Computer Architecture

Lecture 04 – SuperScalar and OOO (Instruction level Parallelism)

Pengju Ren
Institute of Artificial Intelligence and Robotics
Xi'an Jiaotong University
http://gr.xjtu.edu.cn/web/pengjuren

Agenda

SuperScalar Intro

Out-of-Order Processor (OOO)

Speculation and Branches

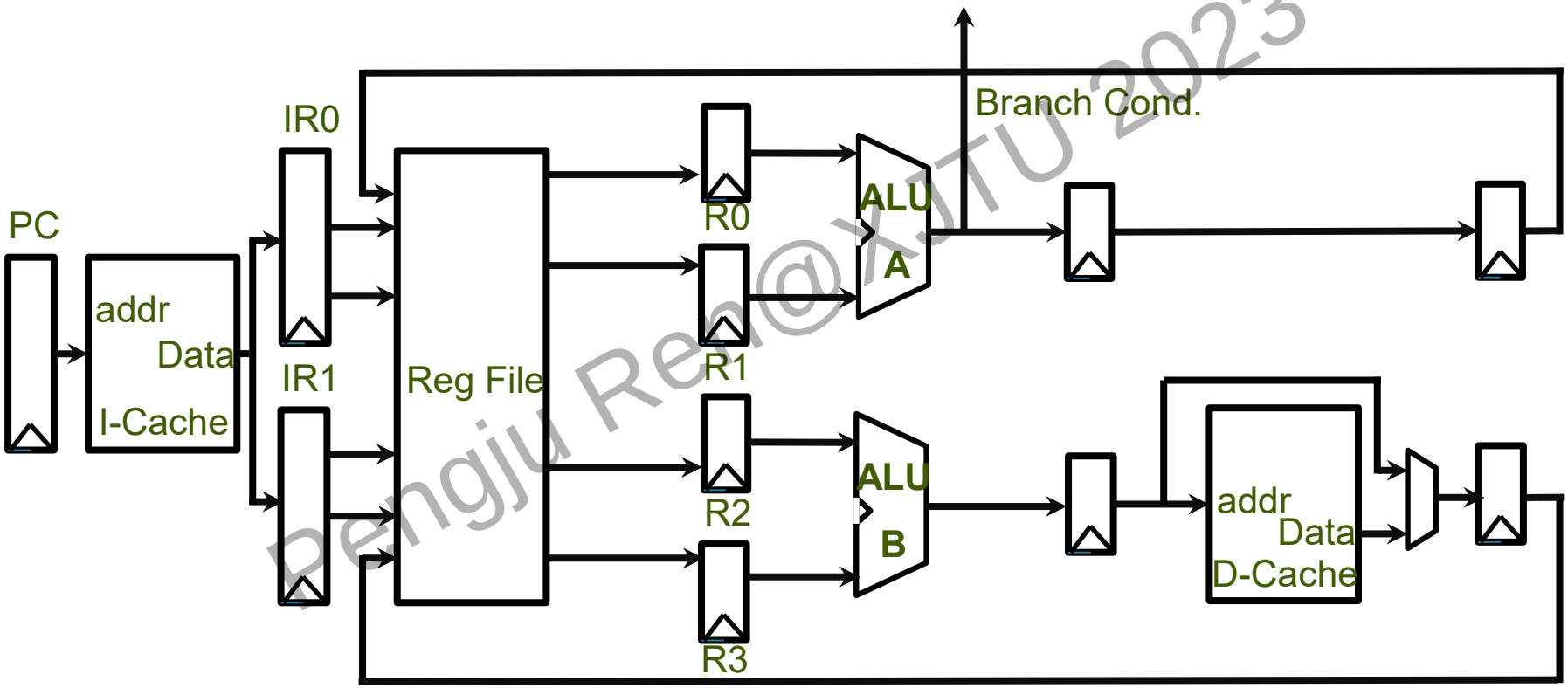
Register Renaming

Memory Disambiguation

Introduction to SuperScalar Processor

- Processors studied so far are fundamentally limited to CPI >= 1
- SuperScalar and VLIW processor enable CPI < 1 (or IPC > 1) by executing **multiple instructions in parallel**
- Can have both **in-order** and **out-of-order(OOO)** SuperScalar processors. We will start with in-order.

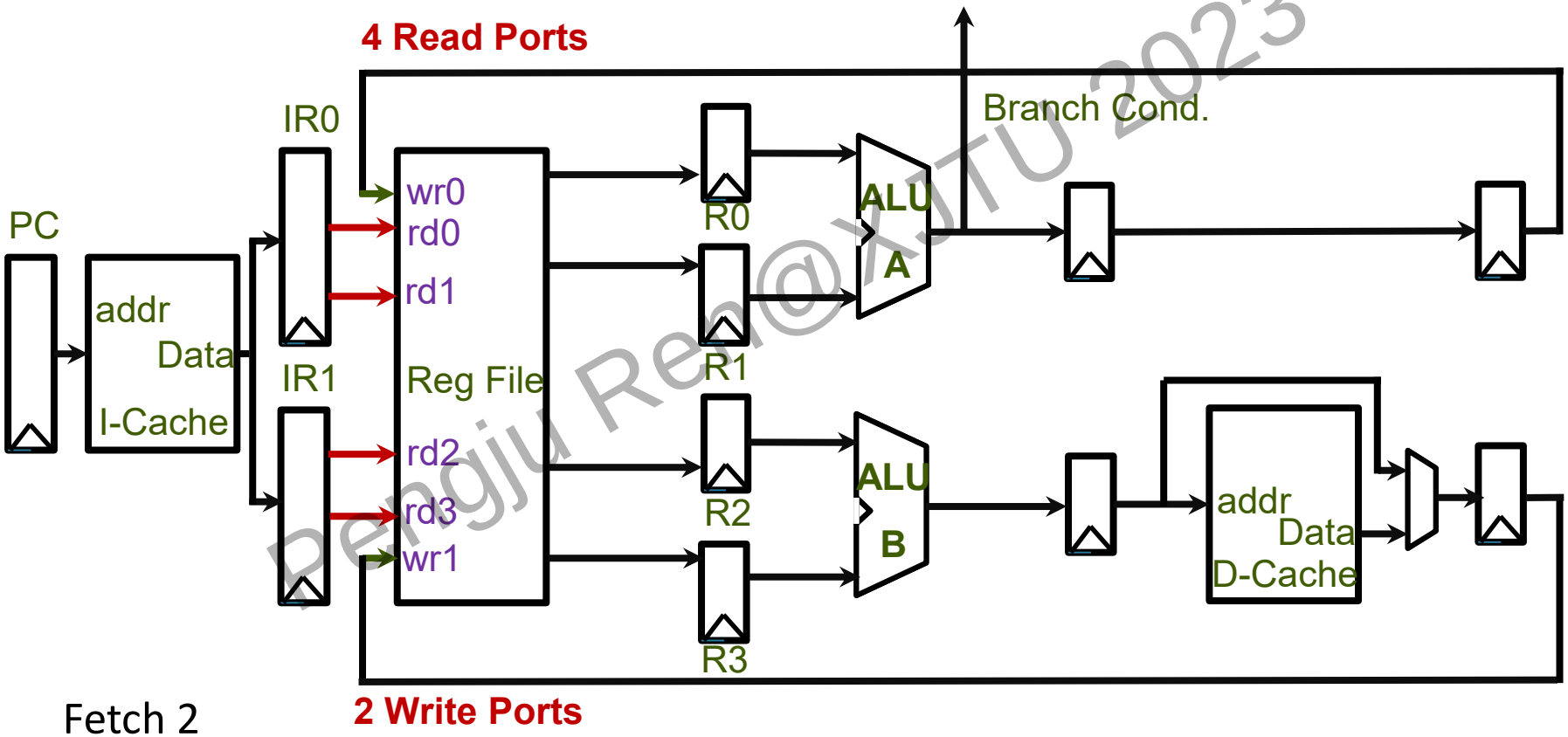
Baseline 2-Way In-Order SuperScalar Processor



Fetch 2 Instructions at the same time

Pipe A : Integer Ops., Branches
Pipe B : Integer Ops., Memory

Baseline 2-Way In-Order SuperScalar Processor

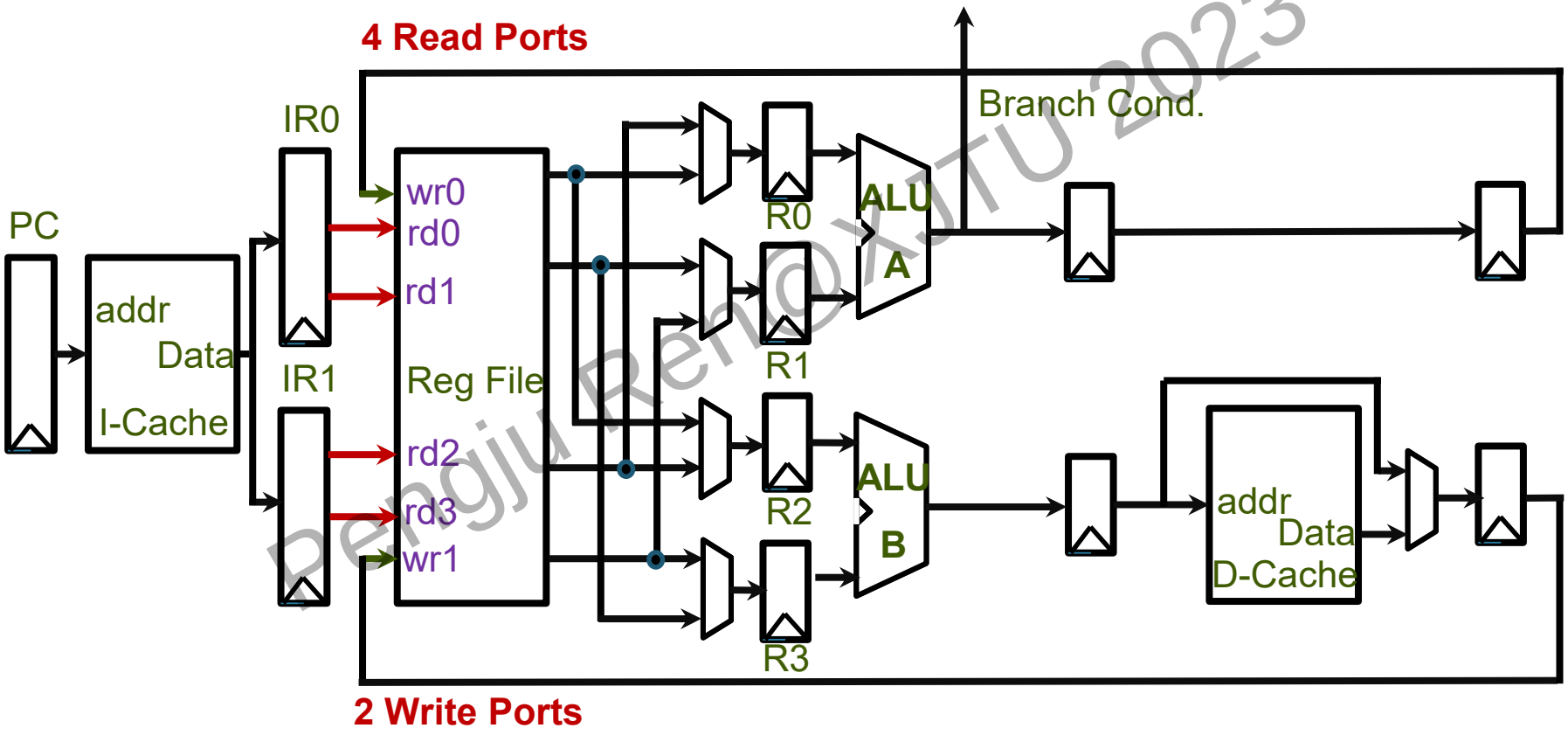


Fetch 2 Instructions at the same time

Pipe A : Integer Ops., Branches

Pipe B : Integer Ops., Memory

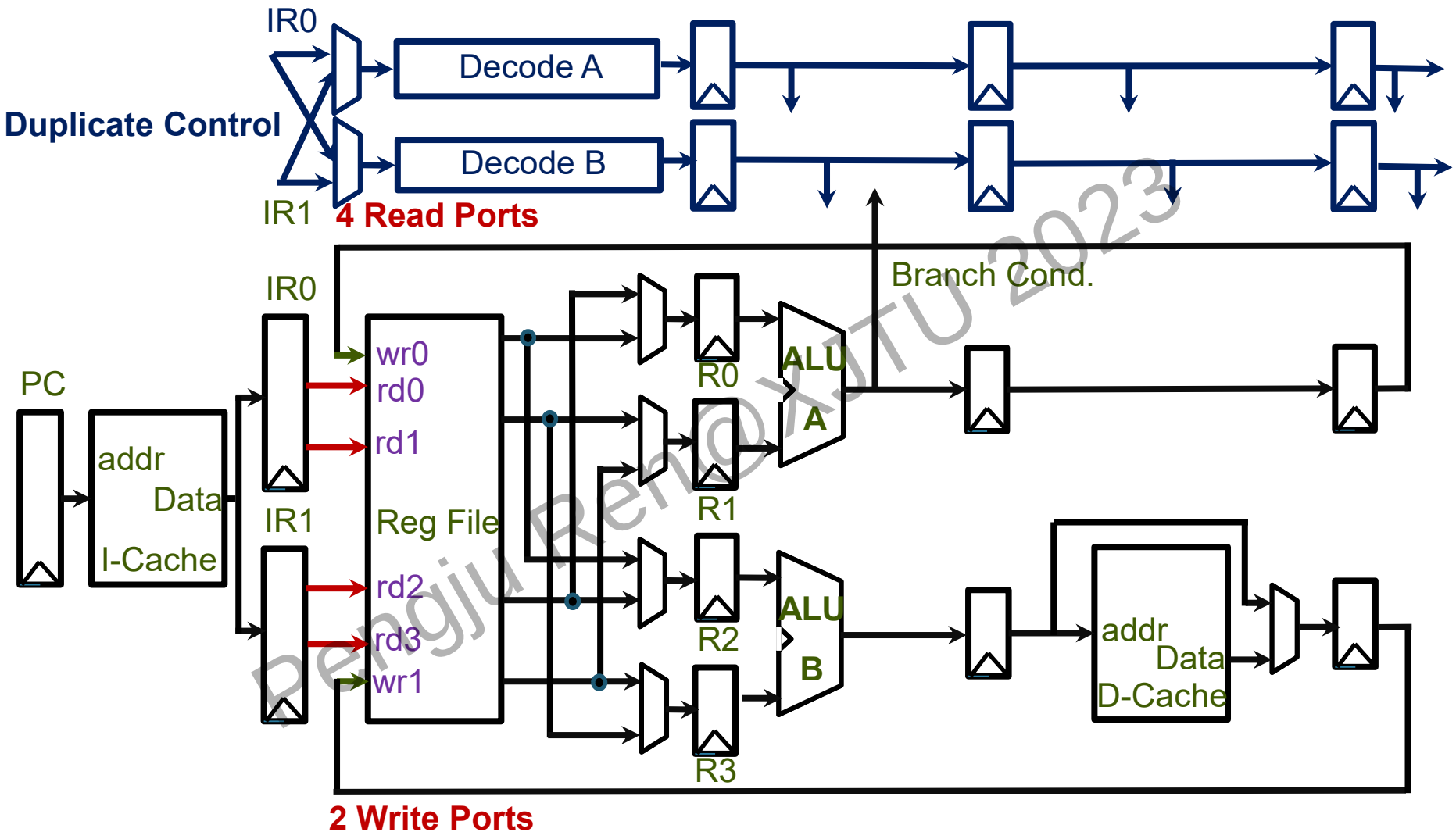
Baseline 2-Way In-Order SuperScalar Processor



Pipe A : Integer Ops., Branches

Pipe B : Integer Ops., Memory

Baseline 2-Way In-Order SuperScalar Processor



Pipe A : Integer Ops., Branches

Pipe B : Integer Ops., Memory

Issue Logic Pipeline Diagrams

OpA	F	D	A0	A1	W		
OpB	F	D	B0	B1	W		
OpC		F	D	A0	A1	W	
OpD		F	D	B0	B1	W	
OpE			F	D	A0	A1	W
OpF			F	D	B0	B1	W

CPI = 0.5 (IPC=2)

Ideally, Double issue pipeline can have two instructions in same stage at the same time

ADD	F	D	A0	A1	W		
LW	F	D	B0	B1	W		
LW		F	D	B0	B1	W	
ADD		F	D	A0	A1	W	
LW			F	D	B0	B1	W
LW			F	D	D	B0	B1

Instruction Issue logic swaps from natural position

Structural Hazard

Dual Issue Data Hazards

No Bypassing:										
ADDI X1, X1, 1		F	D	A0	A1	W				
ADDI X3, X4, 1		F	D	B0	B1	W				
ADDI X5, X6, 1			F	D	A0	A1	W			
ADDI X7, X5, 1			F	D	D	D	D	A0	A1	W

Full Bypassing:										
ADDI X1, X1, 1		F	D	A0	A1	W				
ADDI X3, X4, 1		F	D	B0	B1	W				
ADDI X5, X6, 1			F	D	A0	A1	W			
ADDI X7, X5, 1			F	D	D	A0	A1	W		

After swap:										
ADDI X1, X1, 1		F	D	A0	A1	W				
ADDI X3, X4, 1		F	D	B0	B1	W				
ADDI X7, X5, 1			F	D	A0	A1	W			
ADDI X5, X6, 1			F	D	B0	B1	W			

Order matters

Fetch Logic and Alignment

Cycle	Addr	Instr
0	0x000	OpA
0	0x004	OpB
1	0x008	OpC
1	0x00C	J 0x100
…		
2	0x100	OpD
2	0x104	J 0x204
…		
3	0x204	OpE
3	0x208	J 0x30C
…		
4	0x30C	OpF
4	0x310	OpG
5	0x314	OpH

0x000	0	0	1	1
…				
0x100	2	2		
…				
0x200		3	3	
…				
0x300				4
0x310	4	5		

Fetching across cache lines（e.g Cache block=128bits) is very expensive (need extra ports)

Fetch Logic and Alignment

Cycle	Addr	Instr
0	0x000	OpA
0	0x004	OpB
1	0x008	OpC
1	0x00C	J 0x100
…		
2	0x100	OpD
2	0x104	J 0x204
…		
3	0x204	OpE
3	0x208	J 0x30C
…		
4	0x30C	OpF
4	0x310	OpG
5	0x314	OpH

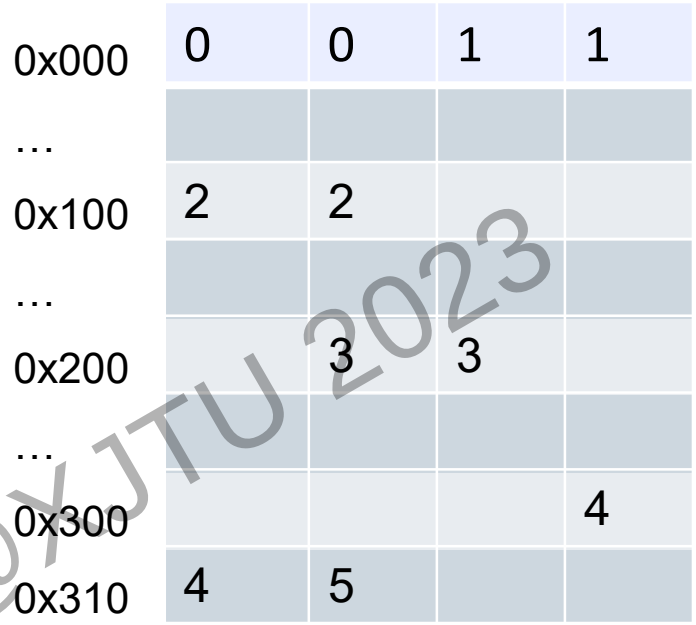
Ideal, No alignment constraints

OpA	F	D	A0	A1	W				
OpB	F	D	B0	B1	W				
OpC		F	D	B0	B1	W			
J		F	D	A0	A1	W			
OpD			F	D	B0	B1	W		
J			F	D	A0	A1	W		
OpE				F	D	B0	B1	W	
J				F	D	A0	A1	W	
OpF					F	D	A0	A1	W
OpG					F	D	B0	B1	W
OpH						F	D	A0	A1	W

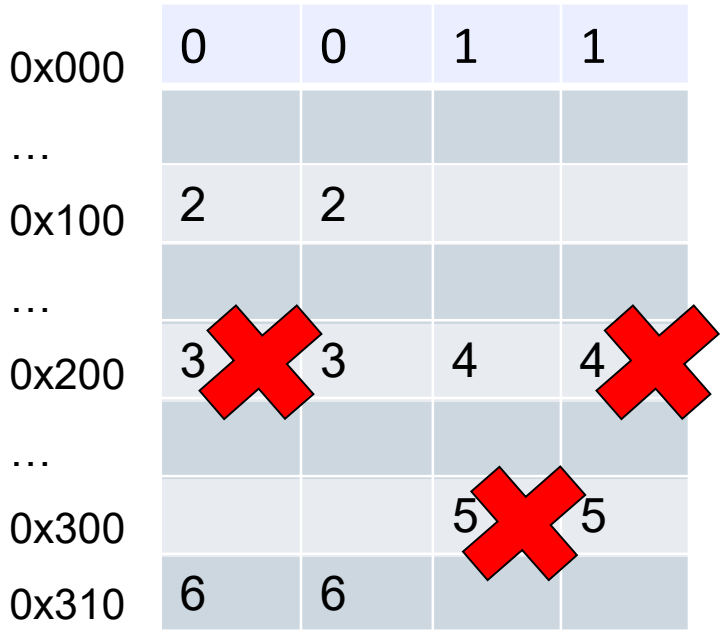
With Alignment Constraints

Cycle	Addr	Instr
?	0x000	OpA
?	0x004	OpB
?	0x008	OpC
?	0x00C	J 0x100
?		
?	0x100	OpD
?	0x104	J 0x204
?		
?	0x204	OpE
?	0x208	J 0x30C
?		
?	0x30C	OpF
?	0x310	OpG
?	0x314	OpH

Fetching across cache lines



With Alignment constraints



With Alignment Constraints

Cycle	Addr	Instr
0	0x000	OpA
0	0x004	OpB
1	0x008	OpC
1	0x00C	J 0x100
2	0x100	OpD
2	0x104	J 0x204
3	0x200	?
3	0x204	OpE
3	0x208	J 0x30C
3	0x20C	?
4	0x308	?
4	0x30C	OpF
5	0x310	OpG
5	0x314	OpH

OpA	F	D	A0	A1	W						
OpB	F	D	B0	B1	W						
OpC		F	D	B0	B1	W					
J 0x100		F	D	A0	A1	W					
OpD			F	D	B0	B1	W				
J 0x204			F	D	A0	A1	W				
?				F	-	-	-	-			
OpE				F	D	A0	A1	W			
J 0x30C					F	D	A0	A1	W		
?					F	-	-	-	-		
?						F	-	-	-	-	
OpF						F	D		A0	A1	W
OpG							F	D	A0	A1	W
OpH							F	D	B0	B1	W

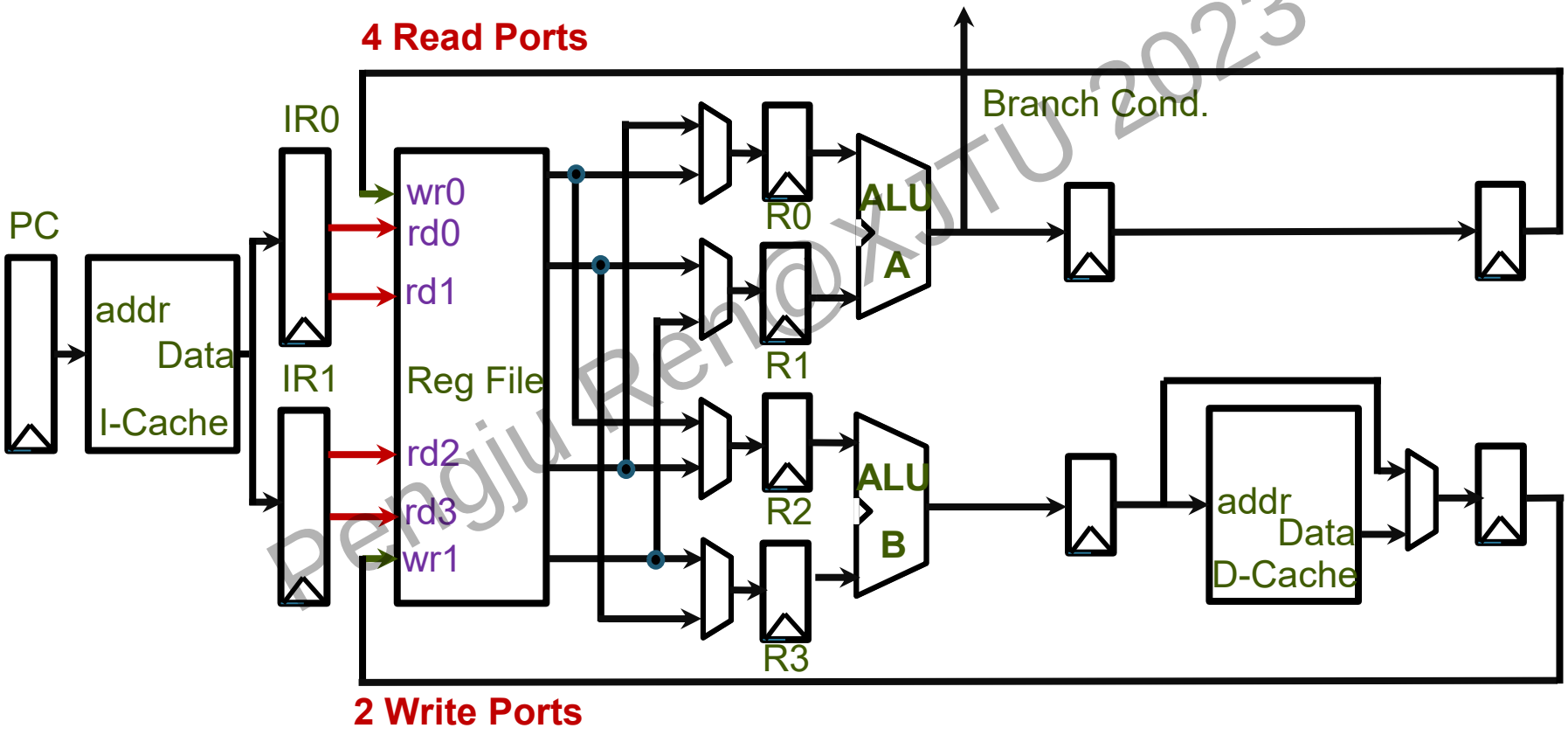
Precise Exceptions and Superscalars

Similar to tracking program order for data dependencies, we need to track order for exceptions

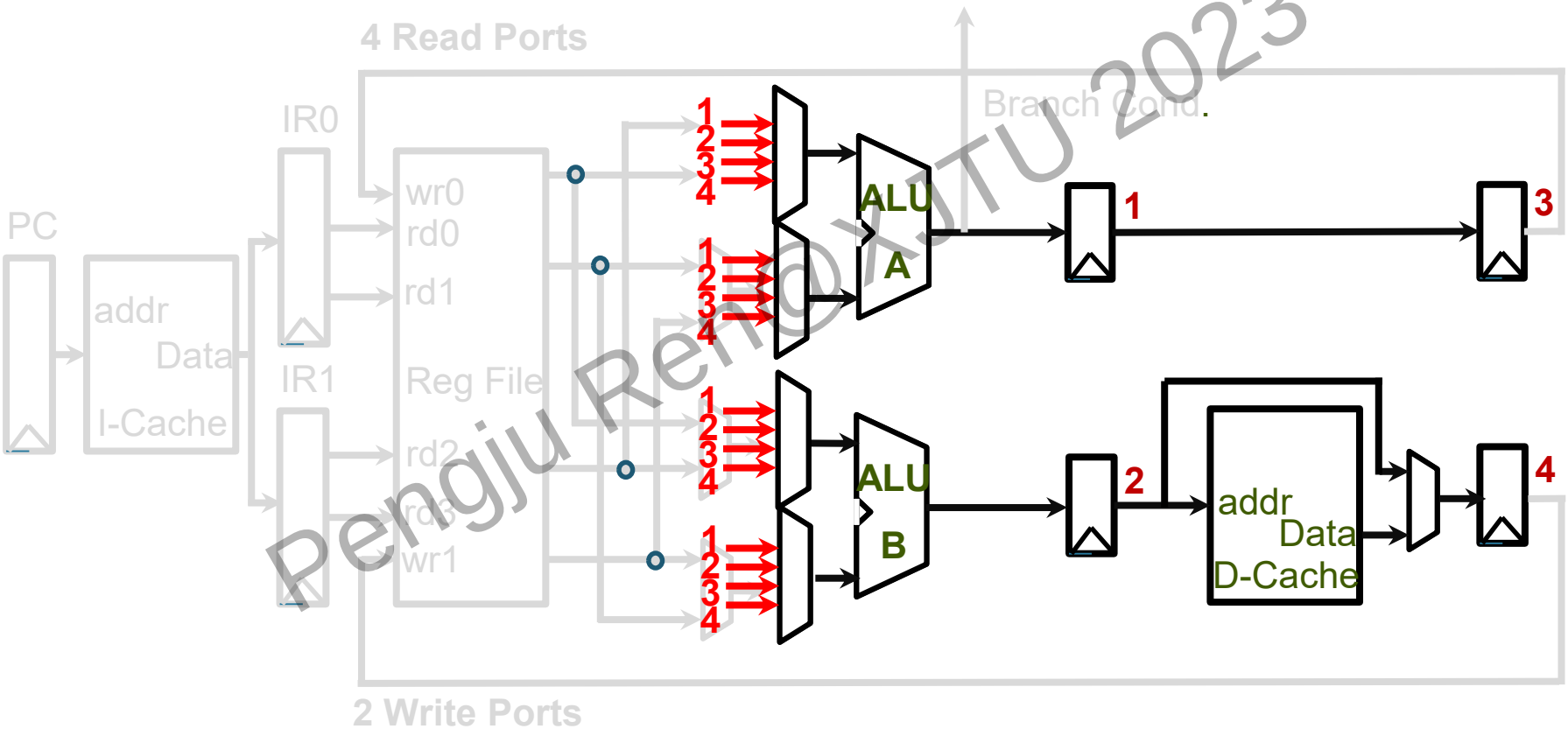
LW	F	D	B0	B1	W
SysCALL	F	D	A0	A1	W

LW is in B pipeline, but commits first in logical order!

Bypassing in SuperScalar Pipelines



Bypassing in SuperScalar Pipelines



Breaking Decode and Issue Stage

Bypass Network can become very complex

Can motivate breaking Decode and Issue Stage

D = Decode, Possibly resolve structural Hazards

I = Register file read, Bypassing, Issue/Steer Instructions to proper unit

OpA	F	D	I	A0	A1	W	
OpB	F	D	I	B0	B1	W	
OpC		F	D	I	A0	A1	W
OpD		F	D	I	B0	B1	W

SuperScalars Multiply Branch Cost

BEQ	F	D	I	A0	A1	W				
OpA	F	D	I	B0	-	-				
OpB		F	D	I	-	-	-			
OpC		F	D	I	-	-	-			
OpD			F	D	-	-	-	-		
OpE			F	D	-	-	-	-		
OpF				F						
OpG				F						
OpH					F	D	I	A0	A1	W
OpI					F	D	I	B0	B1	W

Agenda

SuperScalar Intro

Out-of-Order Processor (OOO)

Speculation and Branches

Register Renaming

Memory Disambiguation

Out-of-Order Execution

Example: Polynomial evaluation

$$value = \sum_{j=0}^{terms} coef[j]x^j$$

Example: Polynomial evaluation

- Compiling on RISC-V

$$value = \sum_{j=0}^{terms} coef[j]x^j$$

```
x6: value
x1: &coef[terms]
x2: x
x3: &coef[0]
x4: power
x5: coef[j]
```

```
int poly(int *coef,
         int terms, int x) {
  int power = 1;
  int value = 0;
  for (int j = 0; j < terms; j++) {
    value += coef[j] * power;
    power *= x;
  }
  return value;
}
```

```
poly:
  ble     x1, x0, .L4
  push    {x4, x5}
  addi    x4, x0, #1
  addi    x6, x0, #0
.L3:
  ld      x5, [x3]
  add     x3, #4
  mul     x7, x4, x5
  add     x6, x6, x7
  mul     x4, x2, x4
  bne     x1  x3 .L3
  pop     {x4, x5}
  bx      lr
.L4:
  addi    x6, x0, #0
  bx      lr
```

Compilers Manage Memory and Registers

Compilers for languages like C/C++:

- Check that program is legal
- Translate into assembly code
- Optimizes the generated code

Compiler performs “register allocation” to decide when to load/store and when to reuse

Example: Polynomial evaluation

```
x6: value
x1: &coef[terms]
x2: x
x3: &coef[0]
x4: power
x5: coef[j]
```

- Compiling on RISC-V

```
int poly(int *coef,
         int terms, int x) {
  int power = 1;
  int value = 0;
  for (int j = 0; j < terms; j++) {
    value += coef[j] * power;
    power *= x;
  }
  return value;
}
```

```
poly:                                  Preamble
  ble     x1, x0, .L4
  push    {x4, x5}
  addi    x4, x0, #1
  addi    x6, x0, #0
.L3:                                   Iteration
  ld      x5, [x3]
  add     x3, #4
  mul     x7, x4, x5
  add     x6, x6, x7
  mul     x4, x2, x4
  bne     x1  x3 .L3
  pop     {x4, x5}                     Finish
  bx      lr
.L4:
  addi    x6, x0, #0
  bx      lr
```

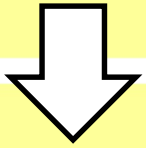
Example: Polynomial evaluation

```
x6: value
x1: &coef[terms]
x2: x
x3: &coef[0]
x4: power
x5: coef[j]
```

- Compiling on RISC-V

Iteration

```
for (int j = 0; j < terms; j++) {
  value += coef[j] * power;
  power *= x;
}
```

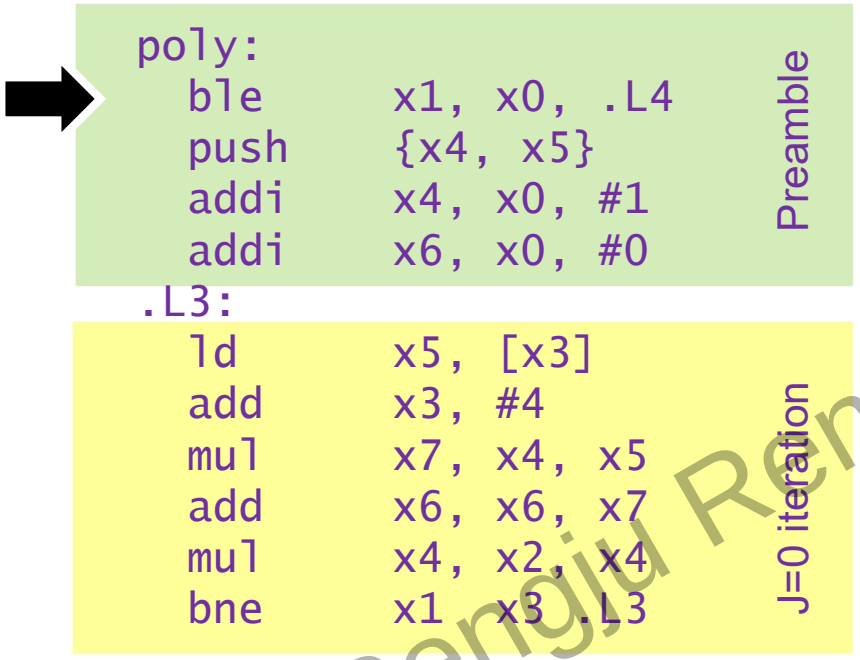


```
.L3:
  ld      x5, [x3]      // r5 <- coef[j];
  add     x3, #4        // j++   (two operations)
  mul     x7, x4, x5    // coef[j]*power
  add     x6, x6, x7    // value += coef[j]*power
  mul     x4, x2, r4     // power *= x
  bne     x1, x3, .L3   // compare: j < terms? repeat?
```

```
ld      x5, [x3]
 add     x3, #4
mul      x7, x4, x5
add      x6, x6, x7
mul      x4, x2, x4
bne      x1  x3 .L3
pop      {x4, x5}
bx       lr
```

Example: Polynomial evaluation

- Executing `poly(A, 3, x)`



Example: Polynomial evaluation

- Executing `poly(A, 3, x)`

```
poly:
  ble     x1, x0, .L4
  push    {x4, x5}
  addi    x4, x0, #1
  addi    x6, x0, #0
.L3:
  ld      x5, [x3]
  add     x3, #4
  mul     x7, x4, x5
  add     x6, x6, x7
  mul     x4, x2, x4
  bne     x1  x3 .L3
```

Preamble

I=0 iteration

```
  ld      x5, [x3]
  add     x3, #4
  mul     x7, x4, x5
  add     x6, x6, x7
  mul     x4, x2, x4
  bne     x1  x3 .L3
```

J=1 iteration

```
  ld      x5, [x3]
  add     x3, #4
  mul     x7, x4, x5
  add     x6, x6, x7
  mul     x4, x2, x4
  bne     x1  x3 .L3
```

J=2 iteration

```
  pop     {x4, x5}
  bx      lr
.L4:
  addi    x6, x0, #0
  bx      lr
```

Finish

Increasing parallelism via dataflow

- Parallelism limited by many *false dependencies*, particularly *sequential program order*

- **Dataflow** tracks how instructions actually depend on each other
  - *True dependence*: read-after-write

Dataflow increases parallelism by eliminating unnecessary dependences

Example: Polynomial evaluation

```
r0: value
r1: &coef[terms]
r2: x
r3: &coef[j]
r4: power
r5: coef[j]
```

- **Compiling on ARM**

```
int poly(int *coef,
         int terms, int x) {
  int power = 1;
  int value = 0;
  for (int j = 0; j < terms; j++) {
    value += coef[j] * power;
    power *= x;
  }
  return value;
}
```

```
poly:                                   Preamble
  cmp     r1, #0
  ble     .L4
  push    {r4, r5}
  mov     r3, r0
  add     r1, r0, r1, lsl #2
  movs    r4, #1
  movs    r0, #0
.L3:                                    Iteration
  ldr     r5, [r3], #4
  cmp     r1, r3
  mla     r0, r4, r5, r0
  mul     r4, r2, r4
  bne     .L3
  pop     {r4, r5}                      Finish
  bx      lr
.L4:
  movs    r0, #0
  bx      lr
```

Example: Dataflow in polynomial evaluation@ARM

Compiling on ARM

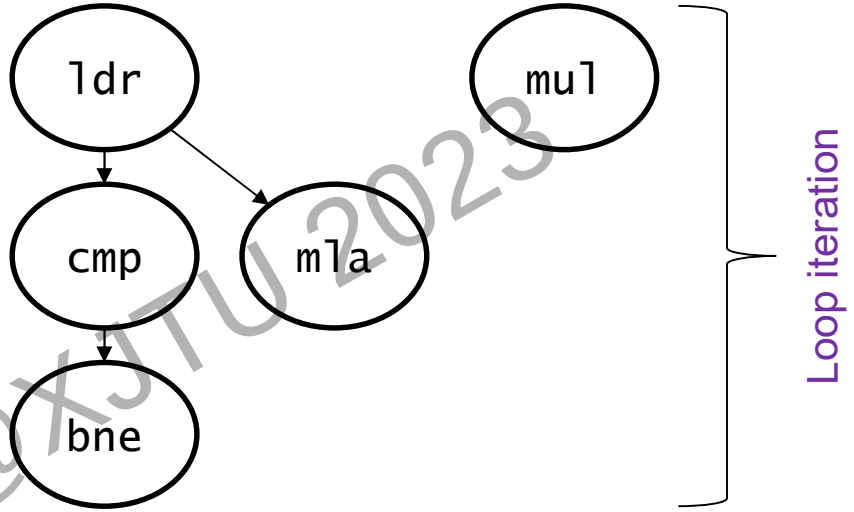
```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

iteration

```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

iteration

...

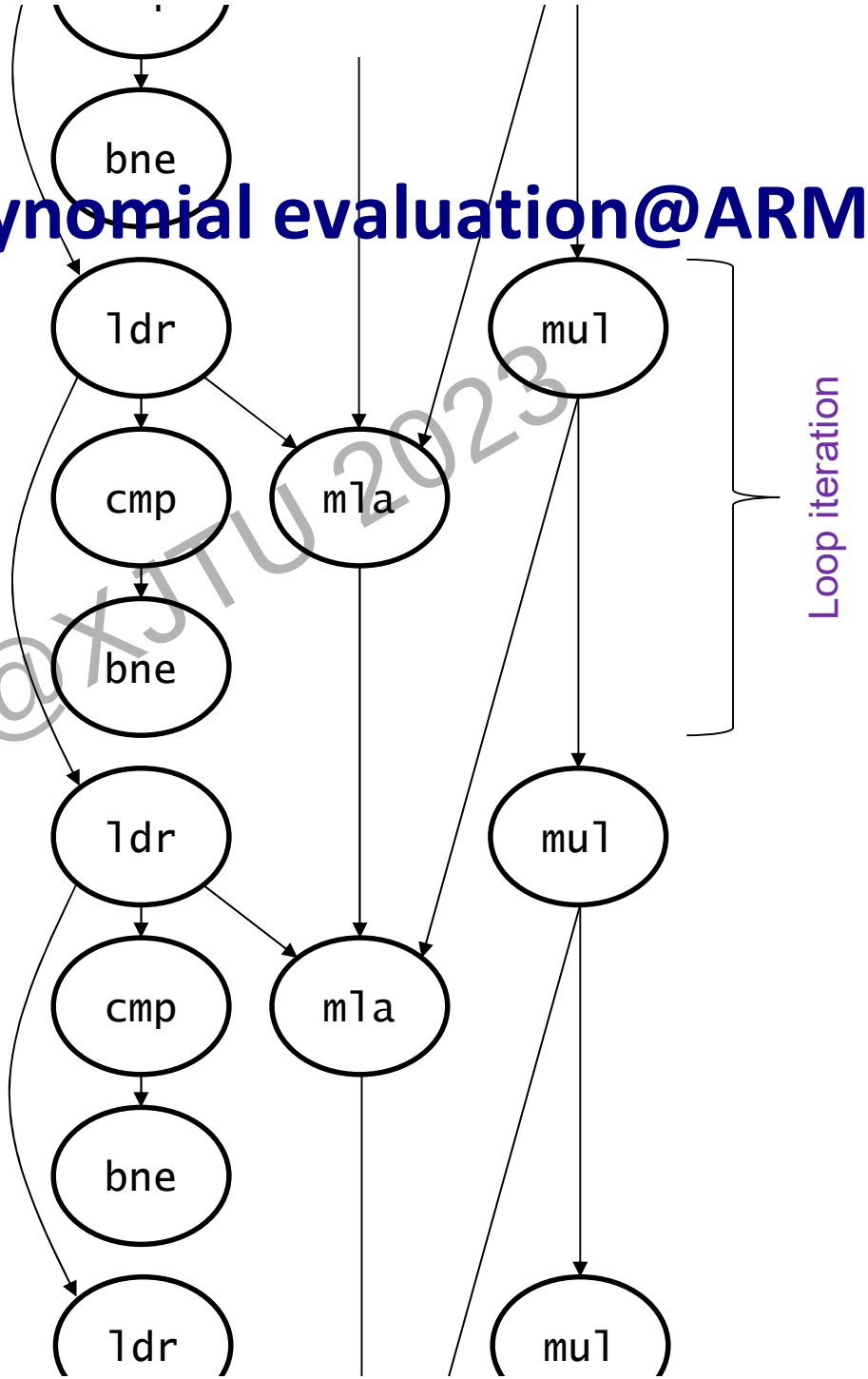


Example: Dataflow in polynomial evaluation@ARM

Compiling on ARM

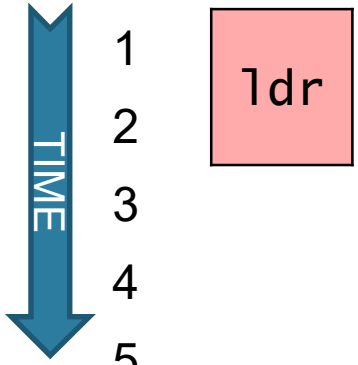
```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3

ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

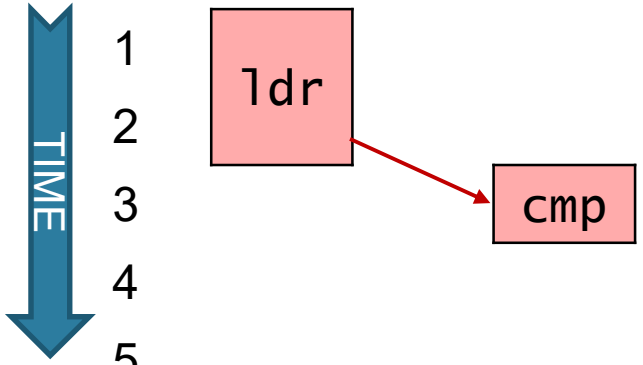


Example: Dataflow polynomial execution@ARM

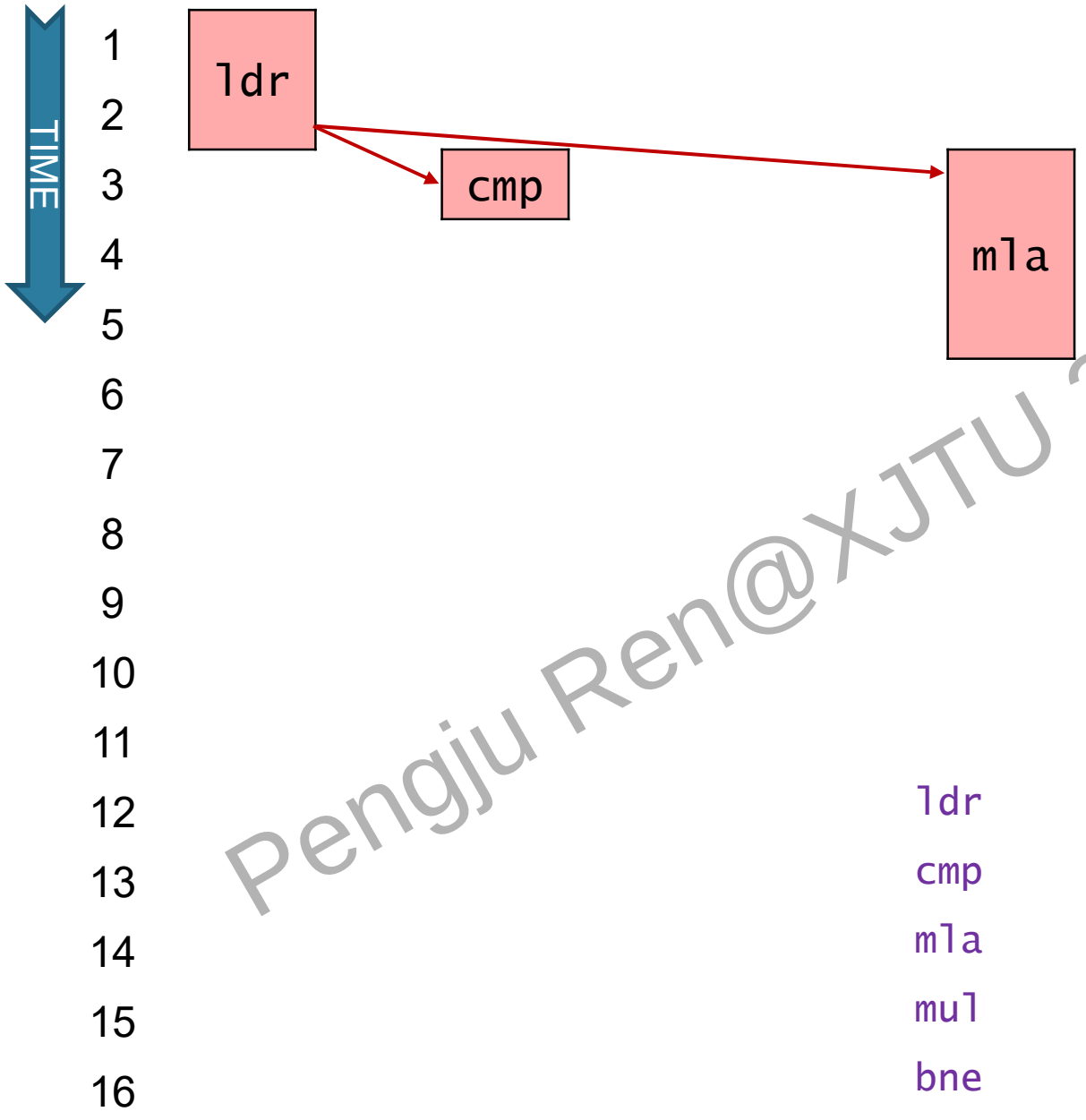
- Execution only, with perfect scheduling & unlimited execution units
  - `ldr`, `mul` execute in 2 cycles
  - `cmp`, `bne` execute in 1 cycle
  - `mla` executes in 3 cycles
- Q: Does dataflow speedup execution? By how much?
- Q: What is the performance bottleneck?



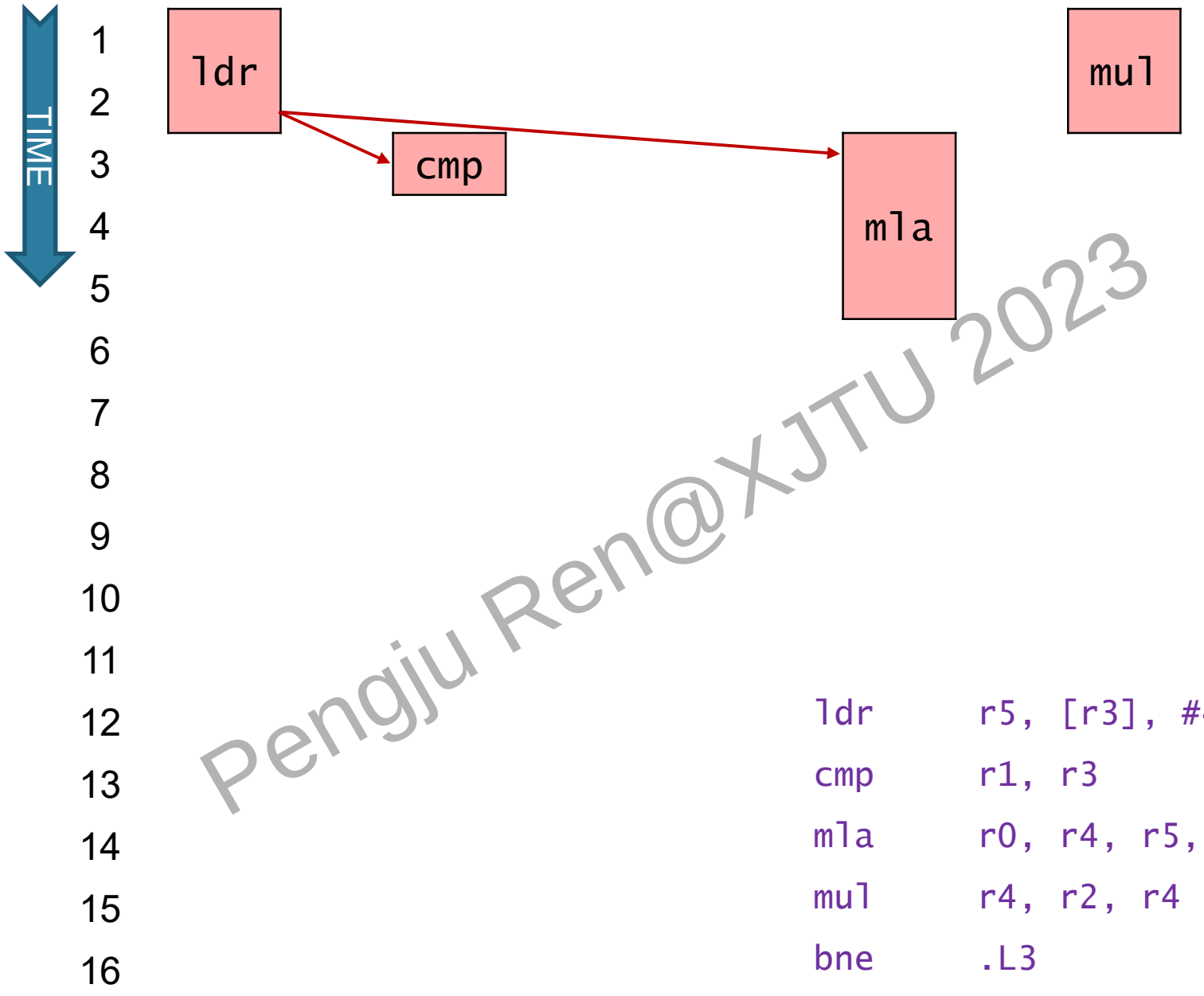

```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```



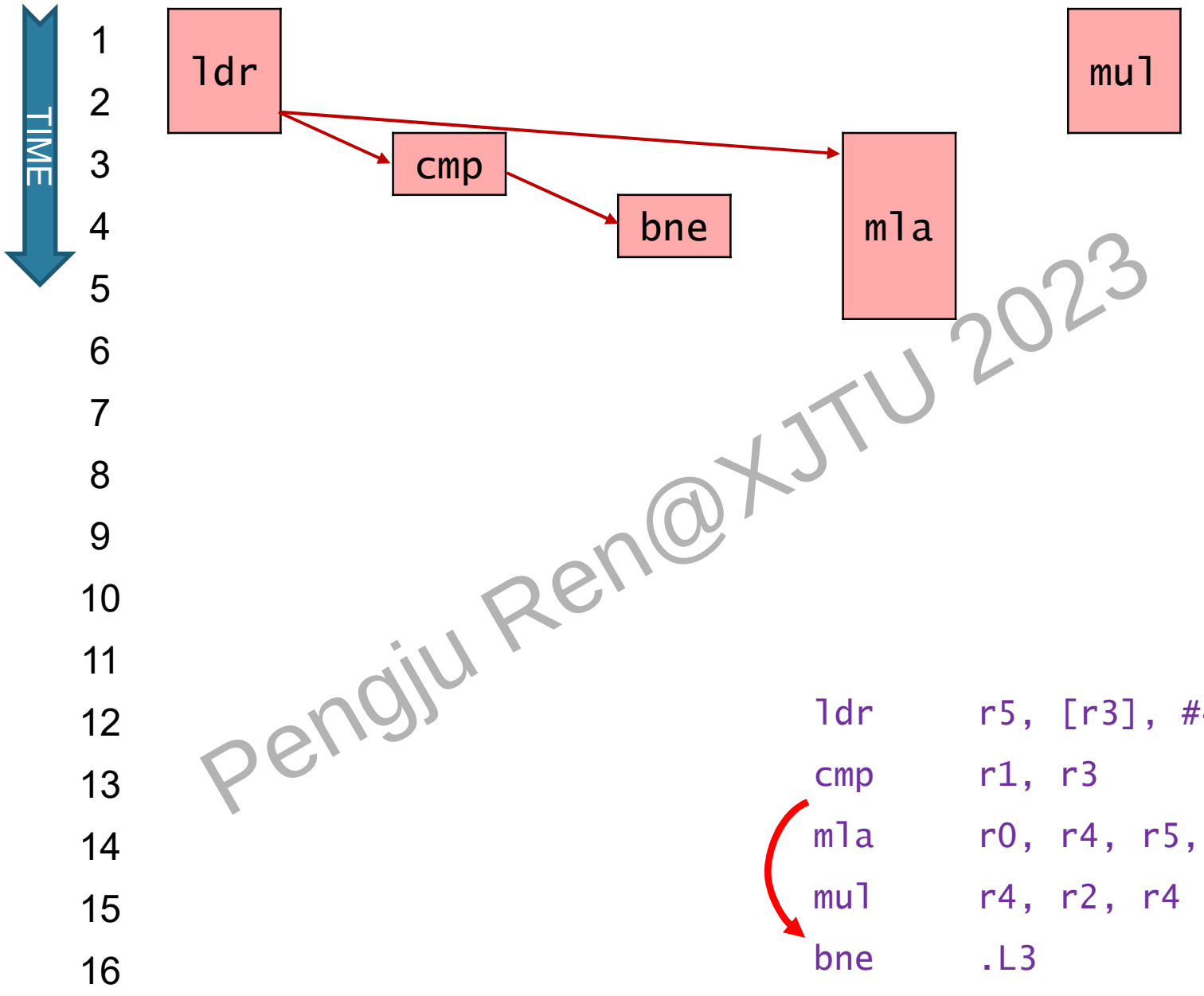

```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```



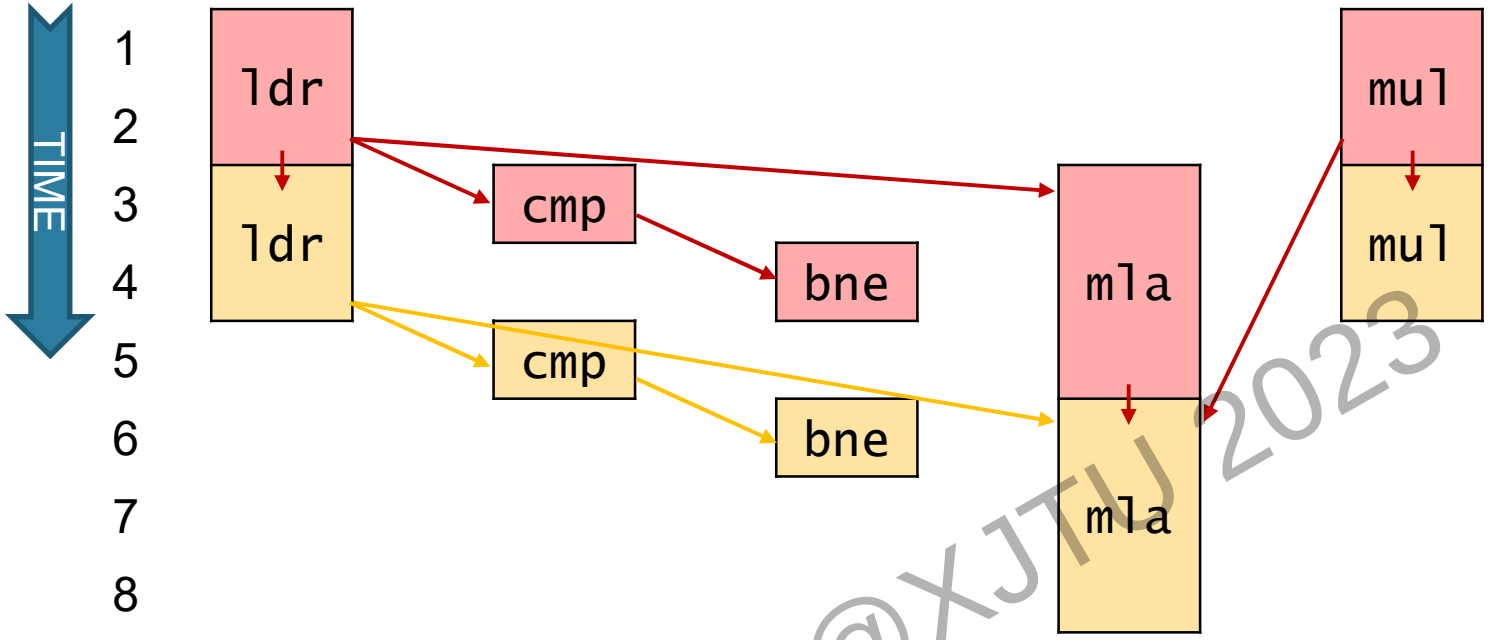

```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```



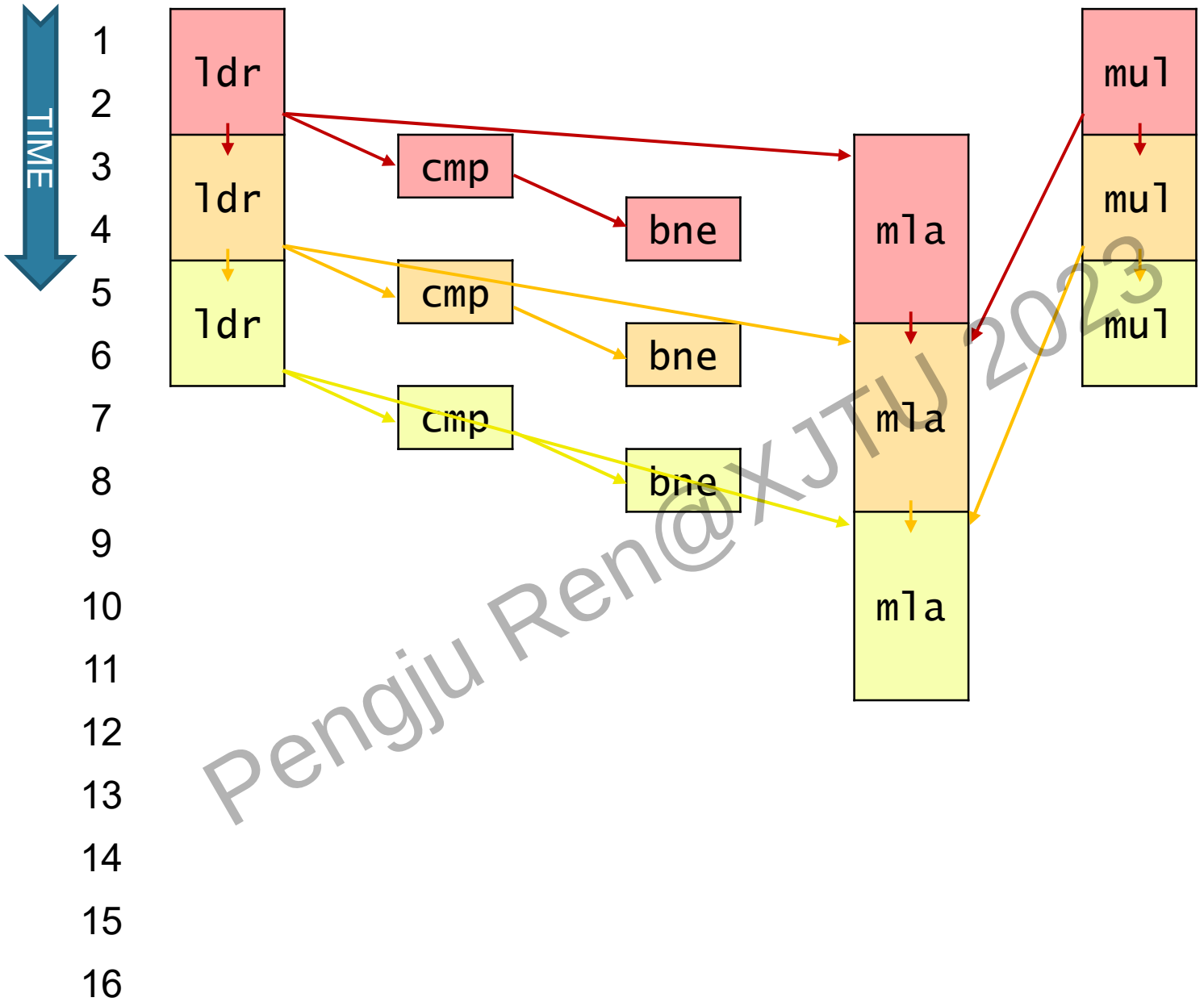

```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

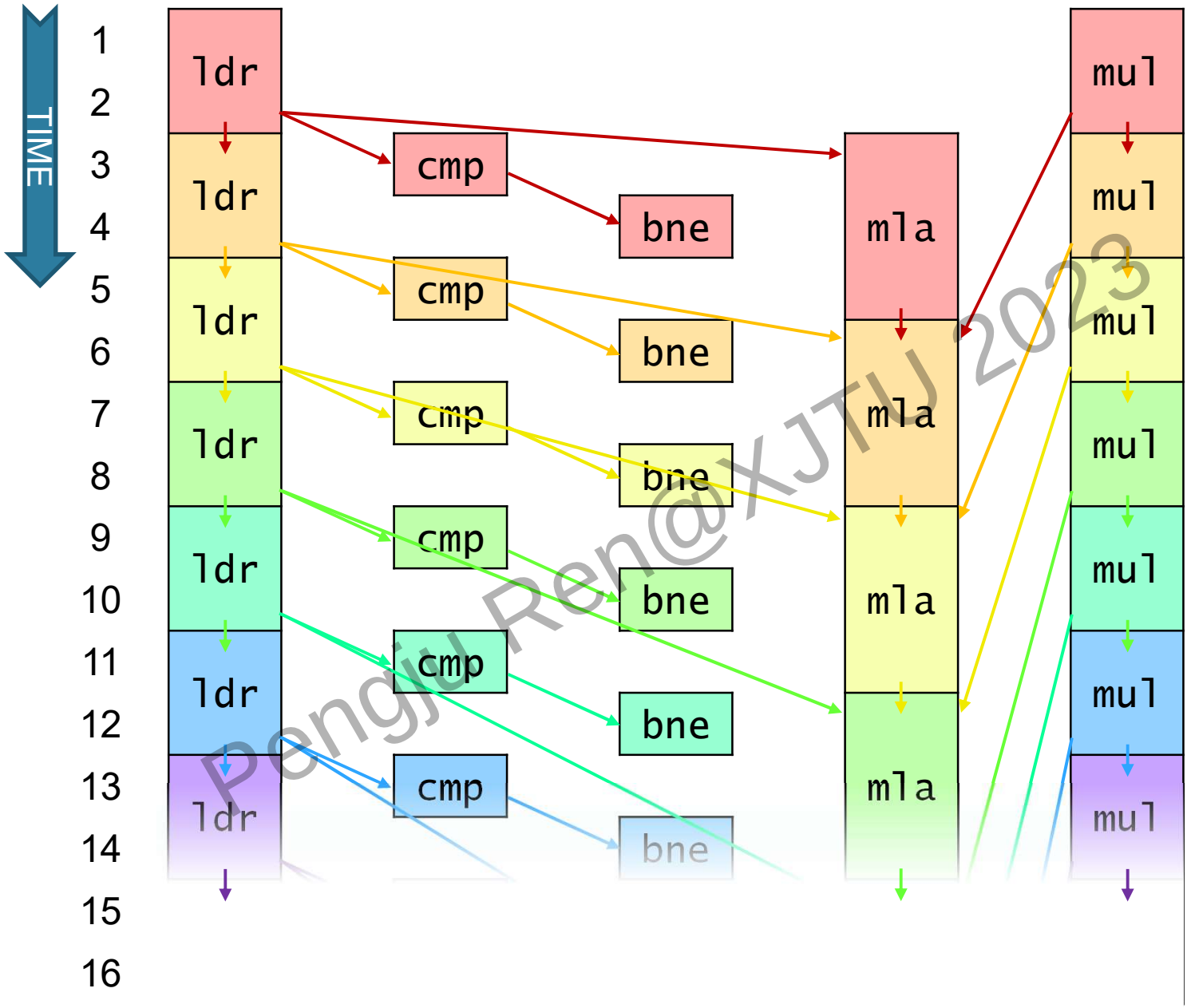



```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```



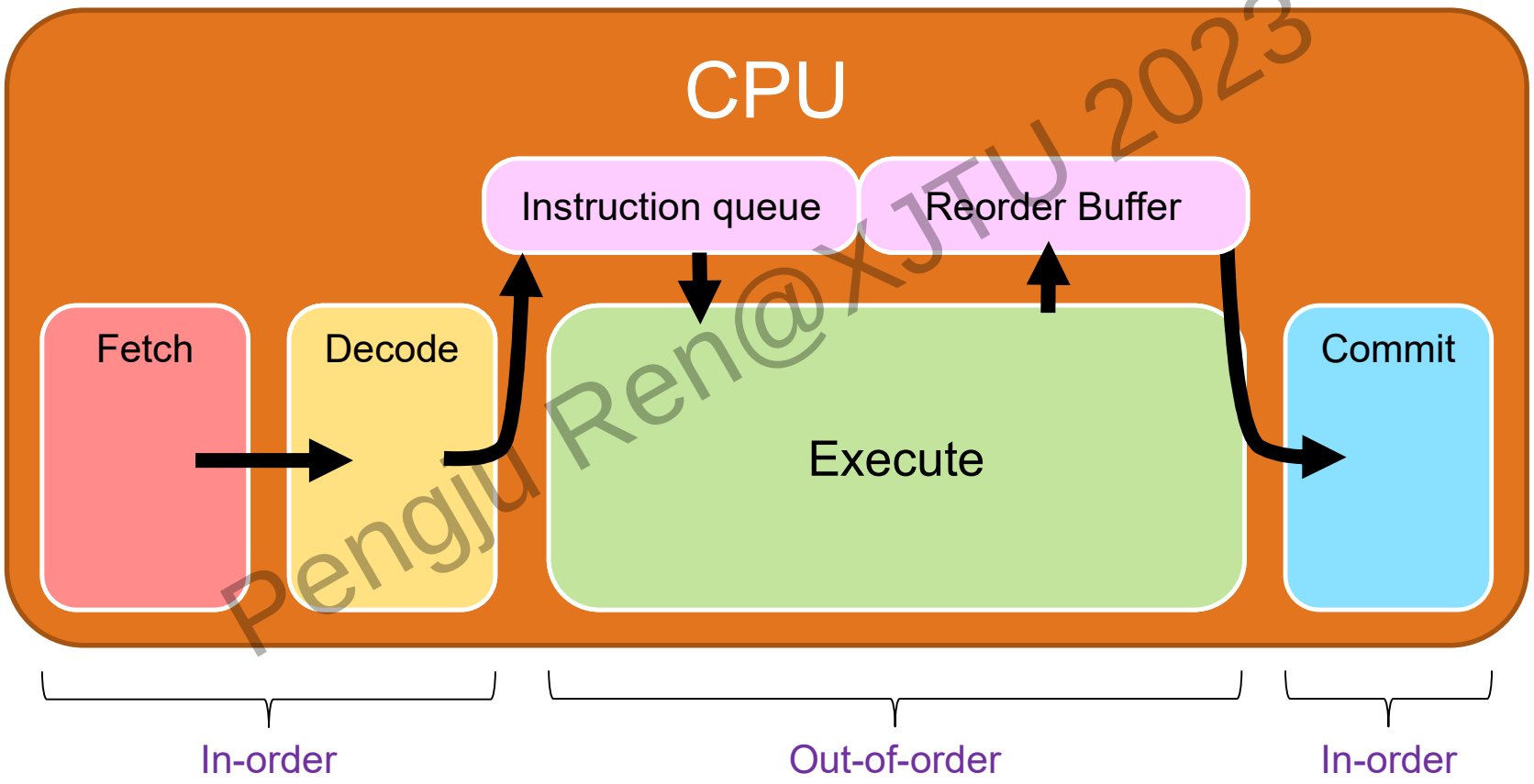

TIME
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
ldr
ldr
ldr
ldr
ldr
ldr
ldr
cmp
cmp
cmp
cmp
cmp
cmp
cmp
bne
bne
bne
bne
bne
bne
bne
mla
mla
mla
mla
mul
mul
mul
mul
mul
mul
mul
Pengju Ren@XJTU 2023

Out-of-order (OoO) execution uses dataflow to increase parallelism

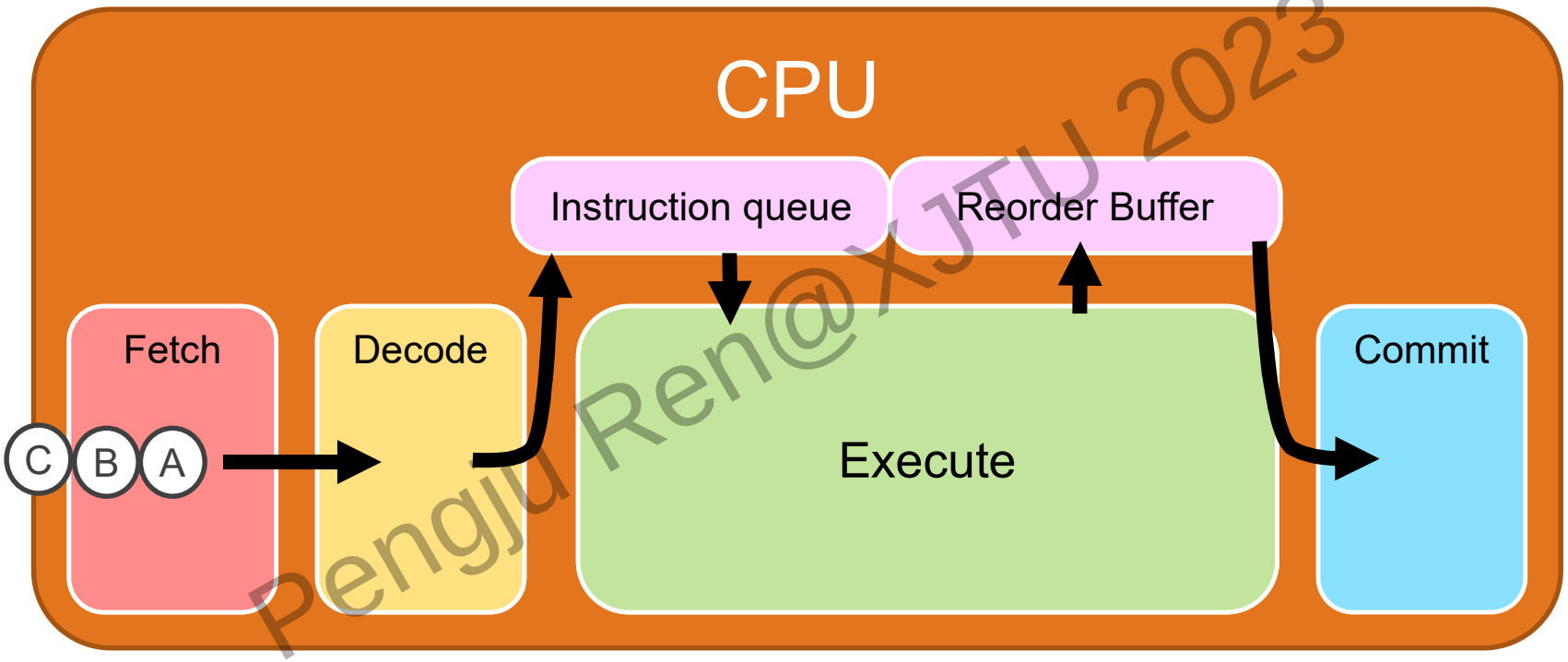
- Idea: Execute programs in dataflow order, but give the *illusion* of sequential execution

- This is a "restricted dataflow" model
  - *Restricted* to instructions near those currently committing
  - (Pure dataflow processors also exist that expose dataflow to software)

High-level OoO microarchitecture

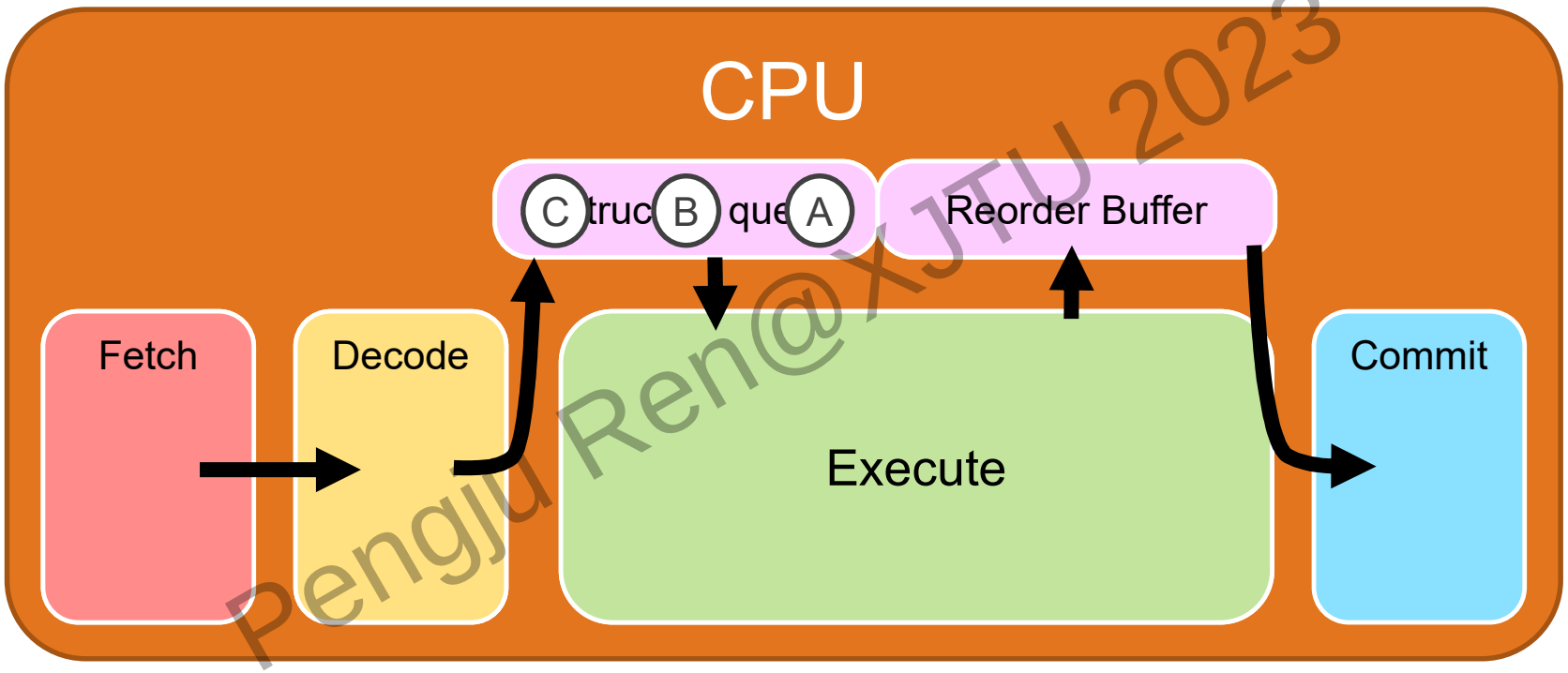


OoO is hidden behind in-order frontend & commit



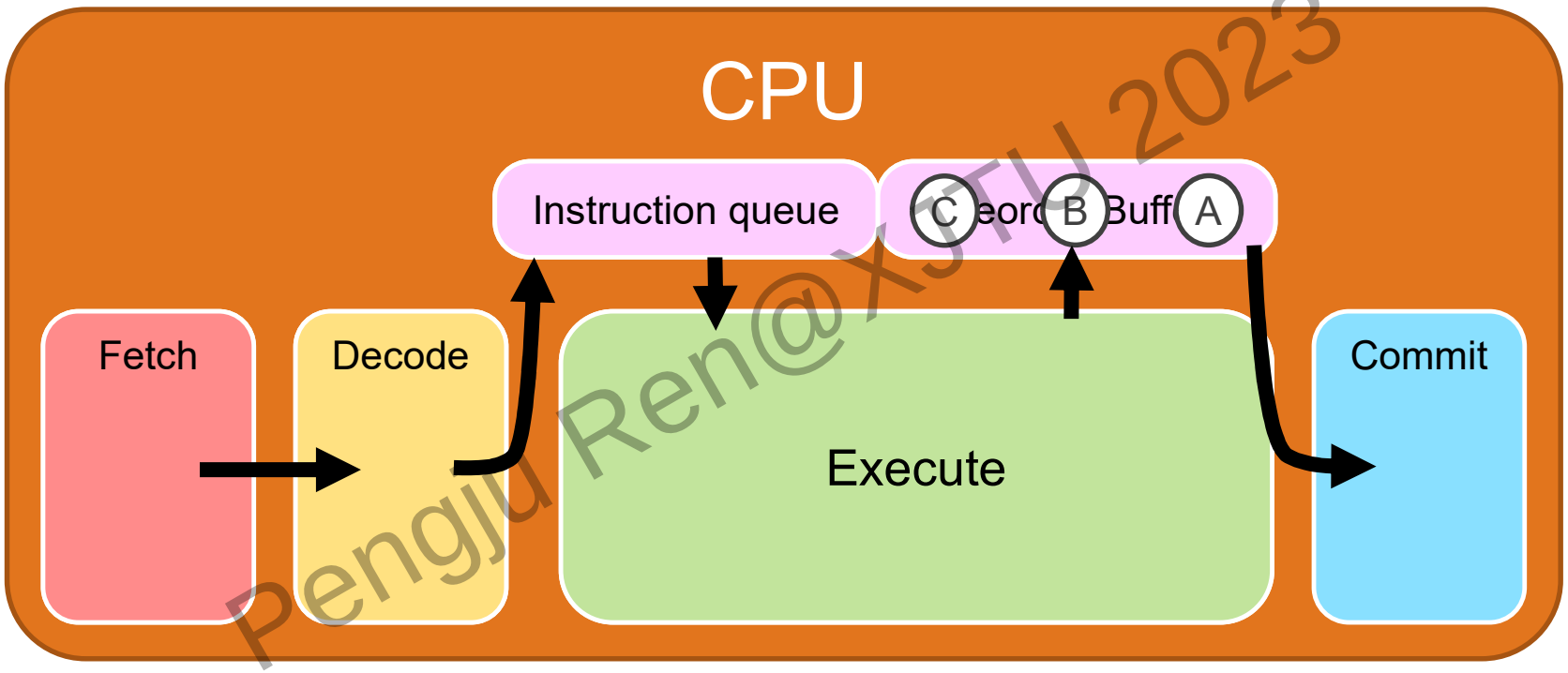
- Instructions only enter instruction queue(IQ) and leave reorder buffer(ROB) in program order; all bets are off in between!

OoO is hidden behind in-order frontend & commit



- Instructions only enter instruction queue(IQ) and leave reorder buffer(ROB) in program order; all bets are off in between!

OoO is hidden behind in-order frontend & commit



- Instructions only enter instruction queue(IQ) and leave reorder buffer(ROB) in program order; all bets are off in between!

Example: OoO polynomial evaluation

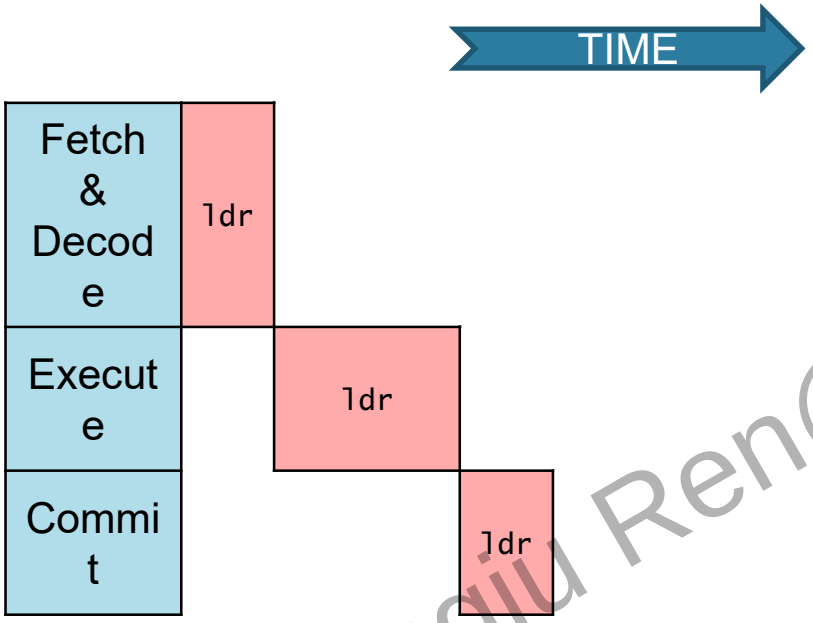
- Q: Does OoO speedup execution? By how much?
- Q: What is the performance bottleneck?
- Assume perfect forwarding & branch prediction

Example: OoO polynomial evaluation pipeline diagram @ARM

`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles

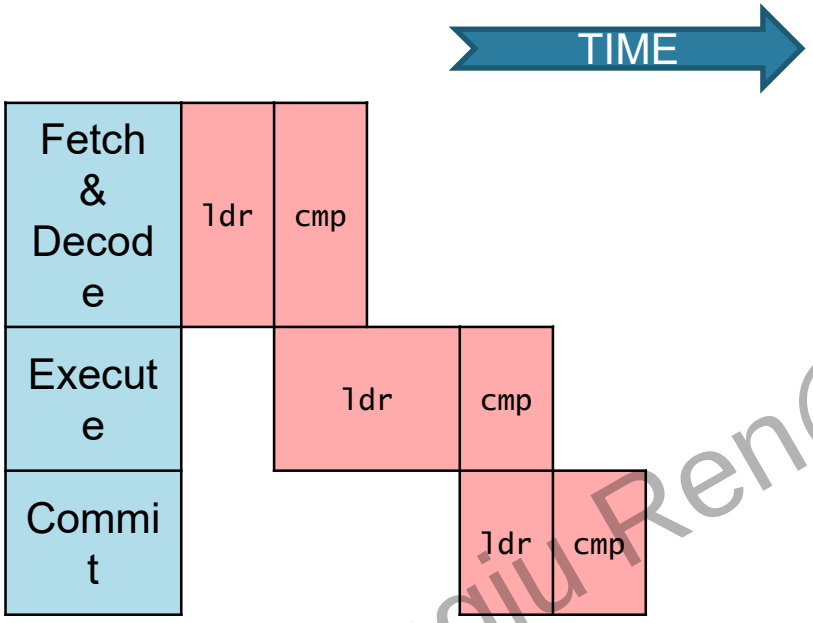


Example: OoO polynomial evaluation pipeline diagram @ARM

`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles

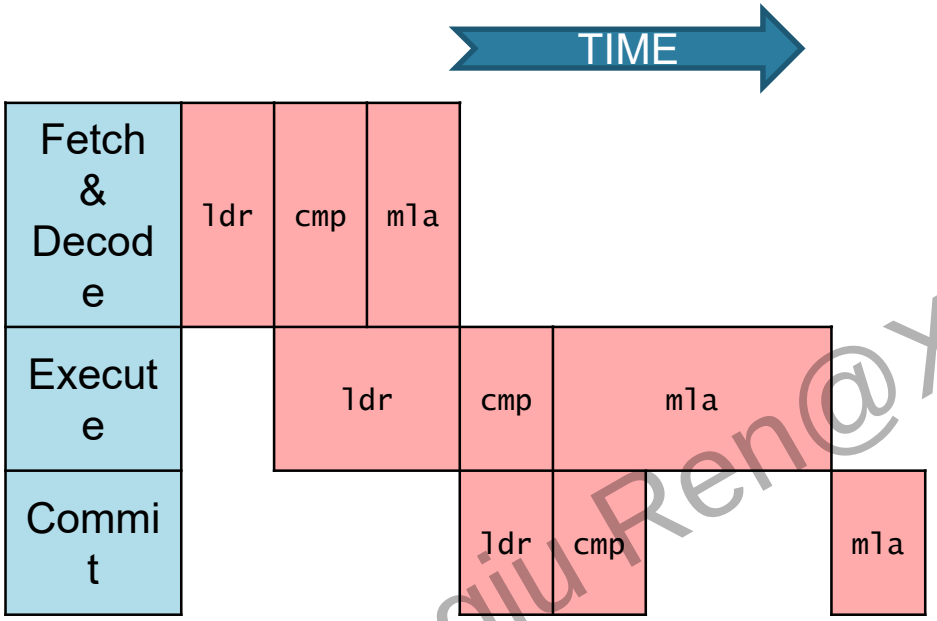


Example: OoO polynomial evaluation pipeline diagram @ARM

`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles

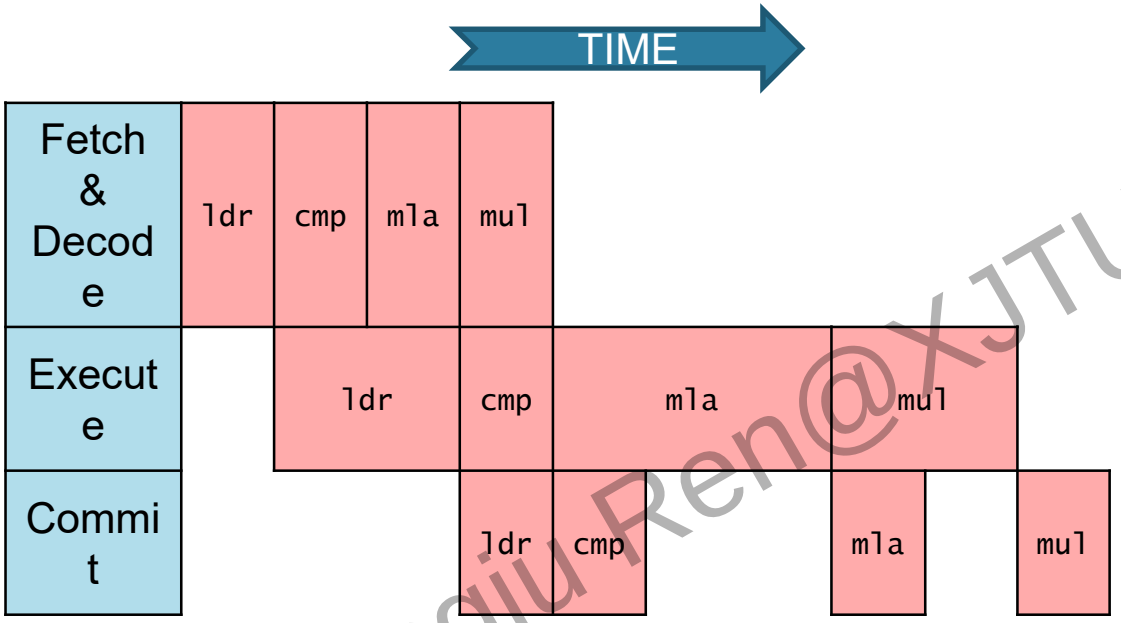


Example: OoO polynomial evaluation pipeline diagram @ARM

`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles

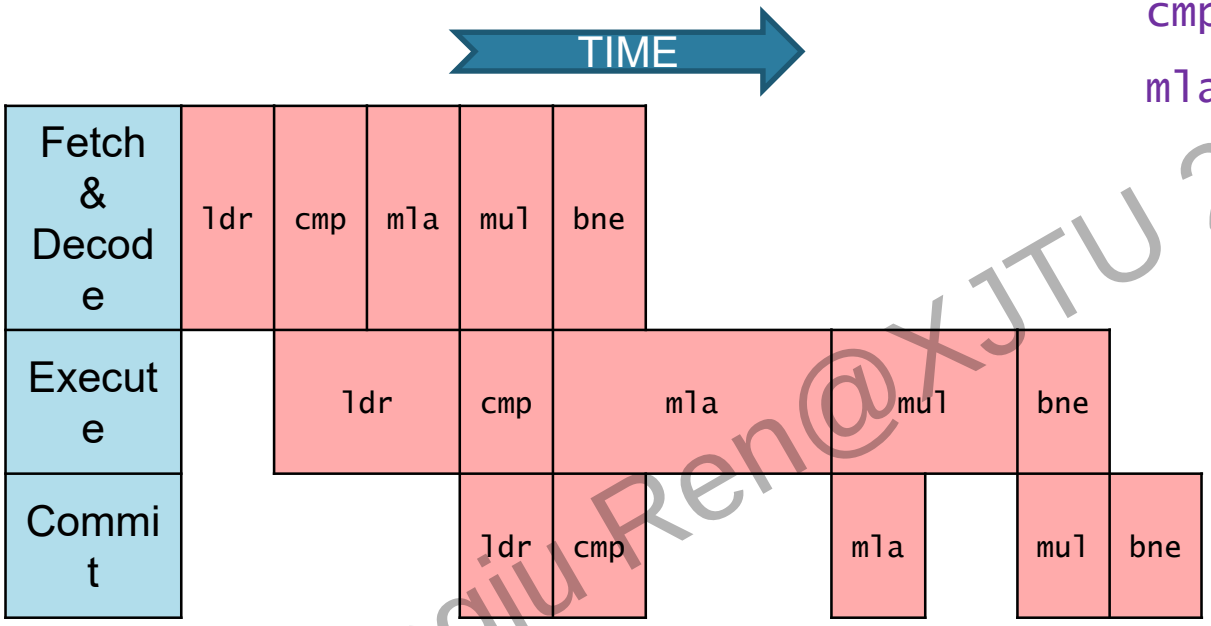


Example: OoO polynomial evaluation pipeline diagram @ARM

`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles

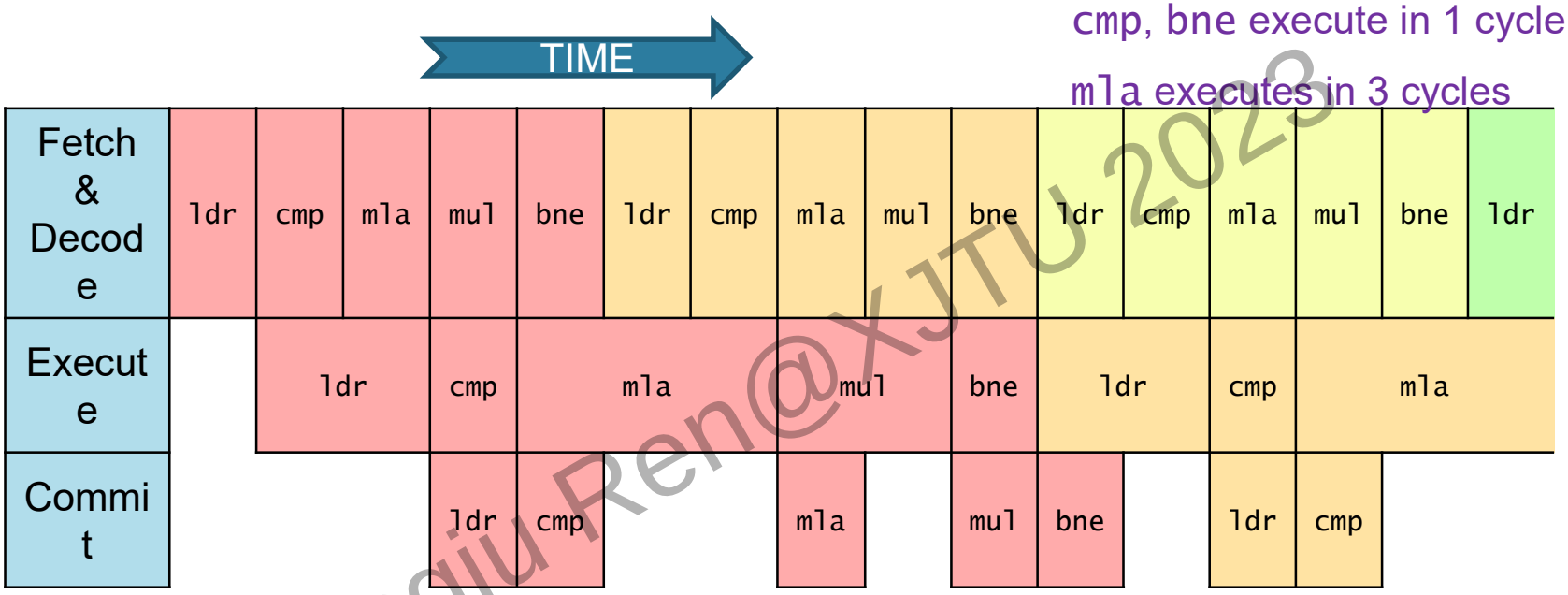


Example: OoO polynomial evaluation pipeline diagram @ARM

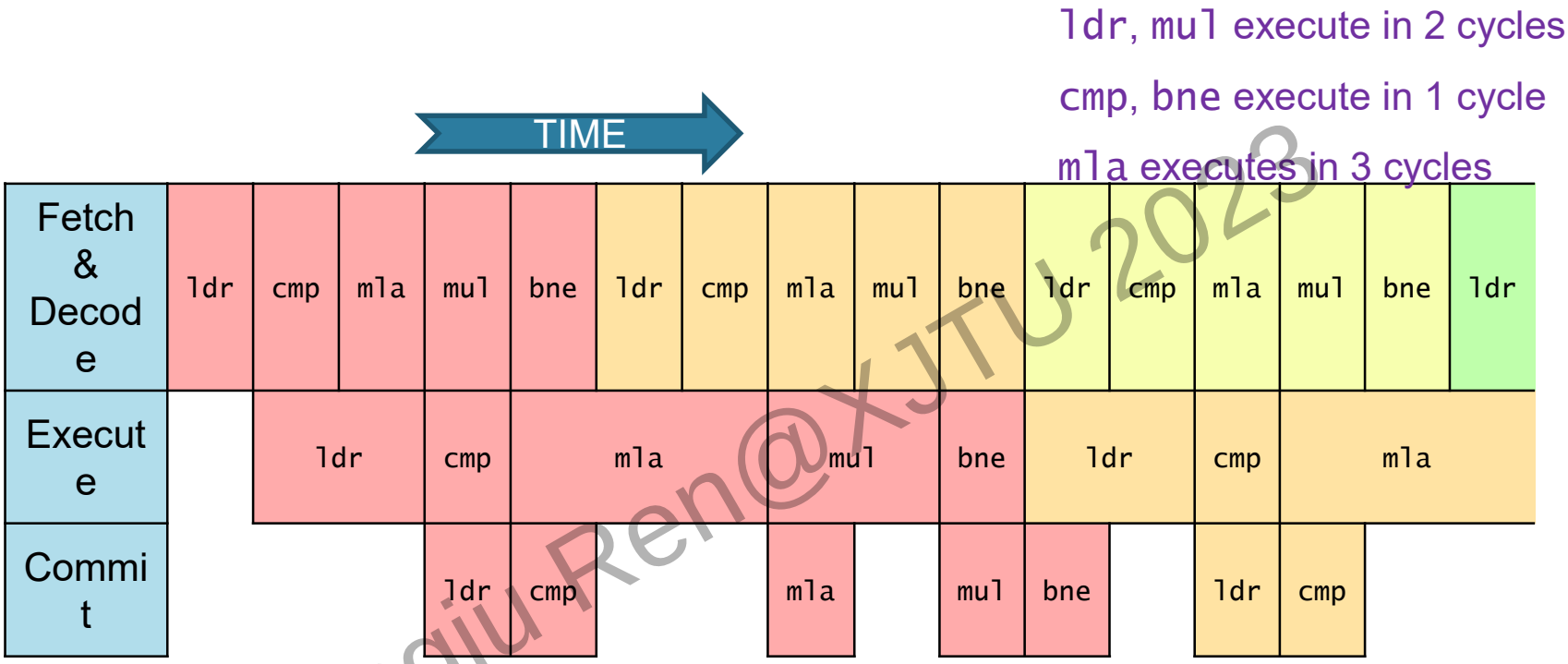
`ldr`, `mul` execute in 2 cycles

`cmp`, `bne` execute in 1 cycle

`mla` executes in 3 cycles



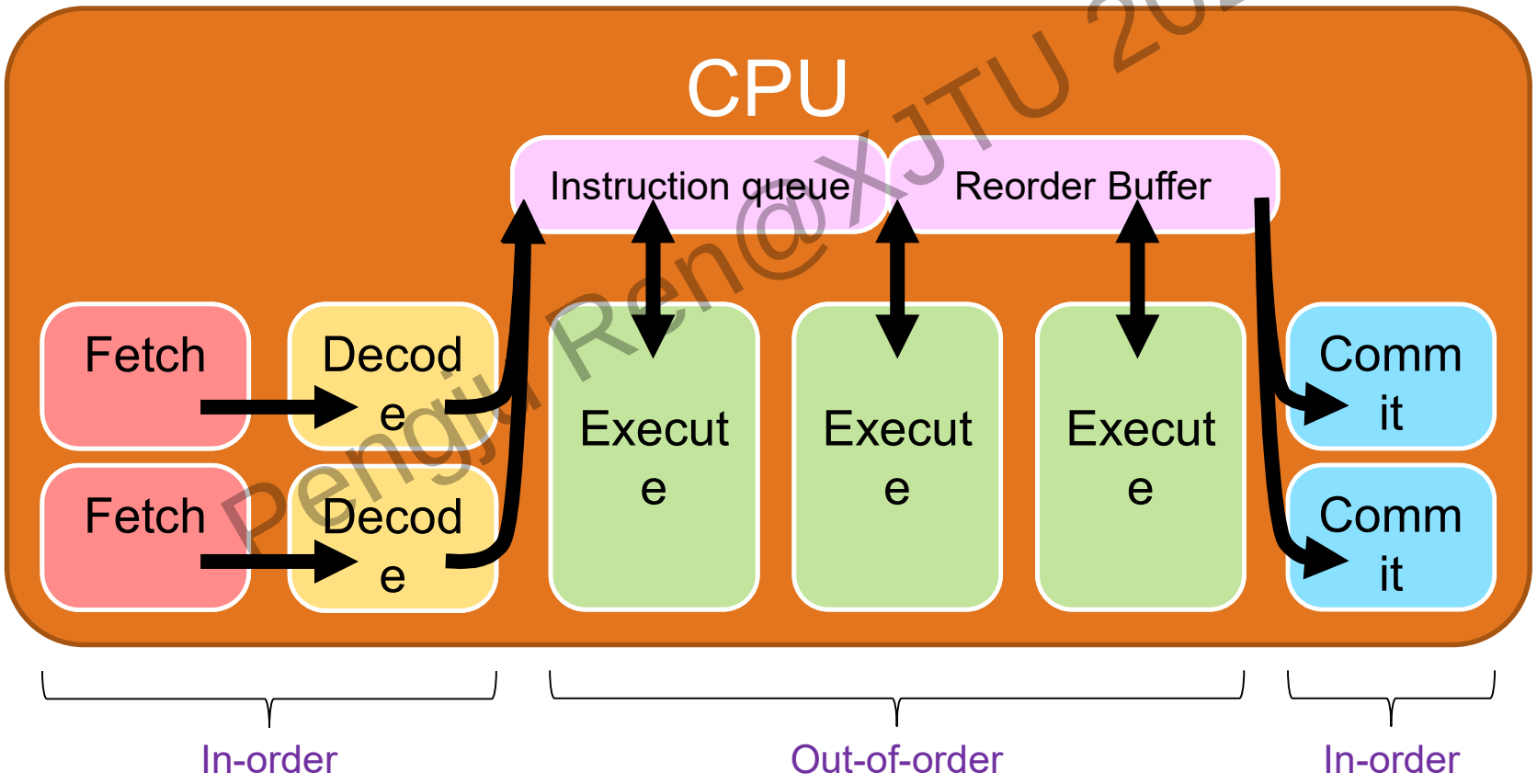
Example: OoO polynomial evaluation pipeline diagram @ARM



- Wait a minute… this isn't OoO… or even faster than a simple pipeline!
- Q: What went wrong?
- A: We're **throughput-limited:** can only exec 1 instr

High-level Superscalar OoO microarchitecture

- Must increase *pipeline width* to increase ILP > 1
  （2-way 3-issue）

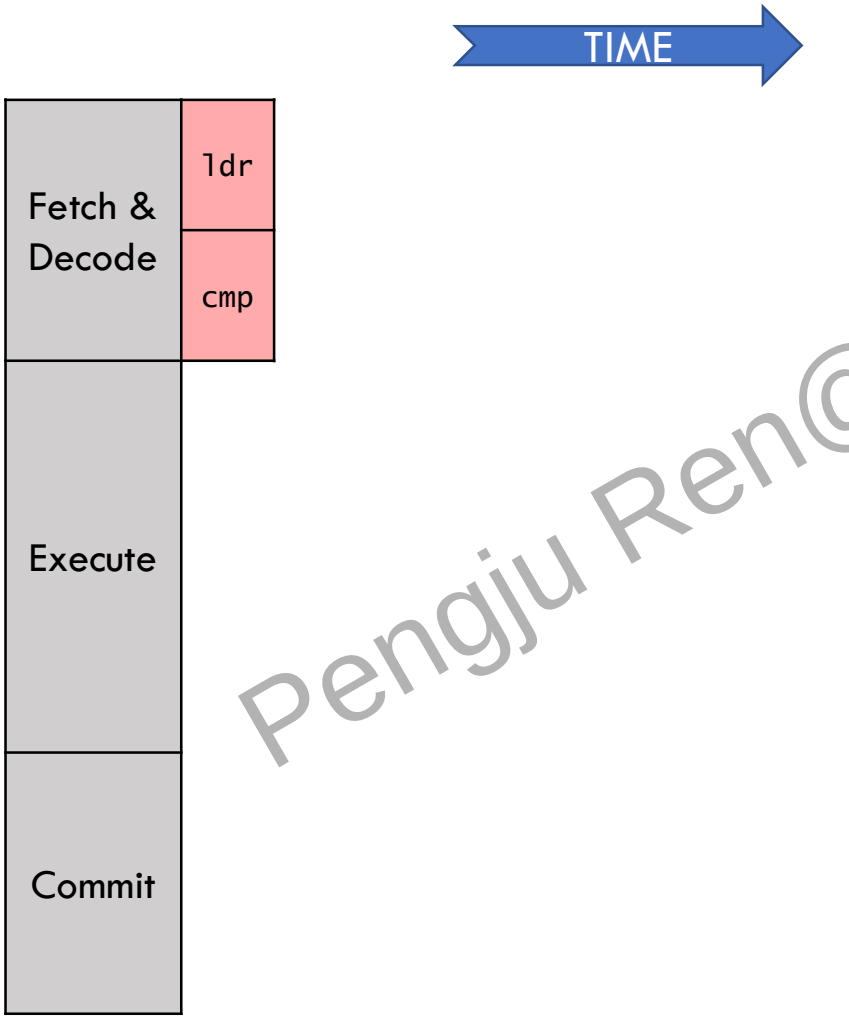


Focus on Execution, not Fetch & Commit

- Goal of OoO design is to only be limited by dataflow execution
- Fetch and commit are *over-provisioned* so that they (usually) do not limit performance
  ➔ Programmers can (usually) ignore fetch/commit

- **Big Caveat:** Programs with *inherently unpredictable* control flow will often be limited by fetch stalls (branch misprediction)
  - E.g., branching based on random data

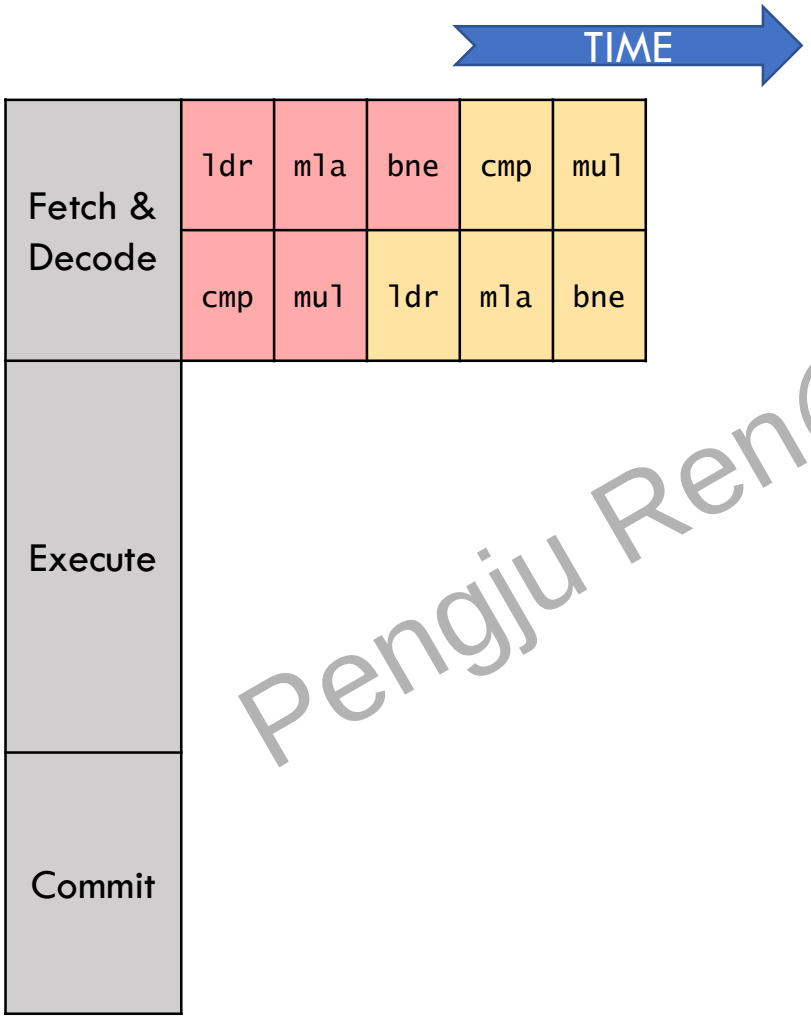
Example: Superscalar OoO polynomial evaluation




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

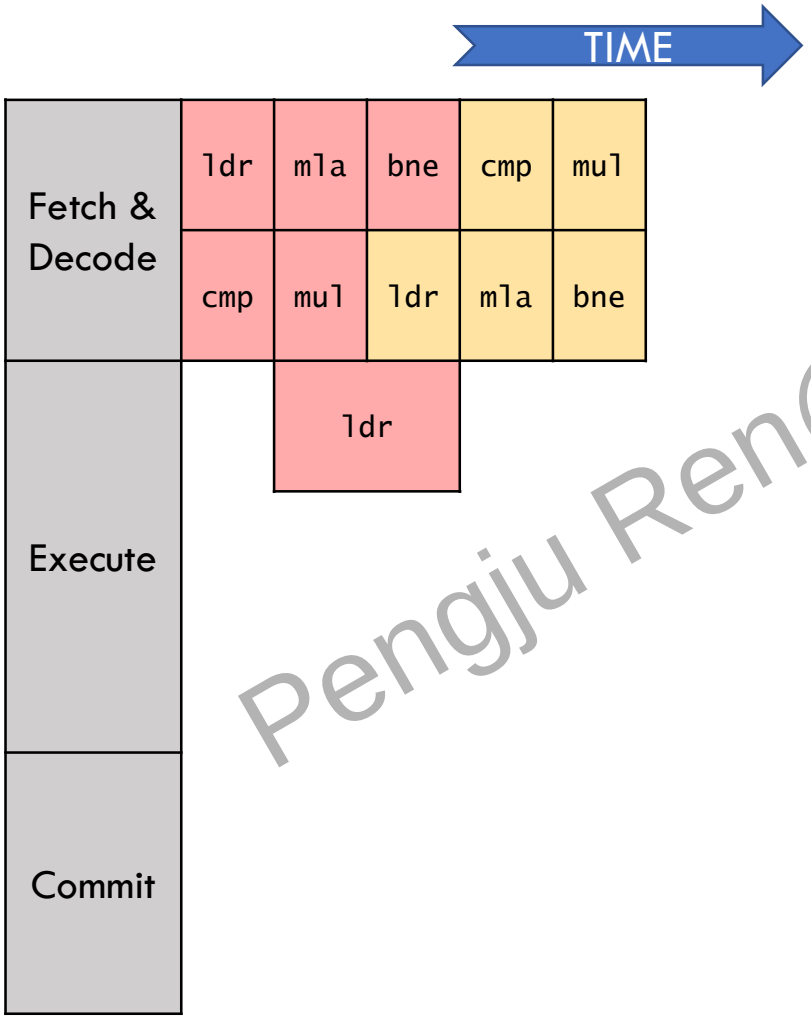
Example: Superscalar OoO polynomial evaluation




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

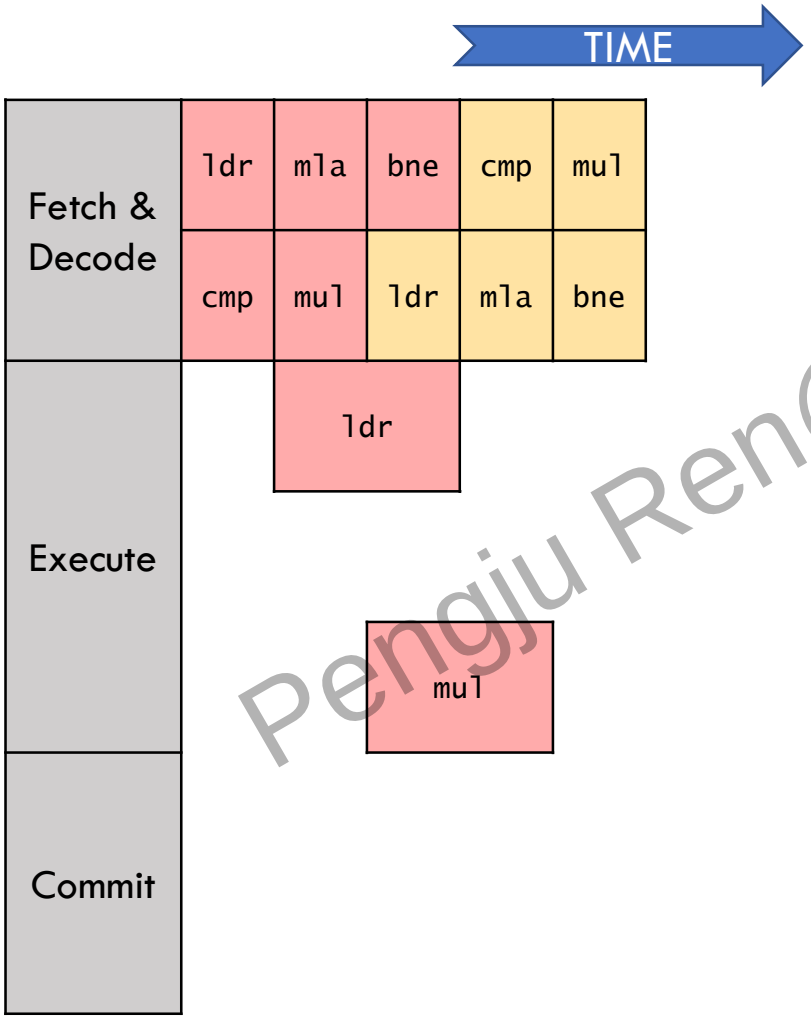
Example: Superscalar OoO polynomial evaluation




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

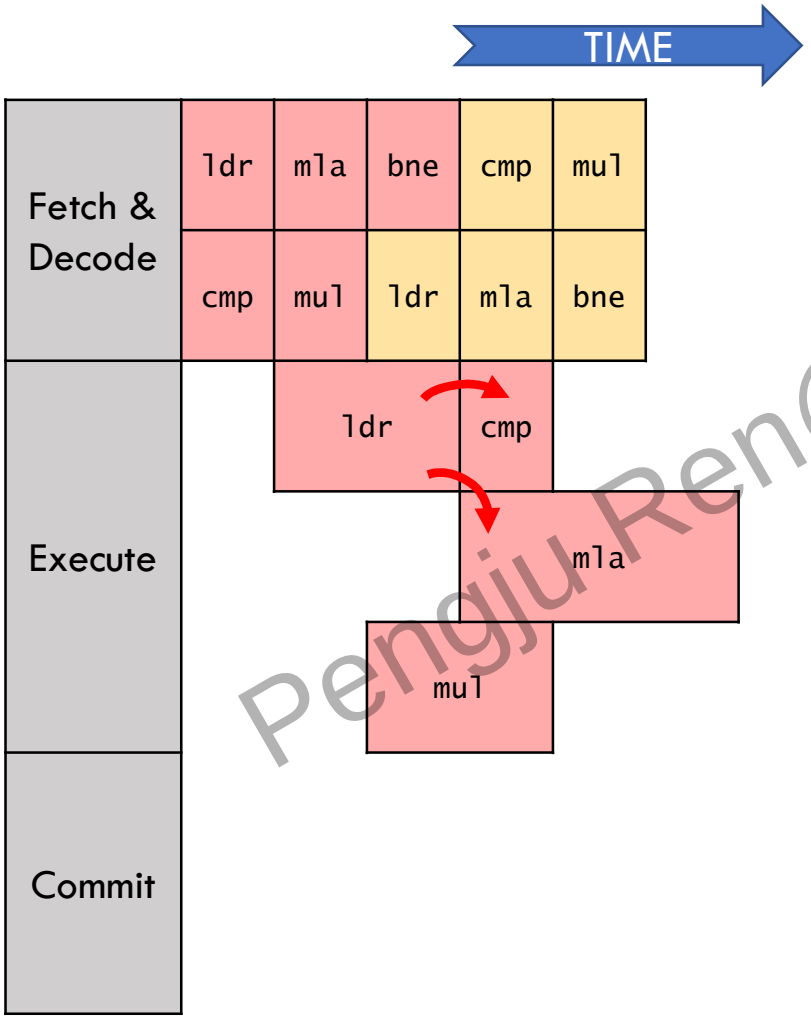
Example: Superscalar OoO polynomial evaluation




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

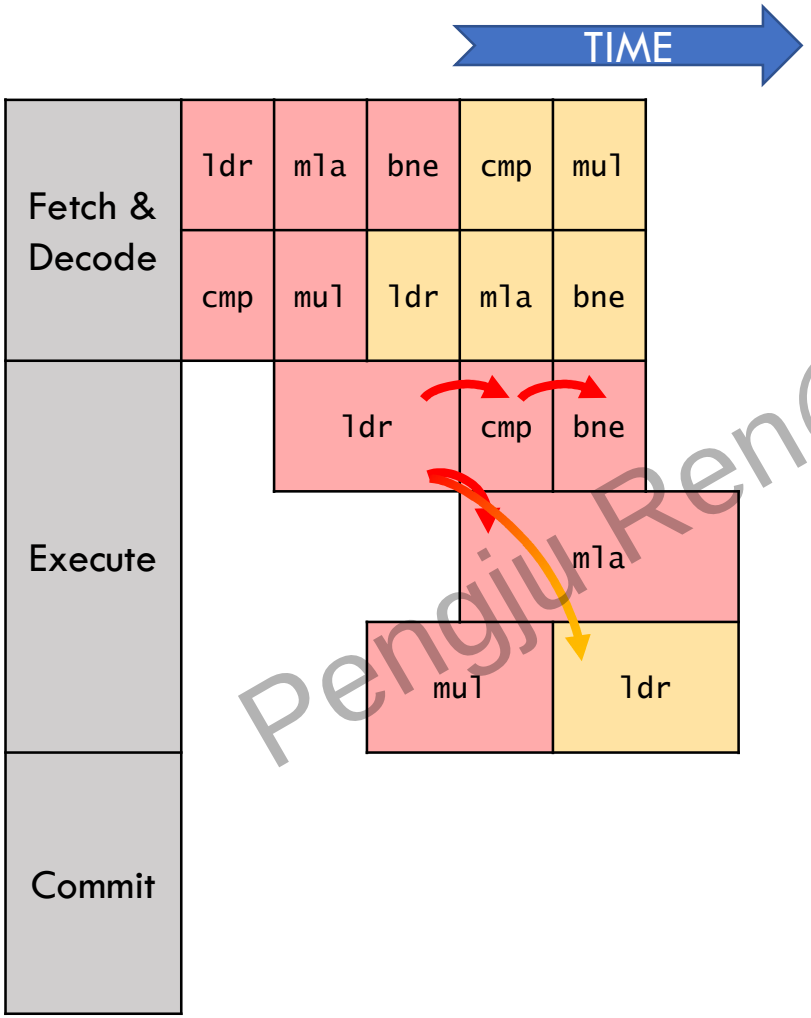
Example: Superscalar OoO polynomial evaluation




```
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
ldr     r5, [r3], #4
cmp     r1, r3
mla     r0, r4, r5, r0
mul     r4, r2, r4
bne     .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

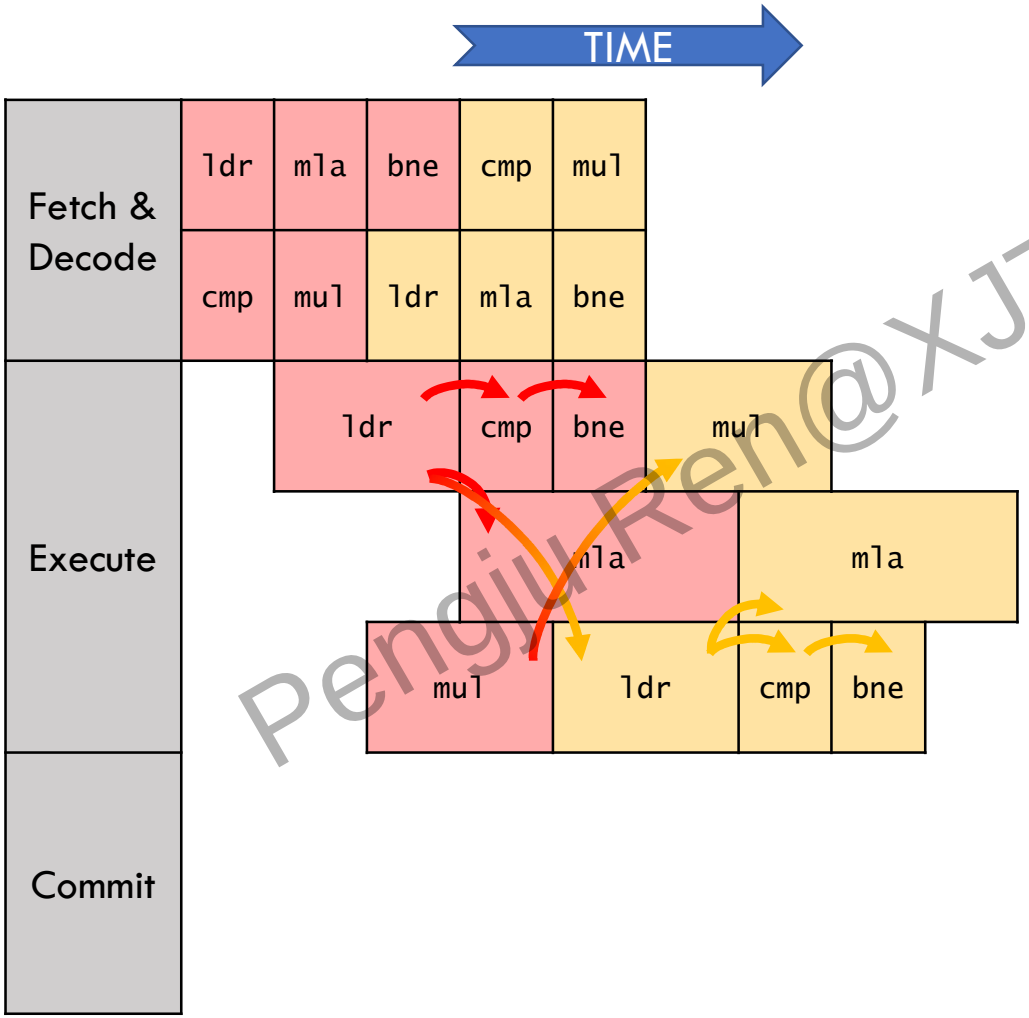
Example: Superscalar OoO polynomial evaluation




```
ldr   r5, [r3], #4
cmp   r1, r3
mla   r0, r4, r5, r0
mul   r4, r2, r4
bne   .L3
ldr   r5, [r3], #4
cmp   r1, r3
mla   r0, r4, r5, r0
mul   r4, r2, r4
bne   .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

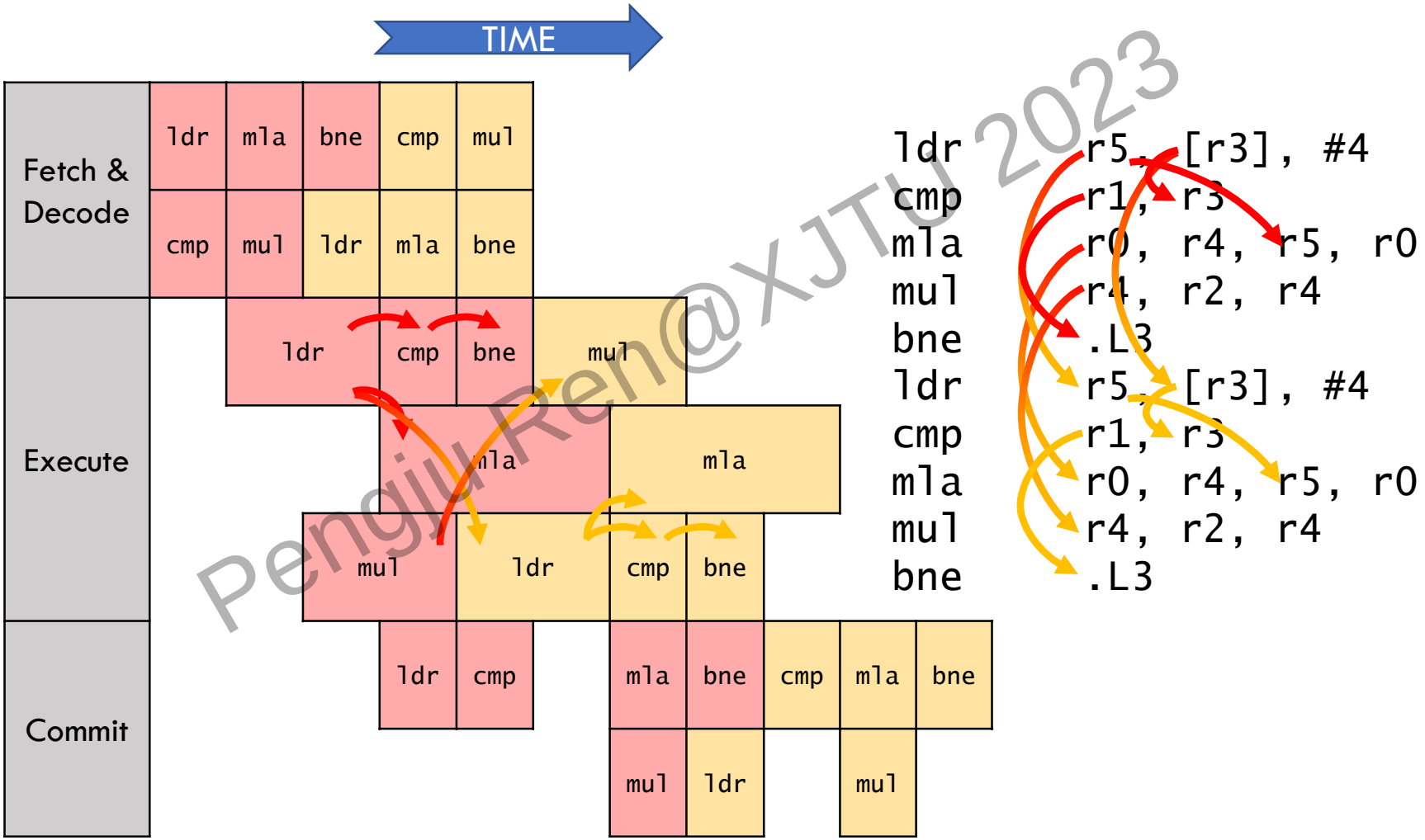
Example: Superscalar OoO polynomial evaluation



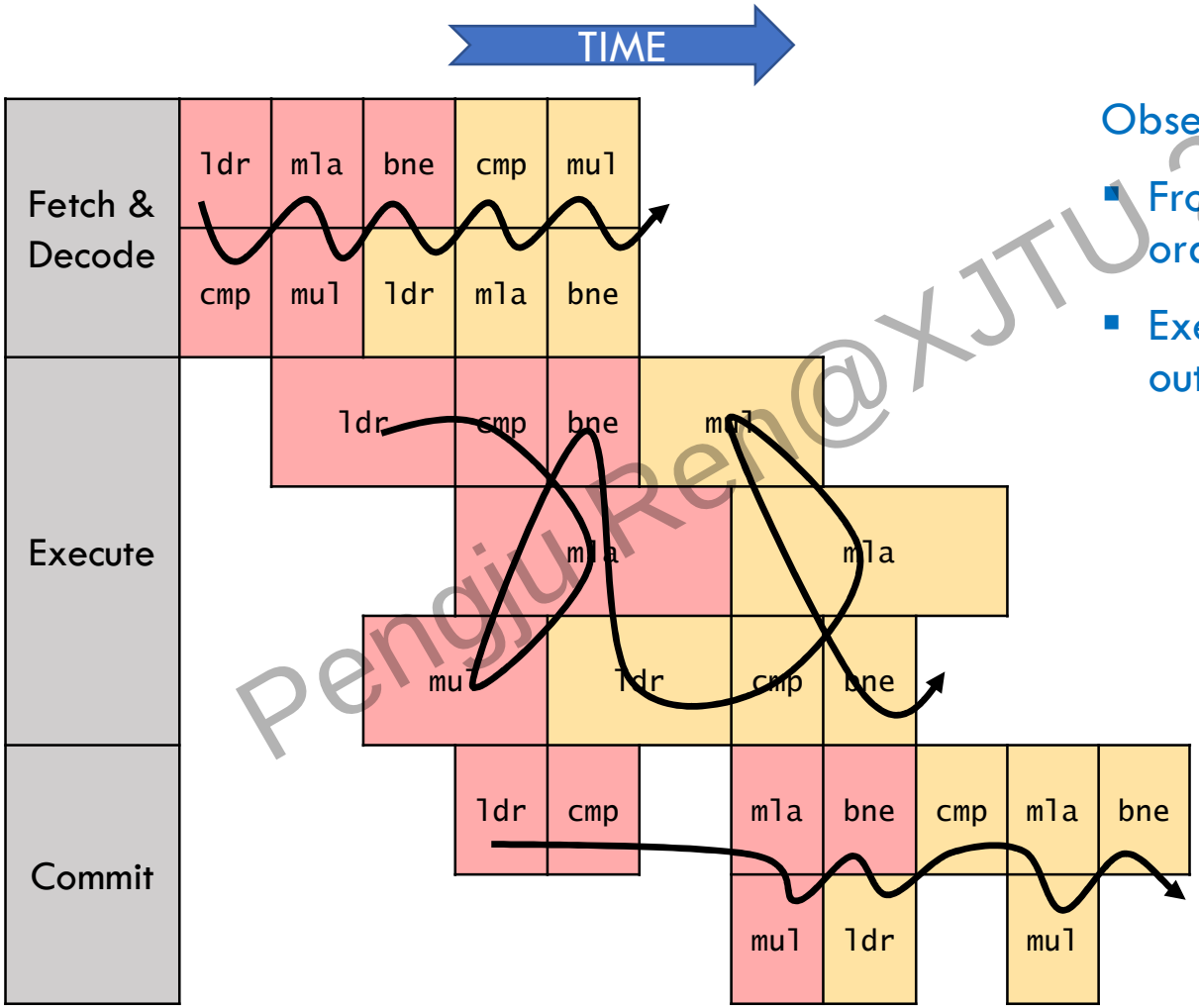

```
ldr   r5, [r3], #4
cmp   r1, r3
mla   r0, r4, r5, r0
mul   r4, r2, r4
bne   .L3
ldr   r5, [r3], #4
cmp   r1, r3
mla   r0, r4, r5, r0
mul   r4, r2, r4
bne   .L3
```

`ldr`, `mul` execute in 2 cycles
`cmp`, `bne` execute in 1 cycle
`mla` executes in 3 cycles

Example: Superscalar OoO polynomial evaluation



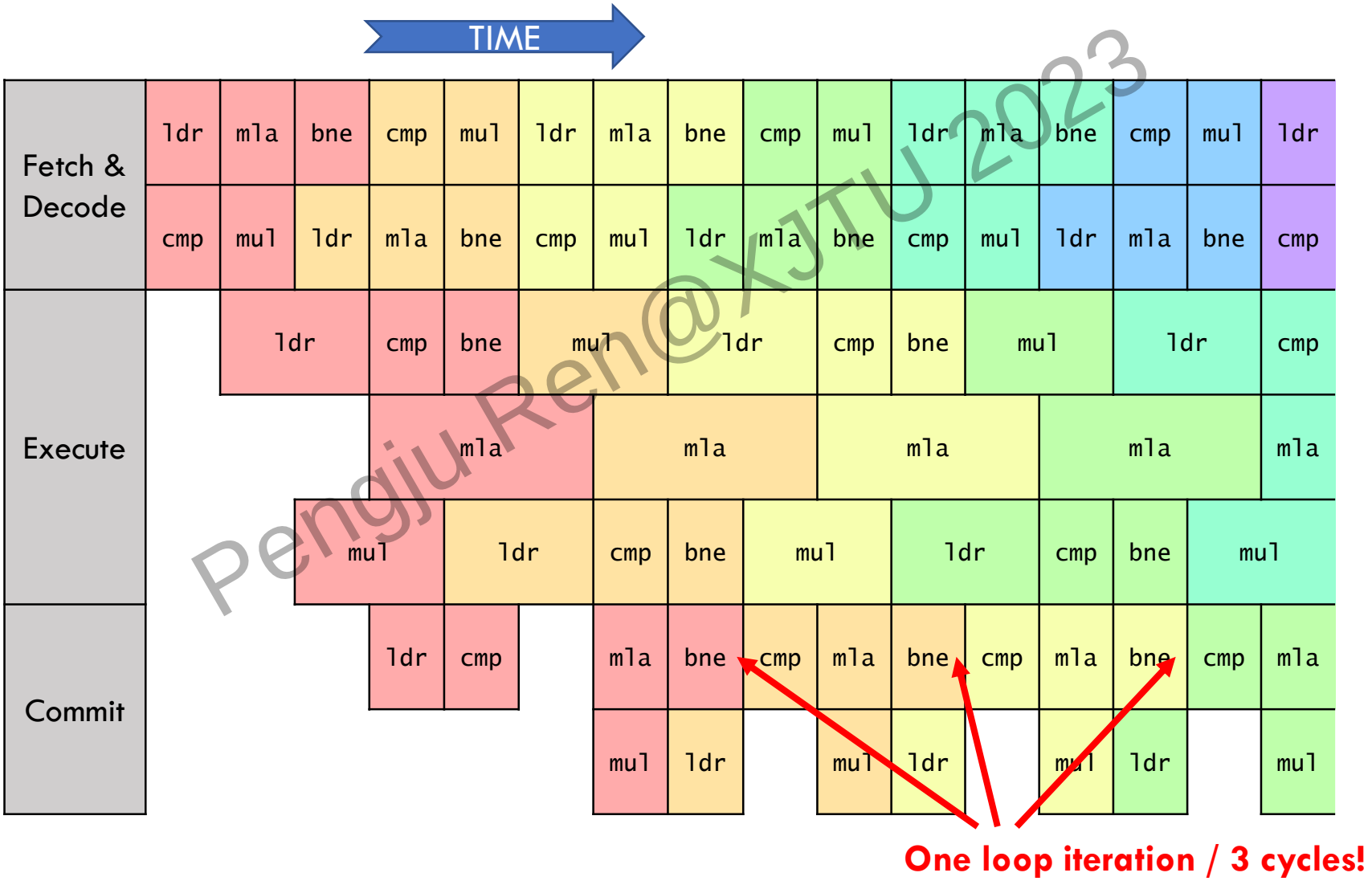
Example: Superscalar OoO polynomial evaluation



Observe:

- Front-end & commit in-order (i.e., left-to-right)
- Execute out-of-order

Example: Superscalar OoO polynomial evaluation



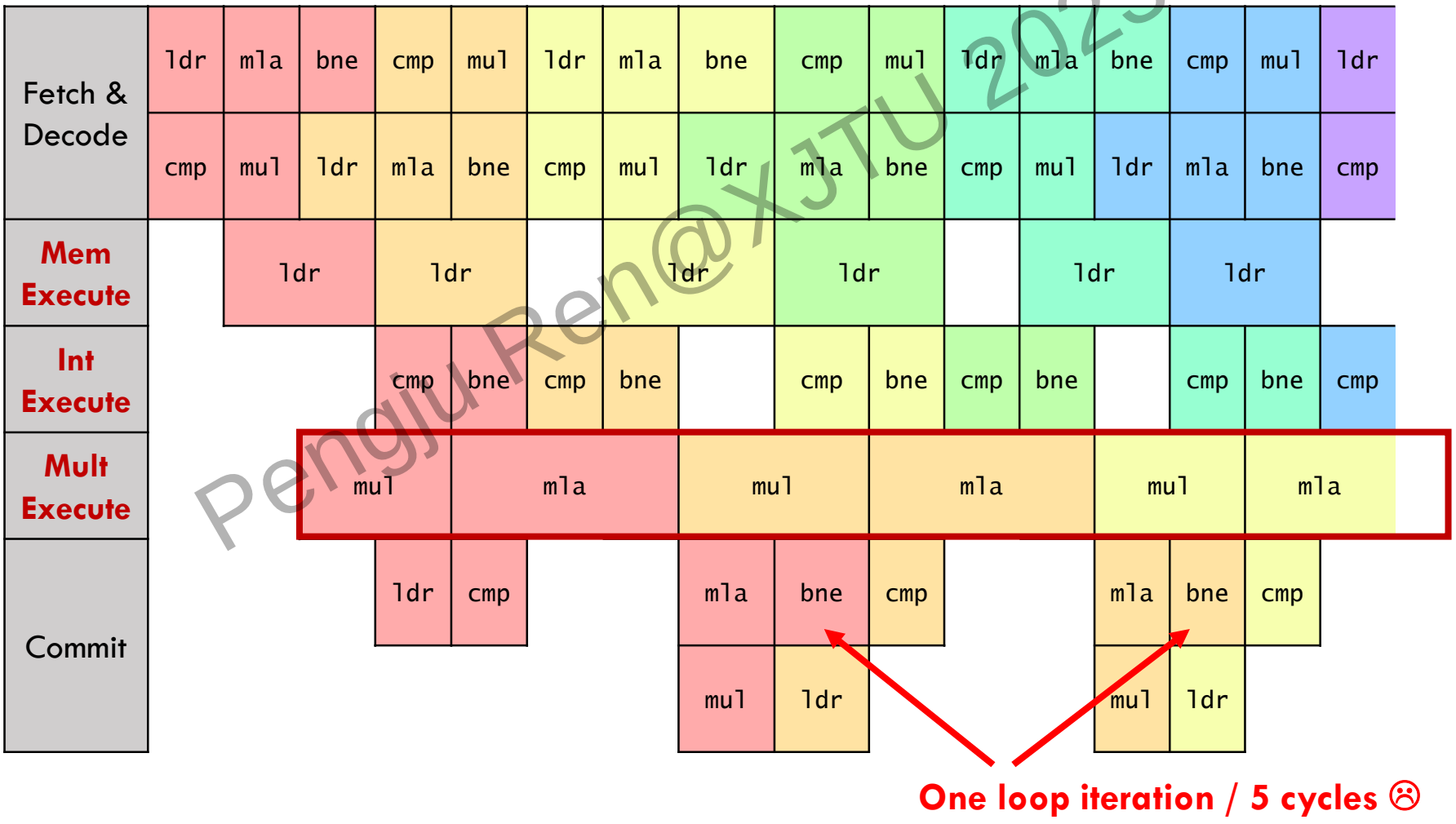
Structural hazards: Other throughput limitations

- Actually execution units are specialized
  - Floating-point (add/multiply)
  - Integer (add/multiply/compare)
  - Memory (load/store)

- Processor designers must choose which execution units to include and how many

- *Structural hazard*: Data is ready, but instr'n cannot issue because no hardware is available

Example: Structural hazards can severely limit performance



Home work (Due date: 3.20 Monday)

- Please do the dataflow analysis using RISC-V ISA to test how many **IPC** improvement can be achieved using OOO compared with In-order execution for *Polynomial evaluation*

  Assuming `ld, mul` execute in 2 cycles

  `add, beq` execute in 1 cycle

Case A: 2-way-3-issue
Fetch & decode 2 instrs
Execution units are specialized
- FPU (add/mutiply) x1
- Integer (add/compare) x1
- Memory (load/store) x1

Case B: 2-way-4-issue
Fetch & decode 2 instrs
Execution units are specialized
- FPU (add/multiply) x2
- Integer (add/compare) x1
- Memory (load/store) x1

Notes: for $value = \sum_{j=0}^{terms} coef[j]x^j$ *, x and coefficients floating numbers*

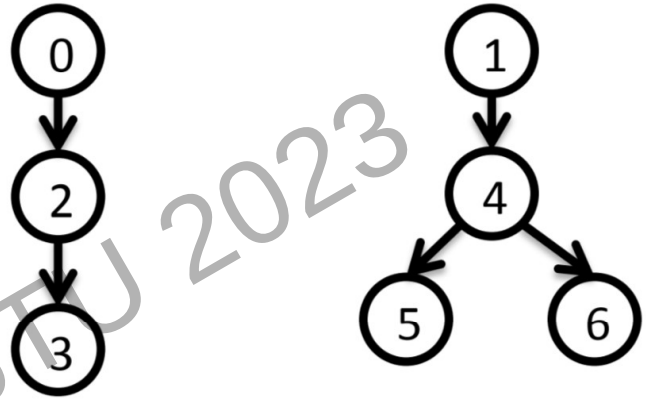
Out-of-Order Execution: Under the Hood

Out-of-Order(OOO) Introduction

Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2OI	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

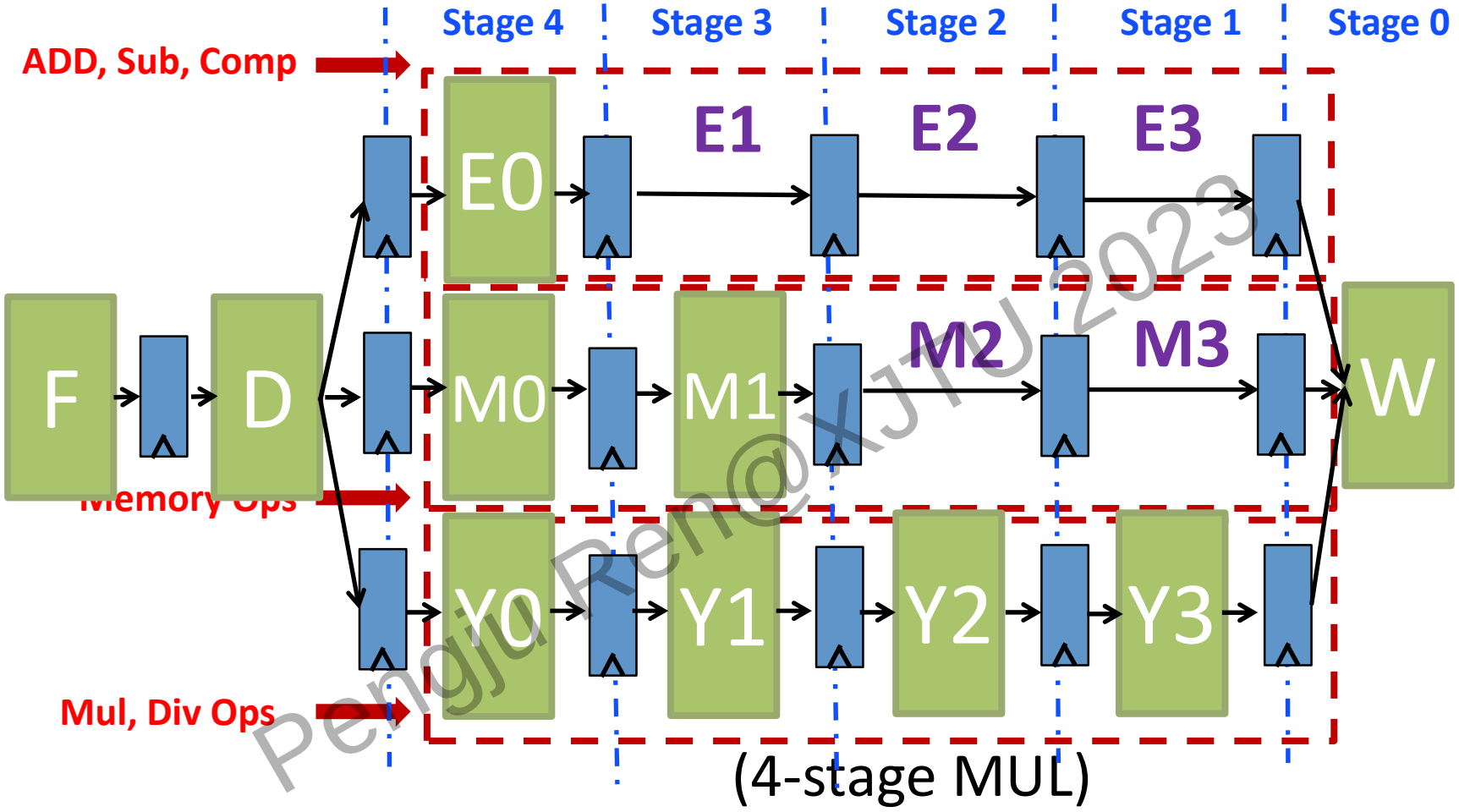
OOO Motivating Code Sequence

0	MUL	X1	X2	X3
1	ADDI	X11	X10	1
2	MUL	X5	X1	X4
3	MUL	X7	X5	X6
4	ADDI	X12	X11	1
5	ADDI	X13	X12	1
6	ADDI	X14	X12	2



- Two independent sequences of instructions enable flexibility in terms of how instructions are scheduled in total order
- We can schedule statically in software or dynamically in hardware

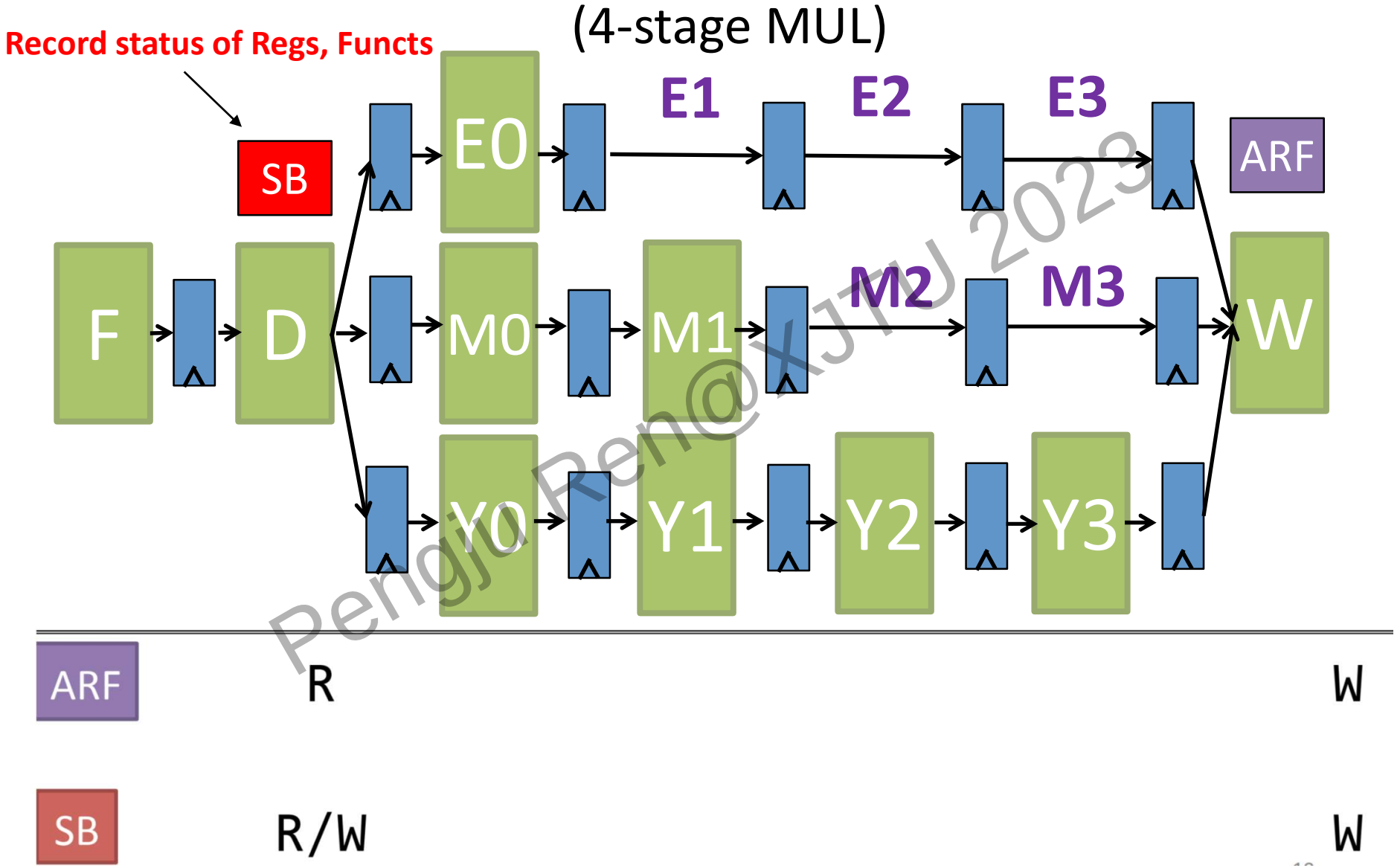
I4: In-order Front End, Issue, WriteBack, Commit



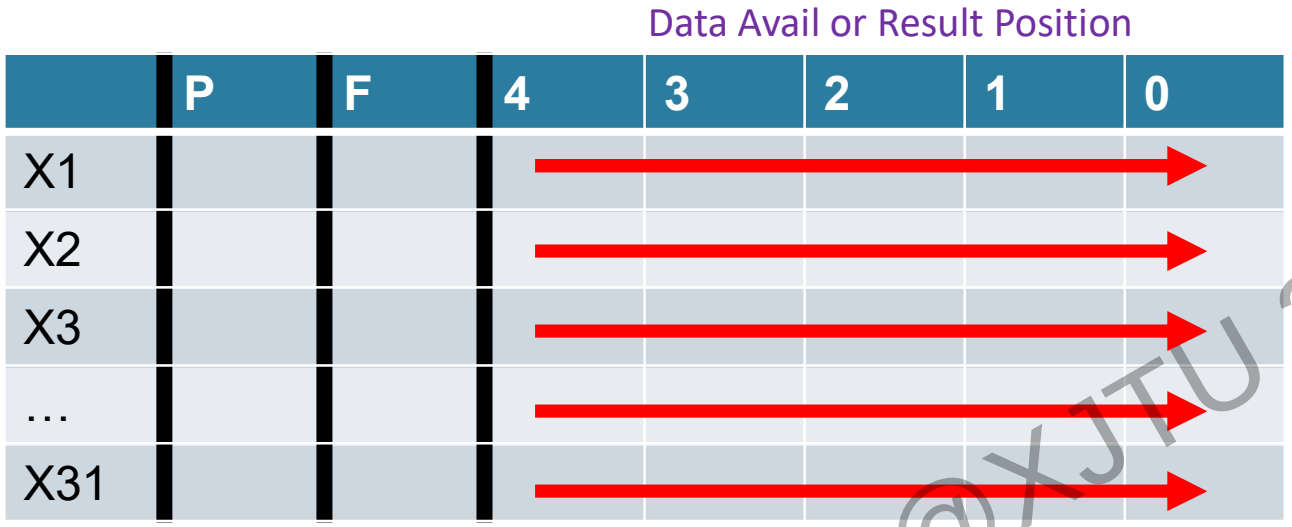
(4-stage MUL)

To avoid increasing CPI, needs full bypassing which can be expensive, To help cycle time, and Issue stage where register file read and instruction "issued" to Functional Unit

I4: In-order Front End, Issue, WriteBack, Commit



Basic Scoreboard



P: Pending, Write to Destination in flight

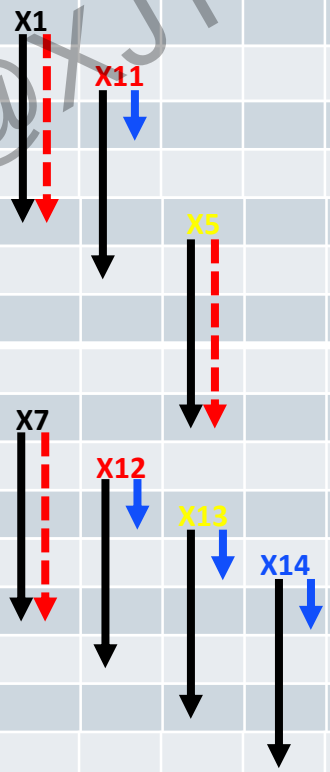
F: Which functional unit is writing register

Data Avail: Where is the write data in the functional unit pipeline

- A One in Data avail. In column '1' means that result data is in stage '1' of functional unit F
- Can use F and Data Avail. Fields to determine when to bypass and where to bypass from
- A one in column zero means that cycle functional unit is in the Writeback stage
- Bits in Data Avail. Field shift right every cycle

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	E1	E2	E3	W										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	E1	E2	E3	W		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	E1	E2	E3	W	
6	ADDI	X14	X12	2												F	D	I	E0	E1	E2	E3	W

Cyc	D	I		4	3	2	1	0		Dest	Regs			
1	0													
2	1	0												
3	2	1		1										
4				1	1									
5					1	1								
6	3	2				1	1							
7				1			1	1						
8					1			1						
9						1								
10	4	3					1							
11	5	4		1				1						
12	6	5		1	1									
13		6		1	1	1								
14				1	1	1	1							
15					1	1	1	1						
16						1	1	1						
17							1	1						
18								1						



Black arrow: Pending

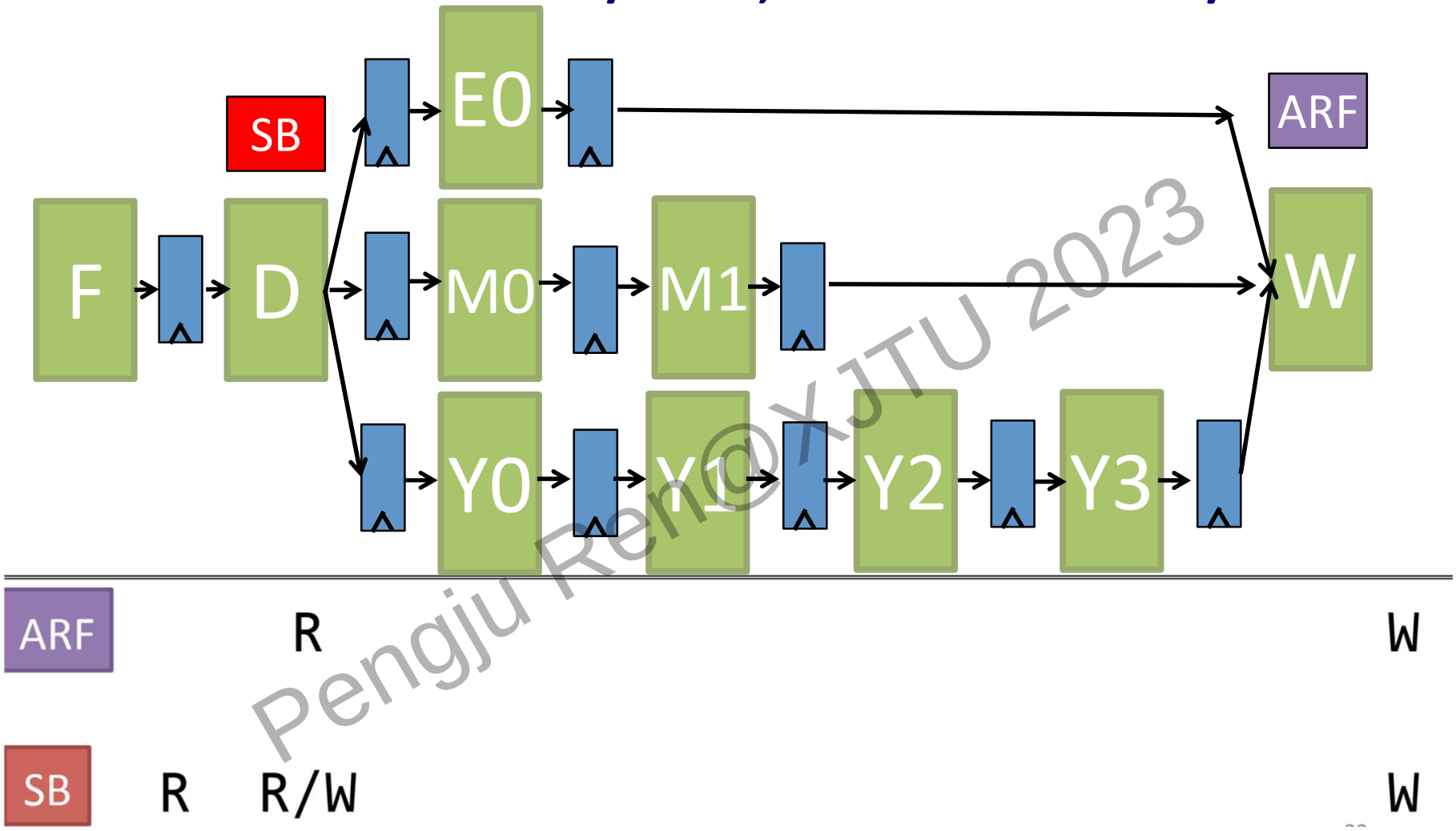
Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

Out-of-Order(OOO) Introduction

Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2OI	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

I2O2: In-order Frontend/Issue, Out-of-order WB/Commit



I2O2 Scoreboard

- Similar to I4, but we can now use it to **track structural hazards** on Writeback port
- Set bit in Data Avail. **According to length of pipeline**
- Architecture conservatively stalls to avoid WAW hazards by **stalling in Decode** therefore current scoreboard sufficient. More complicated scoreboard needed for processing WAW hazards.

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	E1	E2	E3	W										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	E1	E2	E3	W		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	E1	E2	E3	W	
6	ADDI	X14	X12	2												F	D	I	E0	E1	E2	E3	W

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	E0	W			

Structural hazard

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs				
1	0													
2														
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs				
1	0													
2	1	0												
3														
4														
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														
16														
17														
18														

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1						X1					
4															
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check

Black arrow: Pending

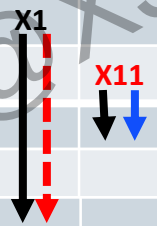
Dotted arrow: MUL(Y) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1											
4	3	2			1		1								
5															
6															
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

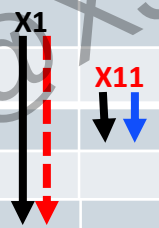
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	X0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	X0	W				
6	ADDI	X14	X12	2												F	D	I	I	X0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs						
1	0															
2	1	0														
3	2	1		1												
4	3	2			1		1									
5	3	2				1		1								
6																
7																
8																
9																
10																
11																
12																
13																
14																
15																
16																
17																
18																

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

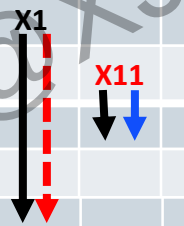
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1											
4	3	2			1		1								
5	3	2				1		1							
6	3	2					1								
7															
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

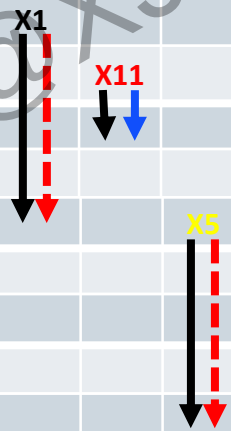
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1											
4	3	2			1		1								
5	3	2				1		1							
6	3	2					1								
7	4	3		1				1							
8															
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

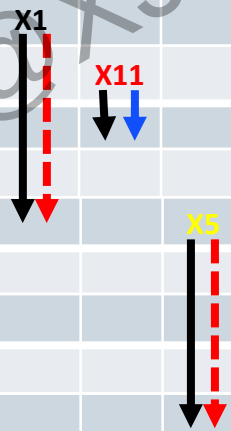
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1											
4	3	2			1		1								
5	3	2				1		1							
6	3	2					1								
7	4	3		1				1							
8	4	3			1										
9															
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

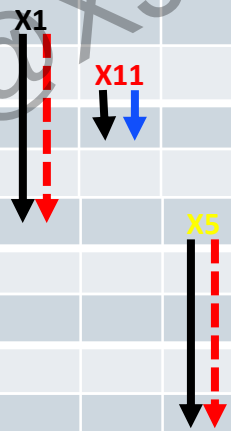
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs					
1	0														
2	1	0													
3	2	1		1											
4	3	2			1		1								
5	3	2				1		1							
6	3	2					1								
7	4	3		1				1							
8	4	3			1										
9	4	3				1									
10															
11															
12															
13															
14															
15															
16															
17															
18															

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

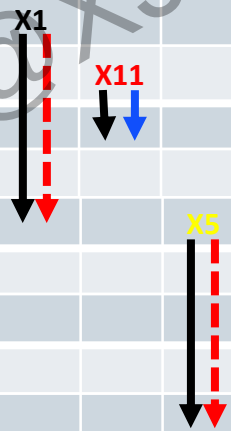
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs						
1	0															
2	1	0														
3	2	1		1												
4	3	2			1		1									
5	3	2				1		1								
6	3	2					1									
7	4	3		1				1								
8	4	3			1											
9	4	3				1										
10	4	3					1									
11																
12																
13																
14																
15																
16																
17																
18																

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

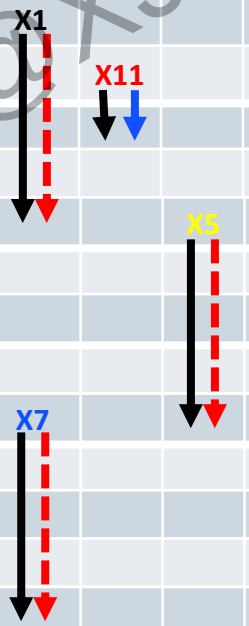
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs						
1	0															
2	1	0														
3	2	1		1												
4	3	2			1		1									
5	3	2				1		1								
6	3	2					1									
7	4	3		1				1								
8	4	3			1											
9	4	3				1										
10	4	3					1									
11	5	4		1				1								
12																
13																
14																
15																
16																
17																
18																

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

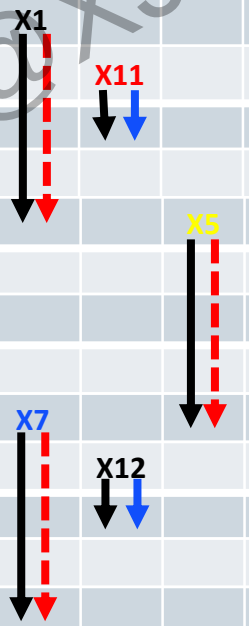
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs						
1	0															
2	1	0														
3	2	1		1												
4	2	1			1		1									
5	2	1				1		1								
6	3	2					1									
7	4	3		1				1								
8	4	3			1											
9	4	3				1										
10	4	3					1									
11	5	4		1				1								
12	6	5			1		1									
13																
14																
15																
16																
17																
18																

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs
1	0									
2	1	0								
3	2	1		1						
4	2	1			1		1			
5	2	1				1		1		
6	3	2					1			
7	4	3		1				1		
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		
12	6	5			1		1			
13		6				1	1	1		
14										
15										
16										
17										
18										

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs
1	0									
2	1	0								
3	2	1		1						
4	2	1			1		1			
5	2	1				1		1		
6	3	2					1			
7	4	3		1				1		
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		
12	6	5			1		1			
13		6				1	1	1		
14		6					1	1		
15										
16										
17										
18										

Black arrow: Pending

Dotted arrow: MUL(Y) unit

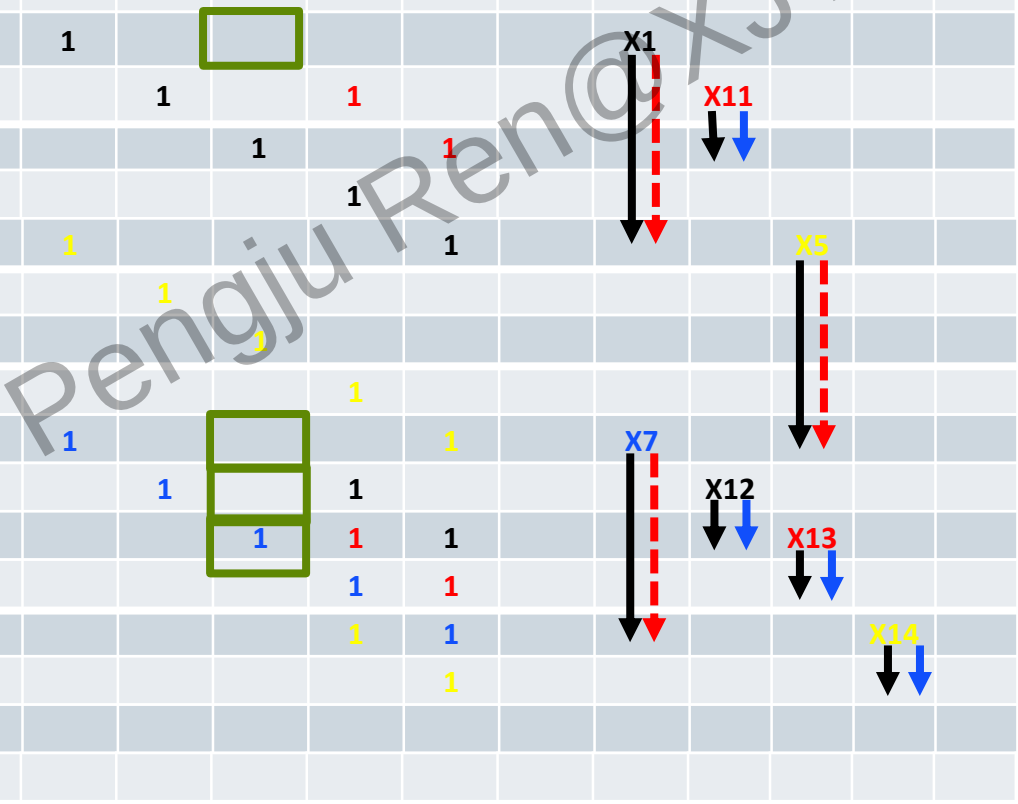
Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

Cyc	D	I		4	3	2	1	0		Dest Regs
1	0									
2	1	0								
3	2	1		1						
4	2	1			1		1			
5	2	1				1		1		
6	3	2					1			
7	4	3		1				1		
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		
12	6	5			1		1			
13		6				1	1	1		
14		6					1	1		
15							1	1		
16								1		
17										
18										

Resource hazard Check



Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

Green rectangle: WB conflict check

Early Commit Point ?

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W											
1	ADDI	X11	X10	1		F	D	I	E0	W													
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W					
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W				
6	ADDI	X14	X12	2												F	D	I	I	E0	W		

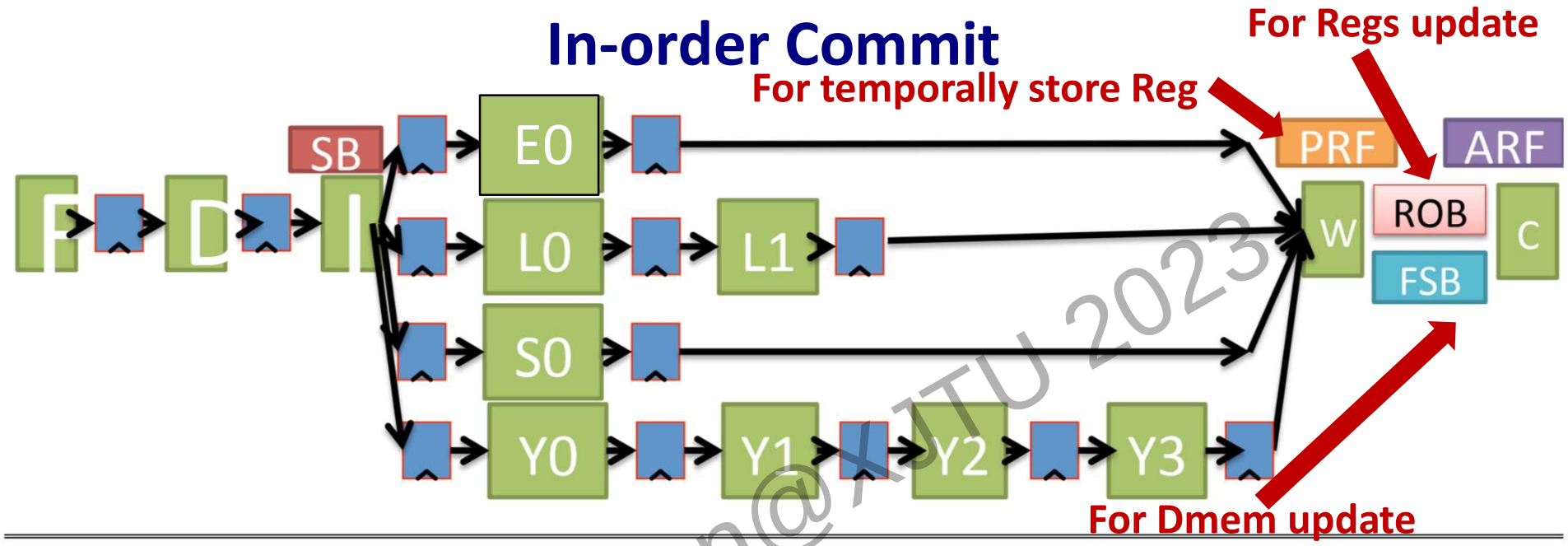
Limits certain types of exceptions.

If exception happens during the execution of Inst0, early write back of Inst1 will arouse problem!

Out-of-Order(OOO) Introduction

Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2OI	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

I2OI:In-order Frontend/Issue, Out-of-order WB, In-order Commit



ARF				W
SB	R/W		W	
PRF	R		W	
ROB	R/W		W	R/W
FSB			W	R/W

PRF = Physical Register File (Future File)

ROB=Reorder Buffer, FSB=Finished Store Buffer (1 Entry)

Reorder Buffer (ROB)

State	S	ST	V	Preg
--				
P	1			
F	1			
P	1			
P				
F				
P				
P				
--				
--				

Next Instruction allocates here(Decoder)

← Tail of ROB

Speculative because branch is in flight

Instruction wrote ROB OOO

← Head of ROB

Commit stage is waiting for Head of ROB to be finished

State: {Empty (--), Pending, Finished}
S: Speculative
ST: Store bit
V: Physical Register File Specifier Valid
Preg: Physical Register File Specifier

Indicating whether Data in PRF is valid?

Finished Store Buffer (FSB)

V	Op	Addr	Data
--			

- Only need one entry if we only support one memory instruction inflight at a time.
- Single Entry FSB makes allocation trivial.
- If support more than one memory instruction, we need to worry about Load/Store address aliasing.

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W						
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W					
6	ADDI	X14	X12	2												F	D	I	I	E0	W			

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

Cyc	D	I		4	3	2	1	0		ROB Regs (Preg same as Areg)
1	0									
2	1	0								P1
3	2	1		1						P11
4	3	2			1		1			

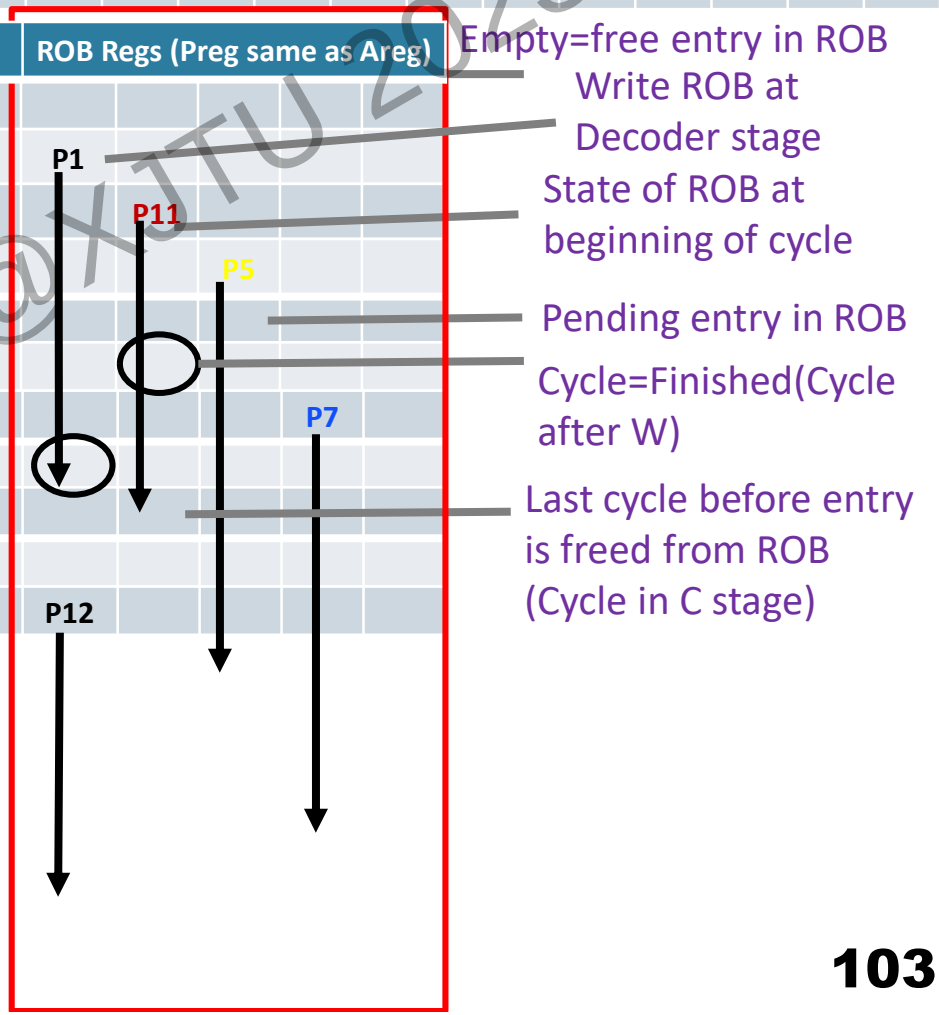
Empty=free entry in ROB

Write ROB at Decoder stage

State of ROB at beginning of cycle

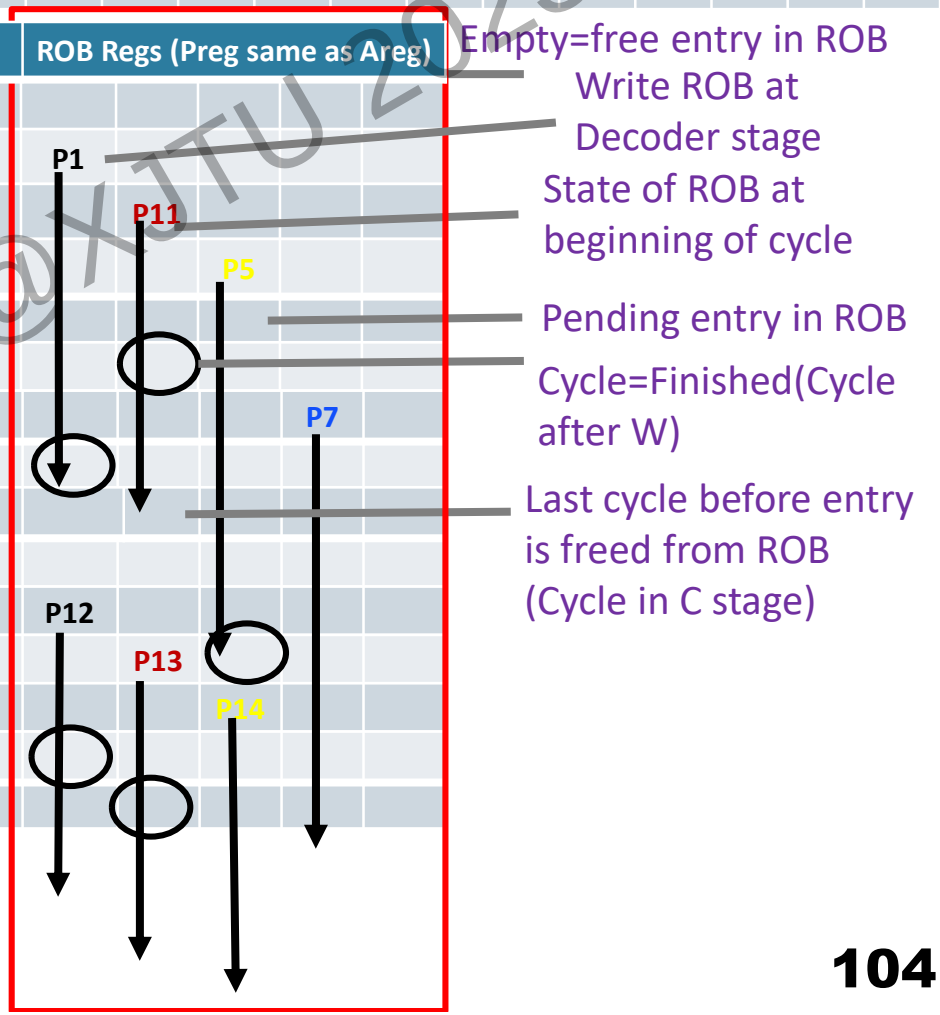
					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

Cyc	D	I		4	3	2	1	0		ROB Regs (Preg same as Areg)
1	0									
2	1	0								P1
3	2	1		1						P11
4	3	2			1		1			P5
5	3	2				1		1		
6	3	2					1			
7	4	3		1				1		P7
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		P12



					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

Cyc	D	I		4	3	2	1	0		ROB Regs (Preg same as Areg)
1	0									
2	1	0								
3	2	1		1						
4	3	2			1		1			
5	3	2				1		1		
6	3	2					1			
7	4	3		1				1		
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		
12	6	5			1		1			
13		6				1	1	1		
14		6					1	1		
15							1	1		



Empty=free entry in ROB

Write ROB at Decoder stage

State of ROB at beginning of cycle

Pending entry in ROB

Cycle=Finished(Cycle after W)

Last cycle before entry is freed from ROB (Cycle in C stage)

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

Cyc	D	I		4	3	2	1	0		ROB Regs (Preg same as Areg)
1	0									
2	1	0								P1
3	2	1		1						P11
4	3	2			1		1			P5
5	3	2				1		1		
6	3	2					1			
7	4	4		1				1		P7
8	4	3			1					
9	4	3				1				
10	4	3					1			
11	5	4		1				1		P12
12	6	5			1		1			P13
13		6				1	1	1		P14
14		6					1	1		
15							1	1		
16								1		
17										
18										
19										

Empty=free entry in ROB

State of ROB at beginning of cycle

Pending entry in ROB

Cycle=Finished(Cycle after W)

Last cycle before entry is freed from ROB (Cycle in C stage)

Entry becomes free and is free on the next cycle

What if First Instruction Causes an Exception

Option 2					0	1	2	3	4	5	6	7	8	9
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	/	
1	ADDI	X11	X10	1		F	D	I	E0	W	r	--	/	
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	/	
3	MUL	X7	X5	X6				F	D	D	D	I	/	
4	ADDI	X12	X11	1					F	F	F	D	/	.

What about Branches ?

Option 2					0	1	2	3	4	5	6	7	8	9
0	BEQ	X1	X2	Target	F	D	I	E0	W	C				
1	ADDI	X11	X10	1		F	D	I	E0	/				
2	MUL	X5	X1	X4			F	D	I	/				
3	MUL	X7	X5	X6				F	D	/				
T	ADDI	X12	X11	1					F	D	I	.	.	.

Squash instructions in ROB when Branch commits

Option 1					0	1	2	3	4	5	6	7	8	9
0	BEQ	X1	X2	Target	F	D	I	E0	W	C				
1	ADDI	X11	X10	1		F	D	I	-					
2	MUL	X5	X1	X4			F	D	-					
3	MUL	X7	X5	X6				F	-					
T	ADDI	X12	X11	1					F	D	I	◦	◦	◦

Squash instructions earlier. Has complexity, ROB needs many ports

Option 3					0	1	2	3	4	5	6	7	8	9
0	BEQ	X1	X2	Target	F	D	I	E0	W	C				
1	ADDI	X11	X10	1		F	D	I	E0	W	/			
2	MUL	X5	X1	X4			F	D	I	Y0	Y1	Y2	...	/
3	MUL	X7	X5	X6				F	D	I	Y0	Y1	...	
T	ADDI	X12	X11	1					F	D	I	E0	W	r

Wait for Speculative instructions reach the Commit stage and Squash in the Commit stage

What about Branches ?

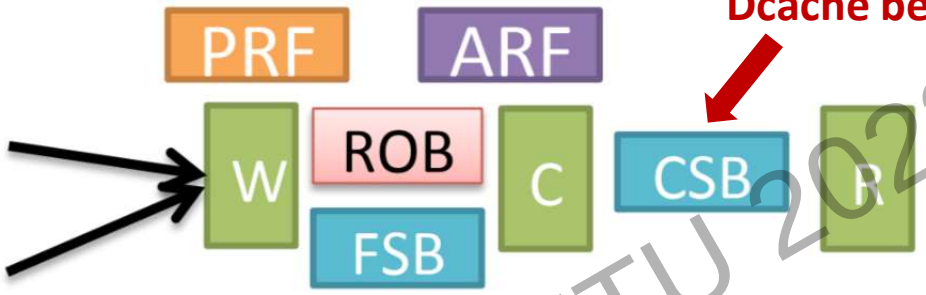
Three possible designs with decreasing complexity based on when to squash speculative instructions and de-allocate ROB entry:

- As soon as branch resolves
- When branch commits
- When Speculative instructions reach commit stage

Base design only allows one branch at a time. Second branch stalls in decode. Can add more bits to track multiple in-flight branches. (*more branches needs a mechanism named checkpoints*)

Avoiding Stalling Commit on Store Miss

CSB=Committed Store Buffer
（To avoid unpredictable Dcache behaver)



0	OpA	F	D	I	X0	W	C			/				
1	SW		F	D	I	S0	W	C	C	C	C			
2	OpB			F	D	I	E0	W	W	W	W	C		
3	OpC				F	D	I	E0	E0	E0	E0	W	C	
4	OpD					F	D	I	I	I	I	E0	W	C

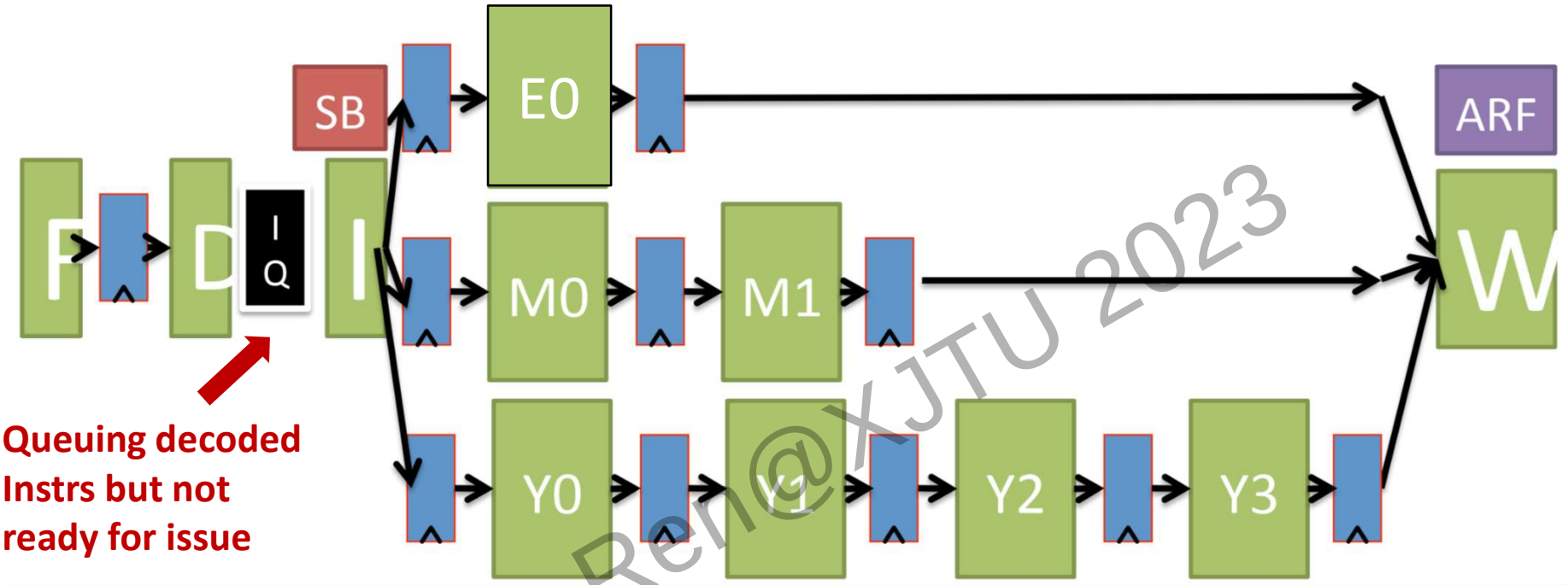
With Retire Stage

0	OpA	F	D	I	X0	W	C			/				
1	SW		F	D	I	S0	W	C	R	R	R			
2	OpB			F	D	I	E0	W	C					
3	OpC				F	D	I	E0	W	C				
4	OpD					F	D	I	E0	W	C			

Out-of-Order(OOO) Introduction

Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2OI	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

IO3: In-order Frontend, Out-of-order Issue/WB/Commit



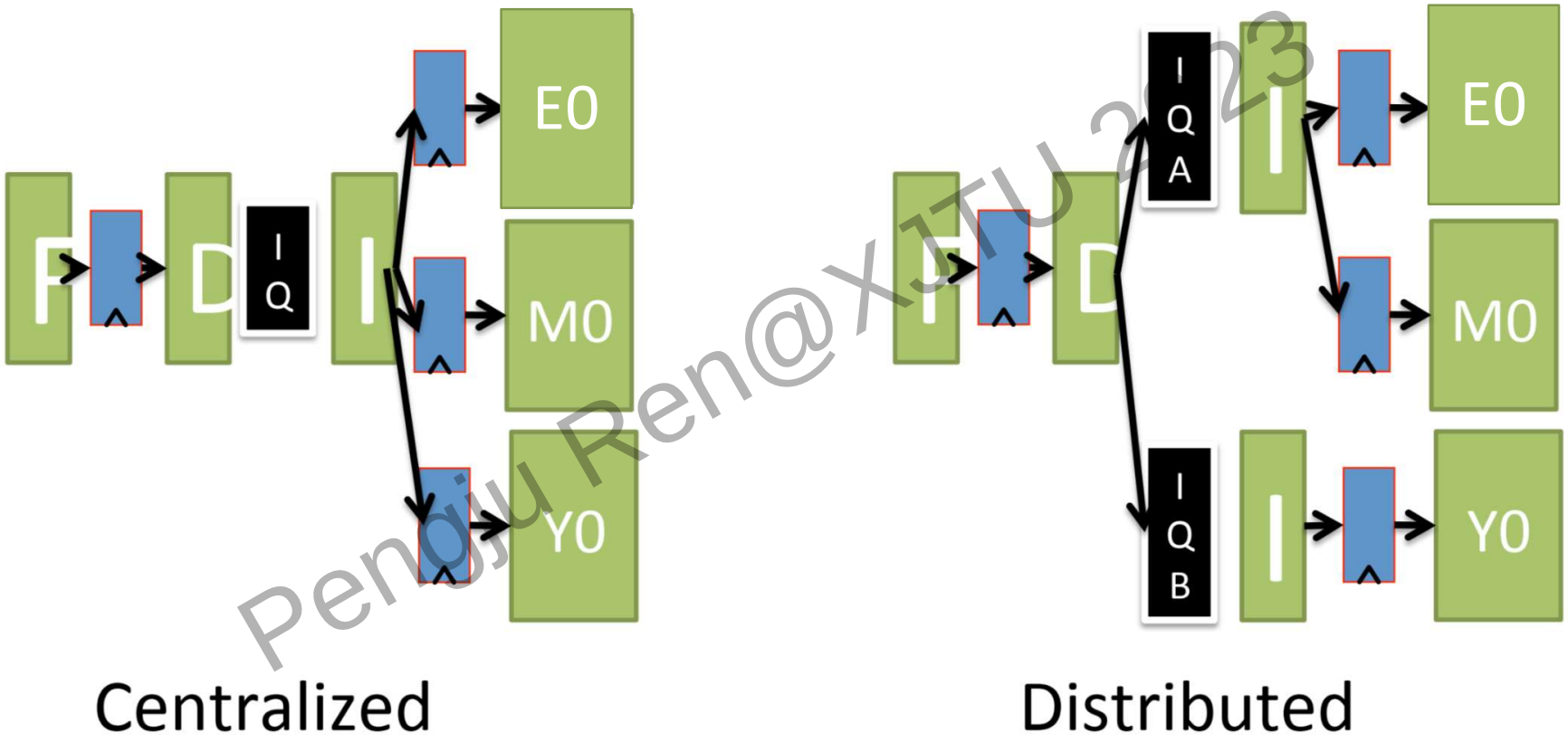
Issue Queue (IQ a.k.a Reservation Station)

Op	Imm	S	V	Dest	V	P	Src0	V	P	Src1

Op: Opcode
Imm.: Immediate
S: Speculative Bit
V: Valid (Instruction has corresponding Src/Dest)
P: Pending (Waiting on operands to be produced)

For high performance, factor in bypassing

Centralized vs. Distributed IQ



Advanced Scoreboard

Data Avail

	P	4	3	2	1	0
X1						
X2						
X3						
…						
X31						

P: Pending, Write to Destination in flight

Data Avail: Where is the write data in the functional unit pipeline

- Data avail. **Now contains functional unit identifier**
- A non-empty value in column zero means that cycle functional unit is the Writeback stage
- Bits in Data Avail. Field shift right every cycle

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

I2OI

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i	I	Y0	Y1	Y2	Y3	W									
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

IO3

i means Instr is writing into IssueQueue

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs				
1	0												

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs				
1	0												
2	1		0										

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3

Circle denotes value present in ARF

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2					1		X11

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	1	X1	1	0	X4

Circle denotes value present in ARF

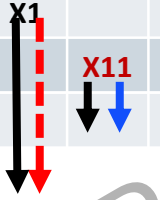
Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	



OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	1	X1	1	0	X4
MUL	-	0	1	X7	1	1	X5	1	0	X6

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		

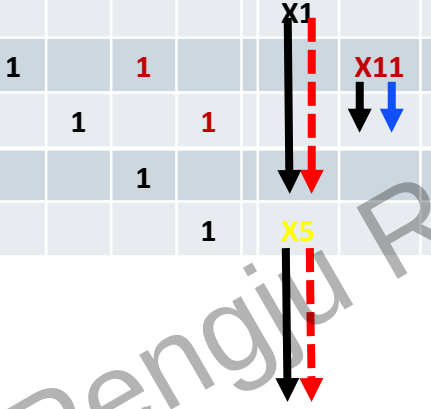
OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	1	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-

Ins2: Pending state of X1 is 1->0
MUL is ready to issue;
Ins4: Delay due to resource hazard
(Single output port of IQ)

Circle denotes value present in ARF
Black arrow: Pending
Dotted arrow: MUL(Y) unit
Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5



OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	1	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	1	X12	-	-	-

X1 bypassed, so no circle

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5
8			5		1		1		X12

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	1	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	1	X12	-	-	-
ADDI	1	0	1	X14	1	1	X12	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5
8			5		1		1		X12
9						1	1	1	X13

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	1	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	0	X12	-	-	-
ADDI	1	0	1	X14	1	0	X12	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5
8			5		1		1		X12
9						1	1	1	X13
10			3				1	1	

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	0	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	0	X12	-	-	-
ADDI	1	0	1	X14	1	0	X12	-	-	-

Ins3 : Pending state of X5 is 1->0
MUL is ready to issue;
Ins6 : Delay due to resource hazard
(Single output port of IQ)

Circle denotes value present in ARF
Black arrow: Pending
Dotted arrow: MUL(Y) unit
Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1					X5
8			5		1		1		X12
9						1	1	1	X13
10			3				1	1	
11			6	1				1	X7

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	0	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	0	X12	-	-	-
ADDI	1	0	1	X14	1	0	X12	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5
8			5		1		1		X12
9						1	1	1	X13
10			3				1	1	
11			6	1				1	X7
12					1		1		X14

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	0	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	0	X12	-	-	-
ADDI	1	0	1	X14	1	0	X12	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Cyc	D	i	I	4	3	2	1	0	Dest Regs
1	0								
2	1		0						
3	2		1	1					X1
4	3	2			1		1		X11
5	4	3				1		1	
6	5	4	2				1		
7	6	5	4	1				1	X5
8			5		1		1		X12
9						1	1	1	X13
10			3				1	1	
11			6	1				1	X7
12					1		1		X14
13						1		1	
14							1		

OP	Imm	S	V	Dest	V	P	Src0	V	P	Src1
MUL	-	0	1	X1	1	0	X2	1	0	X3
ADDI	1	0	1	X11	1	0	X10	-	-	-
MUL	-	0	1	X5	1	0	X1	1	0	X4
MUL	-	0	1	X7	1	0	X5	1	0	X6
ADDI	1	0	1	X12	1	0	X11	-	-	-
ADDI	1	0	1	X13	1	0	X12	-	-	-
ADDI	1	0	1	X14	1	0	X12	-	-	-

Circle denotes value present in ARF

Black arrow: Pending

Dotted arrow: MUL(Y) unit

Blue arrow: ADD(X) unit

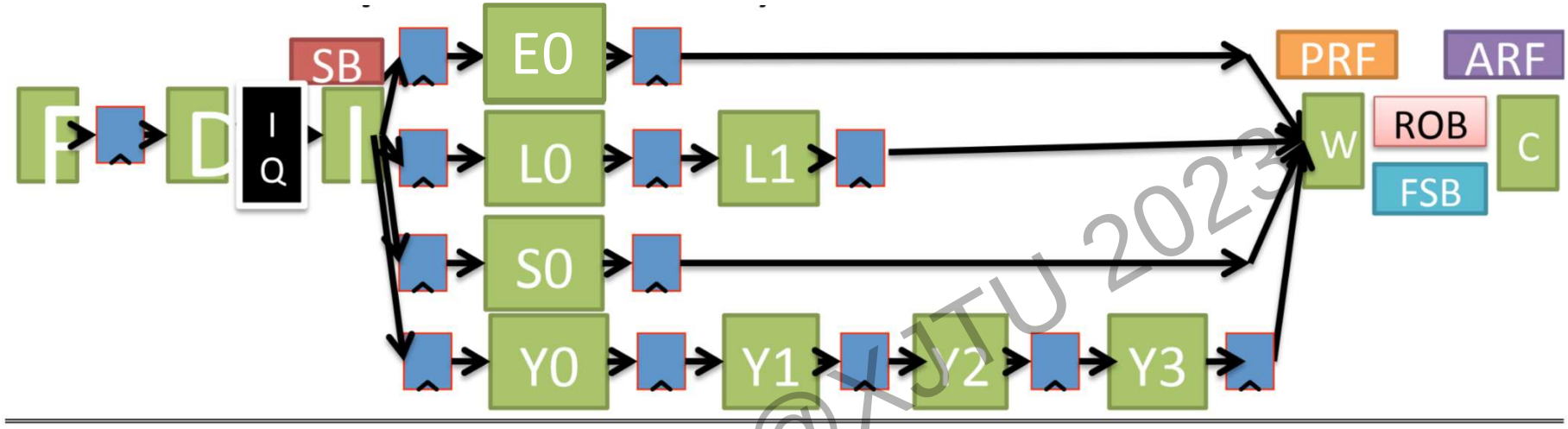
Assume All Instruction in IQ

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	D	D	I	I	I	I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	F	F	D	D	D	D	I	E0	W	r			C		
5	ADDI	X13	X12	1								F	F	F	F	D	I	E0	W	r			C	
6	ADDI	X14	X12	2												F	D	I	I	E0	W	r		C

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W												
1	ADDI	X11	X10	1		F	D	I	E0	W														
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W				
4	ADDI	X12	X11	1					F	D	i	I	E0	W										
5	ADDI	X13	X12	1						F	D	i	I	E0	W									
6	ADDI	X14	X12	2							F	D	i			I	E0	W						

Better performance than previous ?

IO2I:In-order Frontend, Out-of-order Issue/WB, In-order Commit



ARF					W
SB		R/W		W	
PRF		R		W	
ROB	R/W			W	R/W
FSB				W	R/W
IQ	W	R/W			

OOO 2-wide Superscalar with 1 ALU

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	D	i	I	E0	W	r							C		
5	ADDI	X13	X12	1						F	D	i	I	E0	W	r							C	
6	ADDI	X14	X12	2							F	D	i			I	E0	W	r					C

Single port Issue Queue

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	D	I	E0	W	r								C		
5	ADDI	X13	X12	1						F	D	I	E0	W	r								C	
6	ADDI	X14	X12	2							F	D	I	E0	W	r								C

Dual ports Issue Queue

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	ADDI	X11	X10	1		F	D	I	E0	W	r			C										
2	MUL	X5	X1	X4			F	D	i		I	Y0	Y1	Y2	Y3	W	C							
3	MUL	X7	X5	X6				F	D	i					I	Y0	Y1	Y2	Y3	W	C			
4	ADDI	X12	X11	1					F	D	I	E0	W	r								C		
5	ADDI	X13	X12	1						F	D	I	E0	W	r								C	
6	ADDI	X14	X12	2							F	D	I	E0	W	r								C

2way SuperScalar

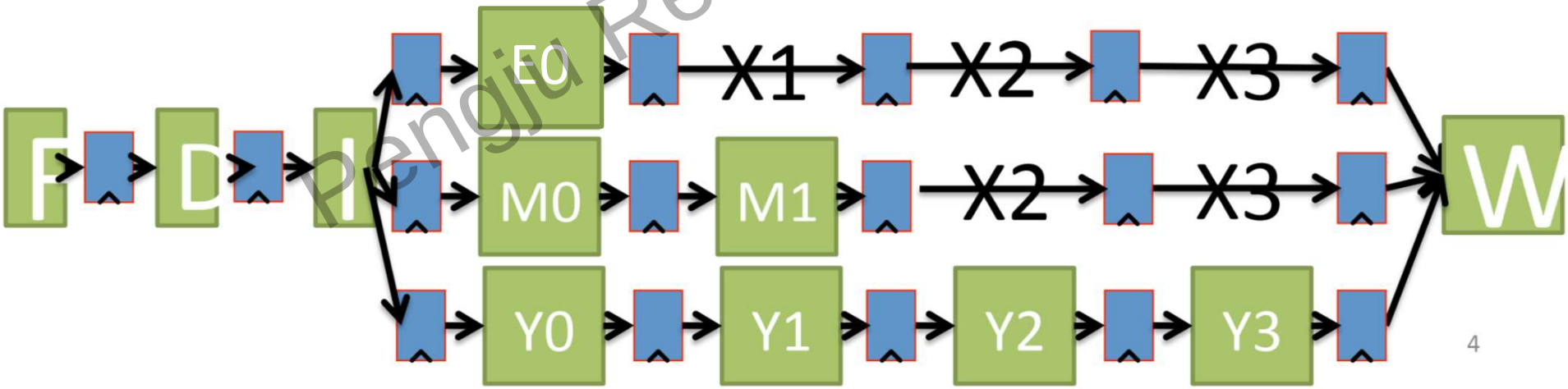
Agenda

SuperScalar Intro
Out-of-Order Processor (OOO)
Speculation and Branches
Register Renaming
Memory Disambiguation

Speculation and Branches: I4

0	MUL X1, X2, X3	F	D	I	Y0	Y1	Y2	Y3	W	/						
1	ADDI X4, X5, 1		F	D	I	E0	E1	E2	E3	W						
2	MUL X6, X1, X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W			
3	BEQ X6, X14, Target				F	D	D	D	I	I	I	I	E0	E1	E2	X3	W
4	ADDI X8, X9, 1					F	F	F	D	D	D.	D	I	--	--	--	--
5	ADDI X10,X11,1								F	F	F	F	D	--	--	--	--
6	ADDI X12,X13,1												F	--	--	--	--
7	T													F	D	I	--

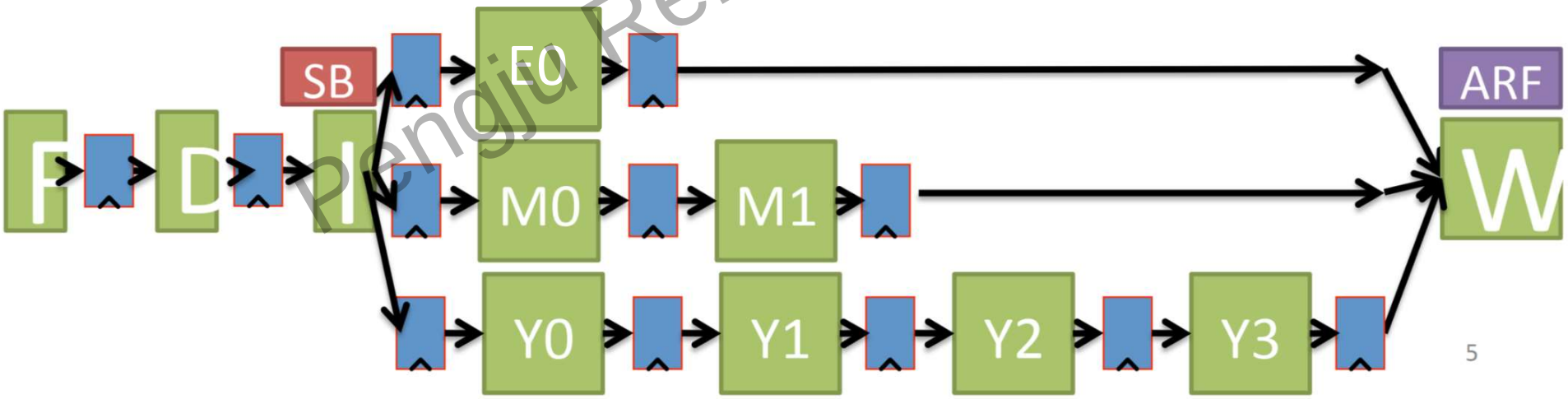
- No Speculative Instructions Commit State



Speculation and Branches: I2O2

0	MUL X1, X2, X3	F	D	I	Y0	Y1	Y2	Y3	W	/						
1	ADDI X4, X5, 1		F	D	I	E0	W									
2	MUL X6, X1, X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W			
3	BEQ X6, X14, Target				F	D	D	D	I	I	I	I	E0	W		
4	ADDI X8, X9, 1					F	F	F	D	D	D.	D	I	--	--	--	--
5	ADDI X10,X11,1								F	F	F	F	D	--	--	--	--
6	ADDI X12,X13,1												F	--	--	--	--
7	T													F	D	I	--

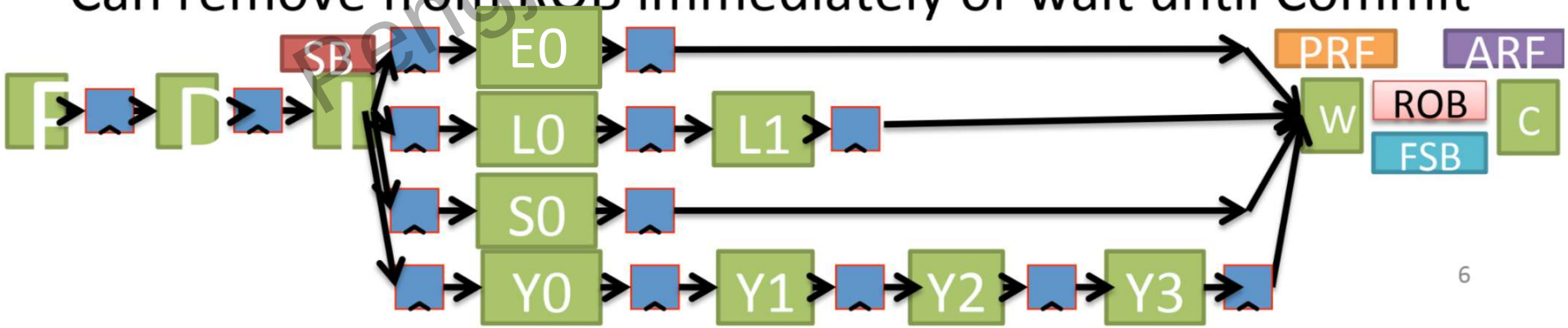
- No Speculative Instructions Commit State



Speculation and Branches: I2OI

0	MUL X1, X2, X3	F	D	I	Y0	Y1	Y2	Y3	W	C						
1	ADDI X4, X5, 1		F	D	I	E0	W	r			C					
2	MUL X6, X1, X4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C		
3	BEQ X6, X14, Target				F	D	D	D	I	I	I	I	E0	W	C	
4	ADDI X8, X9, 1					F	F	F	D	D	D.	D	I	--	--	--	--
5	ADDI X10,X11,1								F	F	F	F	D	--	--	--	--
6	ADDI X12,X13,1												F	--	--	--	--
7	T													F	D	I	--

- Must Squash Instructions in Pipeline after Branch to prevent PRF Write.
- Can remove from ROB immediately or wait until Commit



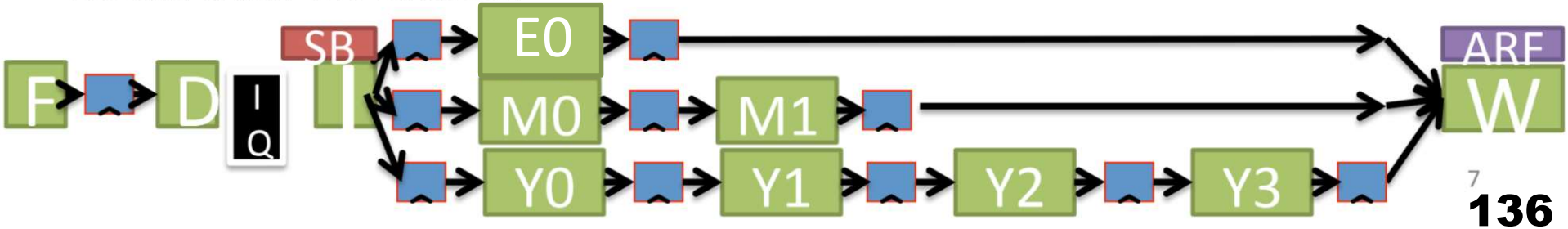
6

Speculation and Branches: IO3

0	MUL X1, X2, X3	F	D	I	Y0	Y1	Y2	Y3	W								
1	ADDI X4, X5, 1		F	D	I	E0	W										
2	MUL X6, X1, X4			F	D	i		I	Y0	Y1	Y2	Y3	W				
3	BEQ X6, X14, Target				F	D	i					I	E0	W			
4	ADDI X8, X9, 1					F	D	i	I	E0	W			--			
5	ADDI X10,X11,1						F	D	i	I	E0	W		--			
6	ADDI X12,X13,1							F	D	i			I	E0	W	--	--
7	? ? ?								F	D							--
8	? ? ?									F	D						
9	? ? ?										F	D					
10	? ? ?											F	D				
11	? ? ?												F	D			
T														F	D	I	

Speculative Instructions Wrote to ARF

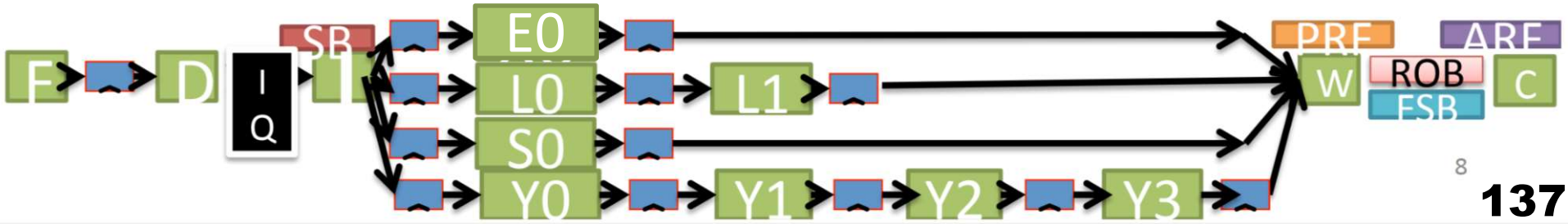
- No Control speculation for IO3
- Could Stall on Branch



Speculation and Branches: IO2I

Need to clean up Speculative state in PRF. Needs Selective Rollback.

0	MUL X1, X2, X3	F	D	I	Y0	Y1	Y2	Y3	W	C						
1	ADDI X4, X5, 1		F	D	I	E0	W	r			C					
2	MUL X6, X1, X4			F	D	i		I	Y0	Y1	Y2	Y3	W	C		
3	BEQ X6, X14, Target				F	D	I					I	E0	W	C	
4	ADDI X8, X9, 1					F	D	i	I	E0	W	r		--	--	--	--
5	ADDI X10,X11,1						F	D	i	I	E0	W		--	--	--	--
6	ADDI X12,X13,1							F	D	i						--	--
7	? ? ?								F	D							--
8	? ? ?									F	D						
9	? ? ?										F	D					
10	? ? ?											F	D				
11	? ? ?												F	D			
T														F	D	I	

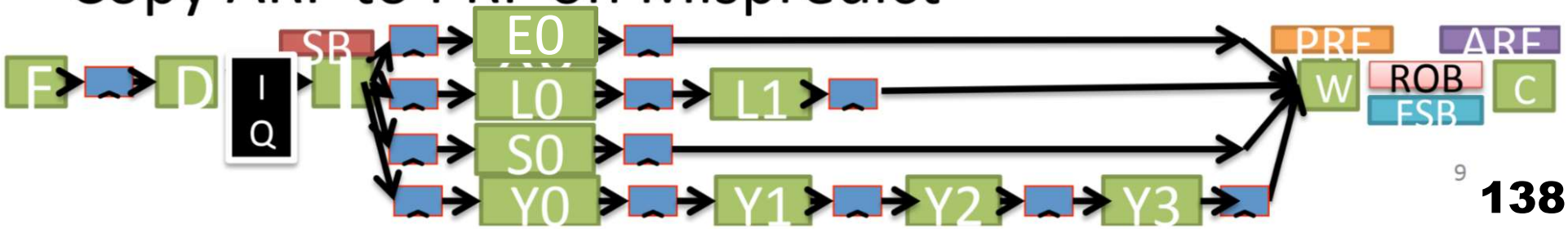


Speculation and Branches: IO2I

0	MUL R1, R2, R3	F	D	I	Y0	Y1	Y2	Y3	W	C							
1	ADDI X4, X5, 1		F	D	I	E0	W	r			C						
2	MUL X6, X1, X4			F	D	i		I	Y0	Y1	Y2	Y3	W	C			
3	BEQ X6, X14, Target				F	D	I					I	E0	W	C		
4	ADDI X8, X9, 1					F	D	i	I	E0	W	r	--	/-	--	--	
5	ADDI X10,X11,1						F	D	i	I	E0	W	R	--	/-	--	--
6	ADDI X12,X13,1							F	D	i			I	E0	/	--	--
7	? ? ?								F	D					/		--
8	? ? ?									F	D				/		
9	? ? ?										F	D			/		
10	? ? ?											F	D		/		
11	? ? ?												F	D	/		
12														F	/		
13															/		
T															F	D	I

Speculative Instructions Wrote to PRF Not ARF

- Copy ARF to PRF on Mispredict



Agenda

SuperScalar Intro
Out-of-Order Processor (OOO)
Speculation and Branches
Register Renaming
Memory Disambiguation

Types of Data Hazards

Consider executing a sequence of register-register instructions of type:

$$X_k \leftarrow X_i \text{ op } X_j$$

Data-dependence

$X_3 \leftarrow X_1$ op X_2 Read-after-Write

$X_5 \leftarrow X_3$ op X_4 **(RAW)** hazard

Anti-dependence

$X_3 \leftarrow X_1$ op X_2 Write-after-Read

$X_1 \leftarrow X_4$ op X_5 **(WAR)** hazard

Output-dependence

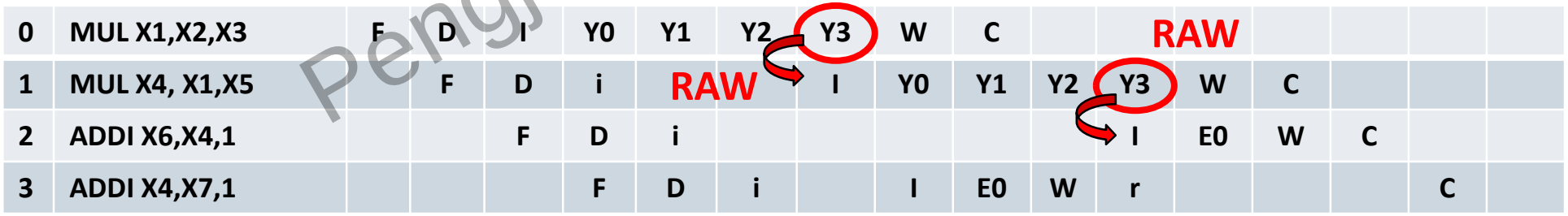
$X_3 \leftarrow X_1$ op X_2 Write-after-Write

$X_3 \leftarrow X_6$ op X_7 **(WAW)** hazard

WAW and WAR "name" Dependencies

- **WAW** and **WAR** are **not "Ture"** data dependencies
- **RAW** is "**Ture**" data dependency because reader needs result of writer
- "Name" dependencies exist <u>because we have limited number of "Names"</u> (register specifiers or memory addresses)

Breaking all "Name" Dependencies (Causes problems)



WAW and WAR "name" Dependencies

- **WAW** and **WAR** are **not "Ture"** data dependencies
- **RAW** is **"Ture"** data dependency because reader needs result of writer
- "Name" dependencies exist <u>because we have limited number of "Names"</u> (register specifiers or memory addresses)

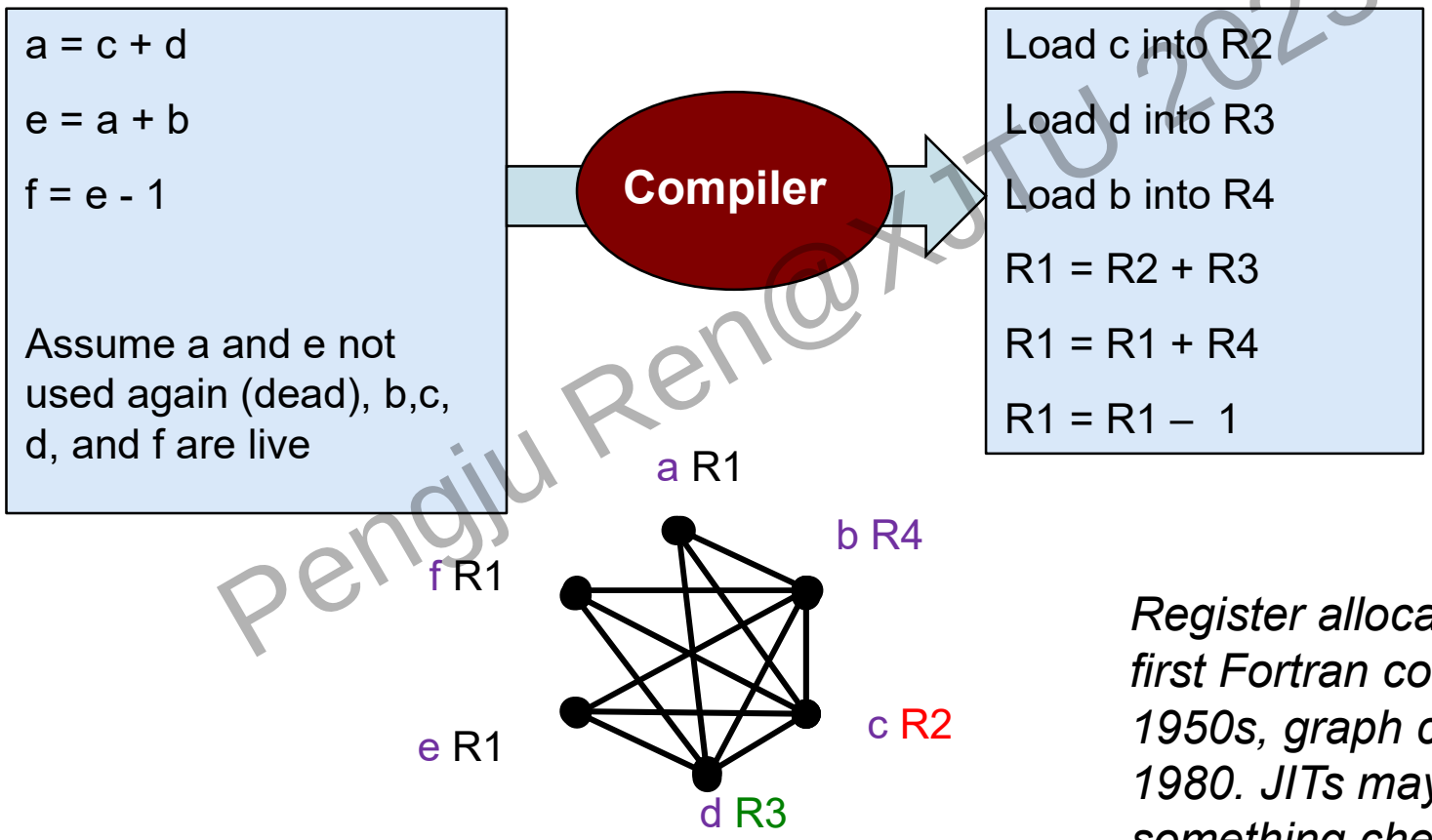
Breaking all "Name" Dependencies (Causes problems)

0	MUL X1,X2,X3	F	D	I	Y0	Y1	Y2	Y3	W	C							
1	MUL X4, X1,X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C			
2	ADDI X6,X4,1			F	D	i				WAW		I	E0	W	C		
3	ADDI X4,X7,1				F	D	i		I	E0	W	r				C	

WAR

Compilers Manage Memory and Registers

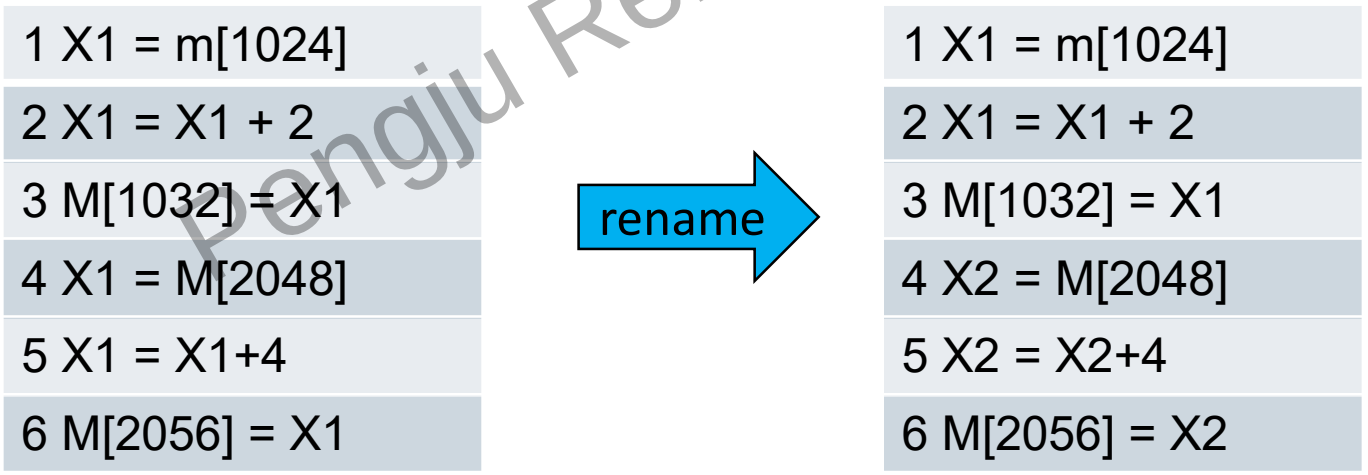
- Compiler performs “register allocation” to decide when to load/store and when to reuse



Register allocation in first Fortran compiler in 1950s, graph coloring in 1980. JITs may use something cheaper

Register Renaming

- Programmers/Compilers (have to) re-use registers for different, unrelated purposes
- Idea: **Re-name on the fly** to resolve (fake) dependencies (anti-dependency)
- Additional benefit: CPU can have more physical register than ISA!
  - Alpha 21264 CPU has 80 integer register; ISA only 32



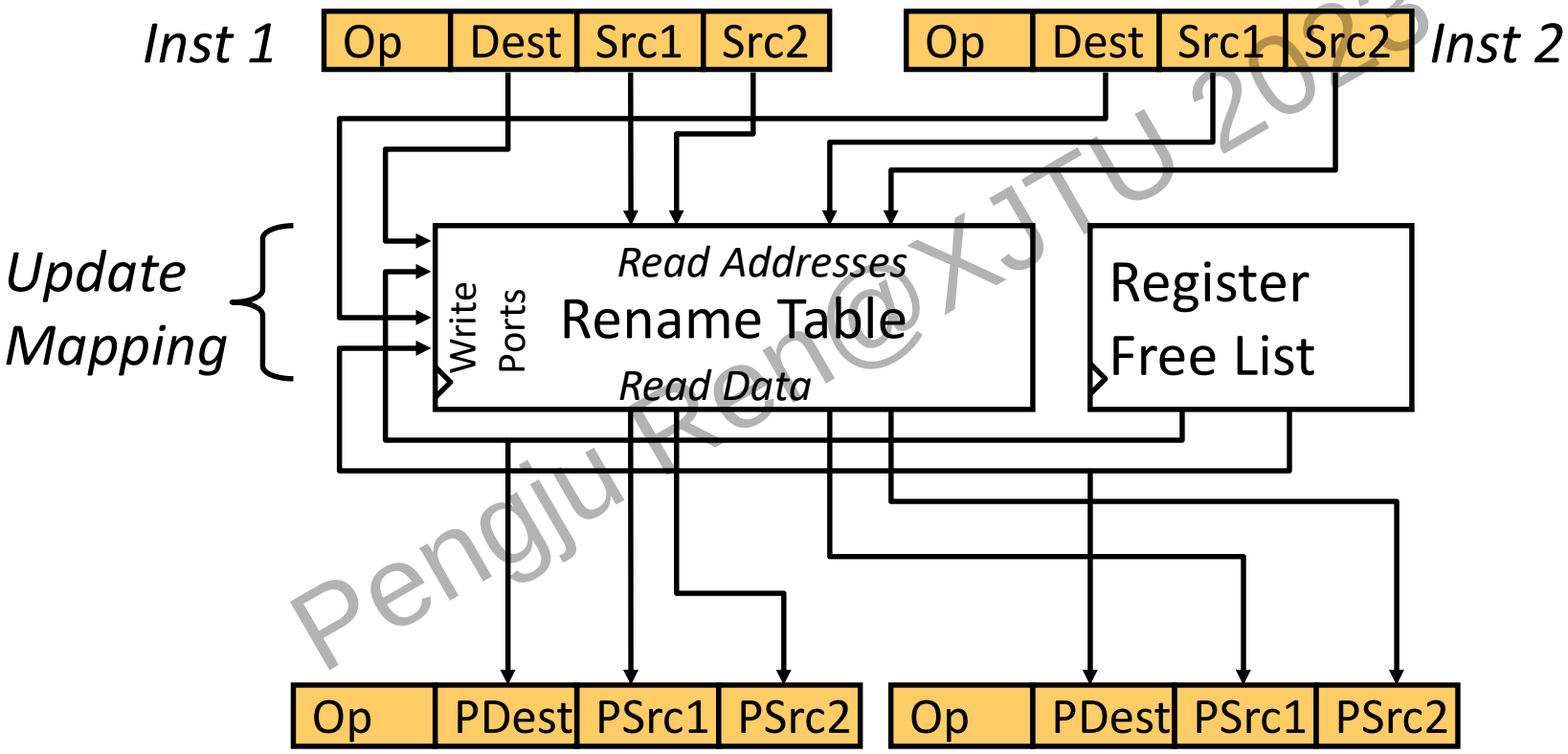
Register Renaming

- Adding more “Name” (registers/memory) removes dependence, but architecture namespaces is limited
  - Registers: Larger namespace requires more bits in instruction encoding. 32 registers = 5bits, 128 registers = 7 bits.

- Register Renaming: Change naming of registers **in hardware to eliminate WAW and WAR hazards**

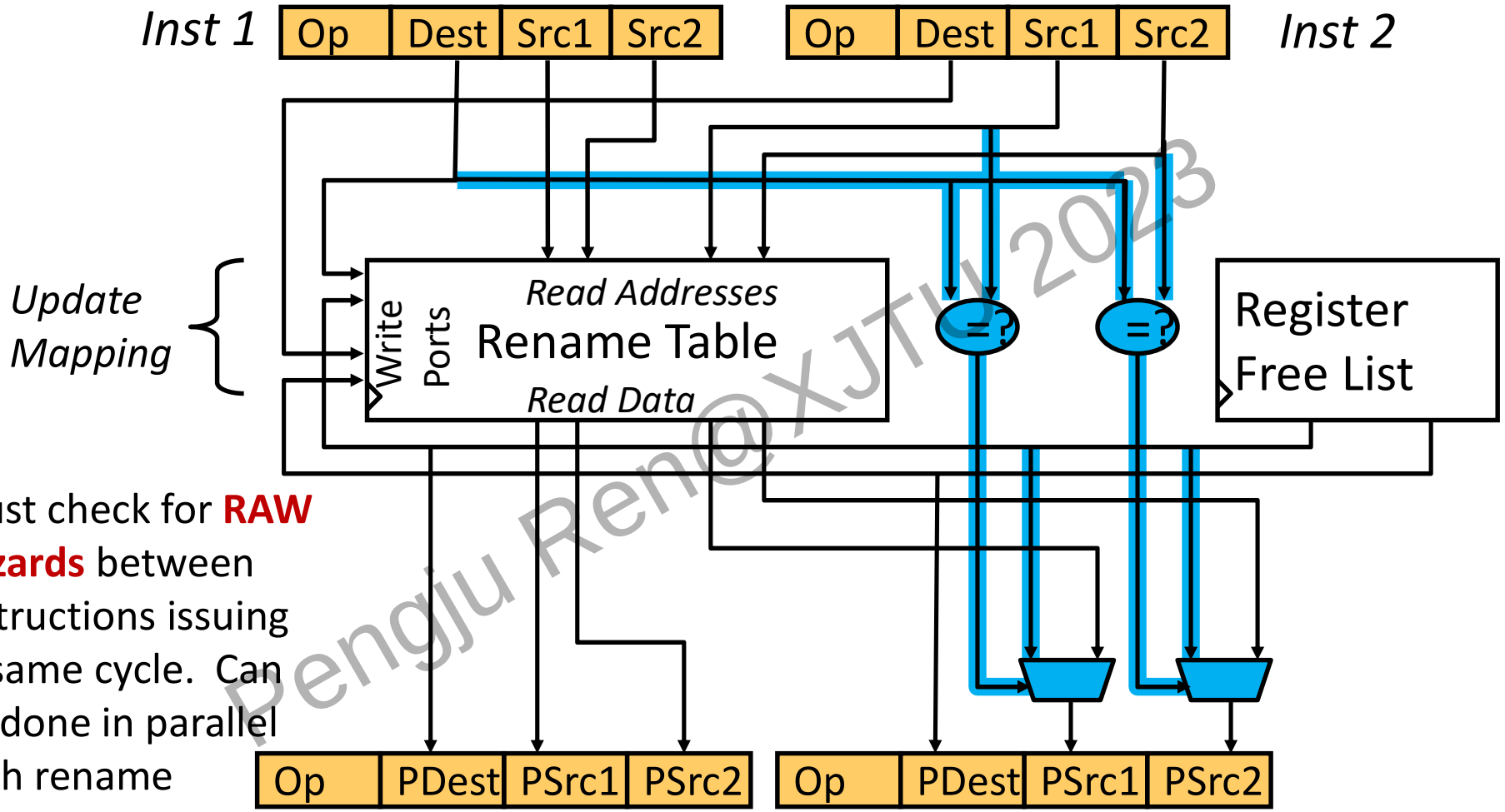
Superscalar Register Renaming

- During **decode**, instructions allocated new physical destination register
- Source operands renamed to physical register with newest value
- Execution unit only sees **physical register numbers**



Does this work?

Superscalar Register Renaming

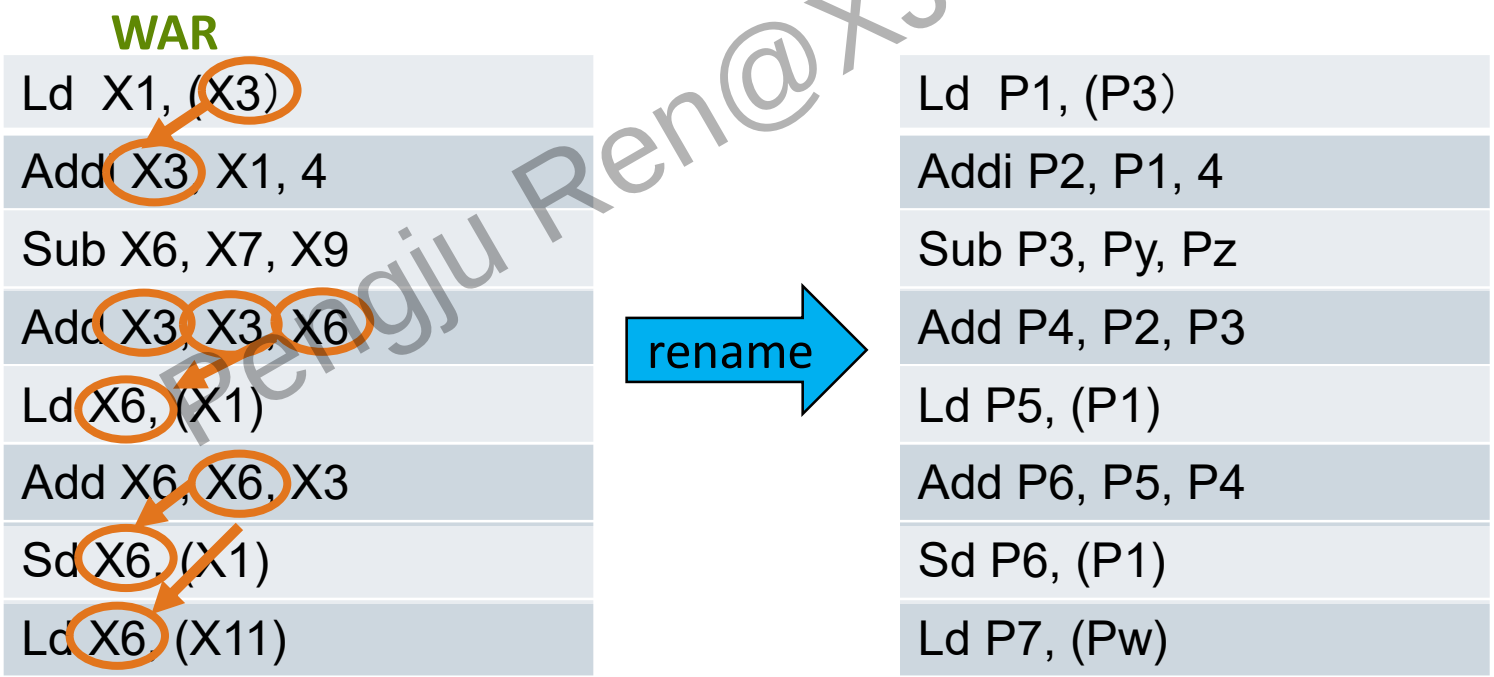


Must check for **RAW hazards** between instructions issuing in same cycle. Can be done in parallel with rename lookup.

MIPS R10K renames 4 serially-RAW-dependent insts/cycle

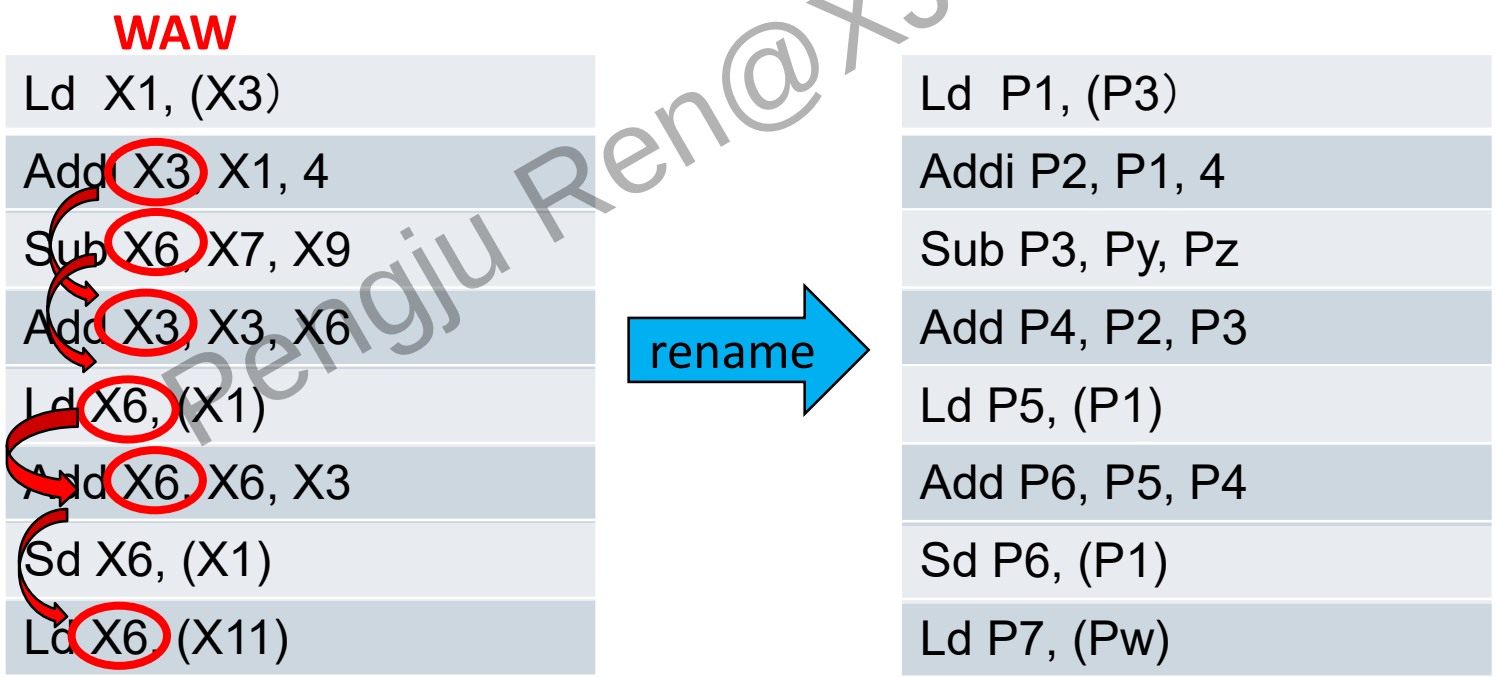
Register Renaming Overview

- 2 Schemes
  - Pointers in the Issue Queue/ReOrder Buffer
  - Values in the Issue Queue/ReOrder Buffer

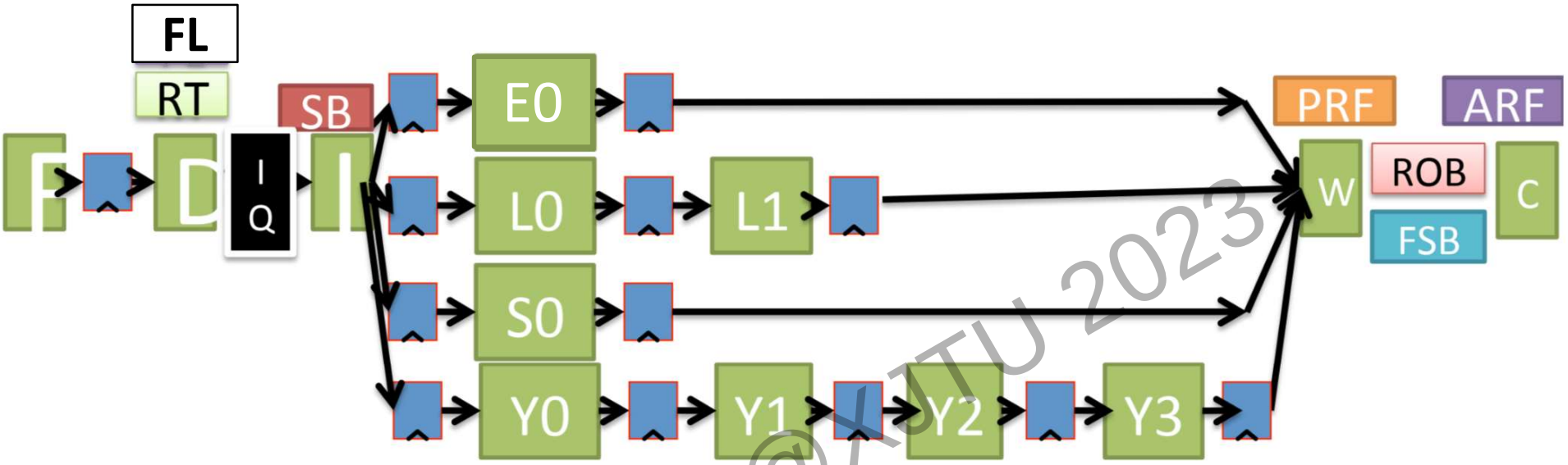


Register Renaming Overview

- 2 Schemes
  - Pointers in the Issue Queue/ReOrder Buffer
  - Values in the Issue Queue/ReOrder Buffer

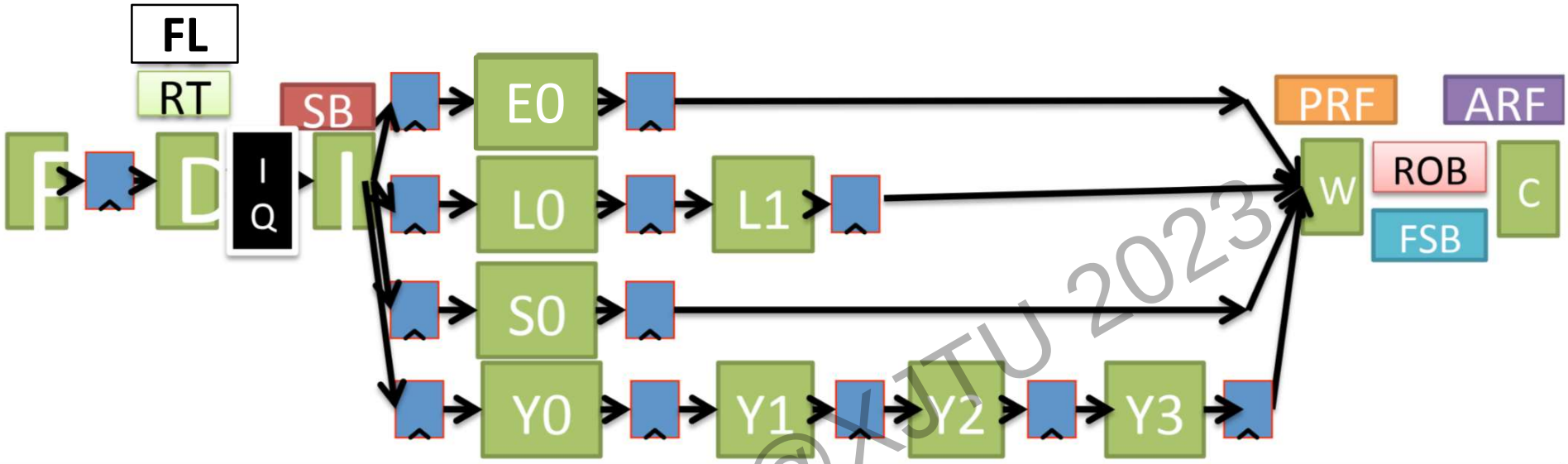


IO2I: Register Renaming with Pointers in IQ and ROB



- All data structures same as in IO2I Except:
  - Add two fields to ROB
  - Add Rename Table (RT) and Free List (FL) of registers
- Increase size of PRF to provide more register "Names"

IO2I: Register Renaming with Pointers in IQ and ROB



ARF					W
SB		R/W		W	
PRF		R		W	
ROB	R/W			W	R/W
FSB				W	R/W
IQ	W			W	
RT	R/W				W
FL	R/W				W

Modified Reorder Buffer (ROB)

State	S	ST	V	Preg	Areg	Ppreg
--						
P						
F						
P						
P						
F						
P						
P						
--						
--						

State: {Free, Pending, Finished}

S: Speculative

ST: Store bit

V: Destination is valid

Preg: Physical Register File Specifier

Areg: Architectural Register File Specifier

Ppreg: Previous Physical Register

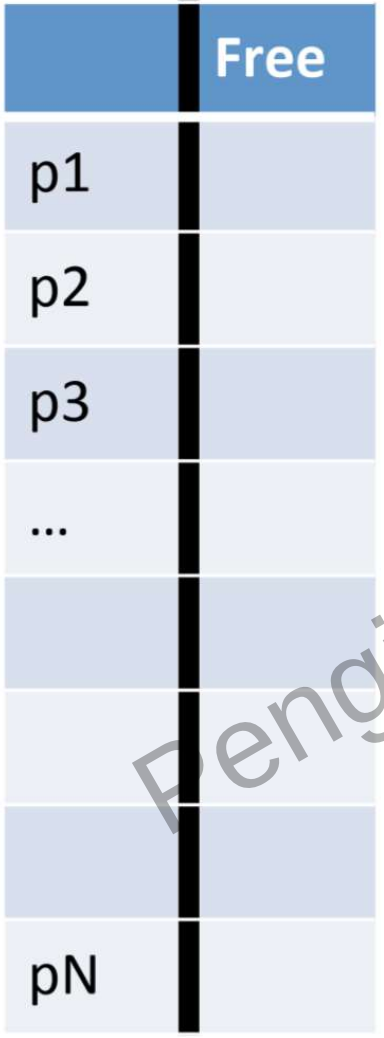
Renaming Table (RT)

	P	Preg
X1		
X2		
X3		
…		
X31		

P: Pending, Write to Destination in flight
Preg: Physical Register Architectural Register maps to.

the Renaming Tables are snapshotted to allow single-cycle recovery on a branch misprediction.

Free List (FL)



Free: Register is free for renaming

If Free == 0, physical register is in use and cannot be used for renaming

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

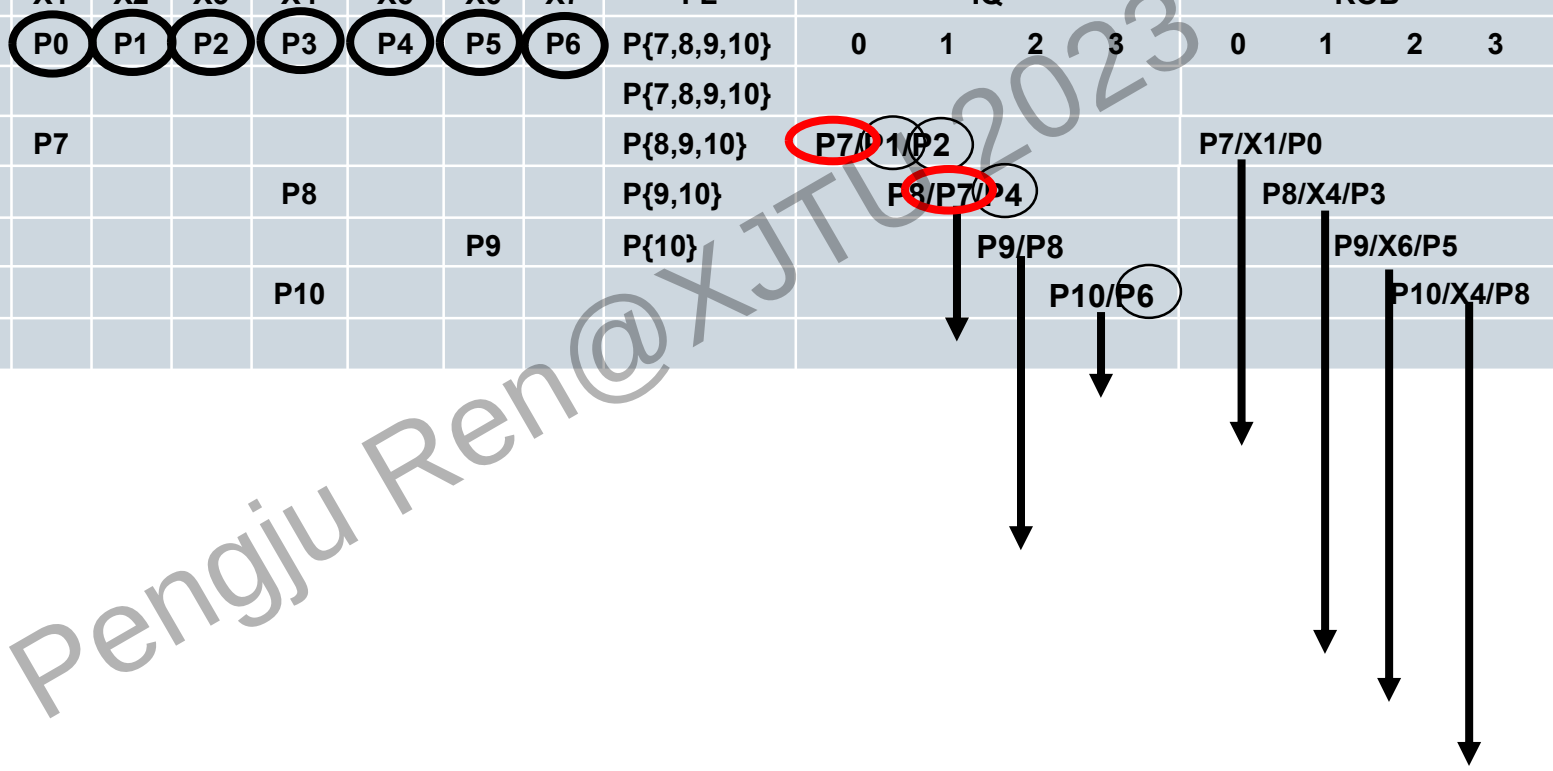
cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL	IQ				ROB			
0						P0	P1	P2	P3	P4	P5	P6	P{7,8,9,10}	0	1	2	3	0	1	2	3
1	0												P{7,8,9,10}								
2	1	0				P7							P{8,9,10}	P7/P1/P2				P7/X1/P0			
3	2								P8				P{9,10}		P8/P7/P4				P8/X4/P3		

Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P0	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											

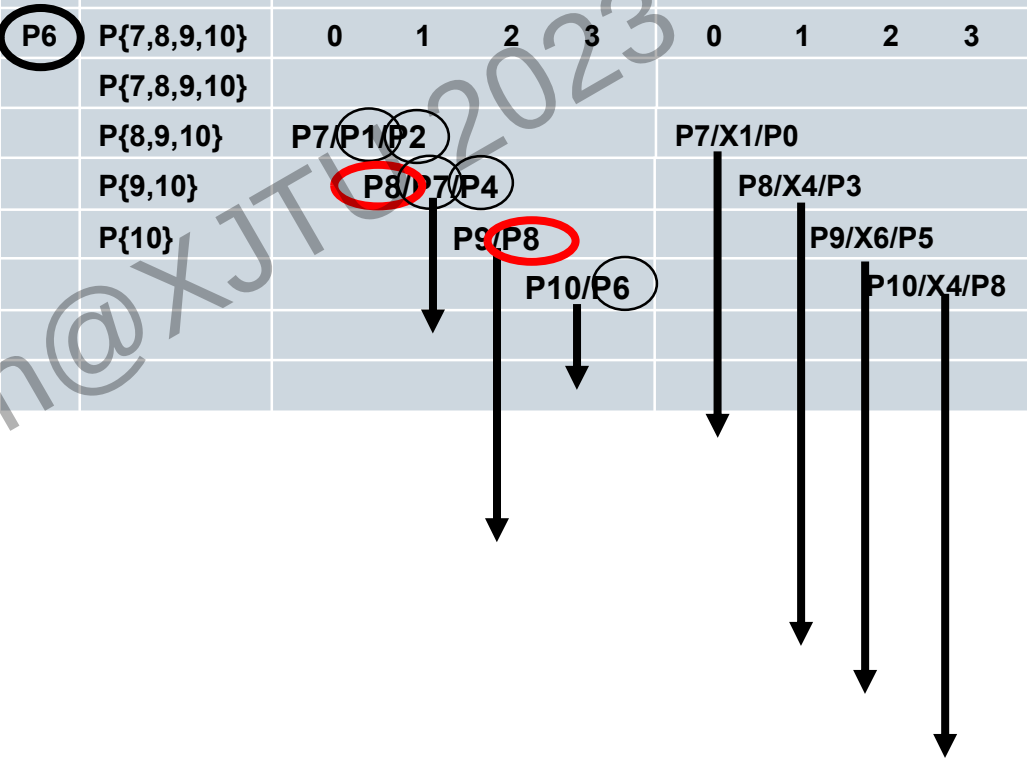


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P0	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										

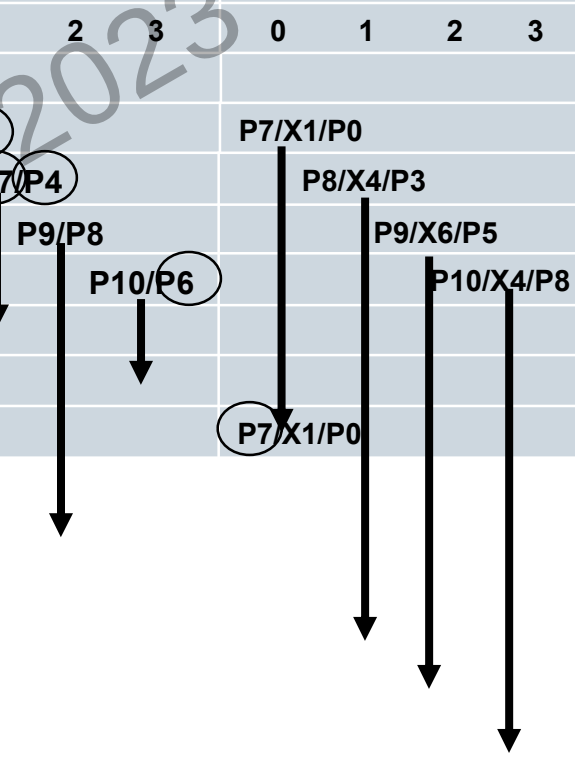


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P0	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									

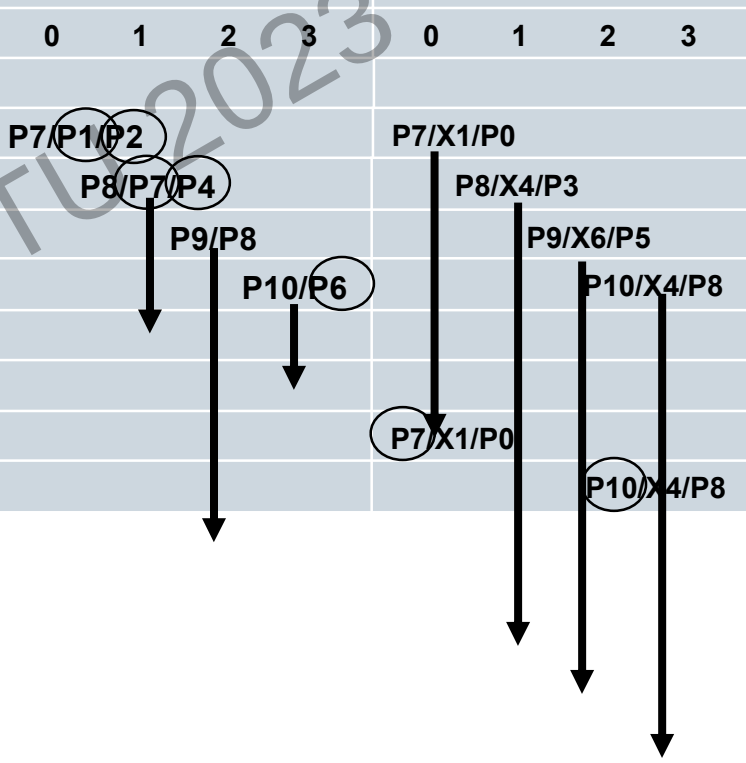


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0

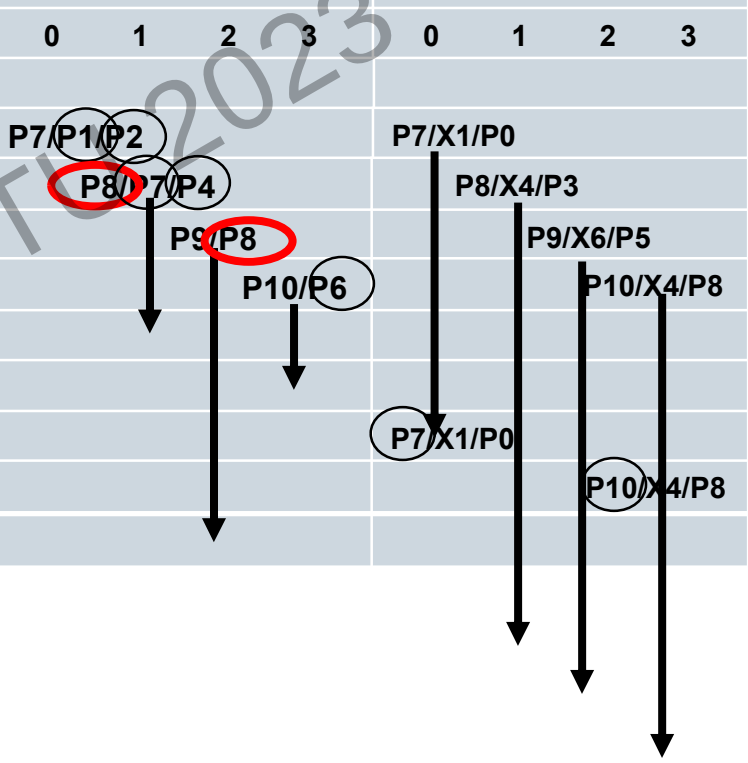


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0
10		2											P0

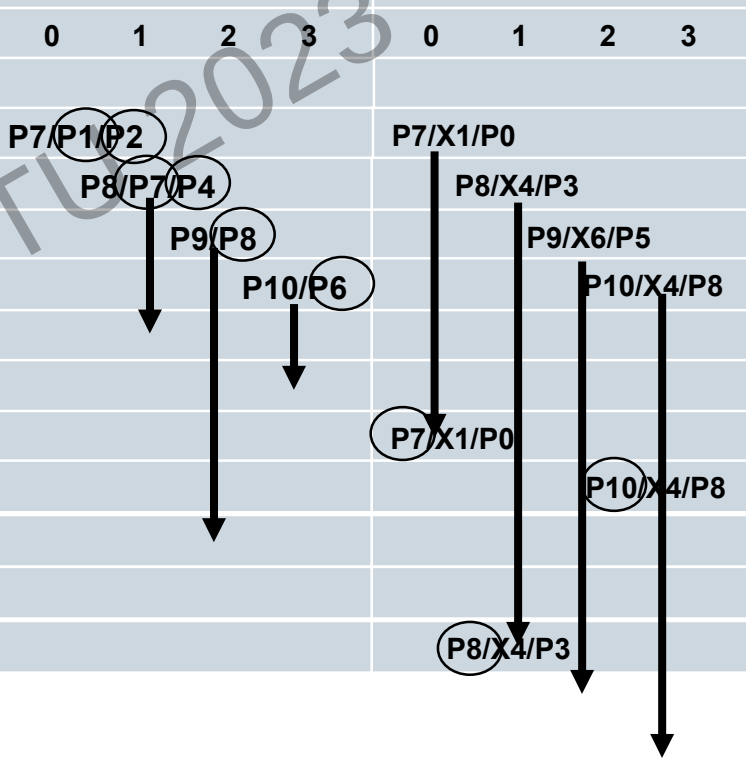


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P3	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0
10		2											P0
11			1										P0
12			2	1									P0

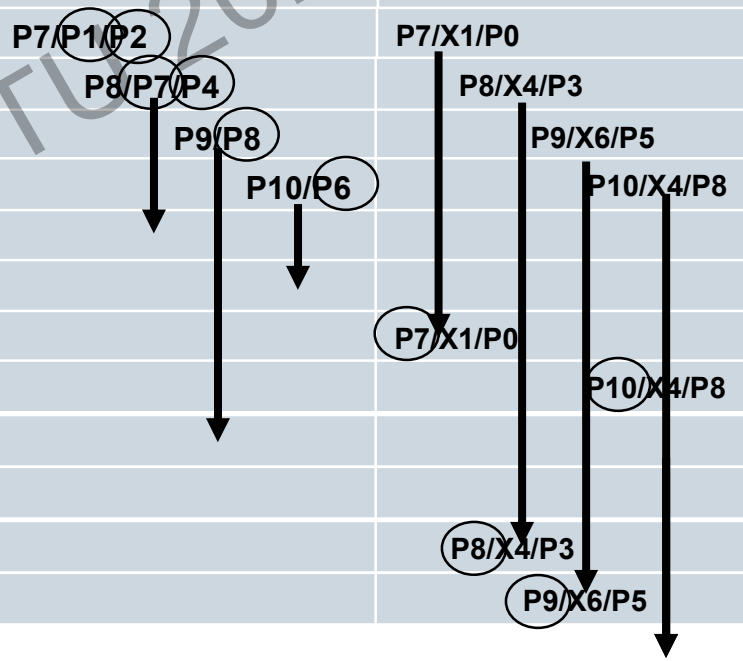


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P8	P4	P5	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0
10		2											P0
11			1										P0
12			2	1									P0
13				2									P{0,3}

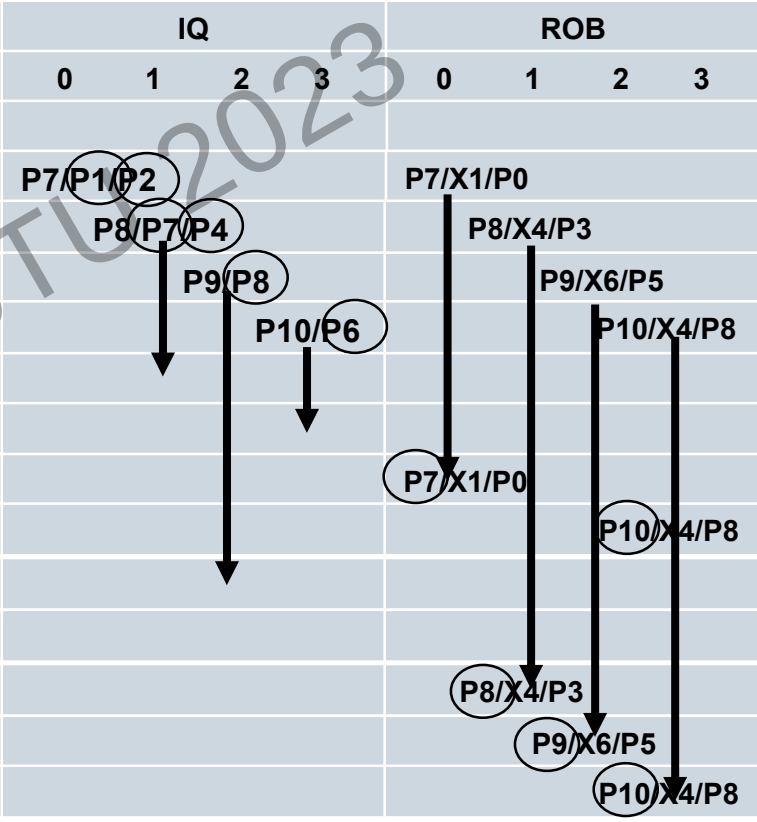


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P8	P4	P9	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0
10		2											P0
11			1										P0
12			2	1									P0
13				2									P{0,3}
14				3									P{0,3,5}

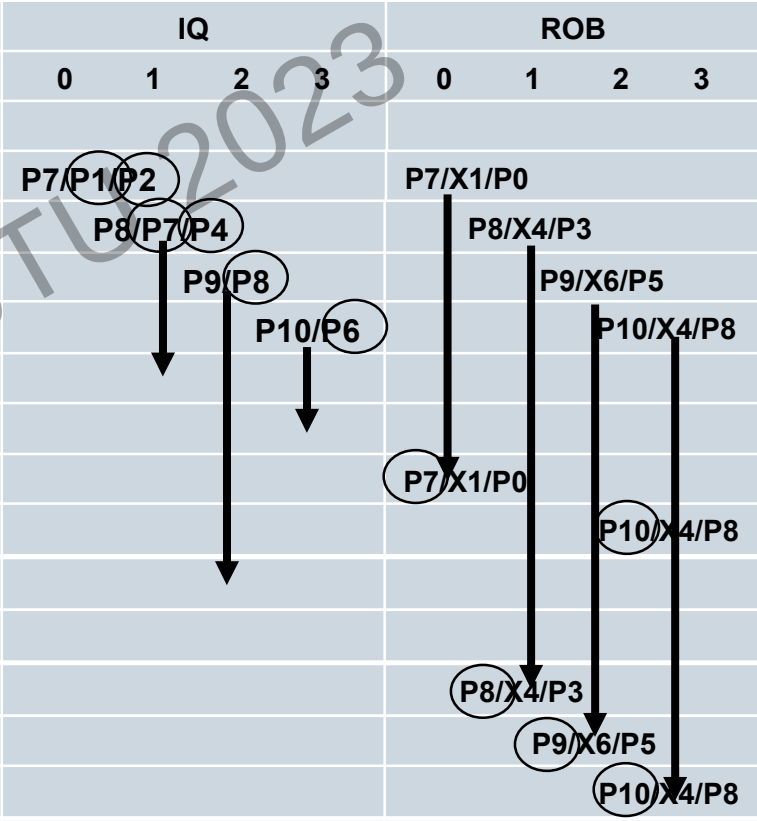


Dest/Src0/Src1 Preg/Areg/Ppreg

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7	FL
0						P7	P1	P2	P10	P4	P9	P6	P{7,8,9,10}
1	0												P{7,8,9,10}
2	1	0				P7							P{8,9,10}
3	2								P8				P{9,10}
4	3										P9		P{10}
5									P10				
6		1											
7		3	0										
8				0									
9			3										P0
10		2											P0
11			1										P0
12			2	1									P0
13				2									P{0,3}
14				3									P{0,3,5}
15													P{0,3,5,8}



Dest/Src0/Src1 Preg/Areg/Ppreg

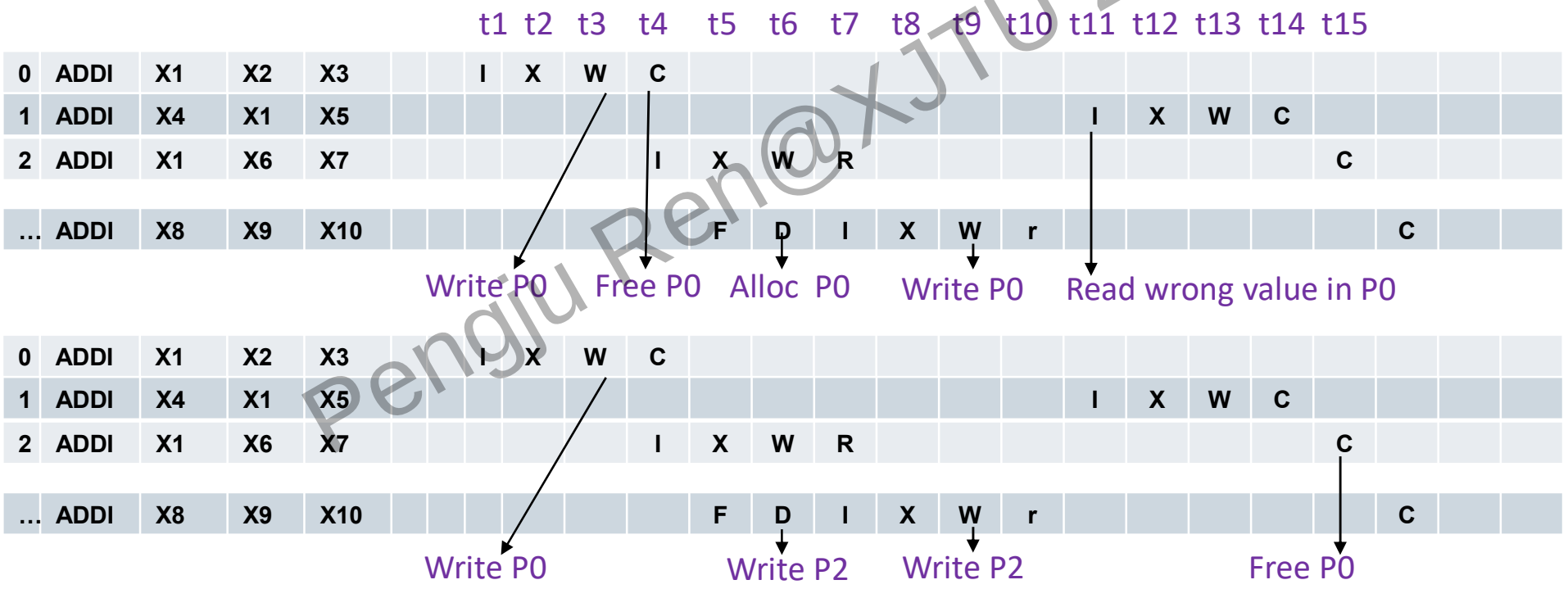
Freeing Physical Registers

1	ADDI	X1	X2	X3
2	ADDI	X4	X1	X5
3	ADDI	X1	X6	X7
...	ADDI	X8	X9	X10

<- Assume Arch.Reg X1 to Phys.Reg P0

<- Assume the value of Arch.Reg X5 is ready **at t11**

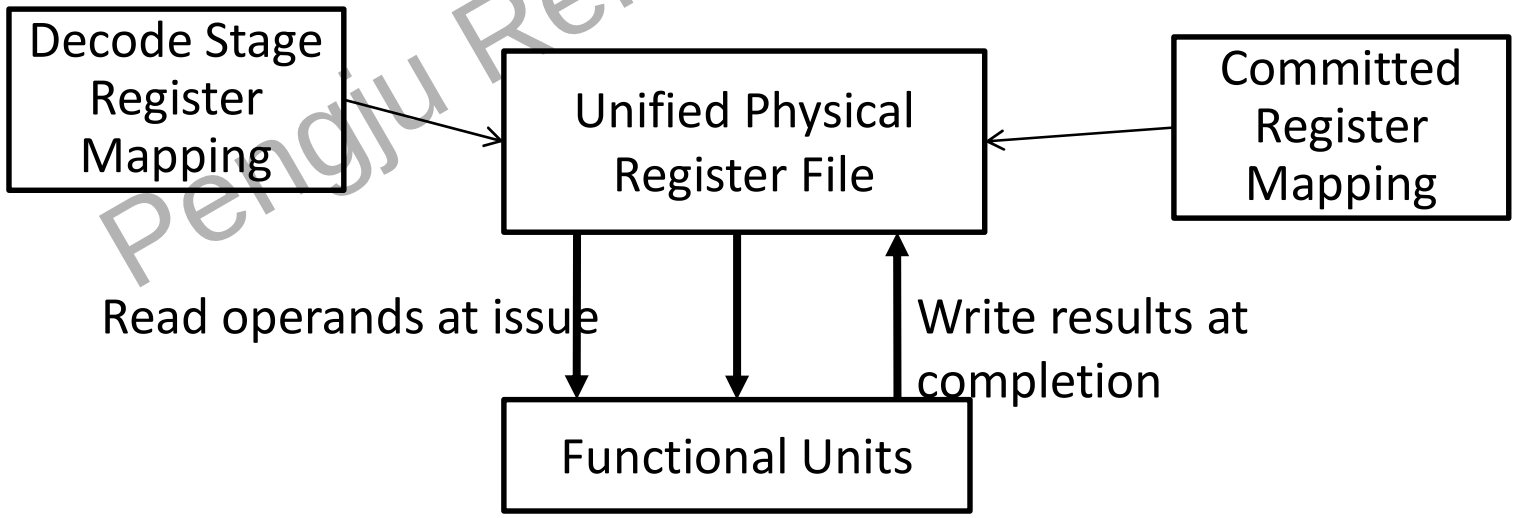
<- Next write of Arch.Reg X1 Mapped to Phys.Reg P1, Arch.Reg X6 is ready at t4



If Arch.Reg Ri mapped to Phys.Reg Pj, we can free Pj when the next instruction that writes Ri commits

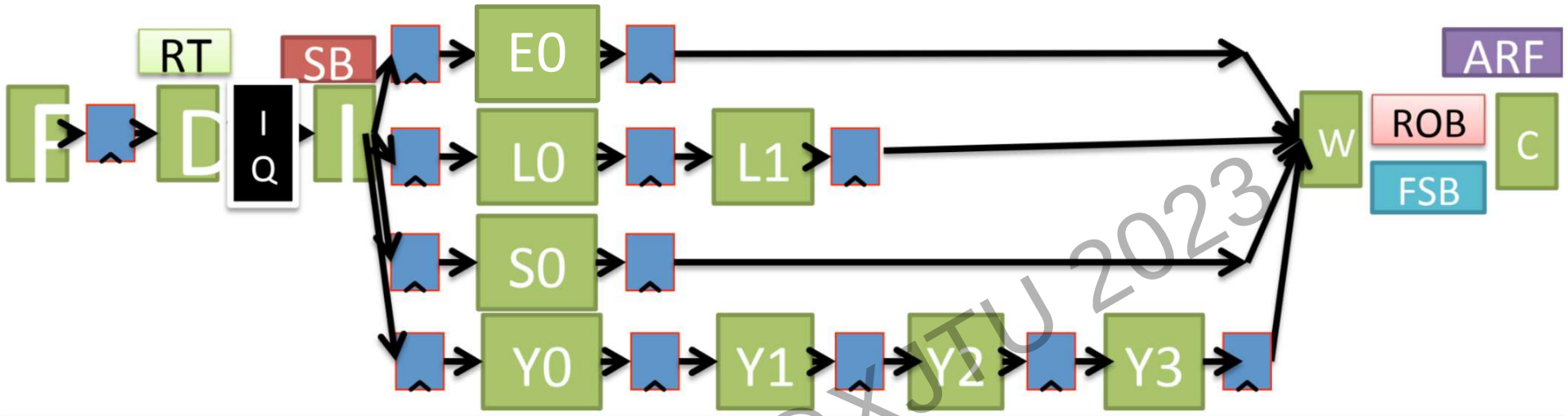
Unified Physical/Architecture Register File

- Rename all architectural registers into a single physical register file during decode, no register values read
- Functional units read and write from single unified register file holding committed and temporary registers in execute
- Commit only updates mapping of architectural register to physical register, **no data movement**
- Unified Physical/Architectural Register file can be small than separate



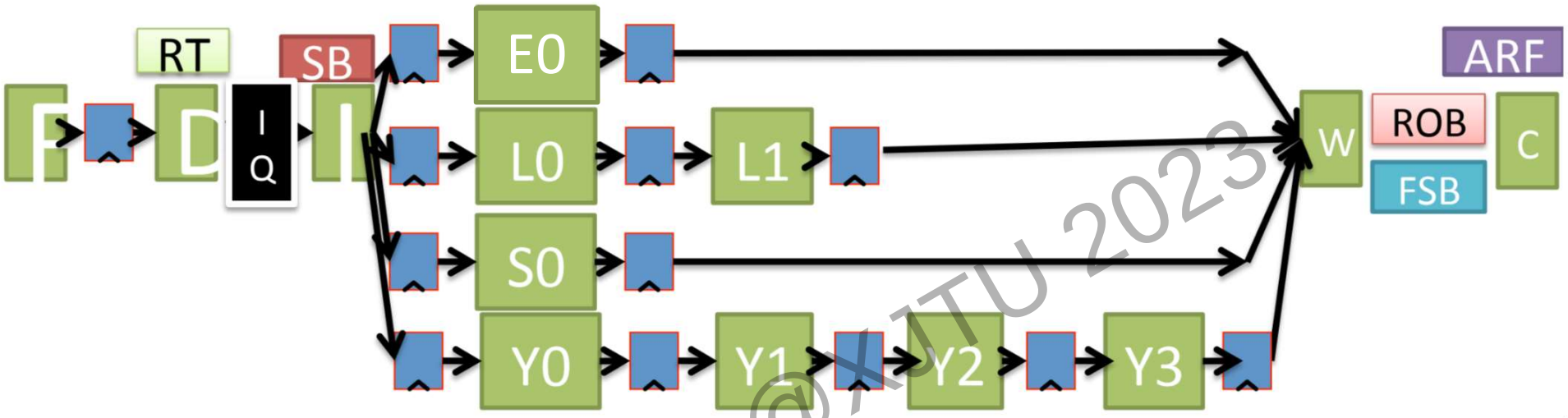
(MIPS R10K, Alpha 21264, Intel Pentium 4 & Sandy/Ivy Bridge)

IO2I: Register Renaming with Values in IQ and ROB



- All data structures same as previous Except:
  - Modified ROB (Values instead of Register Specifier)
  - Modified RT
  - Modified IQ
  - No FL
  - No PRF, values merged into ROB

IO2I: Register Renaming with Values in IQ and ROB



ARF	R				W
SB		R/W		W	
ROB	R/W			W	R/W
FSB				W	R/W
IQ	W			W	
RT	R/W				W

Modified Reorder Buffer (ROB)

State	S	ST	V	Value	Areg
--					
P					
F					
P					
P					
F					
P					
P					
--					
--					

ROB (Before modification)

Preg	Areg	Ppreg

State: {Free, Pending, Finished}

Areg: Architectural Register File Specifier

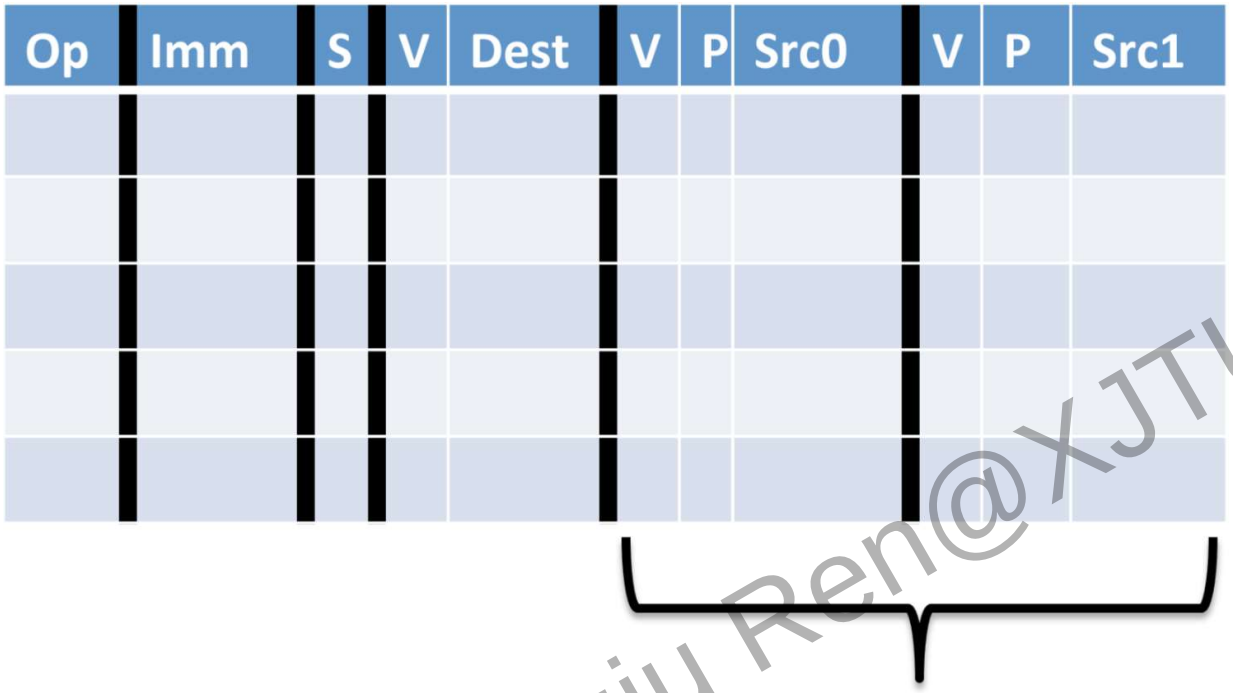
S: Speculative

ST: Store bit

V: Destination is valid

Value: Actual Register Value

Modified Issue Queue

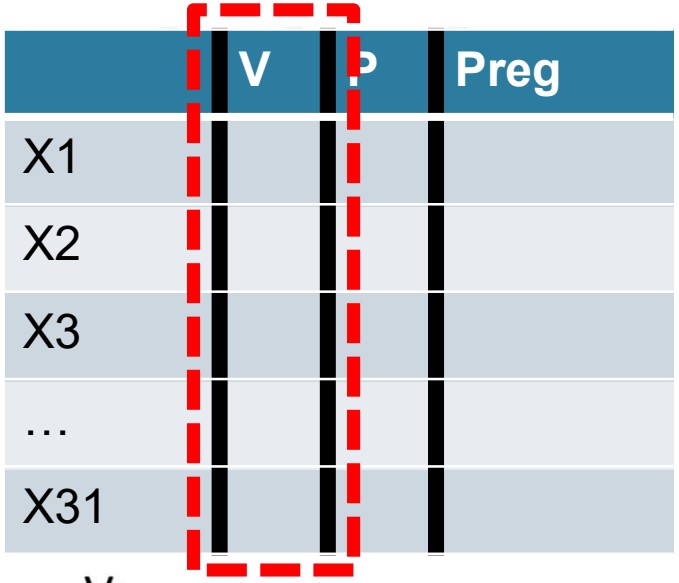


Op: Opcode
Imm.: Immediate
S: Speculative Bit
V: Valid (Instruction has corresponding Src/Dest)
P: Pending (Waiting on operands to be produced)

If Pending, Source Field contains index into ROB. Like a Preg identifier

On Commit, Source Field contains value

Modified Renamed Table (RT)



V: Valid Bit
P: Pending, Write to Destination in flight
Preg: Index into ROB

V:
If V == 0:
Value in ARF is up to date
If V == 1:
Value is in-flight or in ROB

P:
If P == 0:
Value is in ROB
if P == 1:
Value is in flight

					0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	MUL	X1	X2	X3	F	D	I	Y0	Y1	Y2	Y3	W	C											
1	MUL	X4	X1	X5		F	D	i			I	Y0	Y1	Y2	Y3	W	C							
2	ADDI	X6	X4	1			F	D	i						I	E0	W	C						
3	ADDI	X4	X7	1				F	D	i		I	E0	W	r				C					

Renaming Table

cy	D	I	W	C		X1	X2	X3	X4	X5	X6	X7		IQ				ROB			
0														0	1	2	3	0	1	2	3
1	0																				
2	1	0				P0								P0/X2/X3				P0/X1			
3	2								P1						P1/P0/X5				P1/X4		
4	3										P2					P2/P1				P2/X6	
5									P3								P3/X7				P3/X4
6		1																			
7			0																		
8		3		0														P0/X1			
9			3																		
10		2							P3											P3/X4	
11			1																		
12			2	1															P1/X4		
13				2																P2/X6	
14				3																	
15																					

Agenda

SuperScalar Intro
Out-of-Order Processor (OOO)
Speculation and Branches
Register Renaming
Memory Disambiguation

Memory Disambiguation

st X1, 0(X2)

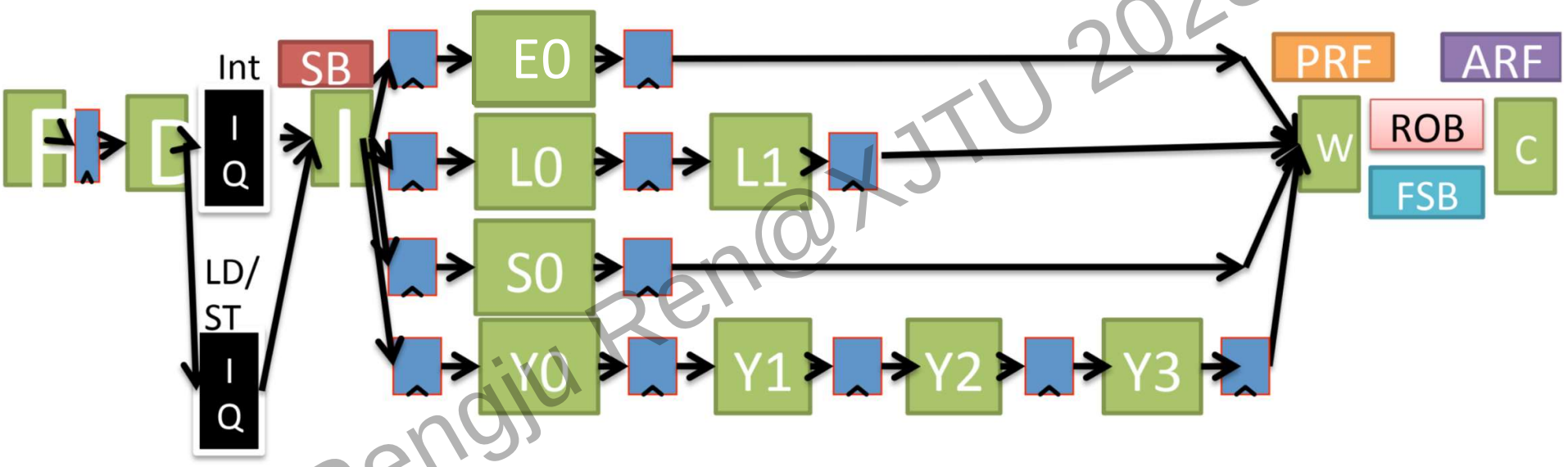
ld X3, 0(X4)

When can we execute the load ?

In-Order Memory Queue

- Execute all loads and stores in program order
  - Load and store cannot leave IQ for execution until all previous loads and stores have completed execution
- Can still execute loads and stores speculatively, and out-of-order with respect to other (non-memory) instructions
- Need a structure to handle memory ordering …

IO2I: With In-Order LD/ST IQ



Conservative OOO Load Execution

(MIPS R10K, 16 entry address queue)

st X1, 0(X2)

ld X3, 0(X4)

- Split execution of store instruction into two phases: **address calculation** and **data write**
- Can execute load therefore store, if addresses known and X4 != X2
- Each load address compared with addresses of all previous uncommitted stores (can use partial conservative check i.e., bottom 12 bits of address)
- *Don't execute load if previous store address not known*

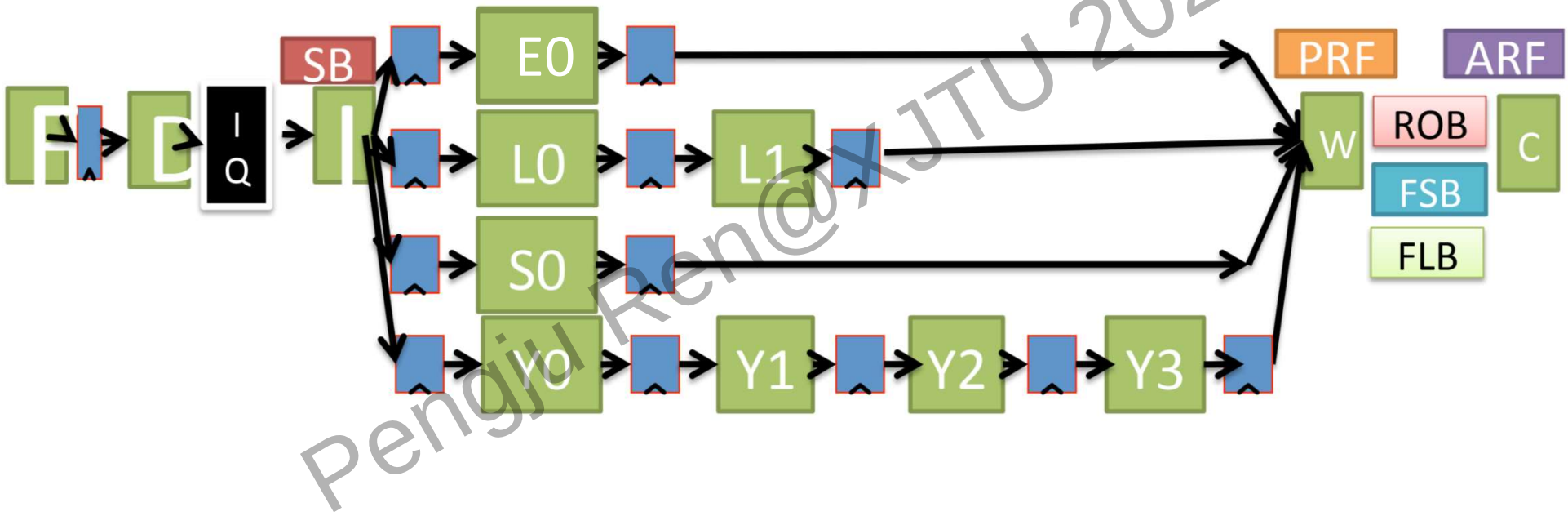
Address Speculation

st X1, 0(X2)

ld X3, 0(X4)

- Guess that X4 != X2
- Execute load before store address known
- Need to hold all completed but uncommitted load/store addresses in program order
- *If subsequently find X4==X2, squash load and **all** following instructions*
  - Large penalty for inaccurate address speculation

IO2I: With OOO Load and Stores



Memory Dependence Prediction

(Alpha 21264)

st X1, 0(X2)

ld X3, 0(X4)

- Guess that X4!=X2 and execute load before store
- If later find X4 == X2, squash load and all following instructions, but mark load instruction as *store-wait*
- Subsequent executions of the same load instruction will wait for all previous stores to complete
- Periodically clear *store-wait* bits

Next Lecture : Memory and Cache

(Memory System)

Acknowledgements

- Some slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - David Patterson (UCB)
  - David Wentzlaff (Princeton University)

- MIT material derived from course 6.823
- UCB material derived from course CS252 and CS 61C