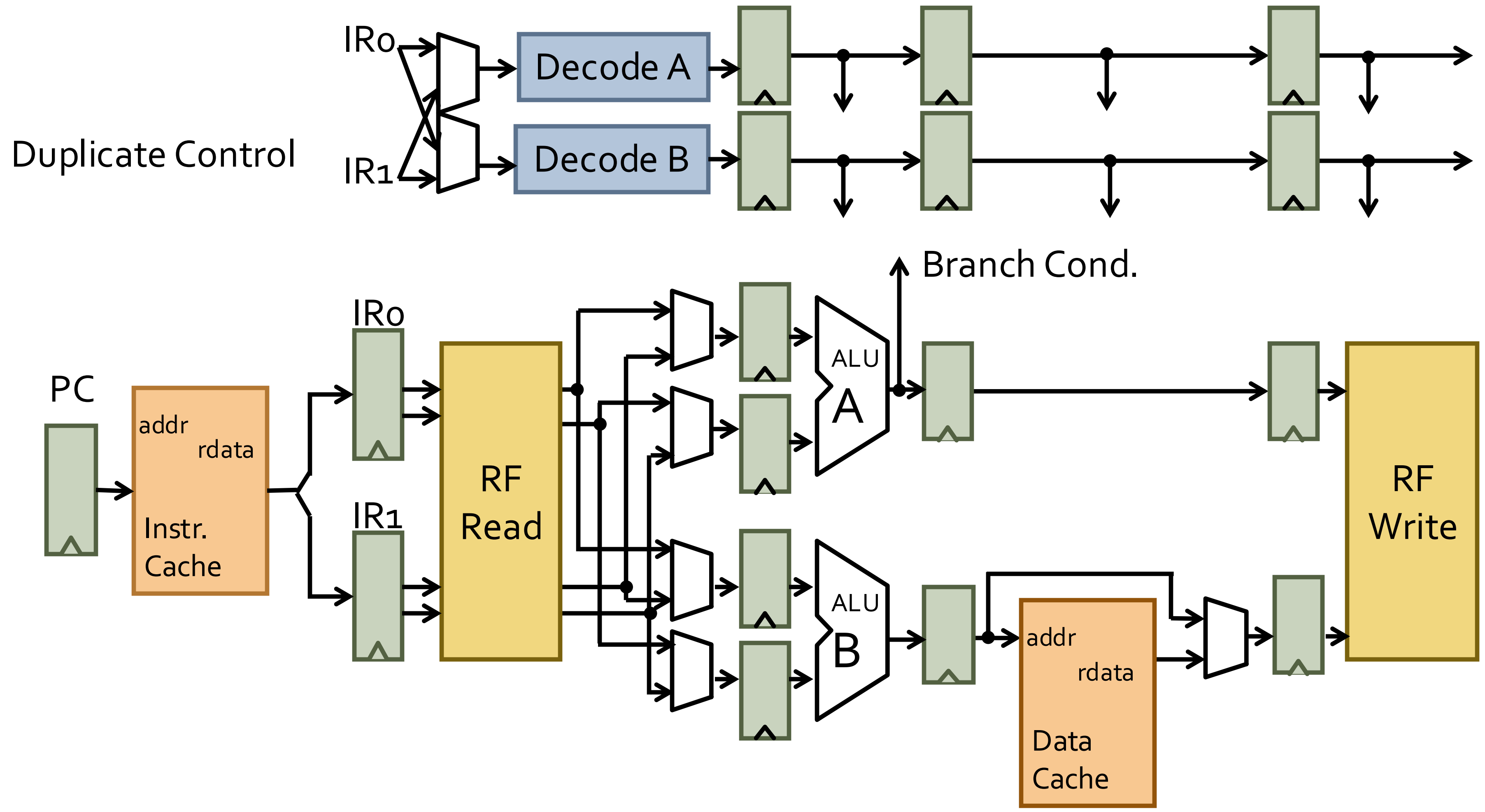
Computer Architecture Superscalar II

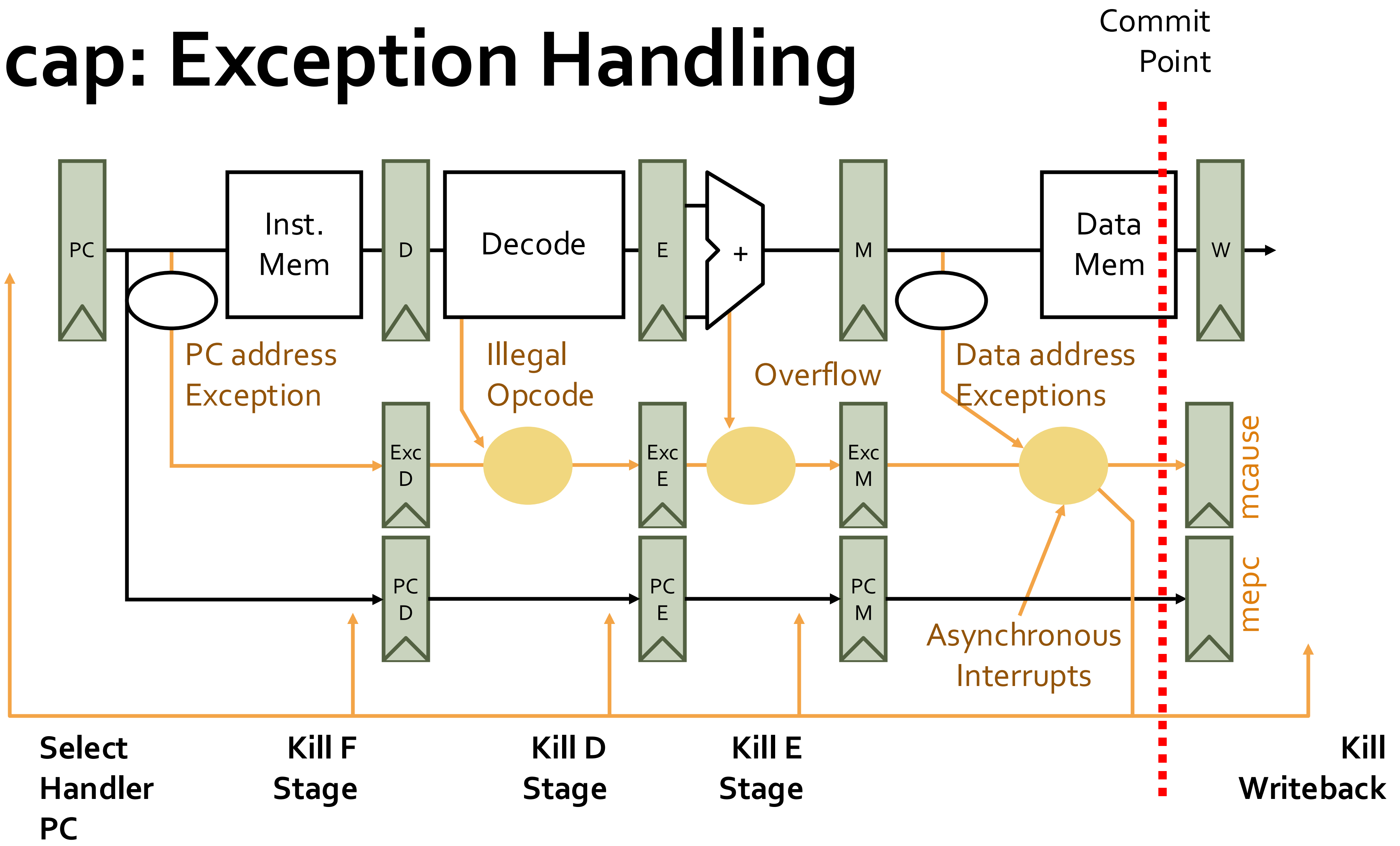
Ting-Jung Chang

NYCU CS

Recap: 2-Way In-Order Superscalar



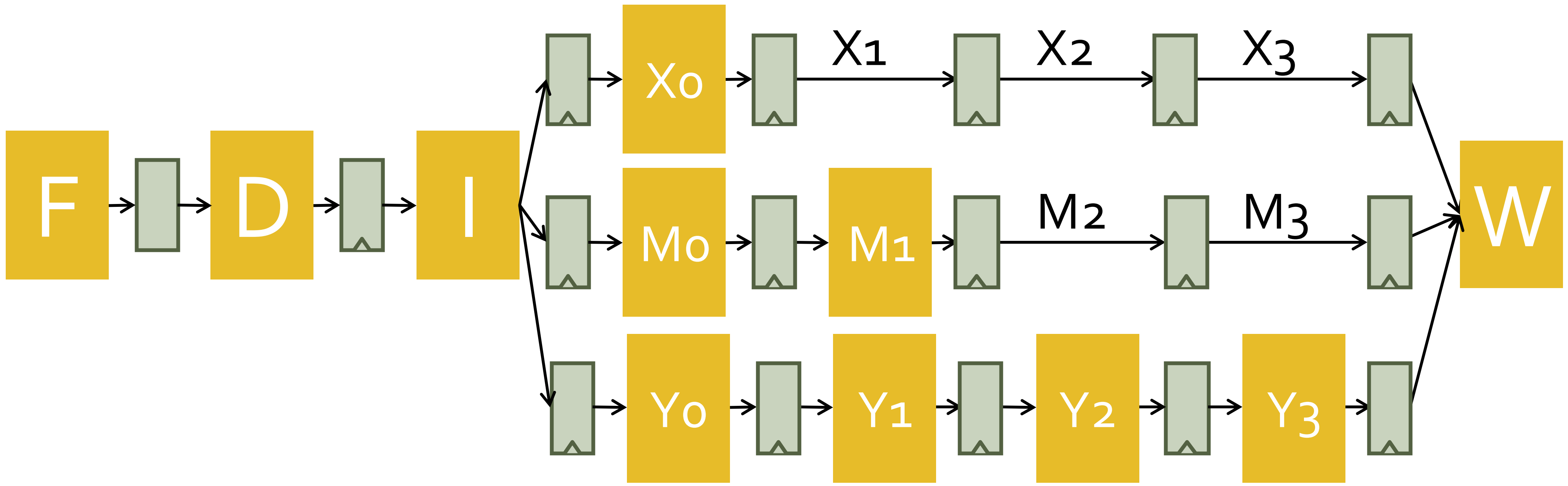
Recap: Exception Handling



Recap: Out-Of-Order (OOO)

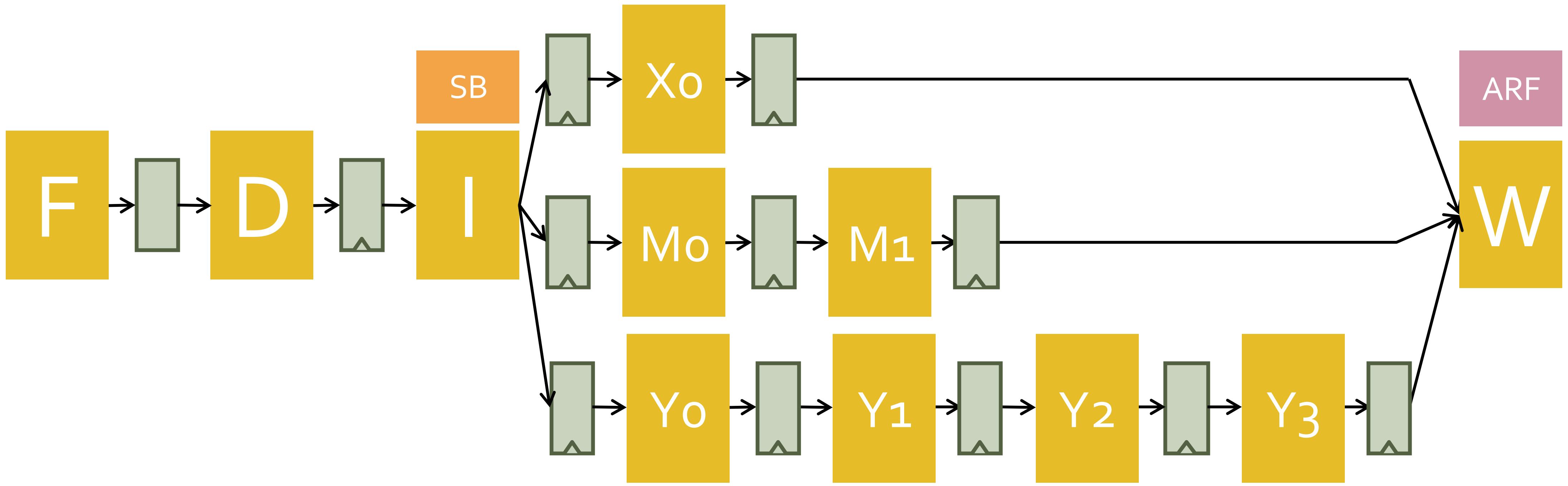
Name	Frontend	Issue	Writeback	Commit	
I4	IO	IO	IO	IO	Fixed Length Pipelines Scoreboard
I2O2	IO	IO	OOO	OOO	Scoreboard
I2OI	IO	IO	OOO	IO	Scoreboard, Reorder Buffer, and Store Buffer
IO3	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

Recap: I4



ARF	R	W
SB	R/W	W

Recap: I2O2



ARF	R	W
SB	R/W	W

Early Commit Point?

```
0 mul   x1, x2, x3 F  D  I  Y0 Y1 Y2 Y3 /
1 addi  x11,x10,1     F  D  I  X0 W     /
2 mul   x5, x1, x4       F  D  I  I  I  /
3 mul   x7, x5, x6          F  D  D  D  /
4 addi  x12,x11,1              F  F  F  /
5 addi  x13,x12,1                       /
6 addi  x14,x12,2
```

- Limits certain types of exceptions

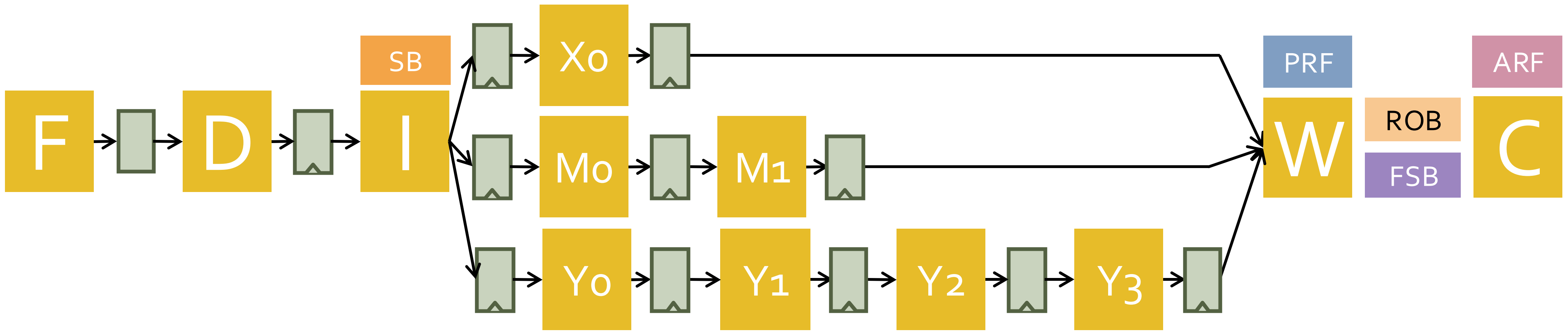
Course Admin

- PS1 solutions are out
- PS2 is out
- Lab1 due next week
  - Double check before submission

Agenda

- Out-of-Order Processors
- Speculation and Branches
- Register Renaming
- Memory Disambiguation

I2OI: In-order Frontend/Issue, Out-of-order Writeback, 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				
--				
--				

Next instruction allocates here in D

Tail of ROB

Speculative because branch is in flight

Instruction wrote ROB out of order

Head of ROB

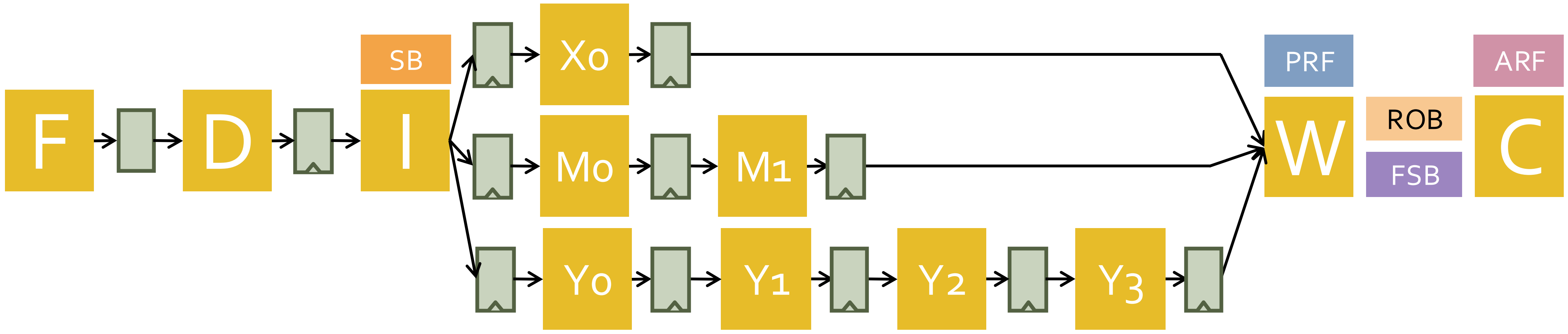
Commit stage is waiting for Head of ROB to be finished

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

Finished Store Buffer (FSB)

V	Op	Addr	Data
--			

- Only need one entry if we only support one memory instruction in flight 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.

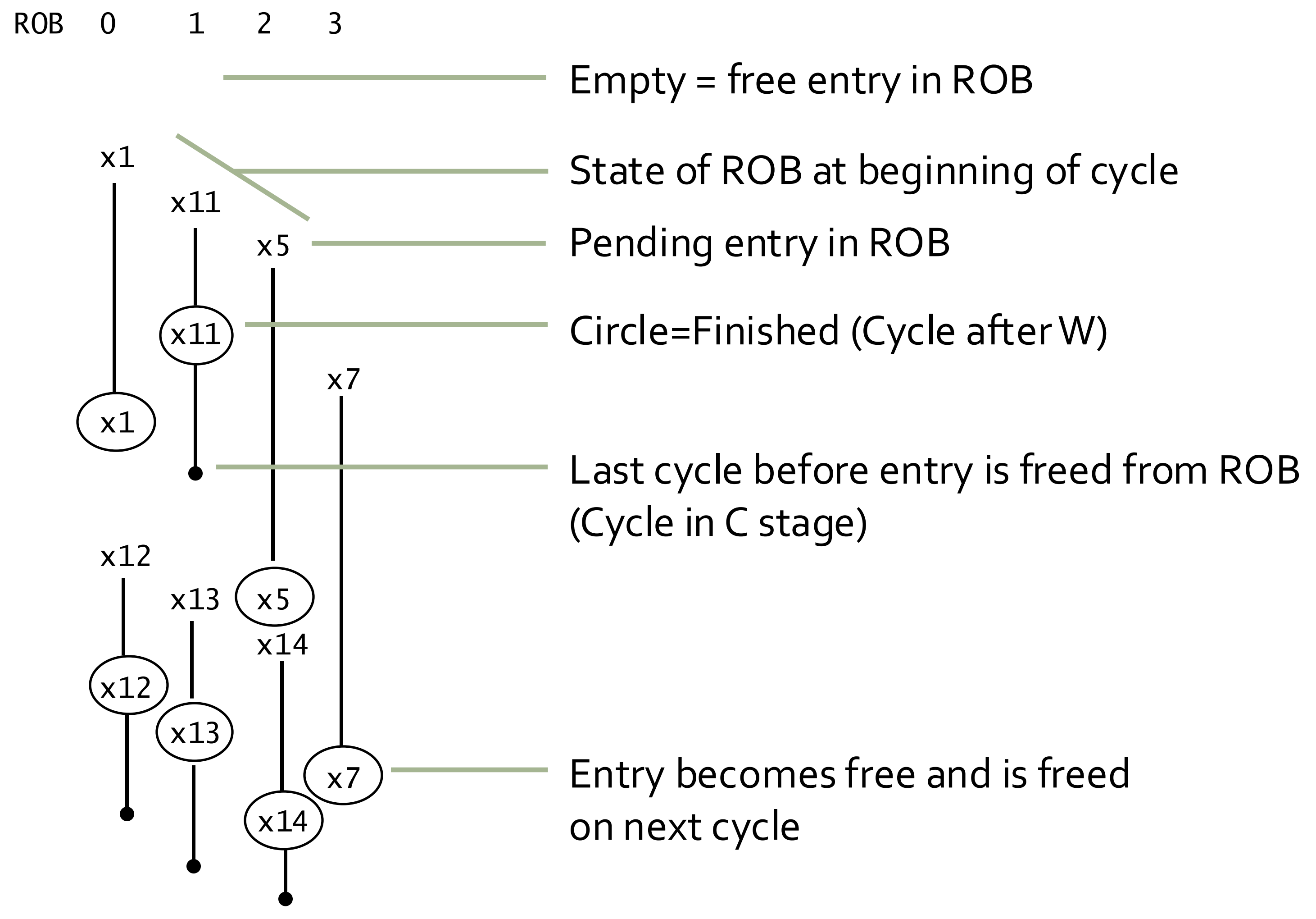


```
                        0   1   2   3   4   5   6   7   8   9   10  11  12  13  14  15  16  17  18  19

0 mul   x1, x2, x3
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
```

```
                      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  X0 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  X0 W  r        C
5 addi  x13,x12,1                           F  F  F  F  D  I  X0 W  r        C
6 addi  x14,x12,2                                       F  D  I  I  X0 W  r        C
```

Cyc	D	I
0		
1	0	
2	1	0
3	2	1
4		
5		
6	3	2
7		
8		
9		
10	4	3
11	5	4
12	6	5
13		
14		6
15		
16		
17		
18		
19		


ROB 0 1 2 3
x1
x11
x5
x11
x7
x1
x12
x13
x5
x14
x12
x13
x7
x14
Empty = free entry in ROB
State of ROB at beginning of cycle
Pending entry in ROB
Circle=Finished (Cycle after W)
Last cycle before entry is freed from ROB (Cycle in C stage)
Entry becomes free and is freed on next cycle

What if First Instruction Causes an Exception?

```
0 mul   x1, x2, x3  F  D  I  Y0 Y1 Y2 Y3 W  /
1 addi  x11,x10,1      F  D  I  X0 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  /
                                            F  D  I. . .
```

What About Branches?

Option 1

```
0 beq   x1, x0, target   F  D  I  X0 W  C
1 addi  x11,x10,1           F  D  I  -
2 add   x5, x1, x4             F  D  -
3 add   x7, x5, x6                F  -
T addi  x12,x11,1                    F  D  I . . .
```

Squash instructions earlier. Has more complexity. ROB needs many ports.

Option 2

```
0 beq   x1, x0, target   F  D  I  X0 W  C
1 addi  x11,x10,1           F  D  I  X0 /
2 add   x5, x1, x4             F  D  I  /
3 add   x7, x5, x6                F  D  /
T addi  x12,x11,1                    F  D  I . . .
```

Squash instructions in ROB when Branch commits.

Option 3

```
0 beq   x1, x0, target   F  D  I  X0 W  C
1 addi  x11,x10,1           F  D  I  X0 W  /
2 add   x5, x1, x4             F  D  I  X0 W  /
3 add   x7, x5, x6                F  D  I  X0 W  /
T addi  x12,x11,1                    F  D  I  X0 W  C
```

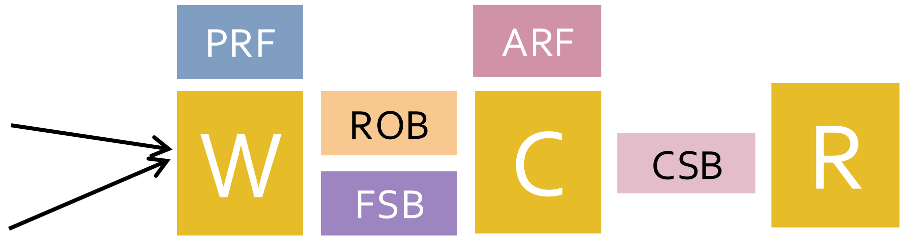
Wait for speculative instructions to reach the Commit stage and squash in Commit stage

What About Branches?

- Three possible designs with decreasing complexity based on when to squash speculative instructions and de-allocate ROB entry:
    1. As soon as branch resolves
    2. When branch commits
    3. When speculative instructions reach commit

- Base design only allows one branch at a time. Second branch stalls in decode. Can add more bits to track multiple in-flight branches.

Avoiding Stalling Commit on Store Miss



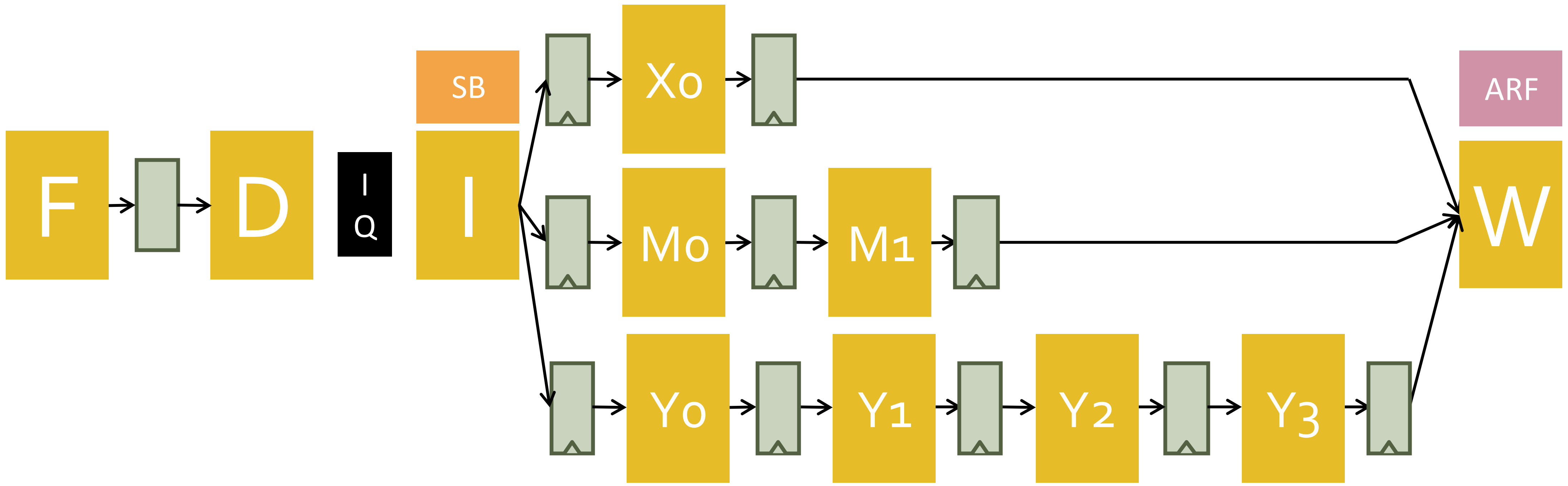
CSB=Committed Store Buffer

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

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



ARF		R	W
SB		R/W	W
IQ	W	R/W	W

Issue Queue (IQ)

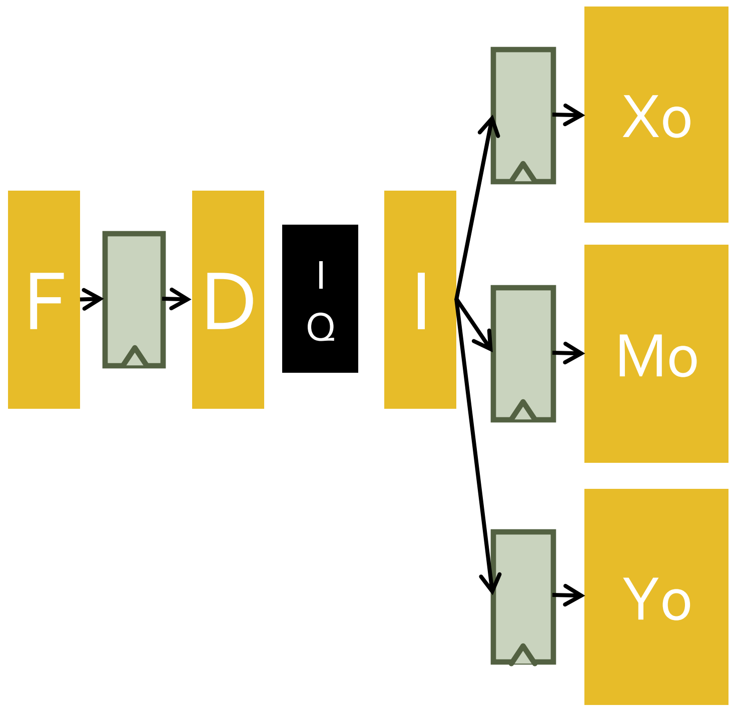
Op	Imm	S	V	Dest	V	P	Src0	V	P	Src1

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

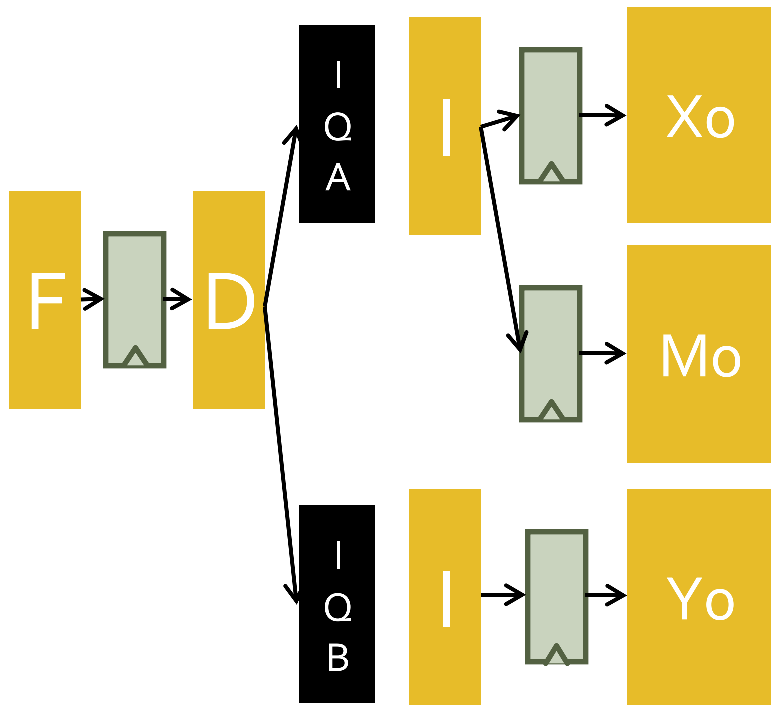
Instruction Ready = (!Vsrc0 || !Psrc0) && (!Vsrc1 || !Psrc1) && no structural hazards

- For high performance, factor in bypassing

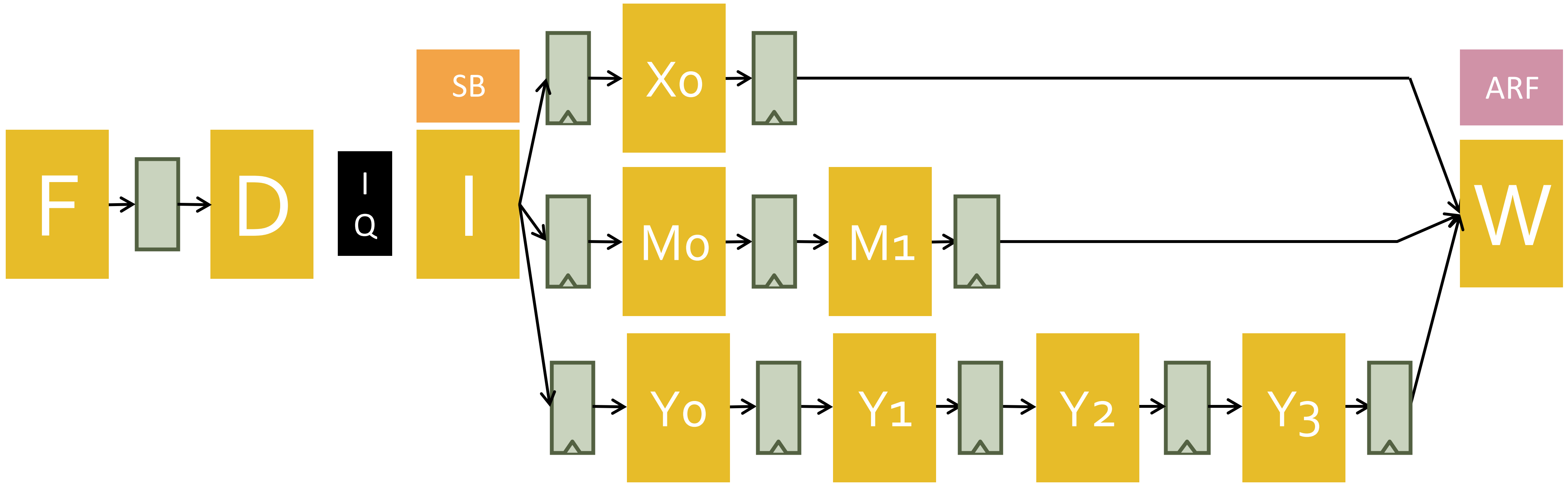
Centralized vs. Distributed Issue Queue



Centralized



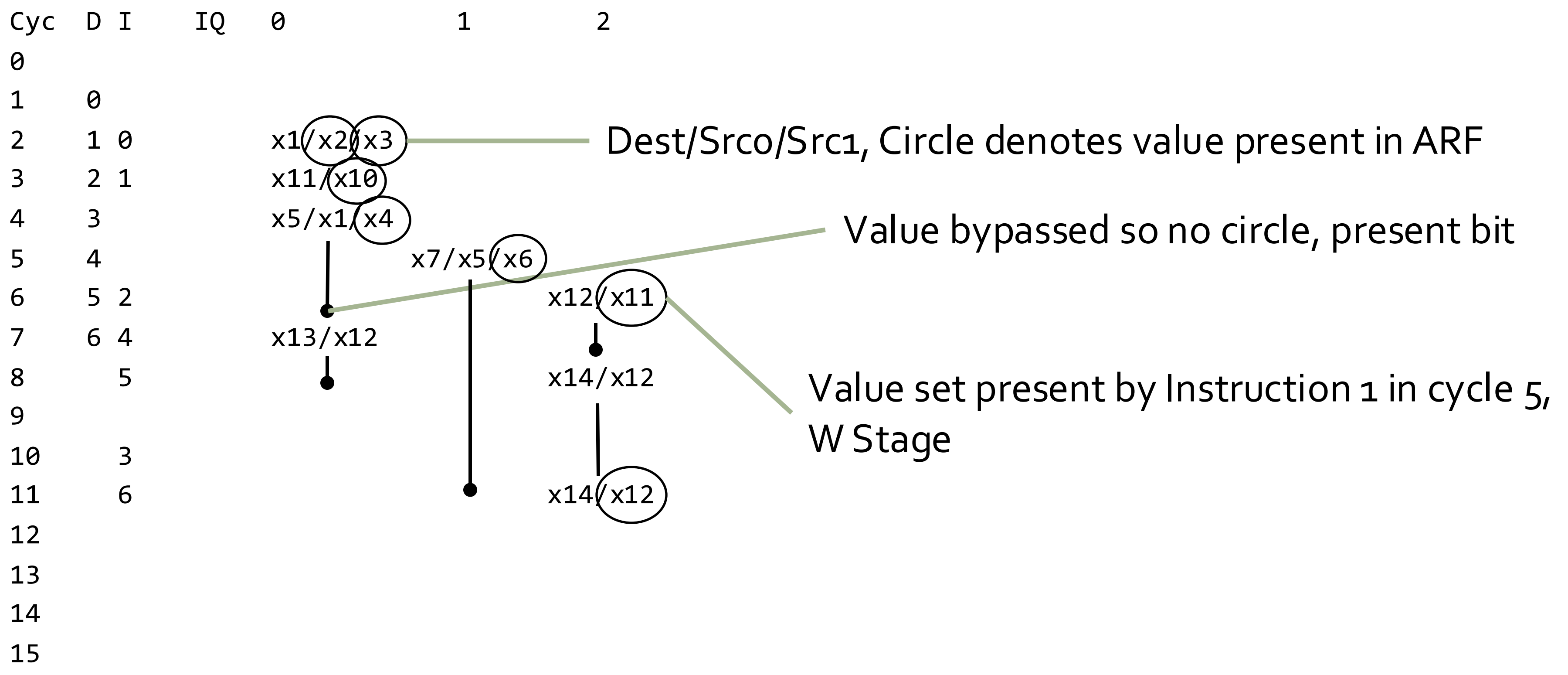
Distributed



```
                    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
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
```

```
                             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
0 mul   x1, x2, x3    F  D  I  Y0 Y1 Y2 Y3 W
1 addi  x11,x10,1        F  D  I  X0 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  X0 W
5 addi  x13,x12,1                    F  D  i  I  X0 W
6 addi  x14,x12,2                       F  D  i        I  X0 W
```



```
Cyc  D I    IQ   0           1        2
0
1    0
2    1 0         x1/x2/x3
3    2 1         x11/x10
4    3           x5/x1/x4
5    4                    x7/x5/x6
6    5 2                           x12/x11
7    6 4         x13/x12
8      5                           x14/x12
9
10     3
11     6                           x14/x12
12
13
14
15
```

Dest/Srco/Src1, Circle denotes value present in ARF

Value bypassed so no circle, present bit

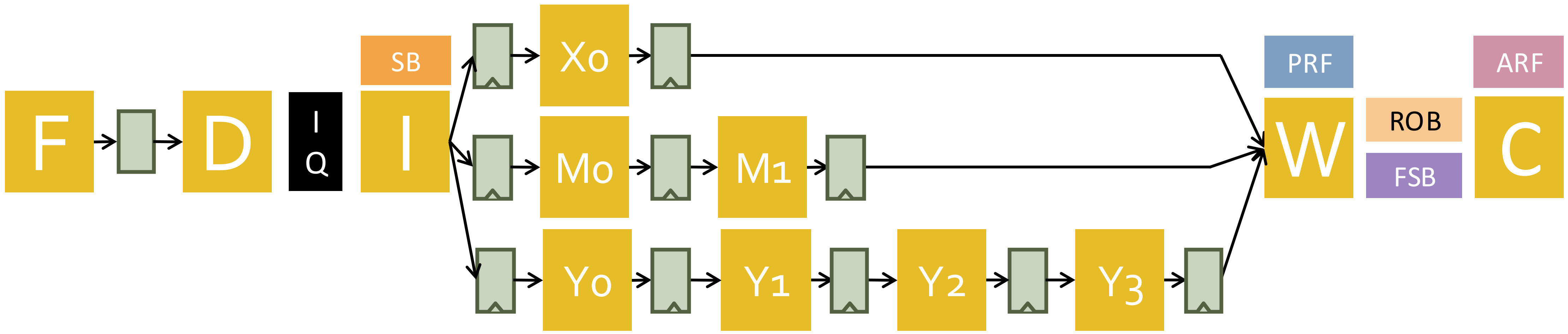
Value set present by Instruction 1 in cycle 5, W Stage

Assume All Instructions in Issue Queue

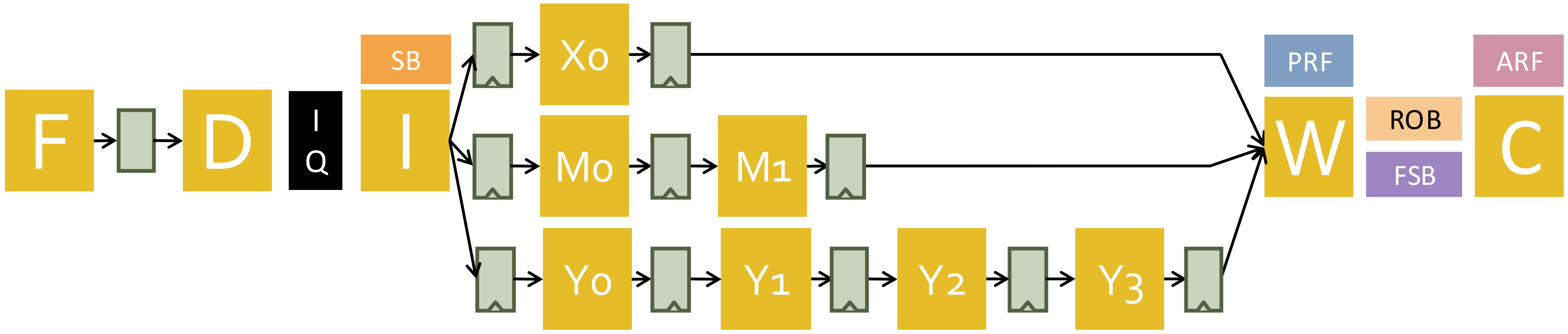
```
                                         0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
0 mul   x1, x2, x3   F  D  i                   I  Y0 Y1 Y2 Y3 W
1 addi  x11,x10,1       F  D  i                   I  X0 W
2 mul   x5, x1, x4         F  D  i                         I  Y0 Y1 Y2 Y3 W
3 mul   x7, x5, x6            F  D  i                                  I  Y0 Y1 Y2 Y3 W
4 addi  x12,x11,1                F  D  i             I  X0 W
5 addi  x13,x12,1                   F  D  i                   I  X0 W
6 addi  x14,x12,2                      F  D  i                   I  X0 W
```

- Better performance than previous?

IO2I: In-order Frontend, Out-of-order Issue/Writeback, 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		



```
                  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
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
```

```
                     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  X0 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  X0 W  r                    C
5 addi  x13,x12,1                   F  D  i  I  X0 W  r                    C
6 addi  x14,x12,2                      F  D  i        I  X0 W  r              C
```

Difference?

```
0 mul   x1, x2, x3   F  D  I  Y0 Y1 Y2 Y3 W  C
1 addi  x11,x10,1       F  D  I  X0 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  X0 W  r                    C
5 addi  x13,x12,1                   F  D  i  I  X0 W  r                    C
6 addi  x14,x12,2                      F  D  i     I  X0 W  r                 C
```

Out-of-order 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	X0	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	X0	W	r										C		
5	addi	x13,x12,1			F	D	i	I	X0	W	r										C	
6	addi	x14,x12,2				F	D	i	I	X0	W	r										C

Out-Of-Order (OOO)

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

Superscalar Intel Processors

Computer Organization and Design RISC-V Edition 2nd Edition Figure 4.74

Microprocessor	Year	Clock Rate	Pipeline Stages	Issue Width	Out-of-Order / Speculation	Cores/Chip	Power
Intel 486	1989	25 MHz	5	1	No	1	5W
Intel Pentium	1993	66 MHz	5	2	No	1	10W
Intel Pentium Pro	1997	200 MHz	10	3	Yes	1	29W
Intel Pentium 4 Willamette	2001	2000 MHz	22	3	Yes	1	75W
Intel Pentium 4 Prescott	2004	3600 MHz	31	3	Yes	1	103W
Intel Core	2006	3000 MHz	14	4	Yes	2	75W
Intel Core i7 Nehalem	2008	3600 MHz	14	4	Yes	2-4	87W
Intel Core Westmere	2010	3730 MHz	14	4	Yes	6	130W
Intel Core i7 Ivy Bridge	2012	3400 MHz	14	4	Yes	6	130W
Intel Core Broadwell	2014	3700 MHz	14	4	Yes	10	140W
Intel Core i9 Skylake	2016	3100 MHz	14	4	Yes	14	165W
Intel Ice Lake	2018	4200 MHz	14	4	Yes	16	185W

- Pentium 4: Marketing demanded higher clock rate ⇒ deeper pipelines & higher power consumption
- Afterwards: Multi-core processors

ARM Cortex-A53 vs. Intel Core i7

Characteristic	ARM A53	Intel Core i7 920
Market	Personal Mobile Device	Server, Cloud
Thermal Design Power	100 milliWatts (1 core @ 1 GHz)	130 Watts
Clock Rate	1.5 GHz	2.66 GHz
Cores/Chip	4 (configurable)	4
Floating Point?	Yes	Yes
Multiple Issue?	Dynamic	Dynamic
Peak Instructions/Clock Cycle	2	4
Pipeline Stages	8	14
Pipeline Schedule	Static In-order	Dynamic Out-of-order with Speculation
Branch Prediction	Hybrid	2-level
1st Level Caches/Core	16-64 KiB I, 16-64 KiB D	32 KiB I, 32 KiB D
2nd Level Cache/Core	128-2048 KiB (shared)	256 KiB (per core)
3rd Level Cache (Shared)	(platform dependent)	2-8 MiB

Computer Organization and Design RISC-V Edition 1st Edition Figure 4.71

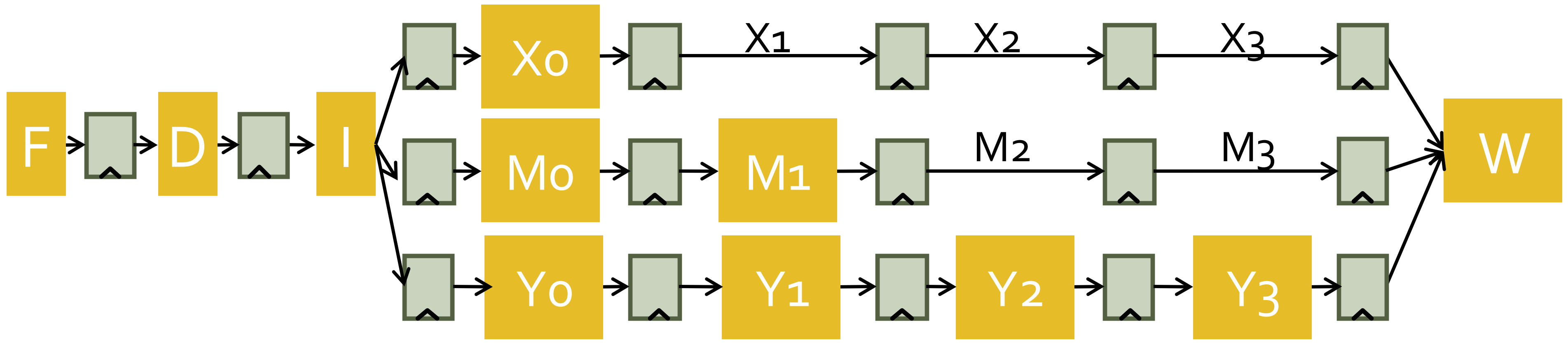
Agenda

- Out-of-Order Processors
- Speculation and Branches
- Register Renaming
- Memory Disambiguation

Speculation and Branches: I4

	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 x4, x5, 1`		F	D	I	X0	X1	X2	X3	W											
`2 mul x6, x1, x4`			F	D	I	I	I	Y0	Y1	Y2	Y3	W								
`3 beq x6, x0, Target`				F	D	D	D	I	I	I	I	X0	X1	X2	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	--	--	--	--	--	--	--	
`T`													F	D	I	. . .				

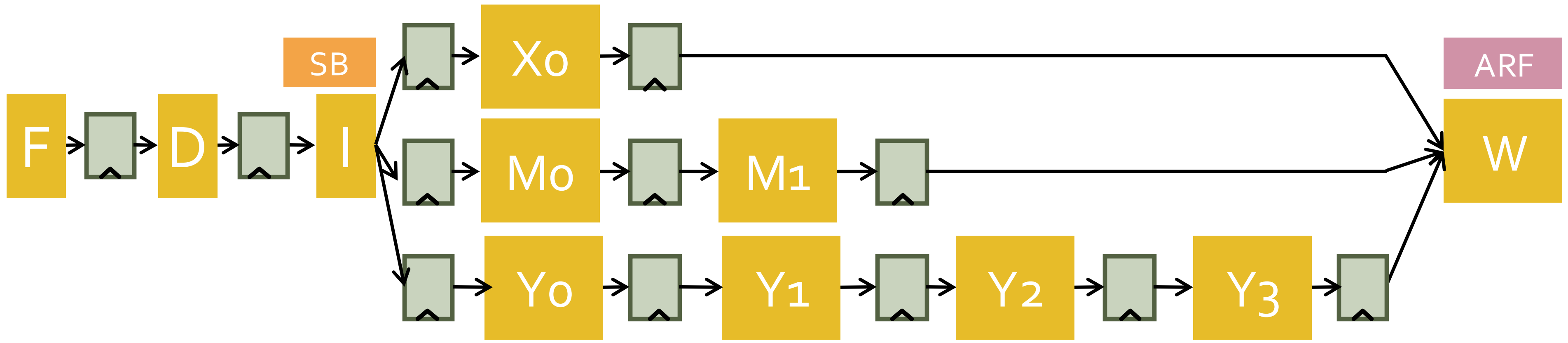
- No speculative instructions commit state



Speculation and Branches: I2O2

	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 x4, x5, 1		F	D	I	X0	W														
2 mul x6, x1, x4			F	D	I	I	I	Y0	Y1	Y2	Y3	W								
3 beq x6, x0, Target				F	D	D	D	I	I	I	I	X0	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	--	--	--	--				
T													F	D	I	. . .				

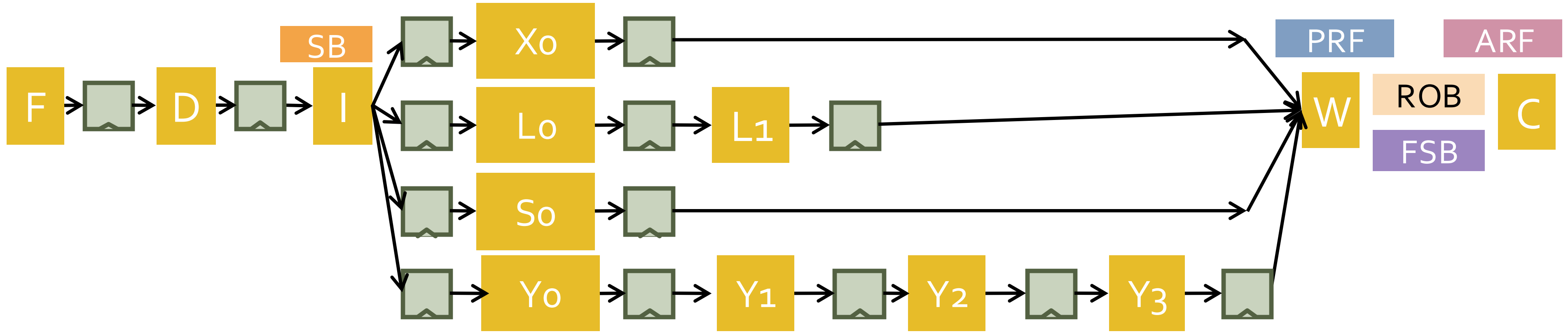
- No speculative instructions commit state



Speculation and Branches: 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	C											
1 addi x4, x5, 1		F	D	I	X0	W	r			C										
2 mul x6, x1, x4			F	D	I	I	I	Y0	Y1	Y2	Y3	W	C							
3 beq x6, x0, Target				F	D	D	D	I	I	I	I	X0	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	--	--	--	--	--			
T													F	D	I	. . .				

- Must squash instructions in pipeline after branch to prevent PRF write
- Can remove from ROB immediately or wait until commit

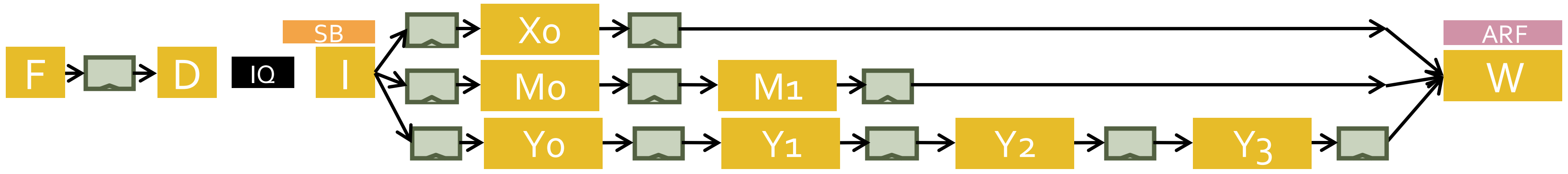


Speculation and Branches: IO3

		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	x4, x5, 1		F	D	I	X0	W														
2 mul	x6, x1, x4			F	D	i		I	Y0	Y1	Y2	Y3	W								
3 beq	x6, x0, Target				F	D	i					I	X0	W							
4 addi	x8, x9 ,1					F	D	i	I	X0	W										
5 addi	x10,x11,1						F	D	i	I	X0	W									
6 addi	x12,x13,1							F	D	i			I	X0	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

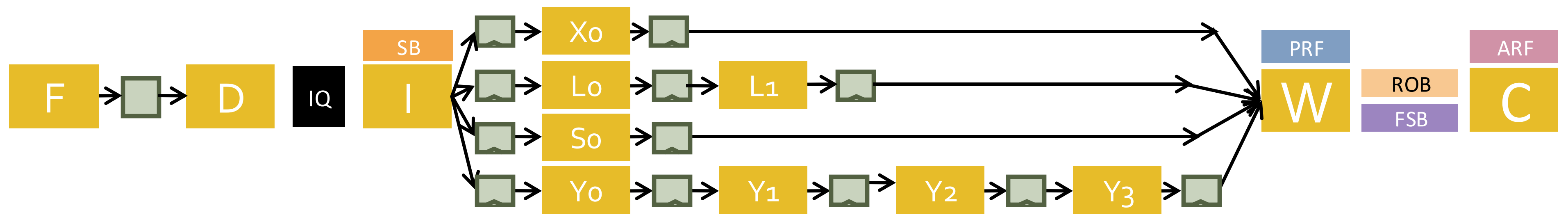
- No control speculation for IO3
- Could stall on branch



Speculation and Branches: IO2I

	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 x4, x5, 1		F	D	I	X0	W	r			C										
2 mul x6, x1, x4			F	D	i		I	Y0	Y1	Y2	Y3	W	C							
3 beq x6, x0, Target				F	D	i					I	X0	W	C						
4 addi x8, x9 ,1					F	D	i	I	X0	W	r	--								
5 addi x10,x11,1						F	D	i	I	X0	W	--								
6 addi x12,x13,1							F	D	i			--								
7 ???								F	D			--								
8 ???									F	D		--								
9 ???										F	D	--								
10???											F	--								
11???												--	D							
T													F	D	I	.	.	.		

Need to clean up Speculative state In PRF. Needs selective rollback



Speculation and Branches: IO2I

			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	x4, x5, 1		F	D	I	X0	W	r			C										
2	mul	x6, x1, x4			F	D	i		I	Y0	Y1	Y2	Y3	W	C							
3	beq	x6, x0, Target				F	D	i					I	X0	W	C						
4	addi	x8, x9 ,1					F	D	i	I	X0	W	r			/						
5	addi	x10,x11,1						F	D	i	I	X0	W	r		/						
6	addi	x12,x13,1							F	D	i			I	X0	/						
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

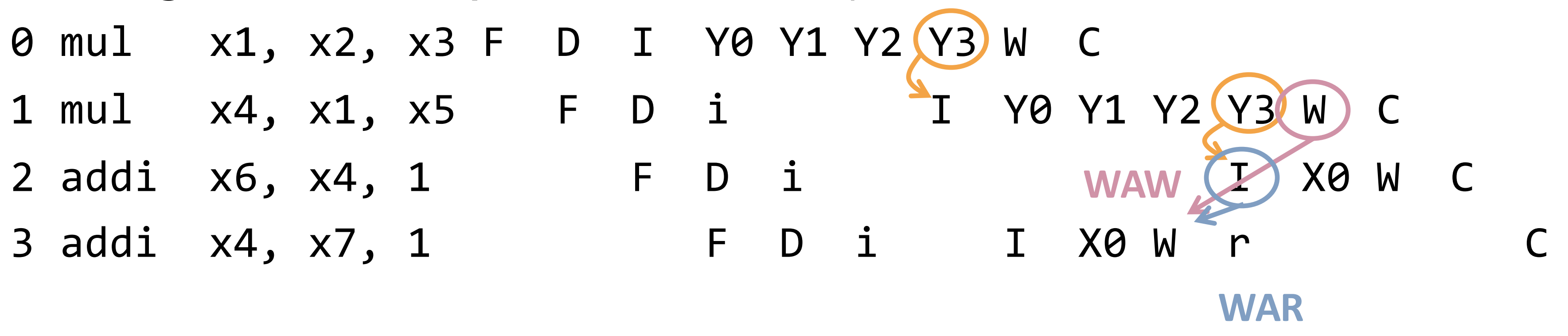
Agenda

- Out-of-Order Processors
- Speculation and Branches
- Register Renaming
- Memory Disambiguation

WAW and WAR “Name” Dependencies

- WAW and WAR are not “True” data dependencies
- RAW is “True” data dependency because reader needs result of writer
- “Name” dependencies exist because we have limited number of “Names” (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  X0 W  C
3 addi  x4, x7, 1           F  D  i     I  X0 W  r           C
                                                WAR
```

Adding More Registers

Breaking all "Name" Dependencies

```
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  X0 W  C
3 addi   x4, x7, 1           F  D  i     I  X0 W  r           C
```

IO2I Microarchitecture Conservatively Stalls

```
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  X0 W  C
3 addi   x4, x7, 1           F  D  D  D  D  D  D  D  D  D  D  I  X0 W  C
```

Manual Register Renaming. What if we could use more registers? Second X4 Write to X8?

```
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  X0 W  C
3 addi   x8, x7, 1           F  D  i     I  X0 W  r           C
```

How many Instructions can be in the pipeline?

- Throughput is limited by number of instructions in flight, but which feature of an ISA limits the number of instructions in the pipeline?

Register Renaming

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

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

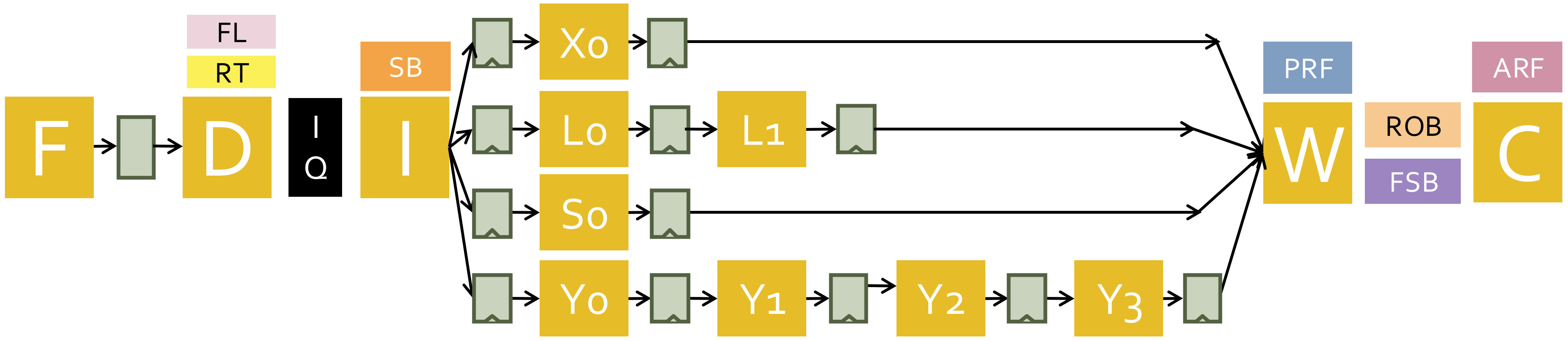
Floating Point pipelines often cannot be kept filled with small number of registers.
$\rightarrow$ IBM 360 had only 4 Floating Point Registers

Register Renaming Overview

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

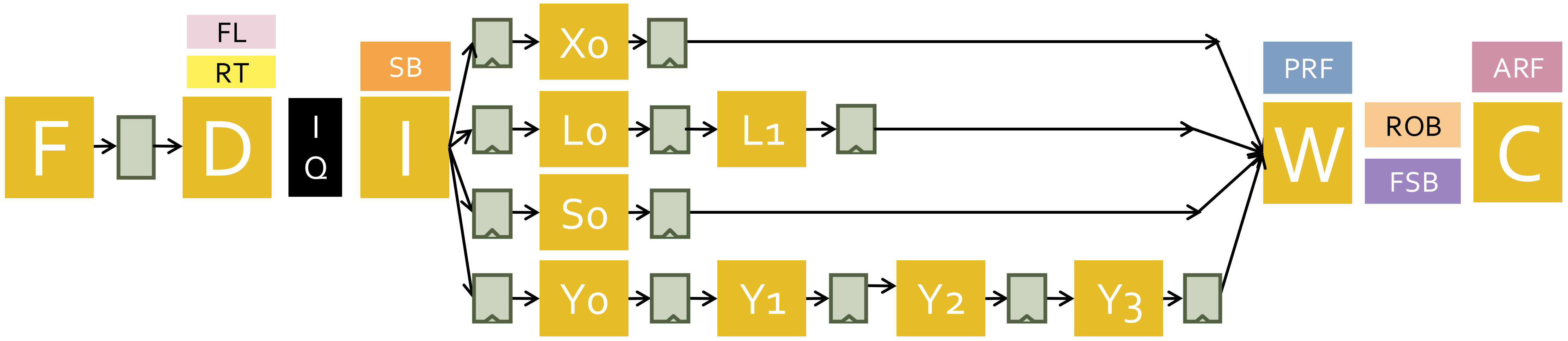
- IO2I uses pointers in IQ and ROB therefore start with that design

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	R/W	W	
FL	R/W			W
RT	R/W		W	

Modified Reorder Buffer (ROB)

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

State: {Empty (--), 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

Rename Table (RT)

	P	Preg
x1		
x2		
x3		
…		
x31		

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

Free List (FL)

	Free
p1	
p2	
p3	
…	
pN	

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
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	X0	W	C		
3 addi	x4, x7, 1				F	D	i		I	X0	W	r				C	

					RT							FL	IQ				ROB			
Cy	D	I	W	C	x1	x2	x3	x4	x5	x6	x7		0	1	2	3	0	1	2	3
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}	p7/p1/p2				p7/x1/p0			
3	2							p8				p{9,10}		p8/p7/p4				p8/x4/p3		
4	3									p9		p10			p9/p8				p9/x6/p5	
5								p10								p10/p6				p10/x4/p8
6		1																		
7		3	0																	
8				0	p7												p7/x1/p0			
9			3									p0								
10		2						p10				p0								p10/x4/p8
11			1									p0								
12			2	1								p0						p8/x4/p3		
13				2						p9		p{0,3}							p9/x6/p5	
14				3								p{0,3,5}								
15												p{0,3,5,8}								

Freeing Physical Registers

```
add  x1,x2,x3     ⇐Assume Arch. Reg X1 maps to Phys. Reg p0
add  x4,x1,x5
add  x1,x6,x7     ⇐Next write of Arch Reg X1, Mapped to Phys. Reg p1
add  x8,x9,x10
```

```
0 add  x1,x2,x3     I  X  W  C
1 add  x4,x1,x5                                 I  X  W  C
2 add  x1,x6,x7              I  X  W  r                     C
3 add  x8,x9,x10                F  D  I  X  W  r               C
             Write p0  Free p0 Alloc p0  Write p0 Read Wrong
                                                  value in p0
```

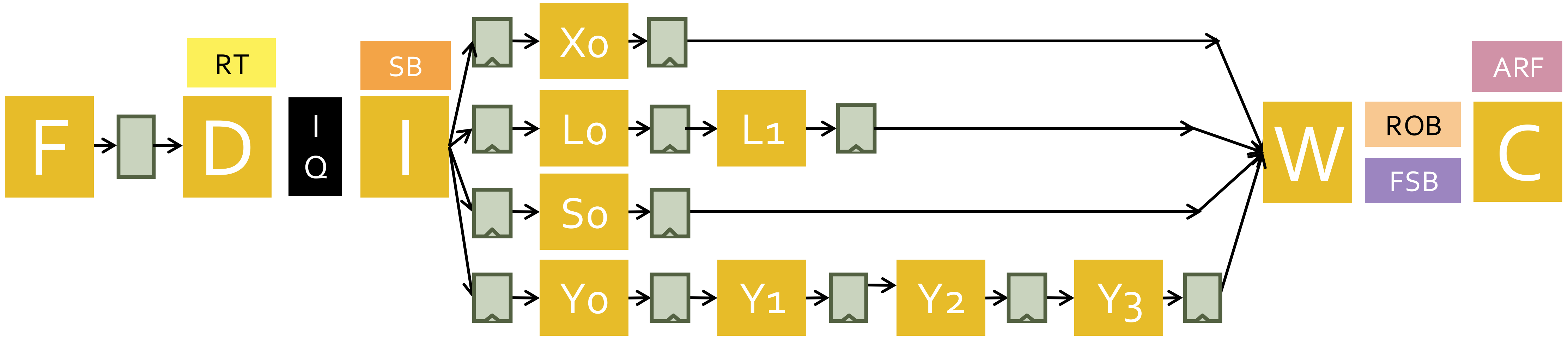
```
0 add  x1,x2,x3     I  X  W  C
1 add  x4,x1,x5                                 I  X  W  C
2 add  x1,x6,x7              I  X  W  r                     C
3 add  x8,x9,x10                F  D  I  X  W  r               C
             Write p0  Alloc p2  Write p2    Dealloc p0
```

- If Arch. Reg Xi mapped to Phys. Reg pj, we can free pj when the next instruction that writes Xi commits

Unified Physical/Architectural Register File

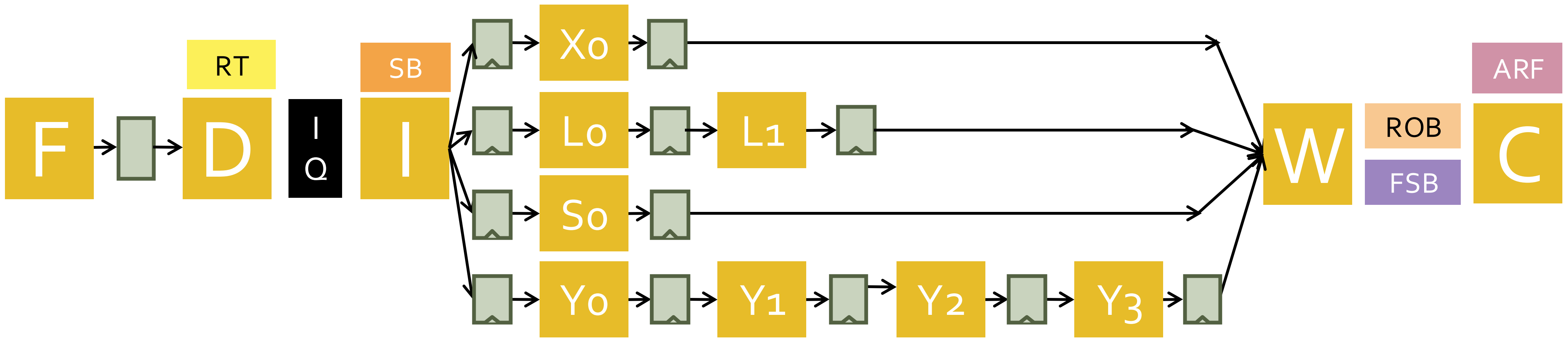
- Combine PRF and ARF into one register file
- Replace ARF with Architectural Rename Table
- Instead of copying **Values**, Commit stage copies Preg pointer into appropriate entry of Architectural Rename Table
- Unified Physical/Architectural Register file can be smaller than separate

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	R/W	W	
RT	R/W			W

Modified Reorder Buffer (ROB)

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

State: {Empty (--), Pending, Finished}
S: Speculative
ST: Store bit
V: Destination is valid
Value: Actual Register Value
Areg: Architectural Register File Specifier

Modified Issue Queue (IQ)

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)

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

On Commit, Source Field contains value

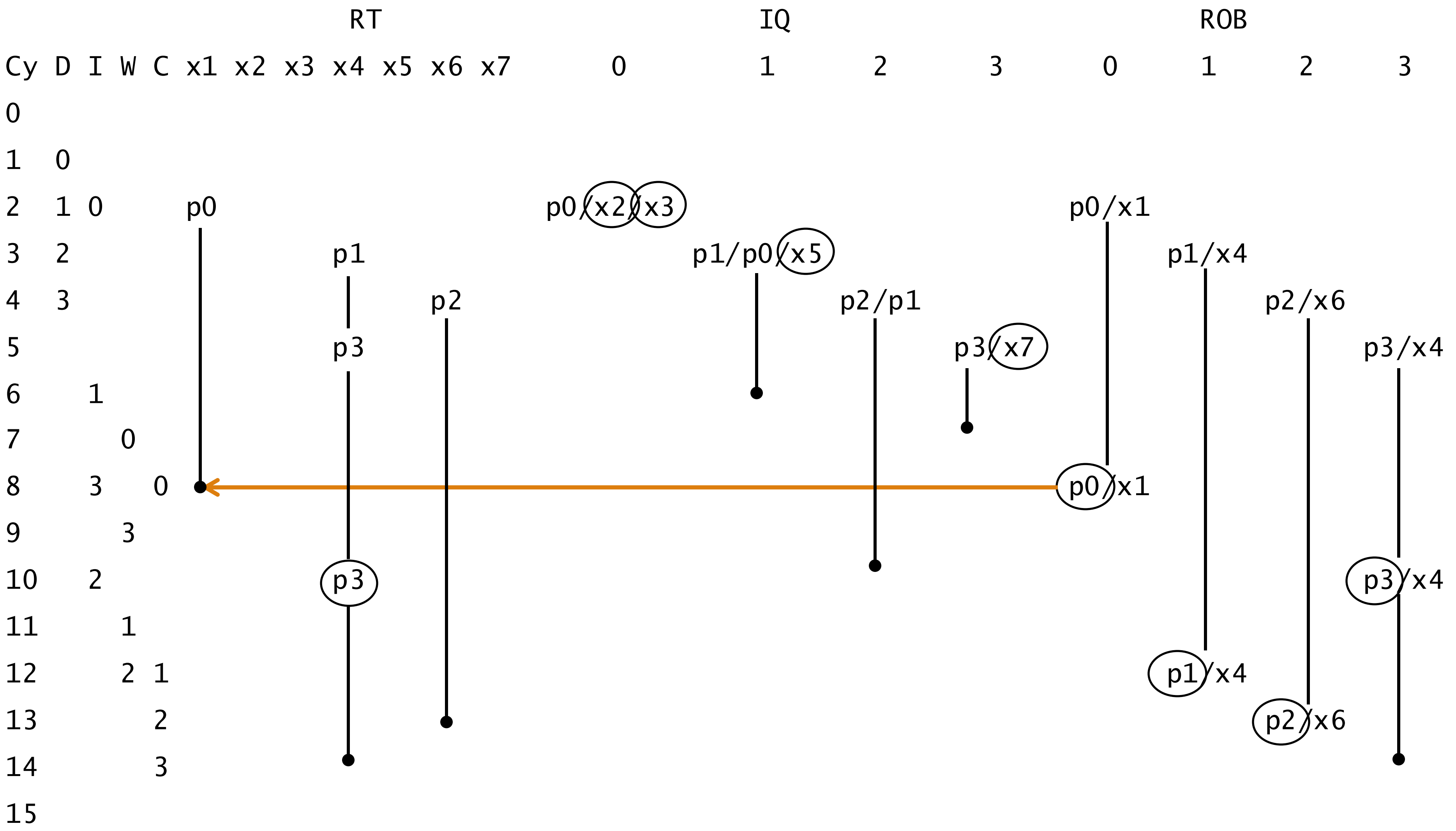
Modified Rename Table (RT)

	V	P	Preg
x1			
x2			
x3			
…			
x31			

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

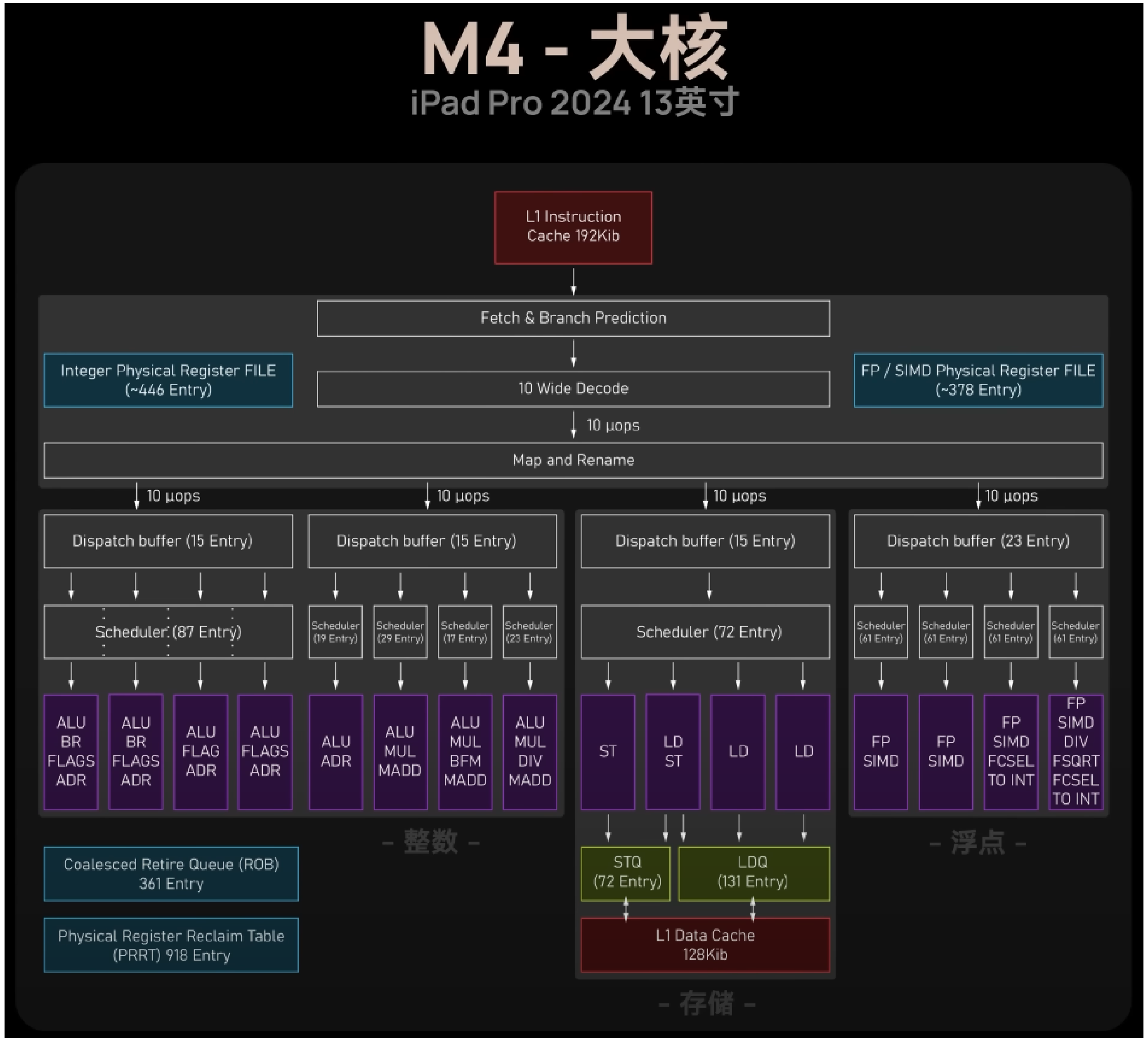
```
V:
   If V == 0:
        Value in ARF is up to date
   If V == 1:
        Value is in-flight or in ROB
P:
   If P == 0:
        Value is in ROB
   if P == 1:
        Value is in flight
```

```
                   0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
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  X0 W  C
3 addi  x4, x7, 1           F  D  i     I  X0 W  r           C
```



```
                         RT                        IQ                                 ROB
Cy D I W C x1 x2 x3 x4 x5 x6 x7        0           1          2        3         0       1       2       3
0
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                                                                  p0/x1
9      3
10   2              p3                                                                                   p3/x4
11     1
12     2 1                                                                               p1/x4
13       2                                                                                       p2/x6
14       3
15
```

- Apple M4 (2024)



Source: https://www.youtube.com/watch?v=EbDPvcbilCs

Agenda

- Out-of-Order Processors
- Speculation and Branches
- Register Renaming
- Memory Disambiguation

Memory Disambiguation

```
sw x1, 0(x2)
lw x3, 0(x4)
```

When can we execute the load?

In-Order Memory Queue

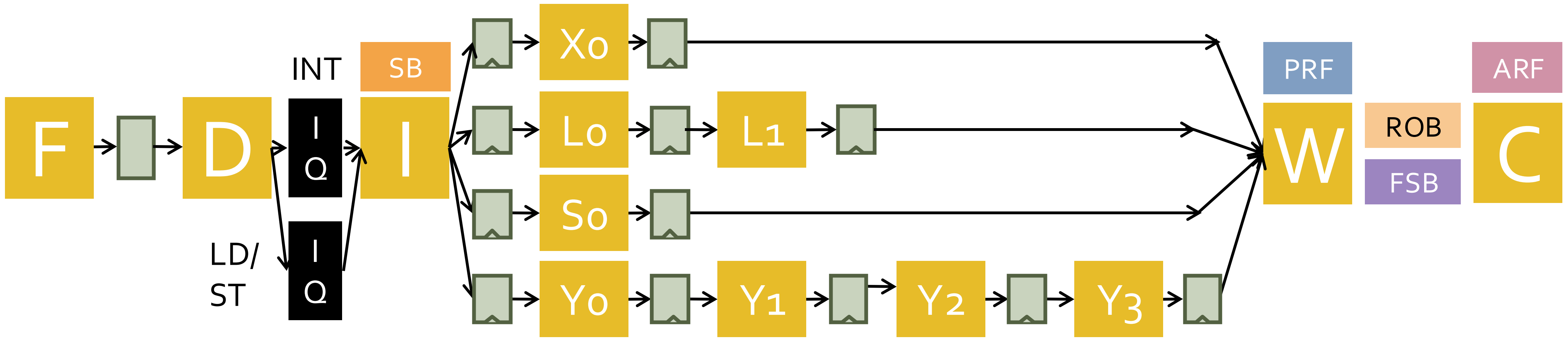
- Execute all loads and stores in program order

  => 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

```
sw x1, 0(x2)
lw x3, 0(x4)
```

- Split execution of store instruction into two phases: address calculation and data write
- Can execute load before 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 any previous store address not known

(MIPS R10K, 16 entry address queue)

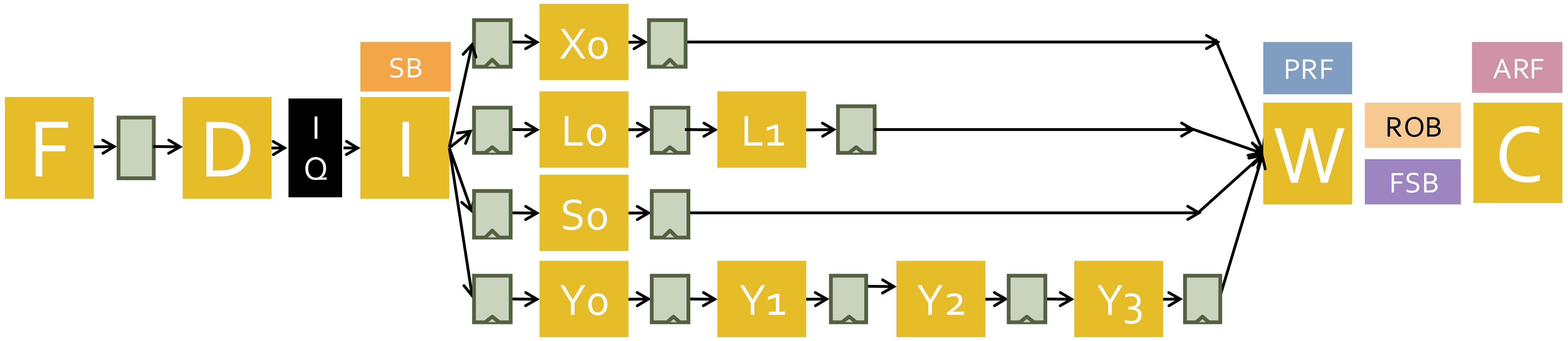
Address Speculation

```
sw x1, 0(x2)
lw 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* dependent instructions

=> Large penalty for inaccurate address speculation

IO2I: With OOO Load and Stores



Memory Dependence Prediction
(Alpha 21264)

```
sw x1, 0(x2)
lw 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

Recap

- Out-of-Order Processors
- Speculation and Branches
- Register Renaming
- Memory Disambiguation

Acknowledgements

- These slides contain material developed and copyright by:
  - Arvind (MIT)
  - Krste Asanovic (MIT/UCB)
  - Joel Emer (Intel/MIT)
  - James Hoe (CMU)
  - John Kubiatowicz (UCB)
  - David Patterson (UCB)
  - Christopher Batten (Cornell)
  - David Wentzlaff (Princeton)
- MIT material derived from course 6.823
- UCB material derived from course CS252
- Cornell material derived from course ECE 4750
- Princeton material derived from course ECE 475