Computer Architecture
ELE 475 / COS 475
Slide Deck 5: Exceptions and Superscalar 2

David Wentzlaff

Department of Electrical Engineering

Princeton University

PRINCETON UNIVERSITY

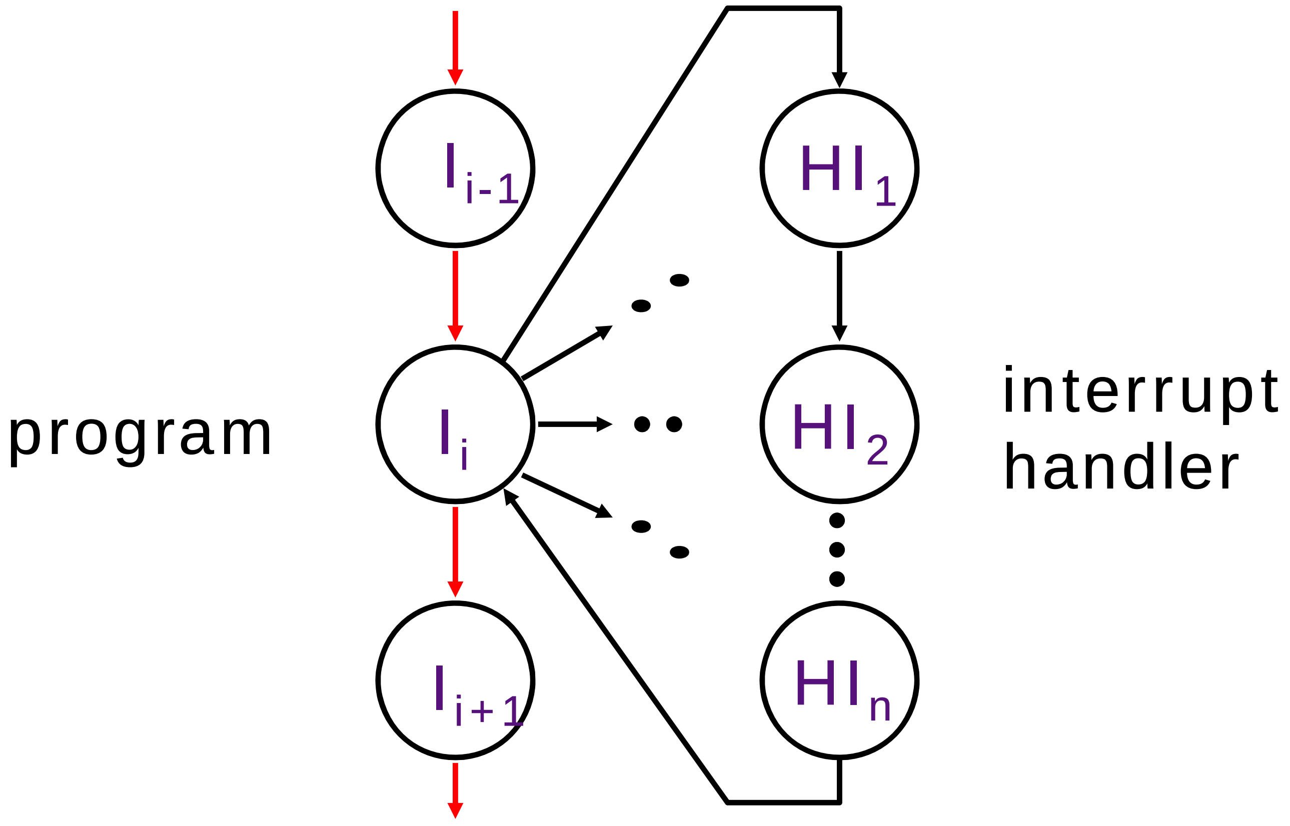
PRINCETON
School of Engineering and Applied Science

Agenda

- Interrupts
- Out-of-Order Processors

Interrupts:

altering the normal flow of control



An external or internal event that needs to be processed by another (system) program. The event is usually unexpected or rare from program’s point of view.

Causes of Exceptions

Interrupt: an *event* that requests the attention of the processor

- Asynchronous: an *external event*
  - input/output device service request
  - timer expiration
  - power disruptions, hardware failure
- Synchronous: an *internal exception (a.k.a. exceptions/trap)*
  - undefined opcode, privileged instruction
  - arithmetic overflow, FPU exception
  - misaligned memory access
  - *virtual memory exceptions:* page faults, TLB misses, protection violations
  - *software exceptions:* system calls, e.g., jumps into kernel

Asynchronous Interrupts:

invoking the interrupt handler

- An I/O device requests attention by asserting one of the *prioritized interrupt request lines*
- When the processor decides to process the interrupt
  - It stops the current program at instruction I_i, completing all the instructions up to I_{i-1} (a *precise interrupt)*
  - It saves the PC of instruction I_i in a special register (EPC)
  - It disables interrupts and transfers control to a designated interrupt handler running in the kernel mode

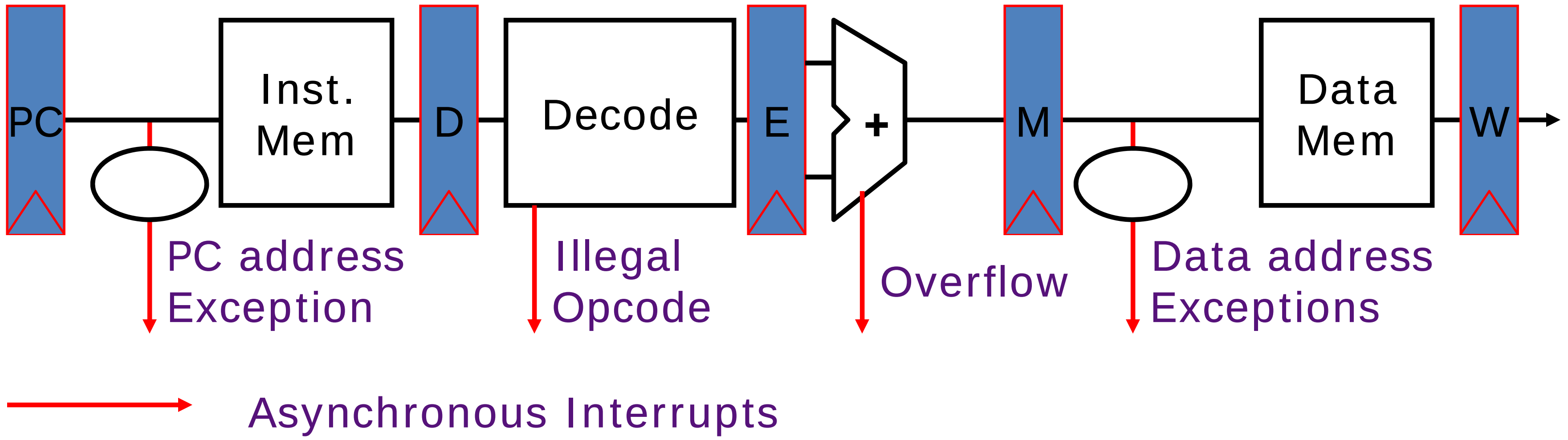
Interrupt Handler

- Saves EPC before re-enabling interrupts to allow nested interrupts $\Rightarrow$
  - need an instruction to move EPC into GPRs
  - need a way to mask further interrupts at least until EPC can be saved
- Needs to read a *status register* that indicates the cause of the interrupt
- Uses a special indirect jump instruction RFE (*return-from-exception*) to resume user code, this:
  - enables interrupts
  - restores the processor to the user mode
  - restores hardware status and control state

Synchronous Interrupts

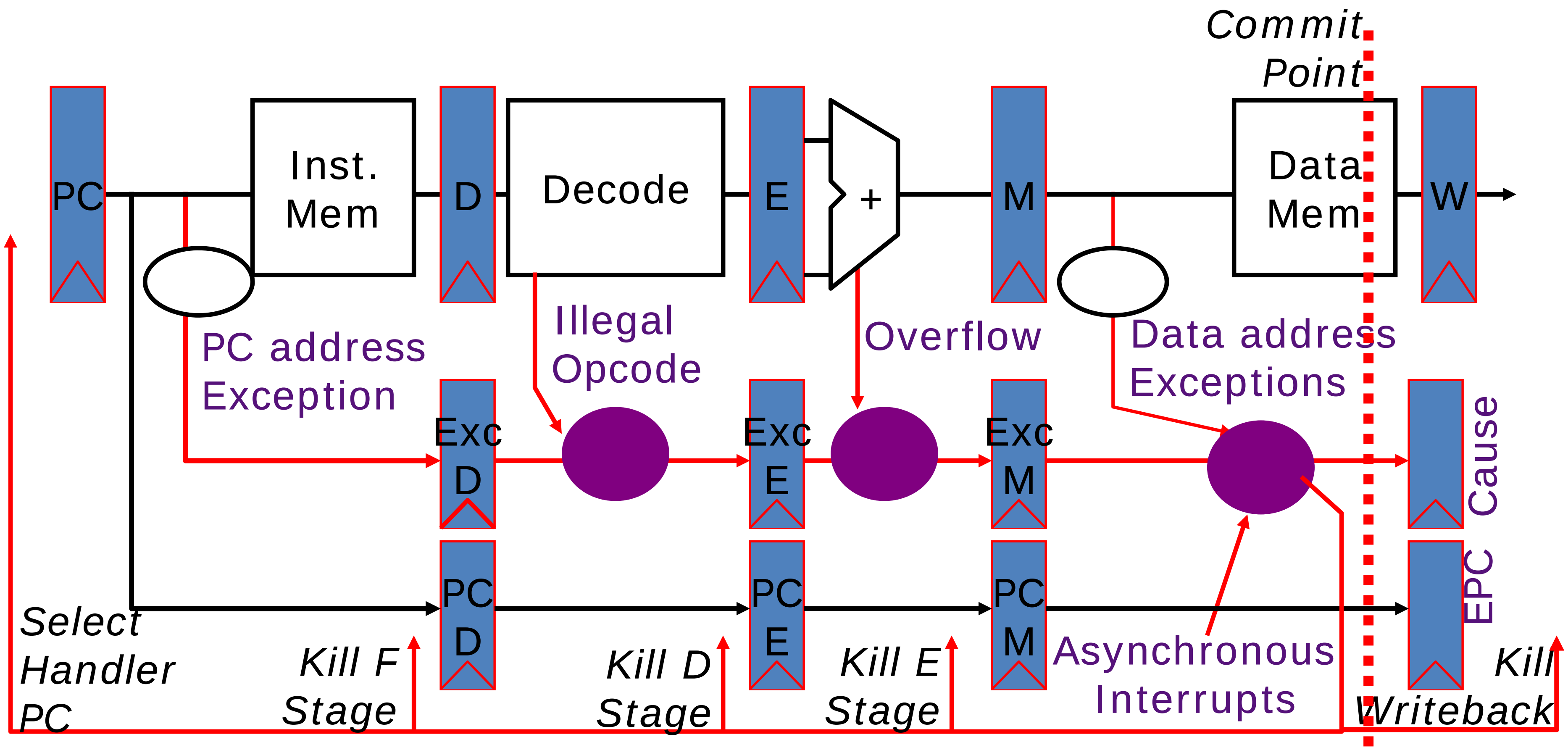
- A synchronous interrupt (exception) is caused by a *particular instruction*
- In general, the instruction cannot be completed and needs to be *restarted* after the exception has been handled
  - requires undoing the effect of one or more partially executed instructions
- In the case of a system call trap, the instruction is considered to have been completed
  - syscall is a special jump instruction involving a change to privileged kernel mode
  - Handler resumes at instruction after system call

Exception Handling 5-Stage Pipeline



- How to handle multiple simultaneous exceptions in different pipeline stages?
- How and where to handle external asynchronous interrupts?

Exception Handling 5-Stage Pipeline



Exception Handling 5-Stage Pipeline

- Hold exception flags in pipeline until commit point (M stage)
- Exceptions in earlier pipe stages override later exceptions *for a given instruction*
- Inject external interrupts at commit point (override others)
- If exception at commit: update Cause and EPC registers, kill all stages, inject handler PC into fetch stage

Speculating on Exceptions

- Prediction mechanism
  - Exceptions are rare, so simply predicting no exceptions is very accurate!
- Check prediction mechanism
  - Exceptions detected at end of instruction execution pipeline, special hardware for various exception types
- Recovery mechanism
  - Only write architectural state at commit point, so can throw away partially executed instructions after exception
  - Launch exception handler after flushing pipeline

- Bypassing allows use of uncommitted instruction results by following instructions

Exception Pipeline Diagram

time

	t0	t1	t2	t3	t4	t5	t6	t7	. . . .
(I_1) 096: ADD	IF_1	ID_1	EX_1	MA_1	nop		*overflow!*		
(I_2) 100: XOR		IF_2	ID_2	EX_2	nop	nop			
(I_3) 104: SUB			IF_3	ID_3	nop	nop	nop		
(I_4) 108: ADD				IF_4	nop	nop	nop	nop	
(I_5) Exc. Handler code					IF_5	ID_5	EX_5	MA_5	WB_5

time

Resource Usage		t0	t1	t2	t3	t4	t5	t6	t7	. . . .
	IF	I_1	I_2	I_3	I_4	I_5				
	ID		I_1	I_2	I_3	nop	I_5			
	EX			I_1	I_2	nop	nop	I_5		
	MA				I_1	nop	nop	nop	I_5	
	WB					nop	nop	nop	nop	I_5

Agenda

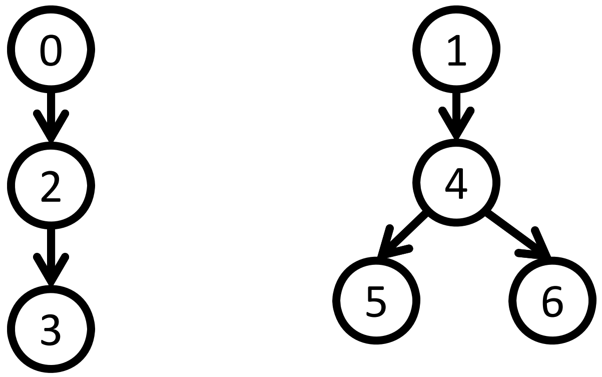
- Interrupts
- Out-of-Order Processors

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
I03	IO	OOO	OOO	OOO	Scoreboard and Issue Queue
IO2I	IO	OOO	OOO	IO	Scoreboard, Issue Queue, Reorder Buffer, and Store Buffer

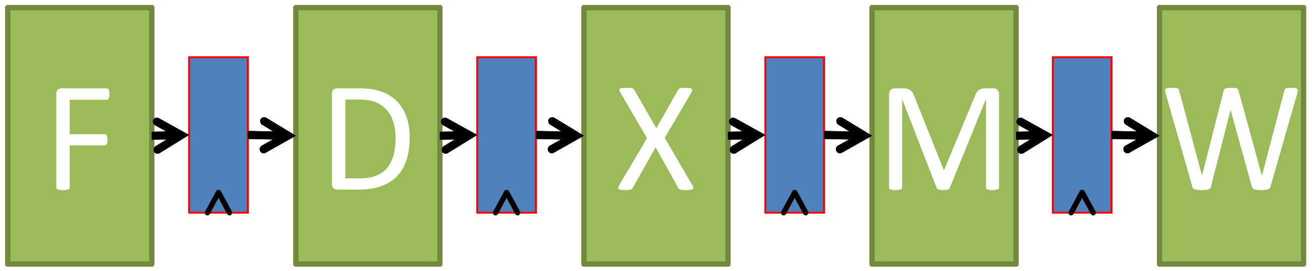
OOO Motivating Code Sequence

```
0 MUL   R1, R2, R3
1 ADDIU R11,R10,1
2 MUL   R5, R1, R4
3 MUL   R7, R5, R6
4 ADDIU R12,R11,1
5 ADDIU R13,R12,1
6 ADDIU R14,R12,2
```

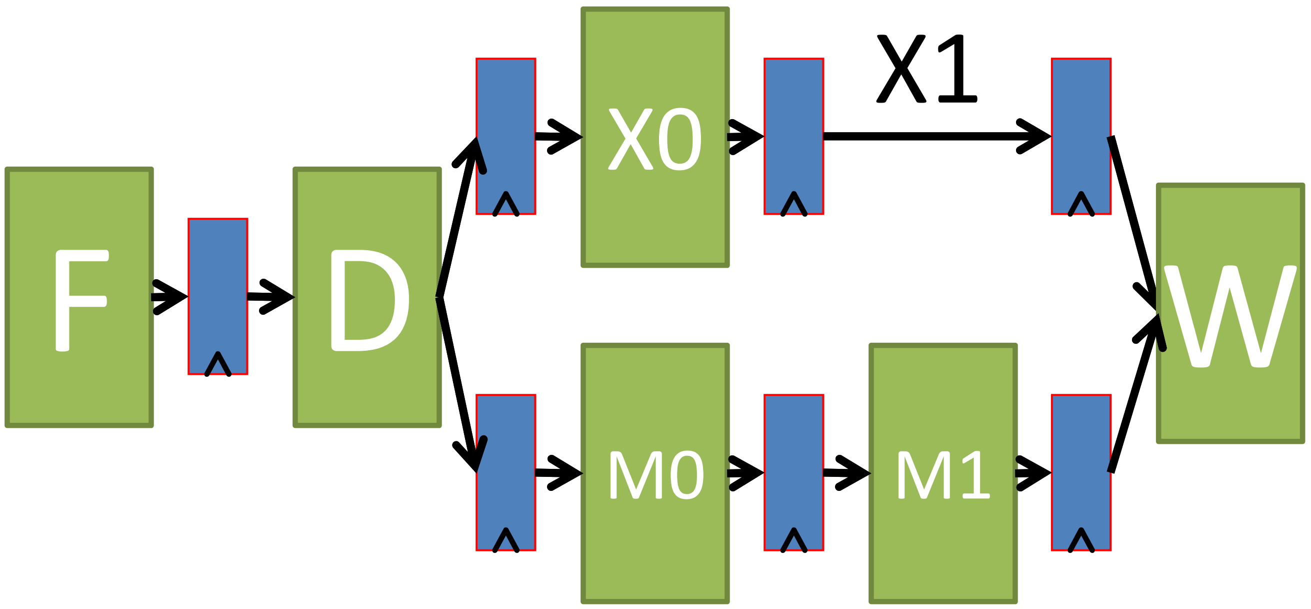

$$\begin{tikzcd} 0 \arrow[d] & & 1 \arrow[d] & \\ 2 \arrow[d] & & 4 \arrow[dl] \arrow[dr] & \\ 3 & 5 & & 6 \end{tikzcd}$$

- 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

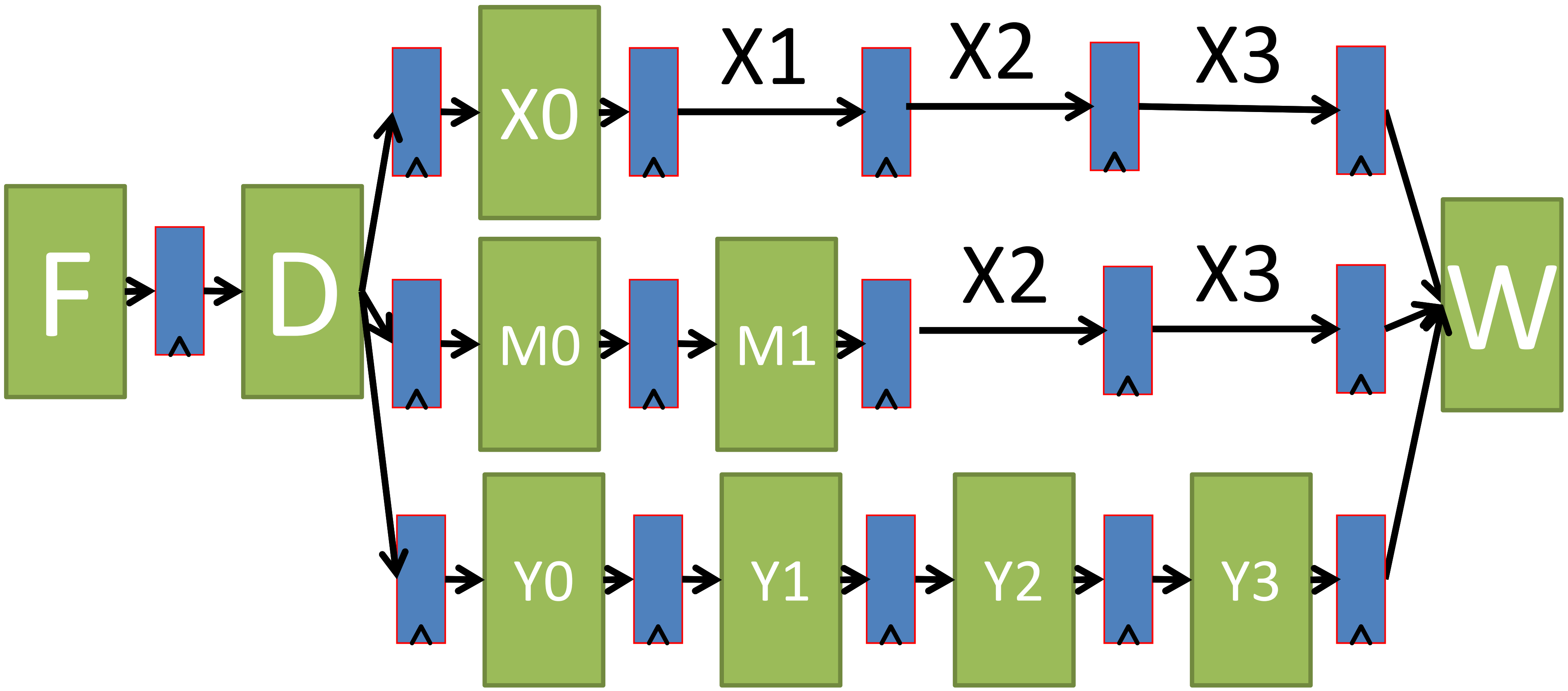
I4: In-Order Front-End, Issue, Writeback, Commit



I4: In-Order Front-End, Issue, Writeback, Commit

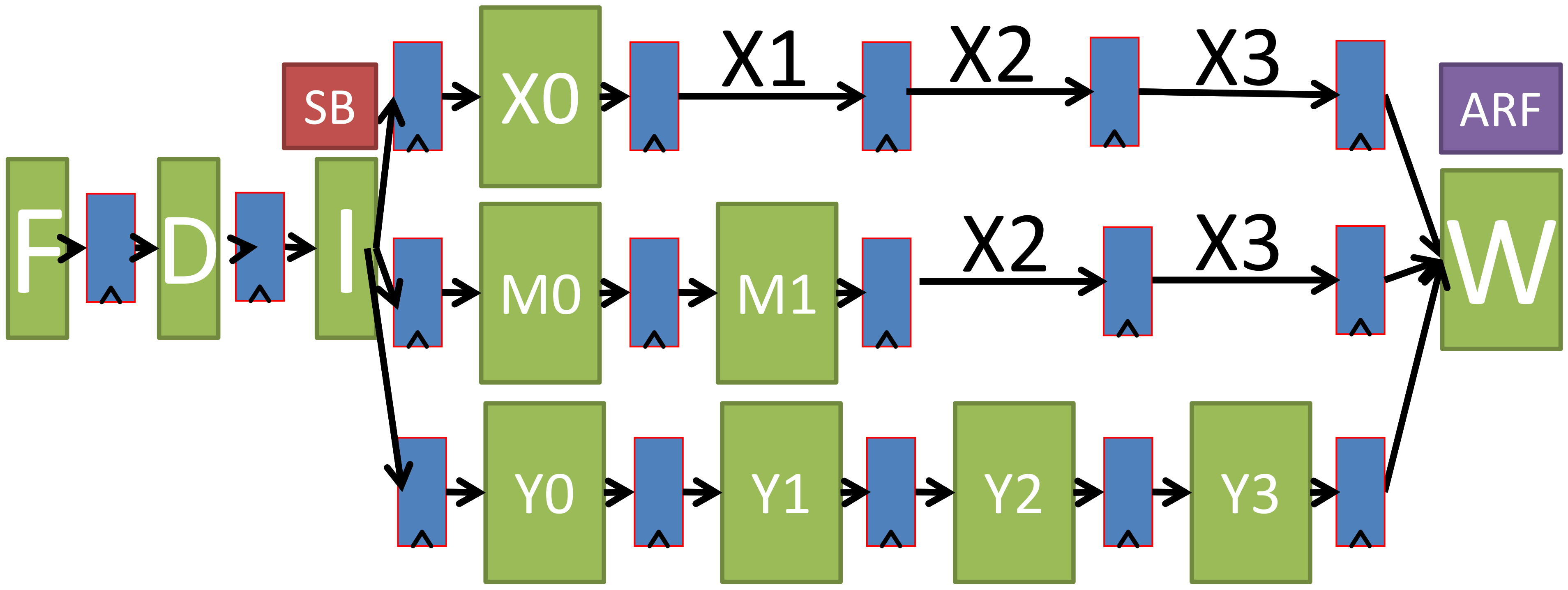


I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)



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

I4: In-Order Front-End, Issue, Writeback, Commit (4-stage MUL)



ARF	R	W
SB	R/W	W

Basic Scoreboard

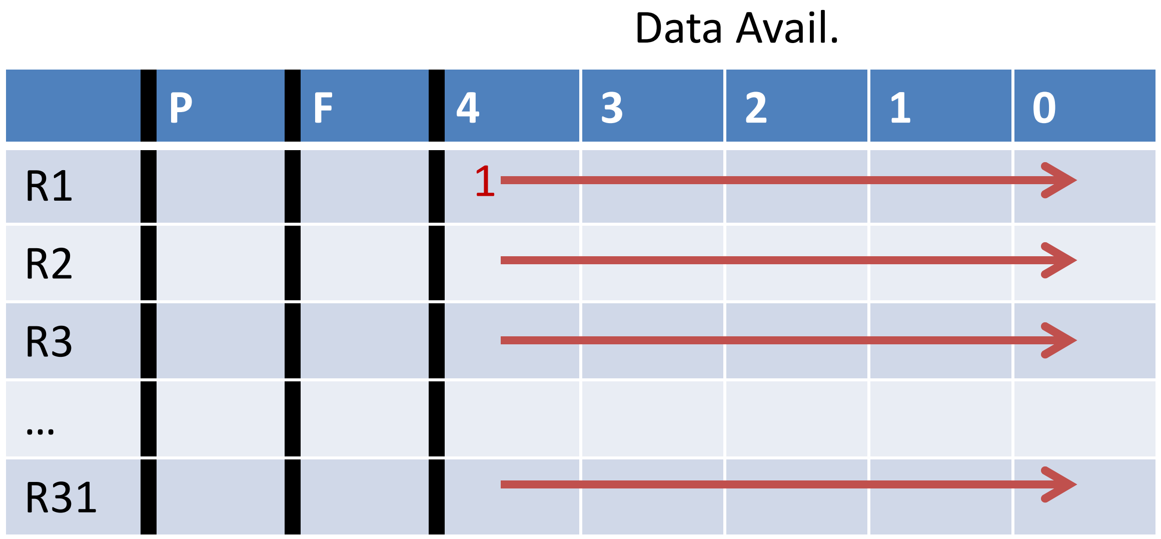
Data Avail.

	P	F	4	3	2	1	0
R1							
R2							
R3							
…							
R31							

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 'I' means that result data is in stage 'I' 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.

Basic Scoreboard



Data Avail.

	P	F	4	3	2	1	0
R1			1				
R2							
R3							
…							
R31							

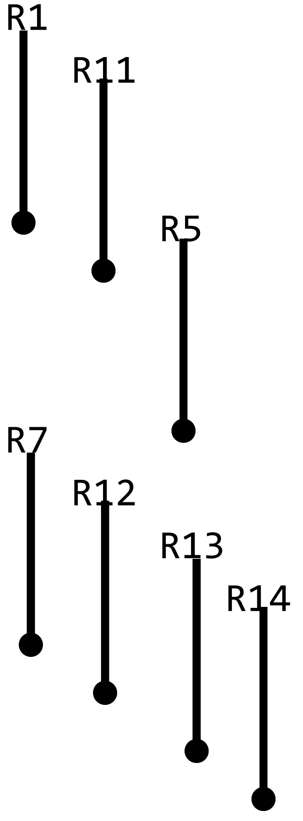
P: Pending, Write to Destination in flight
F: Which functional unit is writing register
Data Avail.: Where is the write data in the functional unit pipeline

- A One in Data Avail. In column ‘I’ means that result data is in stage ‘I’ 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.

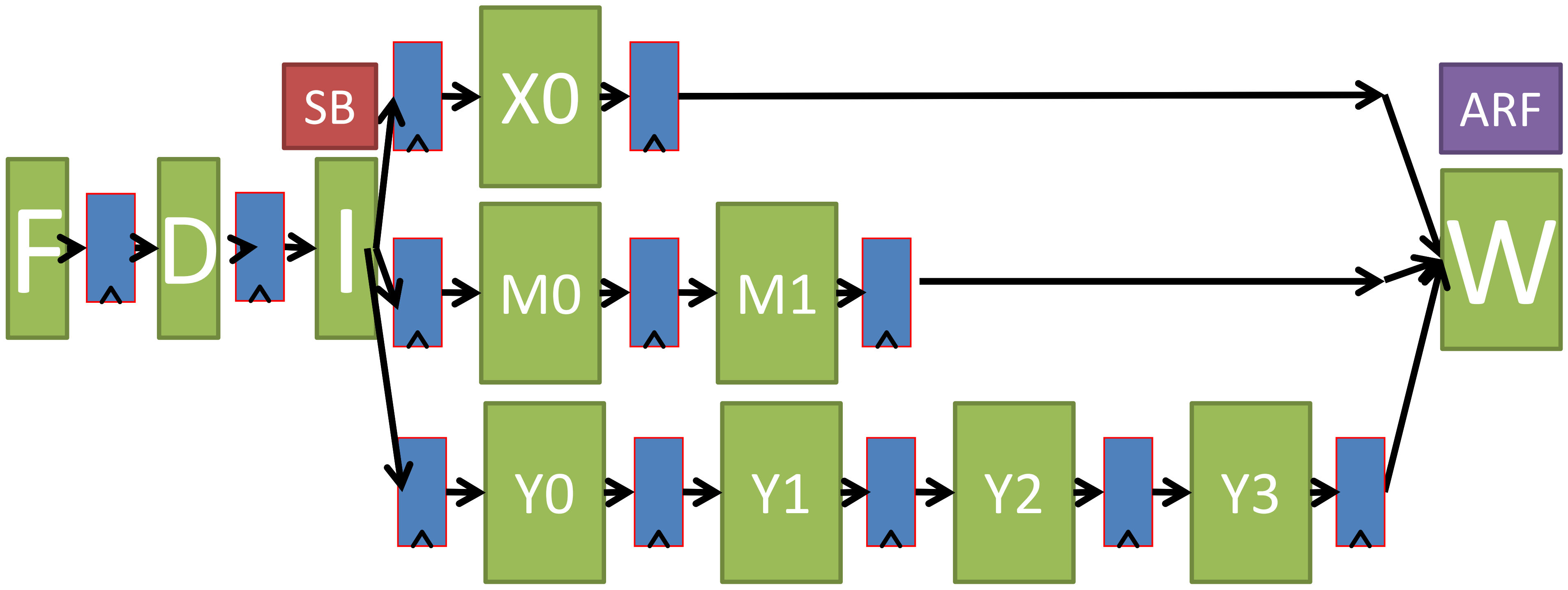
```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU R11,R10,1     F  D  I  X0 X1 X2 X3 W
2 MUL   R5, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W
3 MUL   R7, R5, R6          F  D  D  D  I  I  I  I  Y0 Y1 Y2 Y3 W
4 ADDIU R12,R11,1              F  F  F  D  D  D  D  I  X0 X1 X2 X3 W
5 ADDIU R13,R12,1                       F  F  F  F  D  I  X0 X1 X2 X3 W
6 ADDIU R14,R12,2                                   F  D  I  X0 X1 X2 X3 W
```

```
Cyc  D I         4 3 2 1 0     Dest Regs
1    0
2    1 0
3    2 1         1             R1
4                1 1              R11
5                  1 1
6    3 2             1 1
7                1     1 1            R5
8                  1     1
9                    1
10   4 3               1
11   5 4         1       1     R7
12   6 5         1 1              R12
13     6         1 1 1                R13
14               1 1 1 1                 R14
15                 1 1 1 1
16                   1 1 1
17                     1 1
18                       1
```

RED Indicates if we look at F Field, we can bypass on this cycle



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



ARF		R	W
SB	R	R/W	W

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 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU R11,R10,1     F  D  I  X0 W
2 MUL   R5, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W
3 MUL   R7, R5, R6          F  D  D  D  I  I  I  I  Y0 Y1 Y2 Y3 W
4 ADDIU R12,R11,1              F  F  F  D  D  D  D  I  X0 W
5 ADDIU R13,R12,1                       F  F  F  F  D  I  X0 W
6 ADDIU R14,R12,2                                   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   □         R1
4                 1   1          R11
5                   1   1
6    3 2              1
7               1       1            R5
8                 1
9                   1
10   4 3              1
11   5 4        1   □   1     R7
12   6 5          1 □ 1          R12
13                  1 1 1            R13
14     6            □ 1 1
15                    1 1        R15
16                      1
17
18
```

RED Indicates if we look at F Field, we can bypass on this cycle

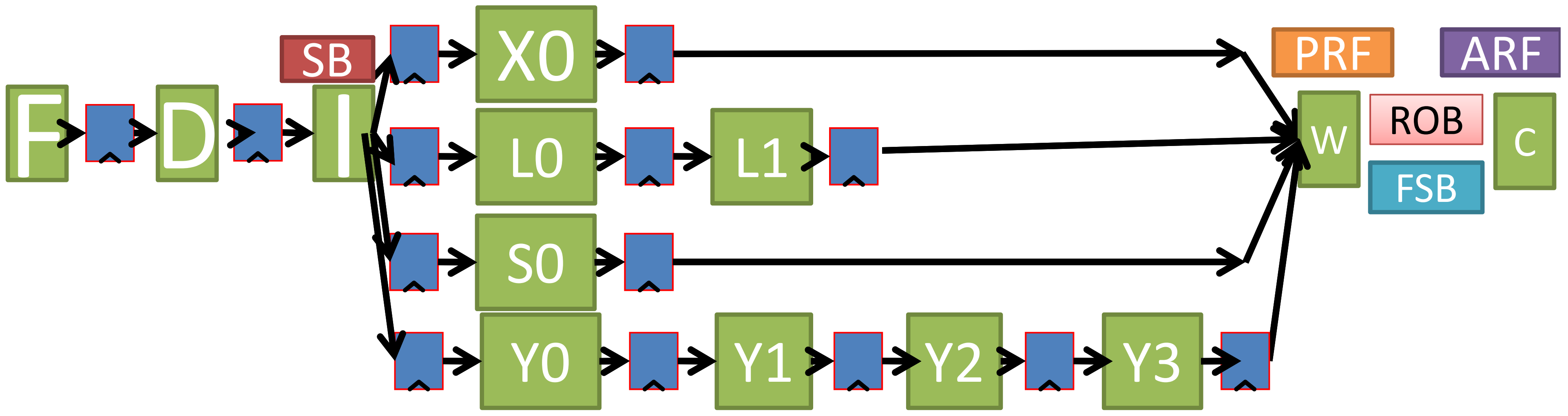
Writes with two cycle latency. Structural Hazard

Early Commit Point?

```
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 /
1 ADDIU R11,R10,1     F  D  I  X0 W     /
2 MUL   R5, R1, R4       F  D  I  I  I  /
3 MUL   R7, R5, R6          F  D  D  D  /
4 ADDIU R12,R11,1              F  F  F  /
5 ADDIU R13,R12,1                       /
6 ADDIU R14,R12,2
```

- Limits certain types of exceptions.

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				
P				
--				
--				

State: {Free, Pending, Finished}
S: Speculative
ST: Store bit
V: Physical Register File Specifier Valid
Preg: Physical Register File Specifier

Reorder Buffer (ROB)

State	S	ST	V	Preg
--				
P	1			
F	1			
P	1			
P				
F				
P				
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: {Free, Pending, Finished}
S: Speculative
ST: Store bit
V: Physical Register File Specifier Valid
Preg: Physical Register File Specifier

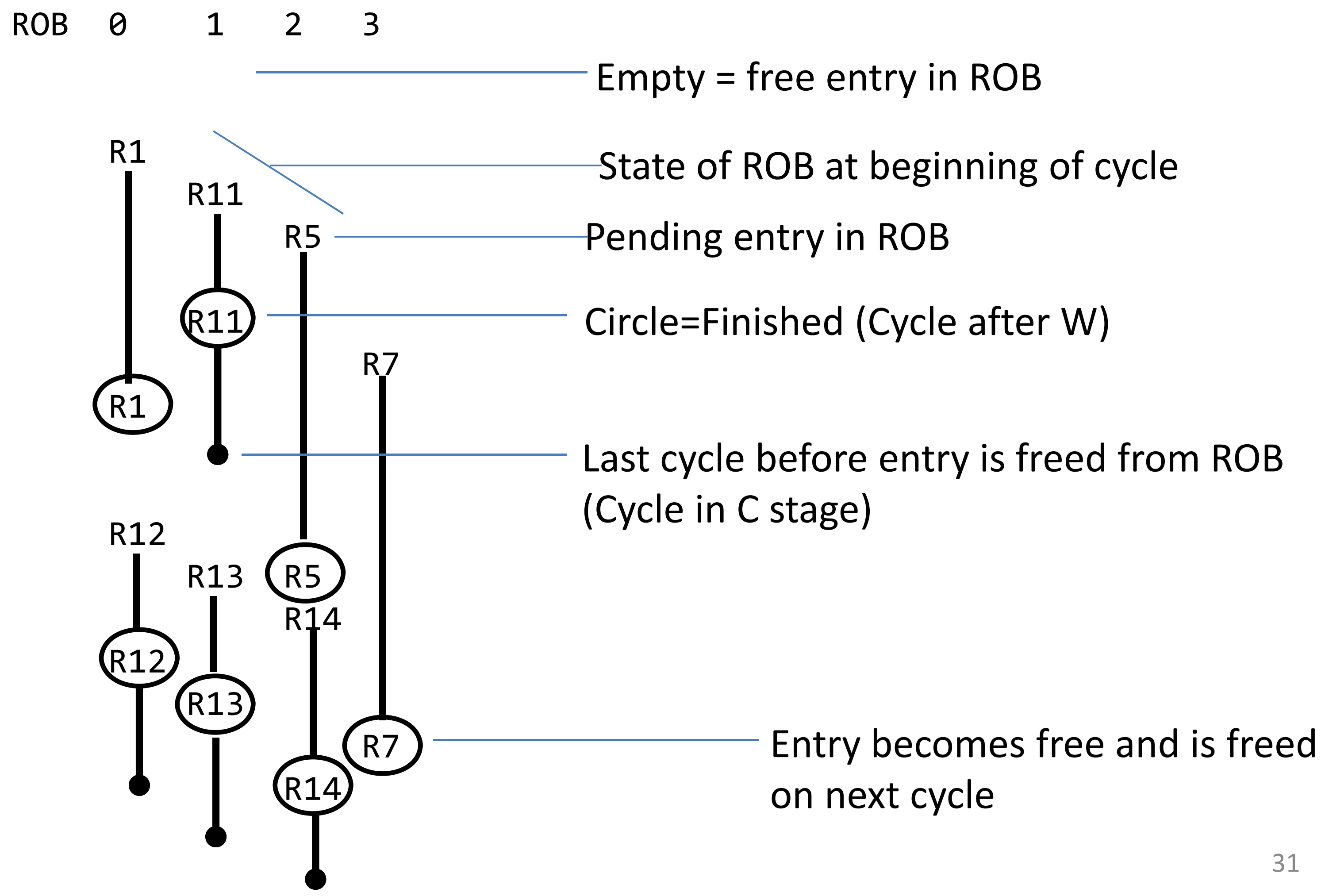
Finished Store Buffer (FSB)

V	Op	Addr	Data
--			

- Only need one entry if we only support one memory instruction 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 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU R11,R10,1     F  D  I  X0 W  r        C
2 MUL   R5, R1, R4       F  D  I  I  I  Y0 Y1 Y2 Y3 W  C
3 MUL   R7, R5, R6          F  D  D  D  I  I  I  I  Y0 Y1 Y2 Y3 W  C
4 ADDIU R12,R11,1              F  F  F  D  D  D  D  I  X0 W  r        C
5 ADDIU R13,R12,1                       F  F  F  F  D  I  X0 W  r        C
6 ADDIU R14,R12,2                                   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
R1
R11
R5
R11
R7
R1
R12
R13
R5
R14
R12
R13
R7
R14
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   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  /
1 ADDIU R11,R10,1     F  D  I  X0 W  r  -- /
2 MUL   R5, R1, R4       F  D  I  I  I  Y0 /
3 MUL   R7, R5, R6          F  D  D  D  I  /
4 ADDIU R12,R11,1              F  F  F  D  /
                                           F  D  I. . .
```

What About Branches?

```
Option 2
0 BEQZ  R1, target F  D  I  X0 W  C
1 ADDIU R11,R10,1     F  D  I  X0 /
2 ADDIU R5, R1, R4       F  D  I  /
3 ADDIU R7, R5, R6          F  D  /
T ADDIU R12,R11,1              F  D  I . . .
```

Squash instructions in ROB when Branch commits

```
Option 1
0 BEQZ  R1, target F  D  I  X0 W  C
1 ADDIU R11,R10,1     F  D  I  -
2 ADDIU R5, R1, R4       F  D  -
3 ADDIU R7, R5, R6          F  -
T ADDIU R12,R11,1              F  D  I . . .
```

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

```
Option 3
0 BEQZ  R1, target F  D  I  X0 W  C
1 ADDIU R11,R10,1     F  D  I  X0 W  /
2 ADDIU R5, R1, R4       F  D  I  X0 W  /
3 ADDIU R7, R5, R6          F  D  I  X0 W  /
T ADDIU R12,R11,1              F  D  I  X0 W  C
```

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

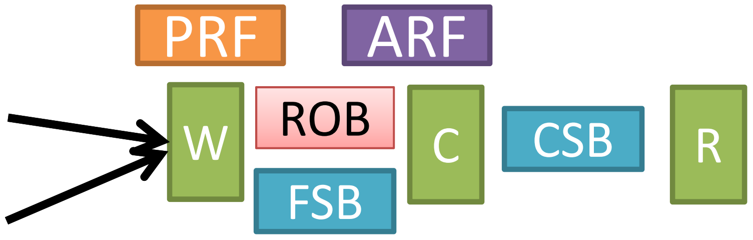
What About Branches?

- 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

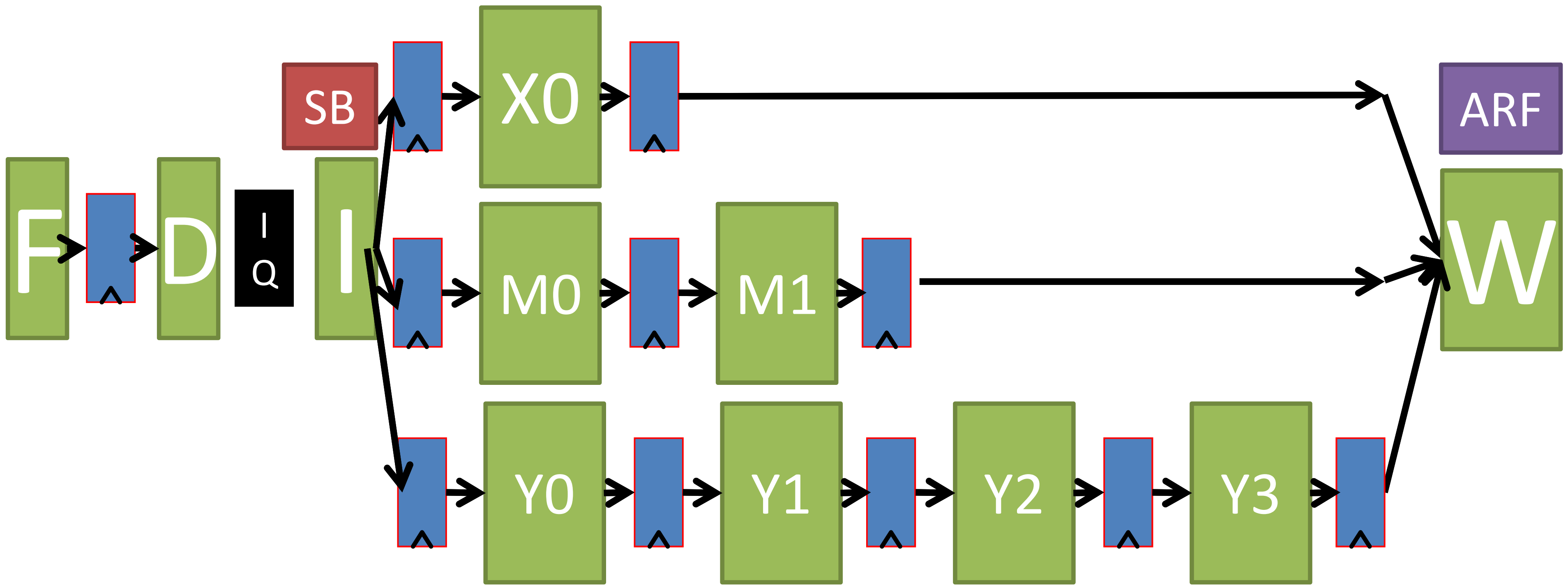


CSB=Committed Store Buffer

```
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 Recovery 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	R/W	W
IQ	W	R/W	W

Issue Queue (IQ)

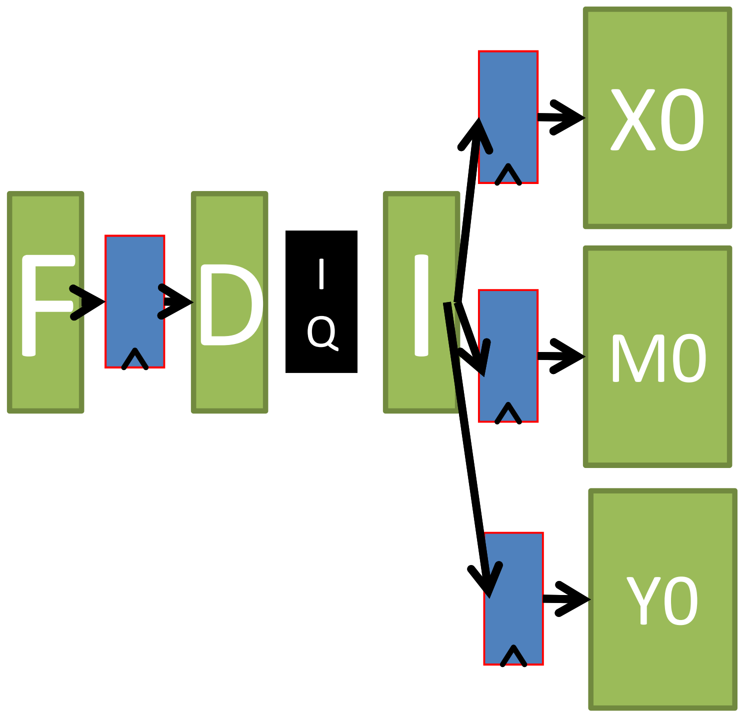
Op	Imm	S	V	Dest	V	P	Src0	V	P	Src1

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

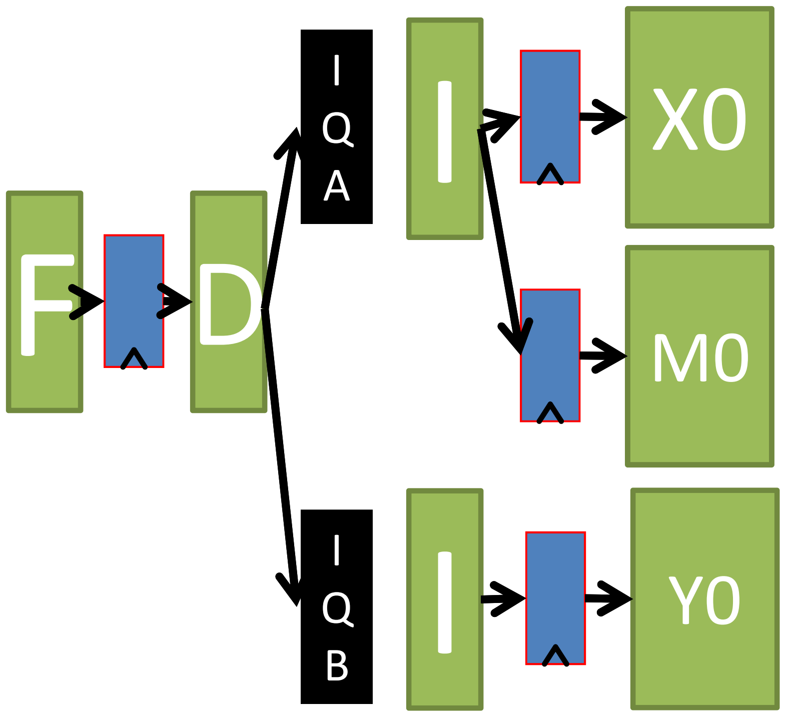
Instruction Ready = (!Vsrc0 || !Psrc0) && (!Vsrc1 || !Psrc1) && no structural hazards

- For high performance, factor in bypassing

Centralized vs. Distributed Issue Queue



Centralized



Distributed

Advanced Scoreboard

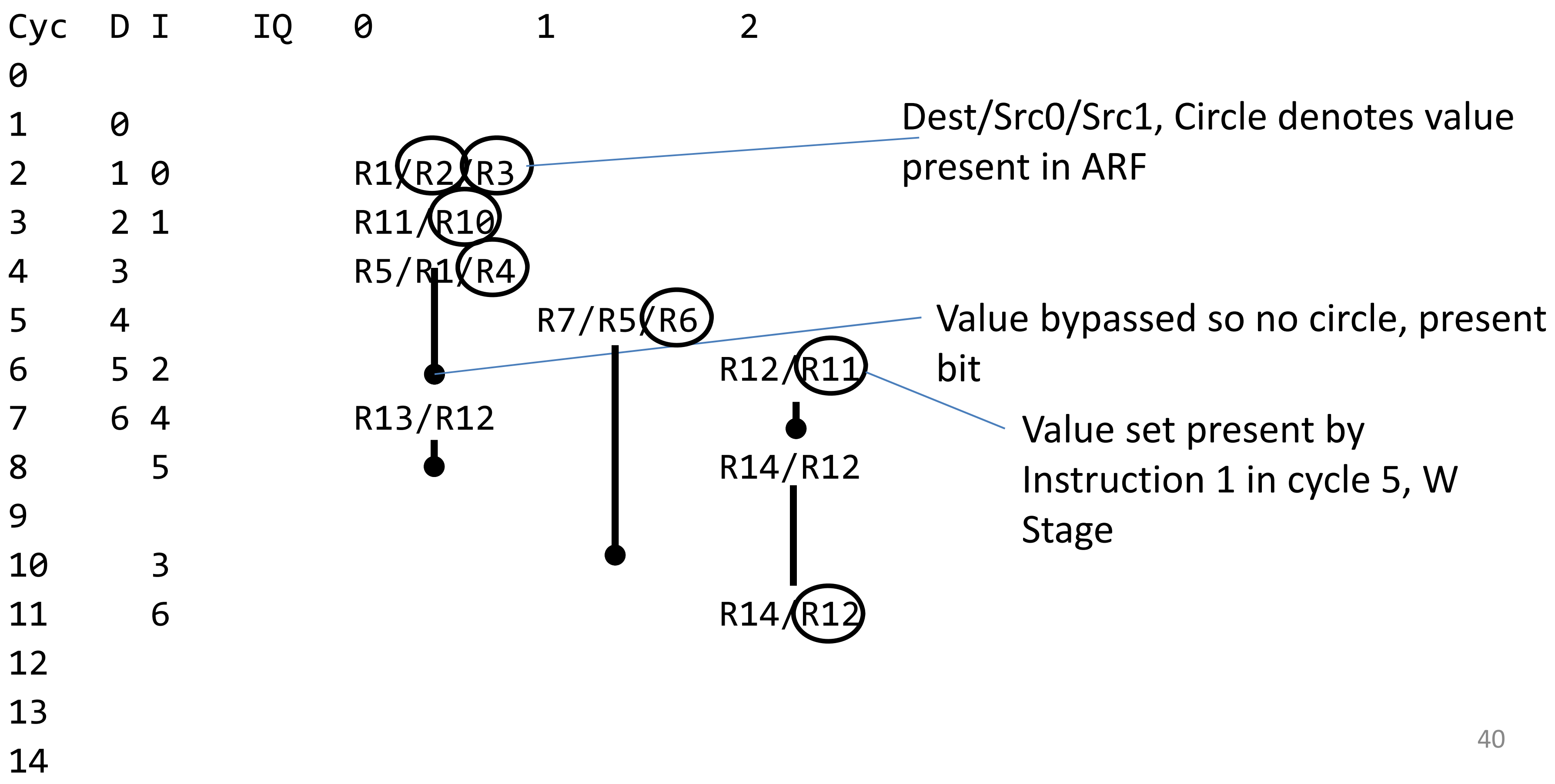
Data Avail.

	P	4	3	2	1	0
R1						
R2						
R3						
…						
R31						

P: Pending, Write to Destination in flight
Data Avail.: Where is the write data in the pipeline and which functional unit

- Data Avail. now contains functional unit identfier
- A non-empty value 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
0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W
1 ADDIU R11,R10,1     F  D  I  X0 W
2 MUL   R5, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W
3 MUL   R7, R5, R6          F  D  i              I  Y0 Y1 Y2 Y3 W
4 ADDIU R12,R11,1              F  D  i  I  X0 W
5 ADDIU R13,R12,1                 F  D  i  I  X0 W
6 ADDIU R14,R12,2                    F  D  i        I  X0 W
```



Dest/Src0/Src1, Circle denotes value present in ARF

Value bypassed so no circle, present bit

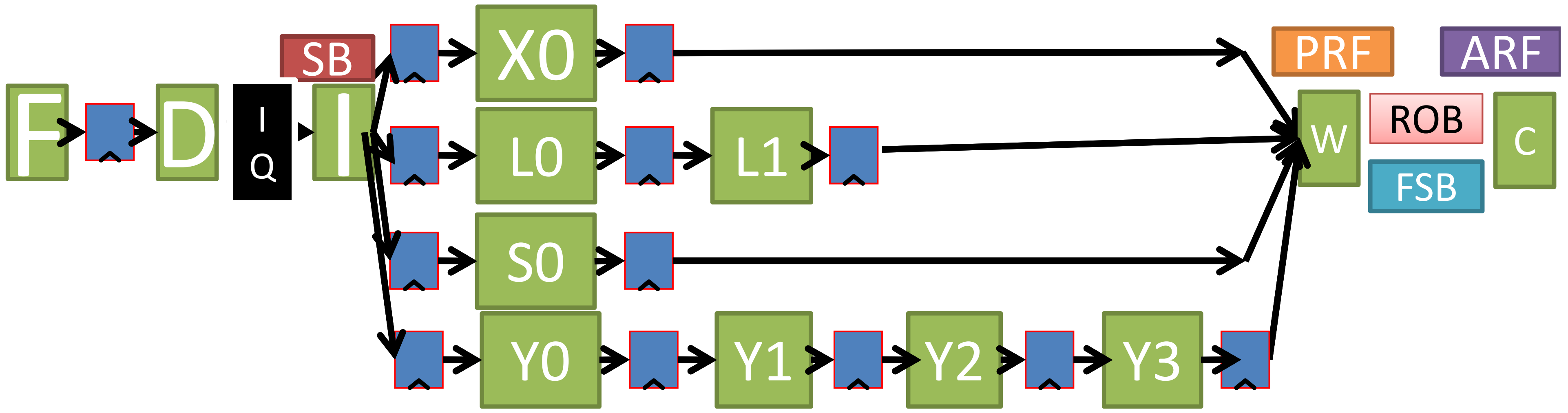
Value set present by Instruction 1 in cycle 5, W Stage

Assume All Instruction in Issue Queue

```
                                       0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15
0 MUL   R1, R2, R3 F  D  i                   I  Y0 Y1 Y2 Y3 W
1 ADDIU R11,R10,1     F  D  i                   I  X0 W
2 MUL   R5, R1, R4       F  D  i                         I  Y0 Y1 Y2 Y3 W
3 MUL   R7, R5, R6          F  D  i                                  I  Y0 Y1 Y2 Y3 W
4 ADDIU R12,R11,1              F  D  i             I  X0 W
5 ADDIU R13,R12,1                 F  D  i                   I  X0 W
6 ADDIU R14,R12,2                    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   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU R11,R10,1     F  D  I  X0 W  r        C
2 MUL   R5, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W  C
3 MUL   R7, R5, R6          F  D  i              I  Y0 Y1 Y2 Y3 W  C
4 ADDIU R12,R11,1              F  D  i  I  X0 W  r                    C
5 ADDIU R13,R12,1                 F  D  i  I  X0 W  r                    C
6 ADDIU R14,R12,2                    F  D  i        I  X0 W  r              C

0 MUL   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU R11,R10,1     F  D  I  X0 W  r        C
2 MUL   R5, R1, R4       F  D  i     I  Y0 Y1 Y2 Y3 W  C
3 MUL   R7, R5, R6          F  D  i              I  Y0 Y1 Y2 Y3 W  C
4 ADDIU R12,R11,1              F  D  i  I  X0 W  r                    C
5 ADDIU R13,R12,1                 F  D  i  I  X0 W  r                    C
6 ADDIU R14,R12,2                    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   R1, R2, R3 F  D  I  Y0 Y1 Y2 Y3 W  C
1 ADDIU R11,R10,1  F  D  I  X0 W  r           C
2 MUL   R5, R1, R4    F  D  i        I  Y0 Y1 Y2 Y3 W  C
3 MUL   R7, R5, R6    F  D  i                    I  Y0 Y1 Y2 Y3 W  C
4 ADDIU R12,R11,1        F  D  I  X0 W  r                             C
5 ADDIU R13,R12,1        F  D  i  I  X0 W  r                             C
6 ADDIU R14,R12,2           F  D  i  I  X0 W  r                             C
```

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)

- MIT material derived from course 6.823
- UCB material derived from course CS252 & CS152
- Cornell material derived from course ECE 4750