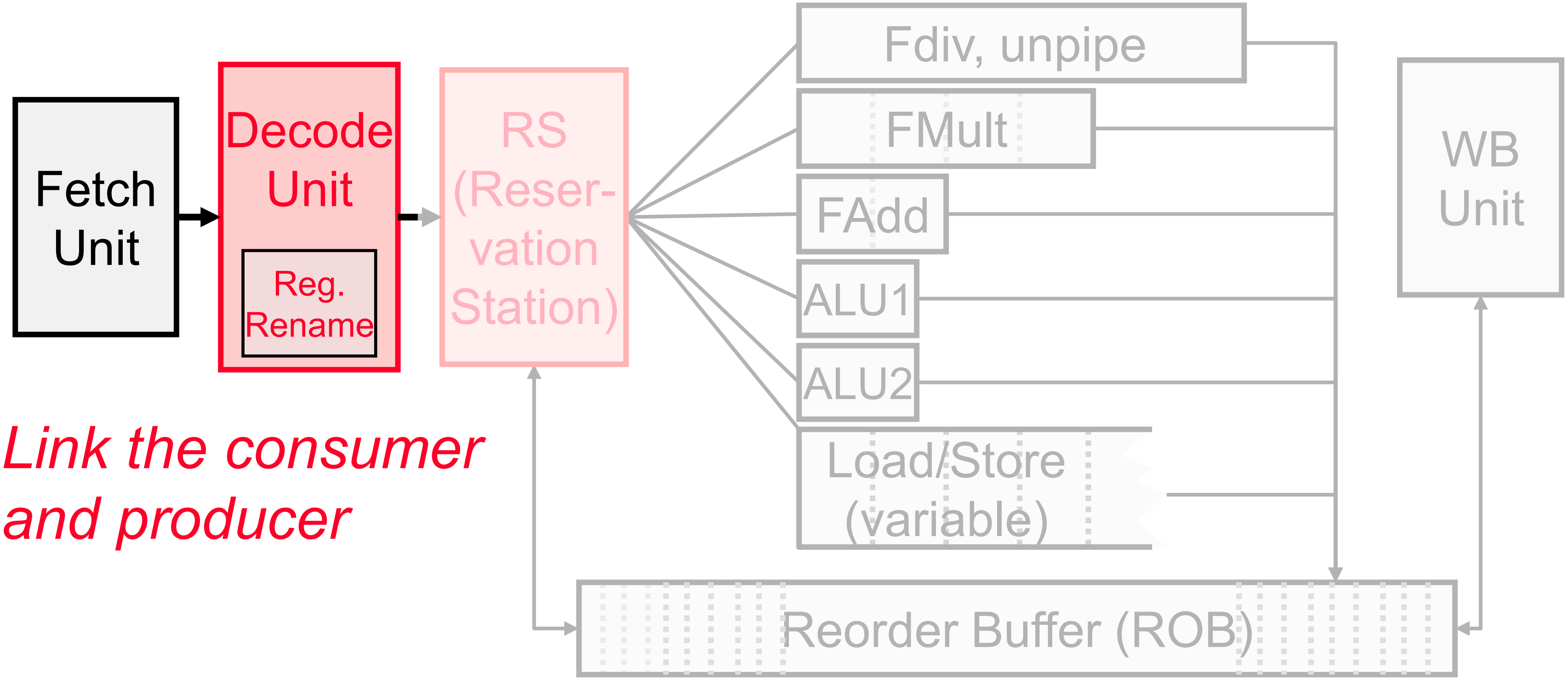
OoO CPU design

- We need to extend the existing architecture to support runtime instruction scheduling
  - 1) Rename register online to mitigate false dependency
  - 2) Decouple fetch and execution using an issue buffer



OoO CPU design

- We need to extend the existing architecture to support runtime instruction scheduling
  - 1) Rename register online to mitigate false dependency
  - 2) Decouple fetch and execution using an issue buffer

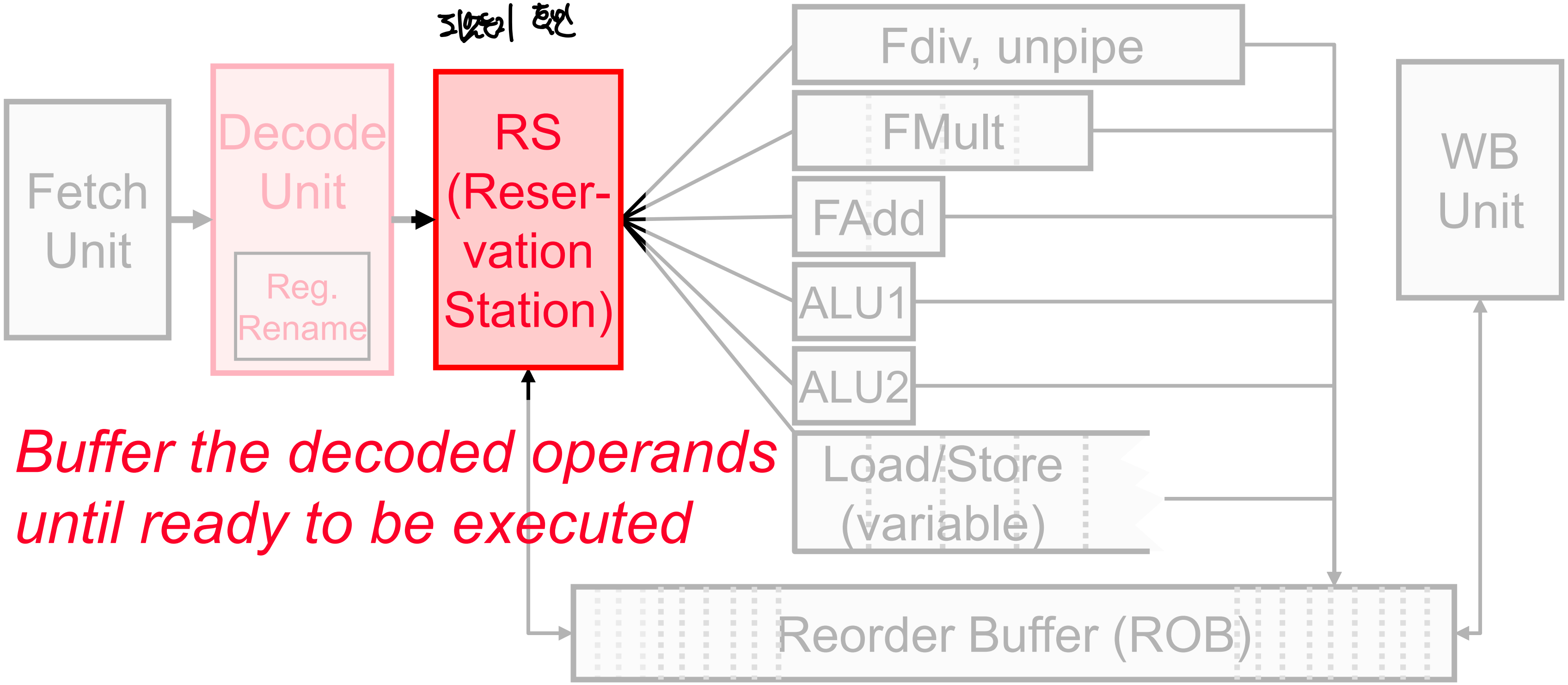


Link the consumer and producer

OoO CPU design

- We need to extend the existing architecture to support runtime instruction scheduling
  - 1) Rename register online to mitigate false dependency
  - 2) Decouple fetch and execution using an issue buffer

실행할 준비가
되었는지 확인

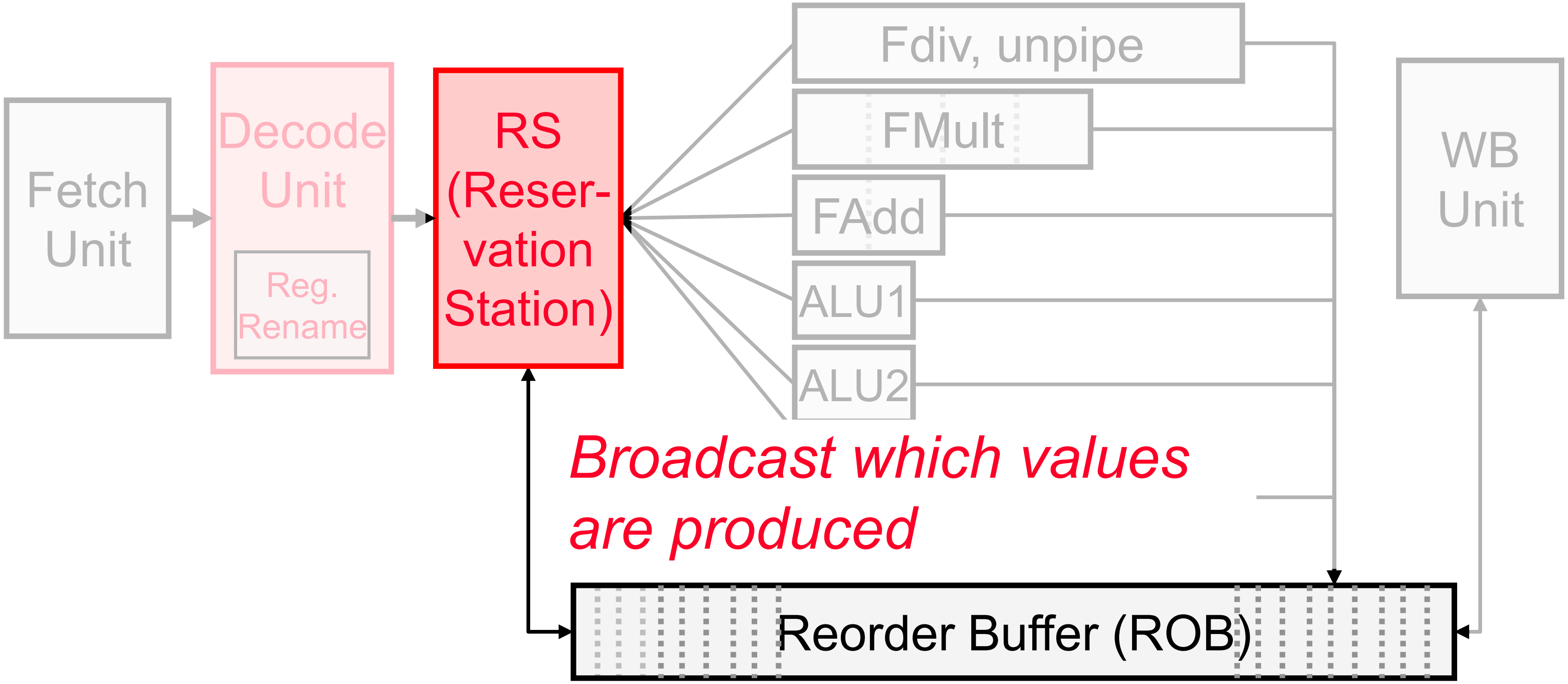


Buffer the decoded operands until ready to be executed

© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

OoO CPU design

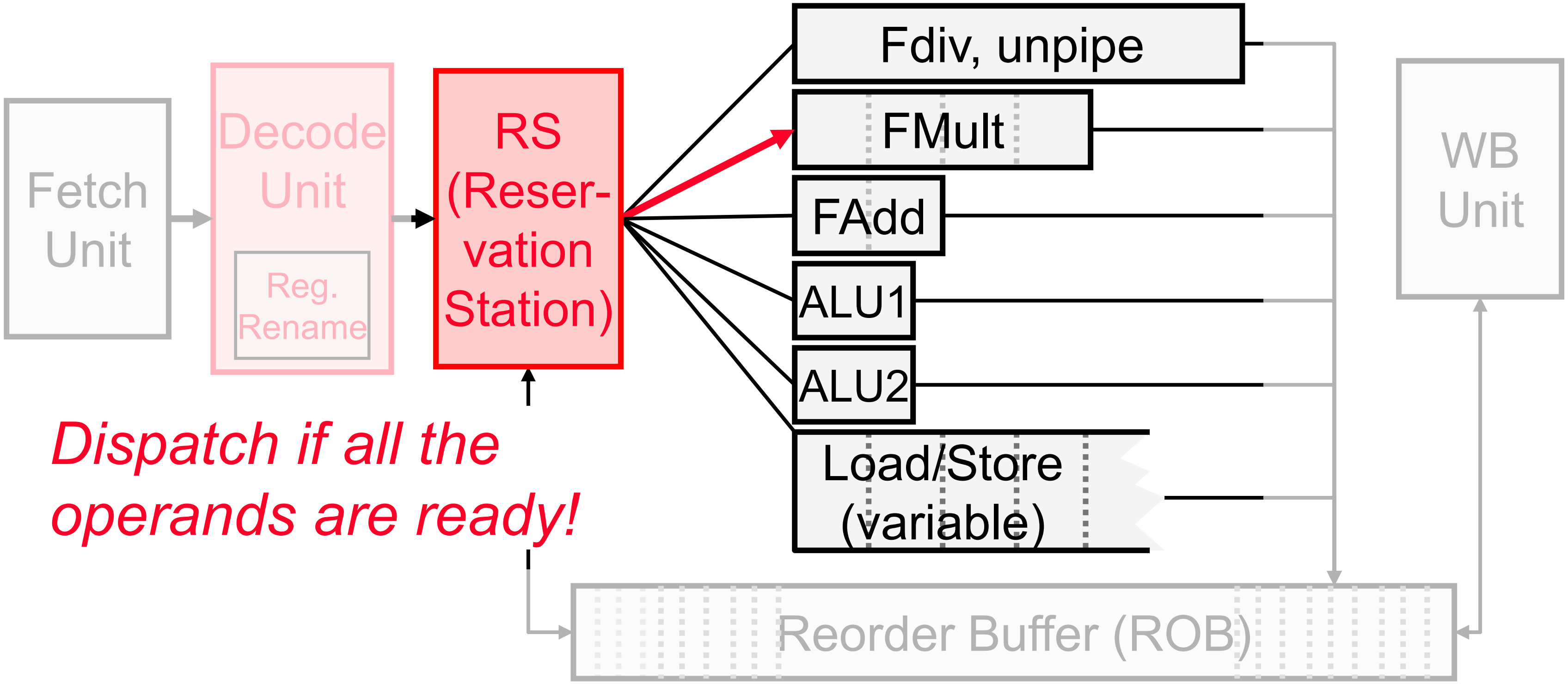
- We need to extend the existing architecture to support runtime instruction scheduling
  - 1) Rename register online to mitigate false dependency
  - 2) Decouple fetch and execution using an issue buffer



Broadcast which values are produced

OoO CPU design

- We need to extend the existing architecture to support runtime instruction scheduling
  - 1) Rename register online to mitigate false dependency
  - 2) Decouple fetch and execution using an issue buffer



Dispatch if all the operands are ready!

Register renaming implementation: Tomasulo’s algorithm

- A physical register is a combination of architectural register + ROB entries
  - The value can either be in the ROB or architectural reg file
  - There is a table to indicate where the data resides

- Let’s start with a reservation station + register file w/o ROB (we’ll get to ROB later on …)

We should extend renaming

```
mul r1, r2, r3   F
add r4, r2, r4
mul r6, r3, r2
add r5, r4, r6
mul r6, r2, r7
```

Register File (RF)

Valid	Name	Value	TAG
1	r0	0	-
1	r1	1	-
1	r2	2	-
1	r3	3	-
1	r4	4	-
1	r5	5	-
1	r6	6	-
1	r7	7	-

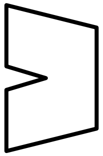
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1						
A2						

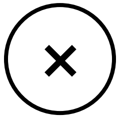
Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1						
M2						

ALU



MUL



We should extend renaming

```
mul r1, r2, r3    F D
add r4, r2, r4      F
mul r6, r3, r2
add r5, r4, r6
mul r6, r2, r7
```

M0가 끝나면 나에게
Write Back 해라

Register File (RF)

2. Access the RF

Val			G
1			
0	r1	-	M0
1	r2	2	-
1	r3	3	-
1	r4	4	-
1	r5	5	-
1	r6	6	-
1	r7	7	-

Rename

Reservation Station (ALU)

4. Rename Entry

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A2						

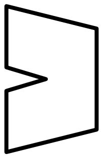
tation (MUL)

3. Reserve Op

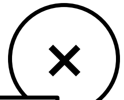
	SRC1 r2			SRC2 r3		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1						
M2						

1. Is RS available (empty)?

ALU



MUL



© Lee 2024 -- Portions © Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, Wenisch, Mutlu, Kim

We should extend renaming

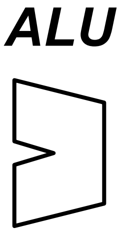
```
mul r1, r2, r3   F D E
add r4, r2, r4     F D
mul r6, r3, r2       F
add r5, r4, r6
mul r6, r2, r7
```

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	M0
1	r2	2	-
1	r3	3	-
0	r4	4	A0
1	r5	5	-
1	r6	6	-
1	r7	7	-

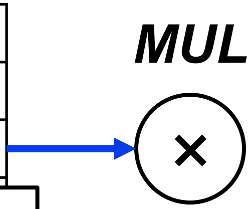
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0	1	-	2	1	-	4
A1						
A2						



Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1						
M2						



Dispatch if possible!

We should extend renaming

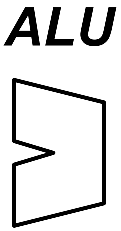
`mul r1, r2, r3`	F	D	E	E
`add r4, r2, r4`		F	D	E
`mul r6, r3, r2`			F	D
`add r5, r4, r6`				F
`mul r6, r2, r7`				

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	M0
1	r2	2	-
1	r3	3	-
0	r4	4	A0
1	r5	5	-
0	r6	6	M1
1	r7	7	-

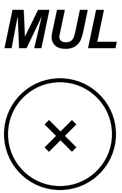
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0	1	-	2	1	-	4
A1						
A2						



Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1	1	-	3	1	-	2
M2						



We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E
`add r4, r2, r4`		F	D	E	E
`mul r6, r3, r2`			F	D	E
`add r5, r4, r6`				F	D
`mul r6, r2, r7`					F

앞선 명령어 EX가 끝나고 사용 가능

Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0	1	-	2	1	-	4
A1	0	A0	-	0	M1	-
A2						

↳여기서 write?

ALU

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	M0
1	r2	2	-
1	r3	3	-
0	r4	4	A0
0	r5	5	A1
0	r6	6	M1
1	r7	7	-

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1	1	-	3	1	-	2
M2						

MUL

×

We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E	E
`add r4, r2, r4`		F	D	E	E	E
`mul r6, r3, r2`			F	D	E	
`add r5, r4, r6`				F	D	
`mul r6, r2, r7`					F	

해제

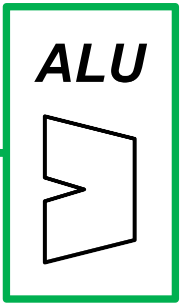
A0 done

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	M0
1	r2	2	-
1	r3	3	-
1	**r4**	**6**	**-**
0	r5	5	A1
0	r6	6	M1
1	r7	7	-

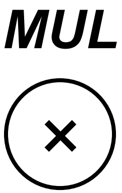
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0	~~1~~	~~-~~	~~2~~	~~1~~	~~-~~	~~4~~
A1	**1**	**A0**	**6**	0	M1	-
A2						



Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1	1	-	3	1	-	2



We should extend renaming

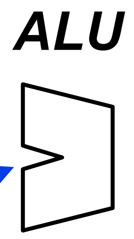
`mul r1, r2, r3`	F	D	E	E	E	E
`add r4, r2, r4`		F	D	E	E	E
`mul r6, r3, r2`			F	D	E	E
`add r5, r4, r6`				F	D	
`mul r6, r2, r7`					F	D

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	M0
1	r2	2	-
1	r3	3	-
1	r4	6	-
0	r5	5	A1
0	r6	6	M2
1	r7	7	-

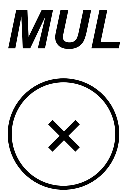
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	A0	6	**0**	M1	-
A2						



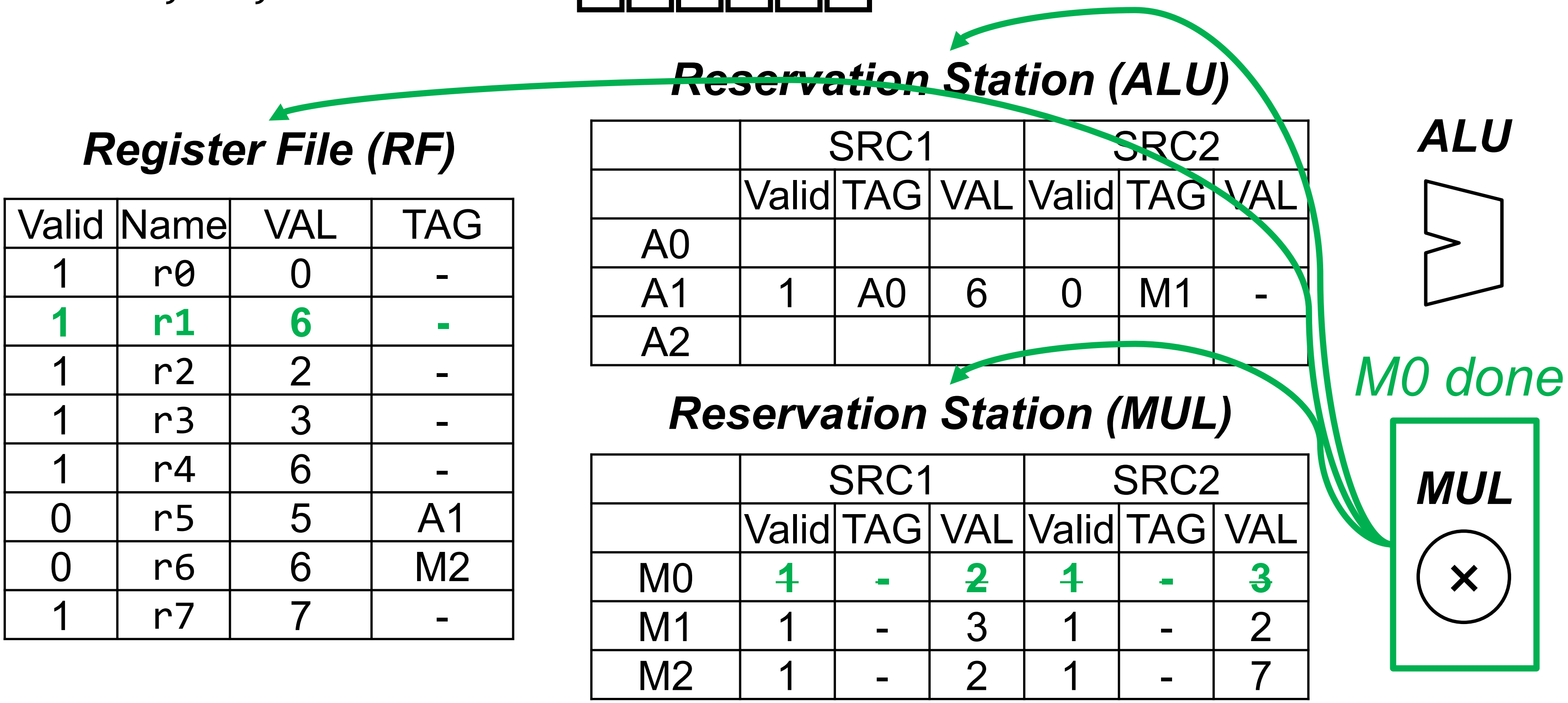
Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1	1	-	3	1	-	2
M2	1	-	2	1	-	7



We should extend renaming

mul r1, r2, r3	F	D	E	E	E	E	E	E	E	E
add r4, r2, r4		F	D	E	E	E	R			
mul r6, r3, r2			F	D	E	E	E	E	E	E
add r5, r4, r6				F	D					
mul r6, r2, r7					F	D	E	E	E	E



Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
1	**r1**	**6**	**-**
1	r2	2	-
1	r3	3	-
1	r4	6	-
0	r5	5	A1
0	r6	6	M2
1	r7	7	-

Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	A0	6	0	M1	-
A2						

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	~~1~~	~~-~~	~~2~~	~~1~~	~~-~~	~~3~~
M1	1	-	3	1	-	2
M2	1	-	2	1	-	7

We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W
`add r4, r2, r4`		F	D	E	E	E	R					
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E
`add r5, r4, r6`				F	D							
`mul r6, r2, r7`					F	D	E	E	E	E	E	E

결! 얘는 그리고
ROB에서 pop 되는 순간에
Register File에 Write함

Register File (RF)

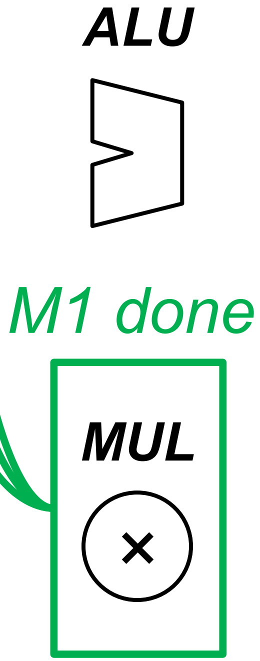
Valid	Name	VAL	TAG
1	r0	0	-
1	r1	6	-
1	r2	2	-
1	r3	3	-
1	r4	6	-
0	r5	5	A1
0	r6	6	M2
1	r7	7	-

Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	A0	6	1	-	6
A2						

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1	~~1~~	~~-~~	~~3~~	~~1~~	~~-~~	~~2~~
M2	1	-	2	1	-	7



We should extend renaming

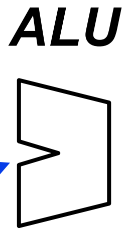
`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W	
`add r4, r2, r4`		F	D	E	E	E	R						W
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R
`add r5, r4, r6`				F	D								E
`mul r6, r2, r7`					F	D	E	E	E	E	E	E	E

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
1	r1	6	-
1	r2	2	-
1	r3	3	-
1	r4	6	-
0	r5	5	A1
0	r6	6	M2
1	r7	7	-

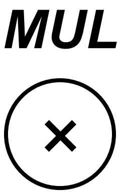
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	A0	6	1	-	6
A2						



Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1						
M2	1	-	2	1	-	7



We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W		
`add r4, r2, r4`		F	D	E	E	E	R						W	
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R	W
`add r5, r4, r6`				F	D								E	E
`mul r6, r2, r7`					F	D	E	E	E	E	E	E	E	E

Register File (RF)

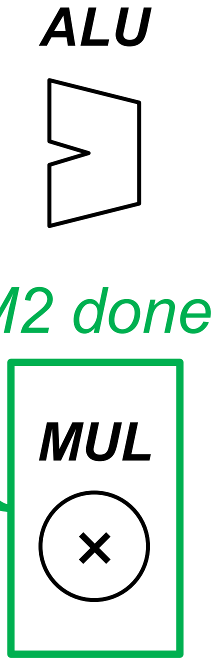
Valid	Name	VAL	TAG
1	r0	0	-
1	r1	6	-
1	r2	2	-
1	r3	3	-
1	r4	6	-
0	r5	5	A1
1	**r6**	**14**	**-**
1	r7	7	-

Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	A0	6	1	-	6
A2						

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1						
M2	~~1~~	~~-~~	~~2~~	~~1~~	~~-~~	~~7~~

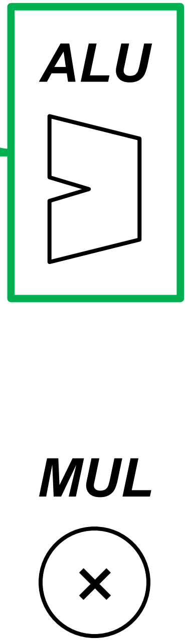


M2 done

We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W			
`add r4, r2, r4`		F	D	E	E	E	R						W		
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R	W	
`add r5, r4, r6`				F	D								E	E	E
`mul r6, r2, r7`					F	D	E	E	E	E	E	E	E	E	R

A1 done



Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
1	r1	6	-
1	r2	2	-
1	r3	3	-
1	r4	6	-
1	r5	12	-
1	r6	14	-
1	r7	7	-

Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	~~1~~	~~A0~~	~~6~~	~~1~~	~~-~~	~~6~~
A2						

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1						
M2						

We should extend renaming

`mul r1, r2, r3`	F	D	E	E	E	E	E	E	E	E	R	W						
`add r4, r2, r4`		F	D	E	E	E	R						W					
`mul r6, r3, r2`			F	D	E	E	E	E	E	E	E	E	R	W				
`add r5, r4, r6`				F	D								E	E	E	R	W	
`mul r6, r2, r7`					F	D	E	E	E	E	E	E	E	E	R			W

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
1	r1	6	-
1	r2	2	-
1	r3	3	-
1	r4	6	-
1	r5	12	-
1	r6	14	-
1	r7	7	-

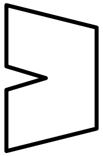
Reservation Station (ALU)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1						
A2						

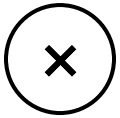
Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0						
M1						
M2						

ALU



MUL



OoO + RS + ROB

Register File (RF)

Valid	Name	VAL	TAG
1	r0	0	-
0	r1	-	**R0**
1	r2	2	-
1	r3	3	-
0	r4	6	**R1**
0	r5	5	**R3**
0	r6	6	**R4**
1	r7	7	-

Reservation Station (ALU)

R1 풀렸다.

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
A0						
A1	1	**R1**	6	0	**R2**	-
A2						

Reservation Station (MUL)

	SRC1			SRC2		
	Valid	TAG	VAL	Valid	TAG	VAL
M0	1	-	2	1	-	3
M1	1	-	3	1	-	2
M2	1	-	2	1	-	7

Timeline

`mul r1, r2, r3`	F	D	E	E	E	E	E
`add r4, r2, r4`		F	D	E	E	E	R
`mul r6, r3, r2`			F	D	E	E	E
`add r5, r4, r6`				F	D		
`mul r6, r2, r7`					F	D	E

Reorder Buffer

Name	Valid	Dst Name	Value	Ready
R0	1	r1	-	0
R1	**1**	**r4**	**6**	**1**
R2	1	r6	-	0
R3	1	r5	-	0
R4	1	r6	-	0

Control dependency can hurt ILP

- We also suffer from control dependency (Some fetched instructions in RS can be potentially invalid)

- Control instructions occupy 14% of an average number of instructions
  - If there are 128 in-flight instructions ➔ there are 18 branches
  - If we have a 90% correct branch predictor → (0.9)^18 → 15% chance of all the branches are correct
    - We only have 83% chance even with 99% correct branch predictor

  - We indeed need an extremely accurate branch predictor!!

Question?

Announcements:	
Reading:	*finish reading P&H Ch.4*
Handouts:	*none*