



CPEN 411: Computer Architecture

Slide Set #6: Multicycle Operations

Instructor: Mieszko Lis

Original Slides: Professor Tor Aamodt



Learning Objectives

- By the end of this slide set, be able to:
 - Describe three types of *data dependencies* and *hazards* and explain the difference between a data dependency and a data hazard
 - Describe the challenges of adding multicycle execution units to a pipelined processor
 - Explain why hazard penalties are larger in the 8-stage MIPS R4000 pipeline



Data Dependencies and Hazards

Data dependence: restriction on the order instructions can execute so they compute the result programmer specified.

Data hazard: ordering of instruction execution that violates a data dependency.

“**Dependence**” = A property of the program completely independent of which processor it runs on.

“**Hazard**” = a property exhibited by a program running on a particular microprocessor microarchitecture.



True Dependence

- Below, Instruction “J” has a **true data dependence** on Instruction “I” since “I” writes to R1, then “J” reads from R1:

I : DADD R1 , R2 , R3

J : DSUB R4 , R1 , R3

- “true” dependence because a value was actually *communicated*.
- A **true dependence** leads to a “**read-after-write**” (**RAW**) hazard if the hardware can perform the read before the write.

	Clock Number					
	1	2	3	4	5	6
DADD R1,R2,R3	F	D	X	M	W	
DSUB R4,R1,R3		F	D	X	M	W



Data Dependence and Hazards



Data Dependence and Hazards

- A hazard is created whenever there is a dependence between instructions **AND** they are close enough that the overlap of execution would change the order of access to the operand involved in the dependence.



Data Dependence and Hazards

- A hazard is created whenever there is a dependence between instructions **AND** they are close enough that the overlap of execution would change the order of access to the operand involved in the dependence.
- Presence of dependence indicates potential for a hazard, but actual hazard and length of any stall is a property of the **pipeline**



Name Dependence

- A **name dependence** occurs when two instructions use same register or memory location but **no communication between the instructions is intended**.
- Dependence is on **data location (name)** not value (contents at location).



Name Dependence #1: Anti-dependence

- Below, instruction “I” should read R1 before instruction “J” writes to R1. Reordering “I” and “J” would change result computed by “I”.

```
I: DSUB R4, R1, R3
J: DADD R1, R2, R3
K: XOR R6, R1, R7
```

This is called an “anti-dependence”.
It results from reuse of the “name” “R1”.

- If anti-dependence causes a hazard in the pipeline, it is called a Write After Read (WAR) hazard



Name Dependence #2: Output dependence

- Below, instruction “I” should write to R1 before instruction “J” writes to R1. Reordering I and J would leave the wrong value in R1 (causing K to compute the wrong value):

```
I : DSUB  R1 , R4 , R3
J : DADD  R1 , R2 , R3
K : XOR   R6 , R1 , R7
```

- Called an “output dependence”
This also results from the reuse of name “R1”
- If an output dependence causes a hazard in the pipeline, called a **Write After Write (WAW) hazard**



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

- A: 5 or fewer
- B: 6
- C: 7
- D: 8
- E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

A: 5 or fewer

B: 6

C: 7 ✓

D: 8

E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

A: 5 or fewer

B: 6

C: 7 ✓

D: 8

E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6,34(R2)
2  L.D      F2,45(R3)
3  MUL.D    F0,F2,F4
4  SUB.D    F8,F6,F2
5  DIV.D    F10,F0,F6
6  ADD.D    F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

A: 5 or fewer

B: 6

C: 7 ✓

D: 8

E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

A: 5 or fewer

B: 6

C: 7 ✓

D: 8

E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6, 34(R2)
2  L.D      F2, 45(R3)
3  MUL.D    F0, F2, F4
4  SUB.D    F8, F6, F2
5  DIV.D    F10, F0, F6
6  ADD.D    F6, F8, F2
```

True dependencies
(possibly leading to
RAW hazards)

How many true
dependencies are there?

A: 5 or fewer

B: 6

C: 7 ✓

D: 8

E: 9 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6, 34(R2)
2  L.D      F2, 45(R3)
3  MUL.D    F0, F2, F4
4  SUB.D    F8, F6, F2
5  DIV.D    F10, F0, F6
6  ADD.D    F6, F8, F2
```

The diagram illustrates dependencies between instructions 1 through 6. True dependencies (red arrows) exist from instruction 1 to 4, 5, and 6; from instruction 2 to 3 and 6; and from instruction 3 to 4. Anti dependencies (blue arrows) exist from instruction 1 to 2 and 4, and from instruction 2 to 4. Output dependencies (green arrows) exist from instruction 4 to 5 and from instruction 5 to 6.

True dependencies
(possibly leading to
RAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D      F6, 34(R2)
2  L.D      F2, 45(R3)
3  MUL.D    F0, F2, F4
4  SUB.D    F8, F6, F2
5  DIV.D    F10, F0, F6
6  ADD.D    F6, F8, F2
```

True dependencies
(possibly leading to
RAW hazards)

Anti dependencies
(possibly leading to
WAR hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)

How many anti
dependencies are
there?

A: none

B: 1

C: 2

D: 3

E: 4 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)

How many anti
dependencies are
there?

A: none

B: 1

C: 2 ✓

D: 3

E: 4 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)

How many anti
dependencies are
there?

A: none

B: 1

C: 2 ✓

D: 3

E: 4 or more



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)

Output dependencies
(possibly leading to
WAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

True dependencies
(possibly leading to
RAW hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Anti dependencies
(possibly leading to
WAR hazards)

```
1  L.D    F6,34(R2)
2  L.D    F2,45(R3)
3  MUL.D  F0,F2,F4
4  SUB.D  F8,F6,F2
5  DIV.D  F10,F0,F6
6  ADD.D  F6,F8,F2
```

Output dependencies
(possibly leading to
WAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

How many output dependencies are there?

- A: none
- B: 1
- C: 2
- D: 3
- E: 4 or more

(possibly leading to RAW hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F0,F8,F2
```

Anti dependencies
(possibly leading to WAR hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Output dependencies
(possibly leading to WAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

How many output dependencies are there?

- A: none
- B: 1 ✓
- C: 2
- D: 3
- E: 4 or more

(possibly leading to RAW hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F0,F8,F2
```

Anti dependencies
(possibly leading to WAR hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Output dependencies
(possibly leading to WAW hazards)



Example Problem

Find true, anti and output dependencies in this code:

How many output dependencies are there?

- A: none
- B: 1 ✓
- C: 2
- D: 3
- E: 4 or more

(possibly leading to RAW hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F0,F8,F2
```

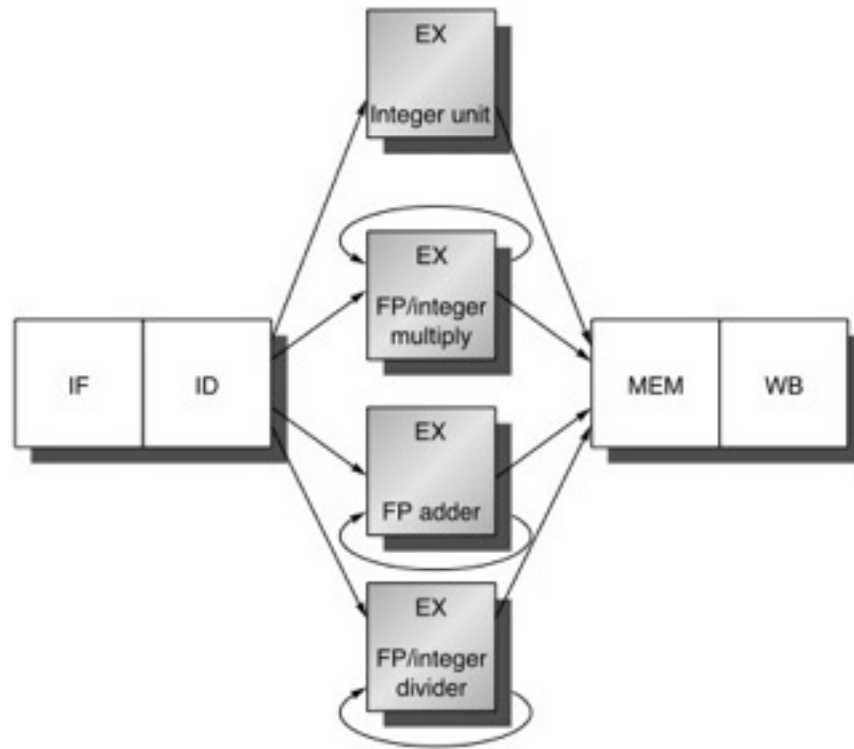
Anti dependencies
(possibly leading to WAR hazards)

```
L.D    F6,34(R2)
L.D    F2,45(R3)
MUL.D  F0,F2,F4
SUB.D  F8,F6,F2
DIV.D  F10,F0,F6
ADD.D  F6,F8,F2
```

Output dependencies
(possibly leading to WAW hazards)

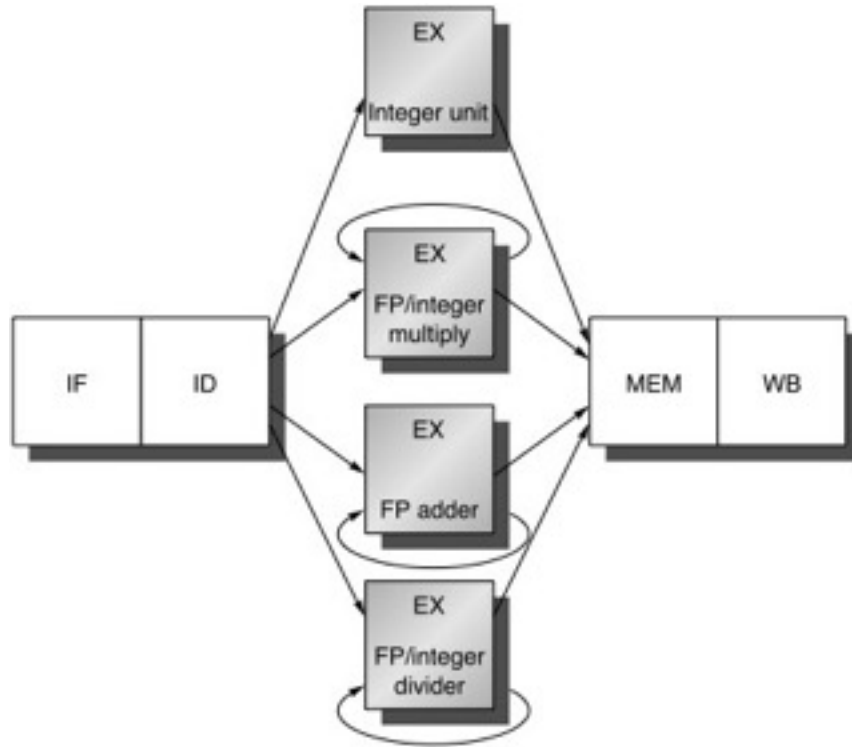


Multicycle Operations





Multicycle Op

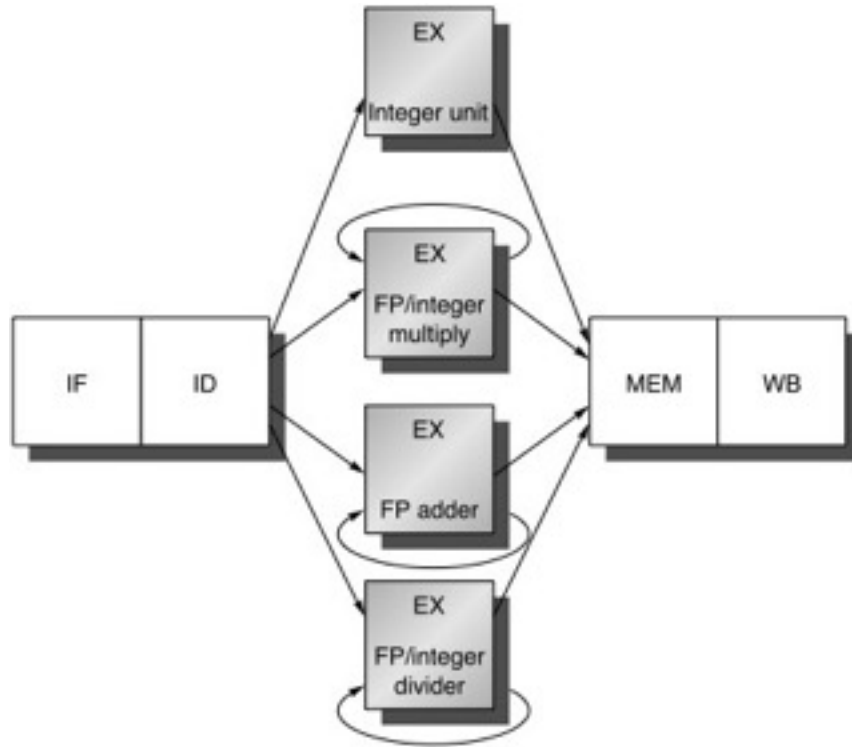


Which hazards can occur with this pipeline organization?

- A: RAW
- B: WAR
- C: WAR and RAW
- D: WAW and RAW
- E: RAW, WAR and WAW



Multicycle Op



Which hazards can occur with this pipeline organization?

A: RAW

B: WAR

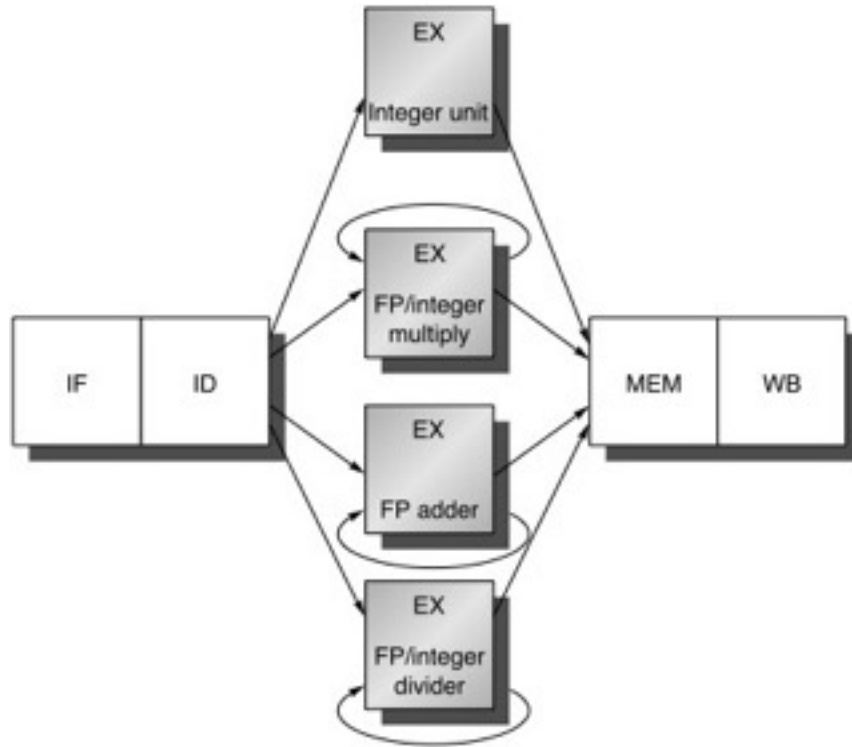
C: WAR and RAW

D: WAW and RAW ✓

E: RAW, WAR and WAW

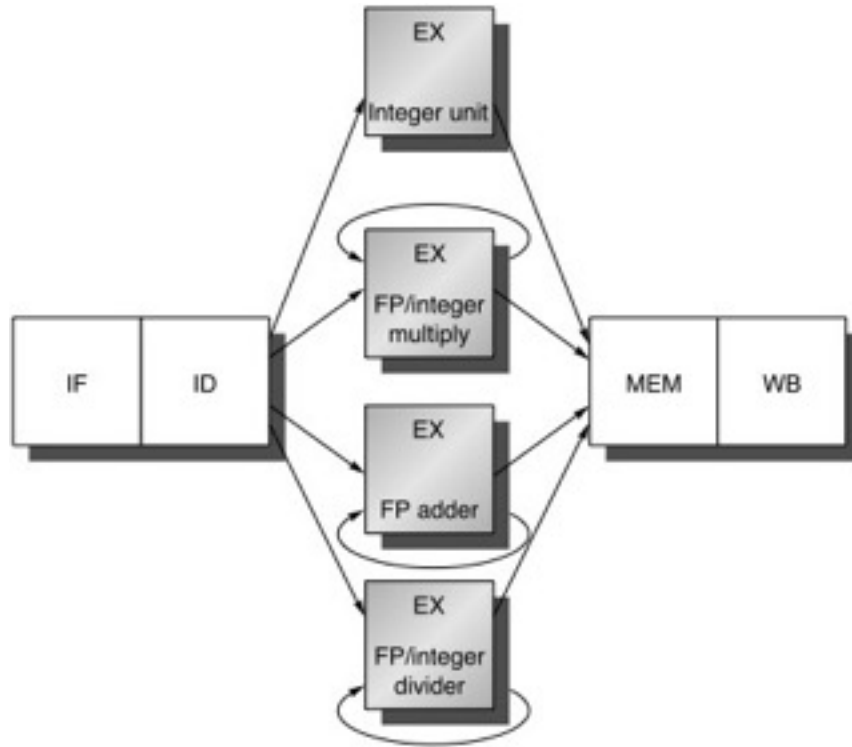


Multicycle Operations





Multicycle Operations

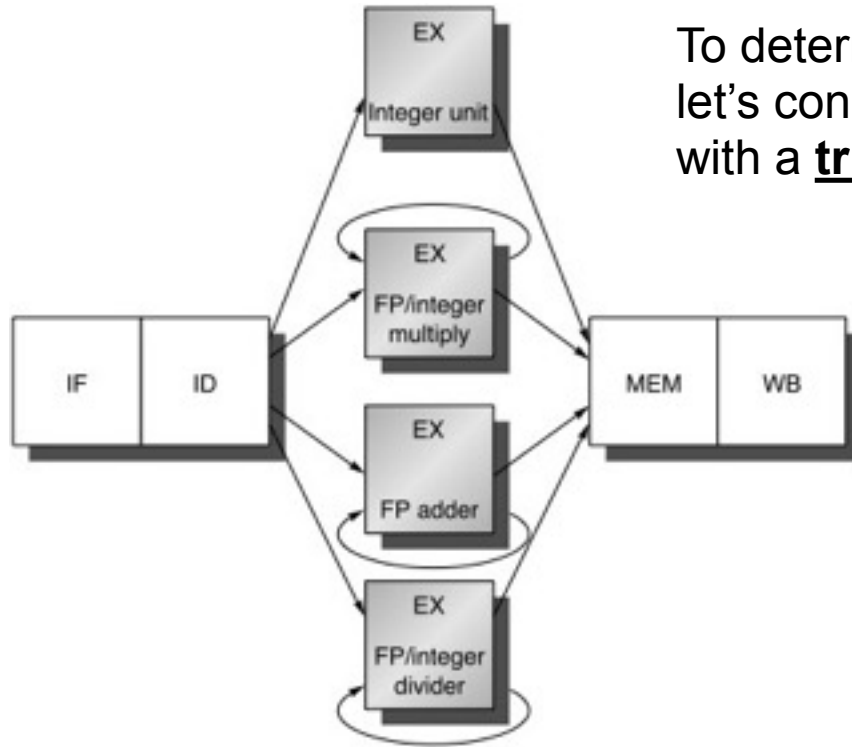


- RAW hazards?



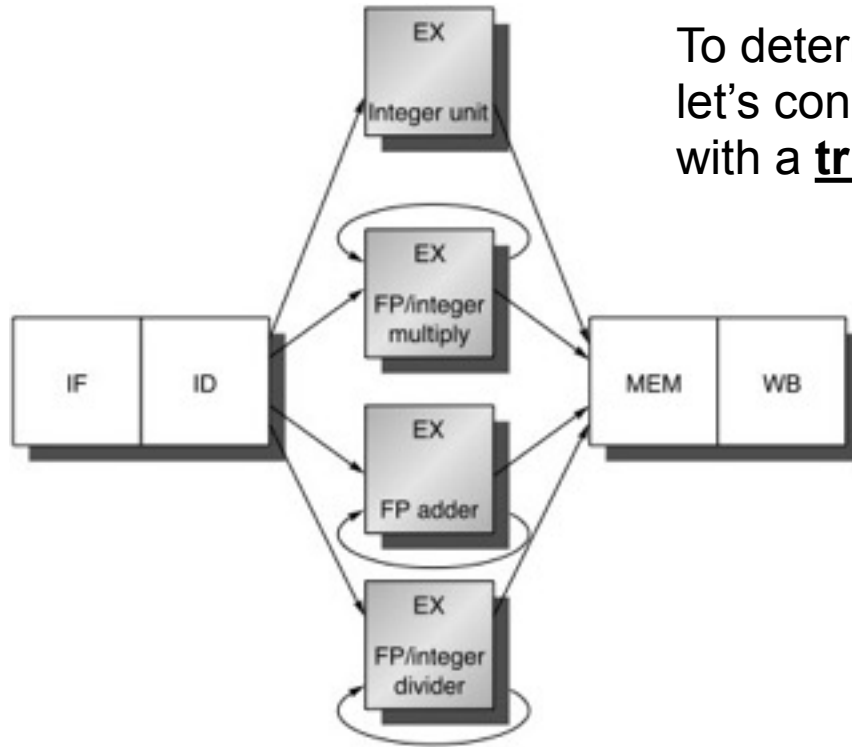
Multicycle Operations

To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:



- RAW hazards?

Multicycle Operations

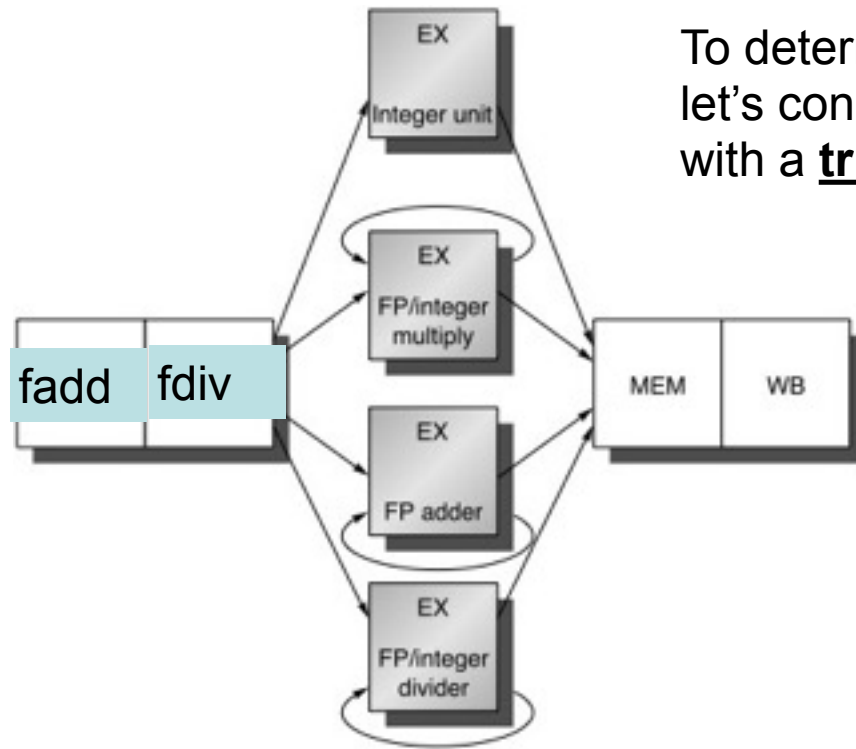


To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:

```
i: fdiv F1,...,...
j: fadd ...,F1,...
```

- RAW hazards?

Multicycle Operations

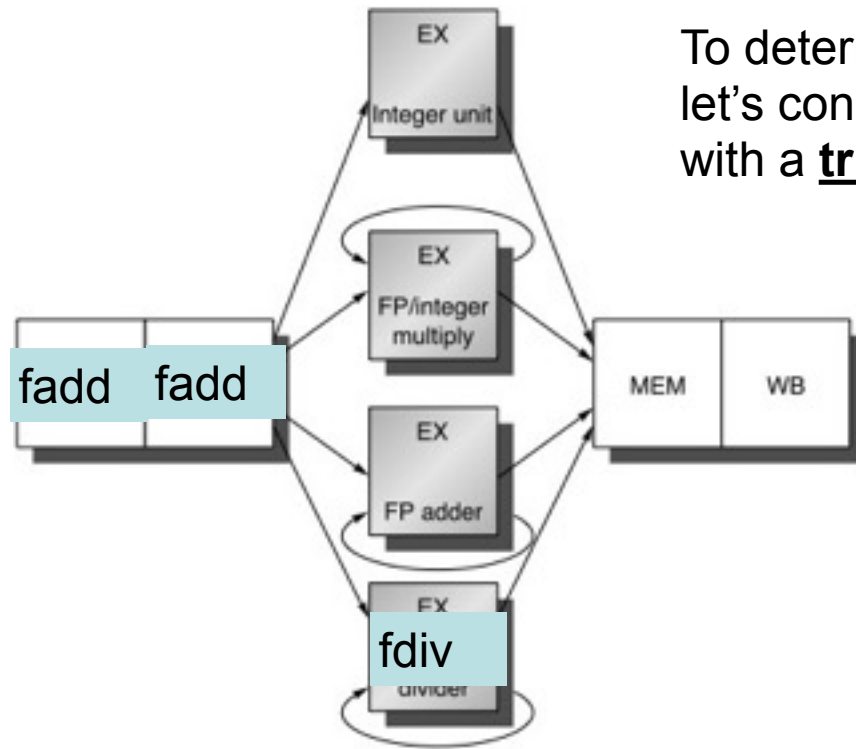


To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:

```
i: fdiv  F1, ..., ...
j: fadd  ..., F1, ...
```

- RAW hazards?

Multicycle Operations

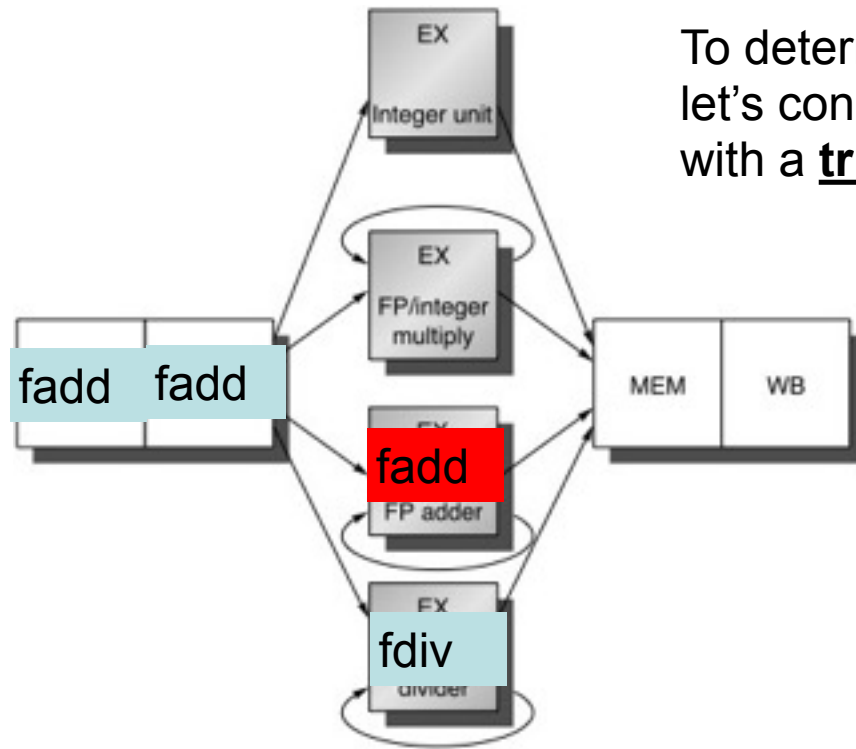


To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:

```
i: fdiv  F1,...,...
j: fadd  ...,F1,...
```

- RAW hazards?

Multicycle Operations

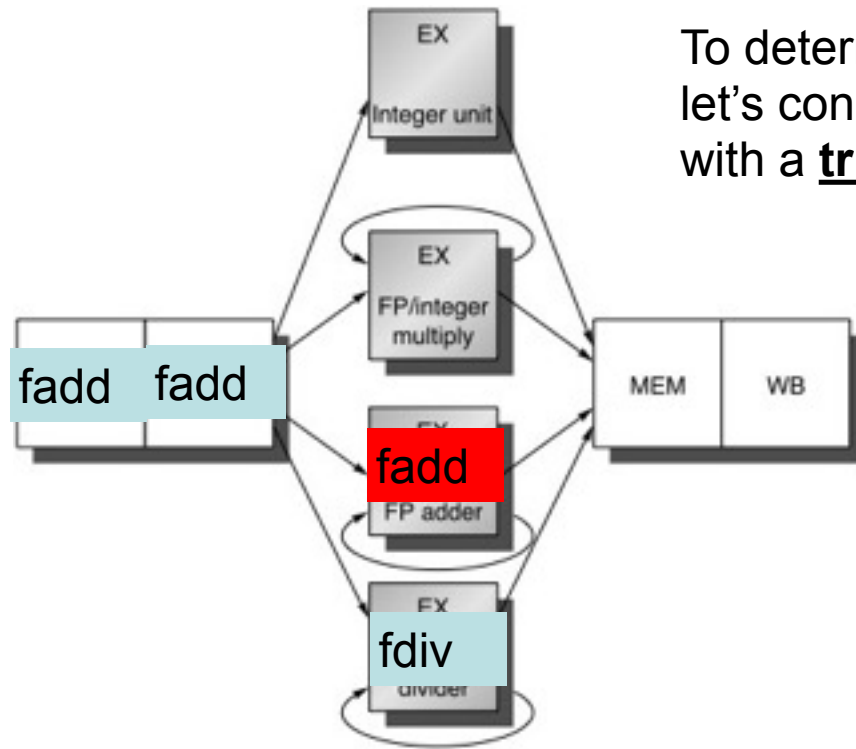


To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:

```
i: fdiv  F1, ..., ...
j: fadd  ..., F1, ...
```

- RAW hazards?

Multicycle Operations

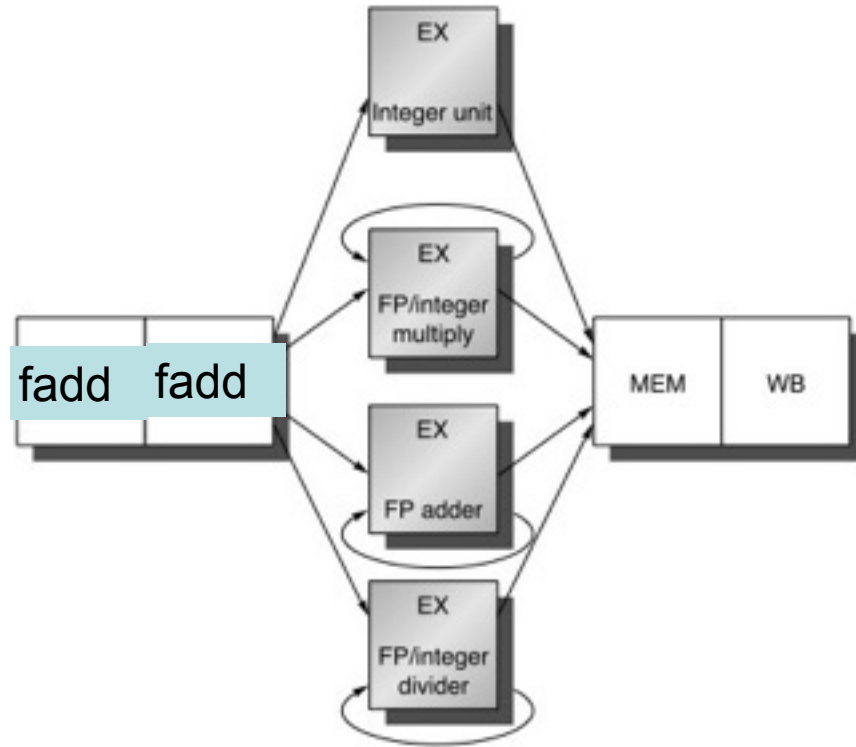


To determine if there can be a RAW hazard let's consider a sequence of two instructions with a **true dependence**:

```
i: fdiv  F1, ..., ...
j: fadd  ..., F1, ...
```

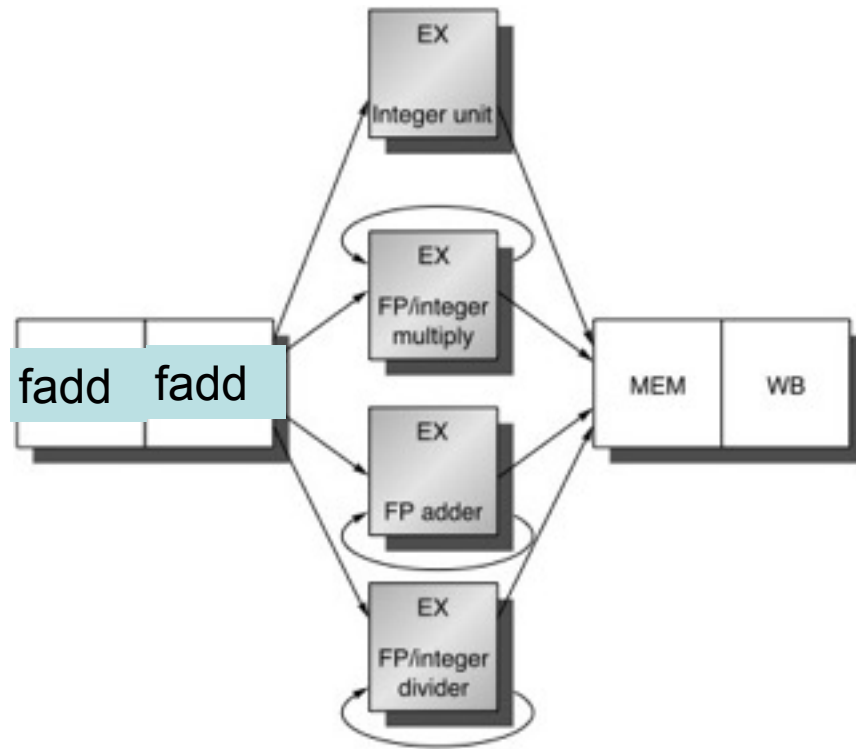
- RAW hazards? Yes

Multicycle Operations



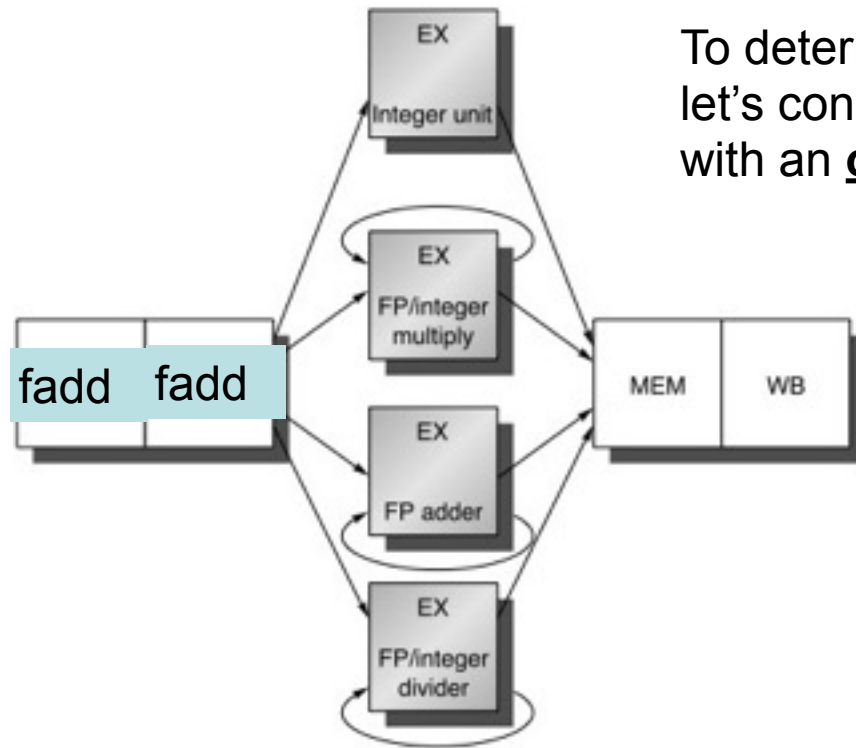
- RAW hazards? Yes

Multicycle Operations



- RAW hazards? Yes
- WAW hazards?

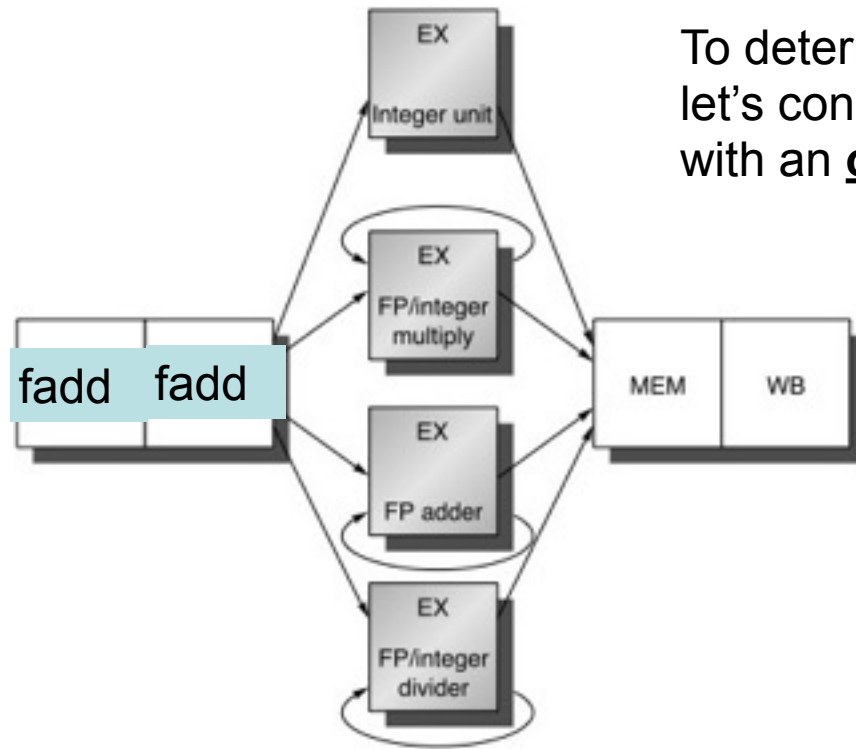
Multicycle Operations



To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

- RAW hazards? Yes
- WAW hazards?

Multicycle Operations

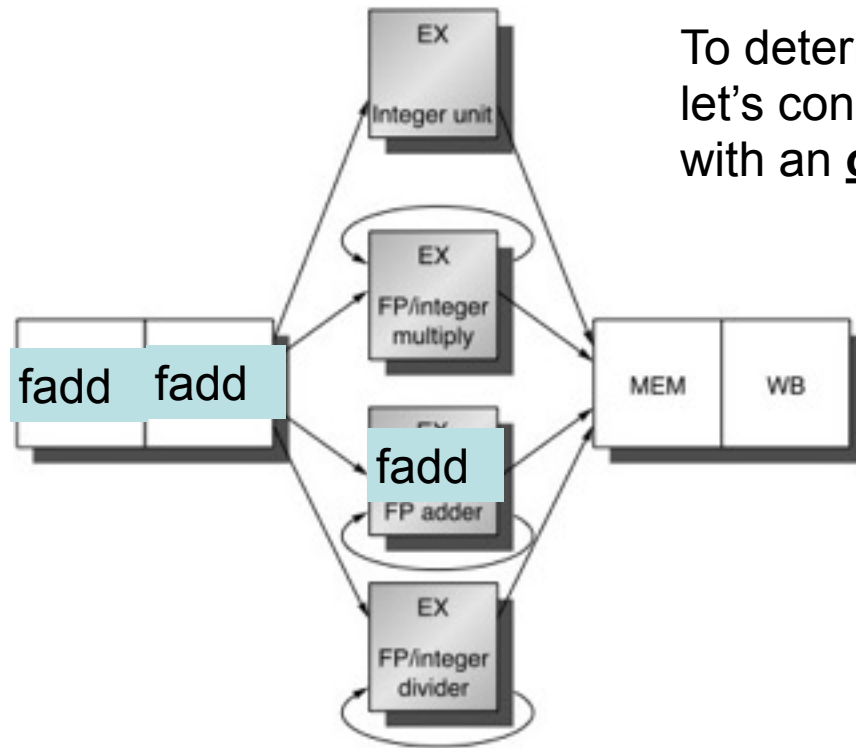


To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

```
i: fdiv F1, ..., ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards?

Multicycle Operations

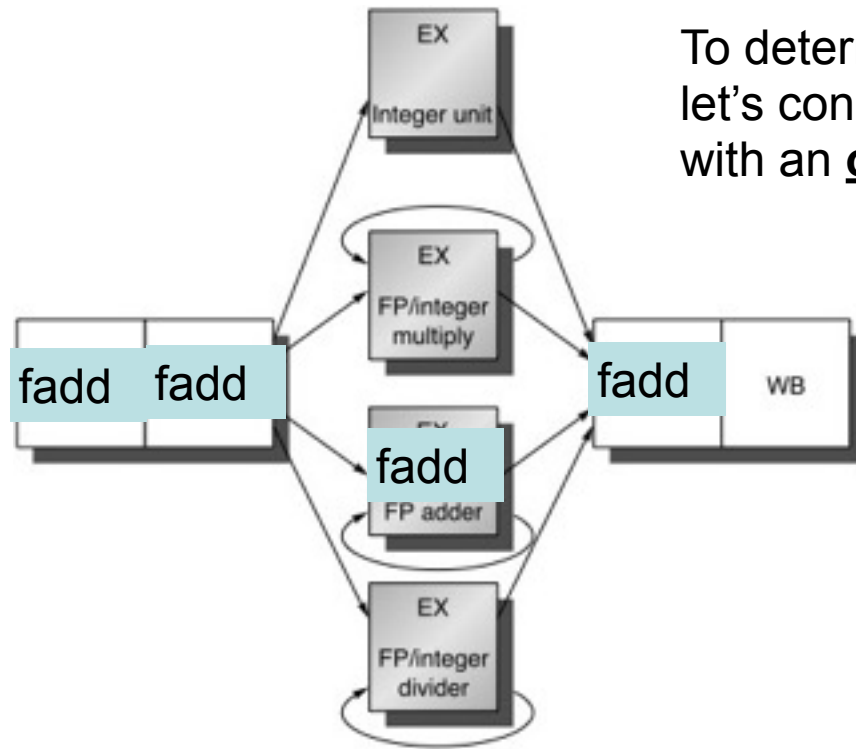


To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

```
i: fdiv F1, ..., ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards?

Multicycle Operations

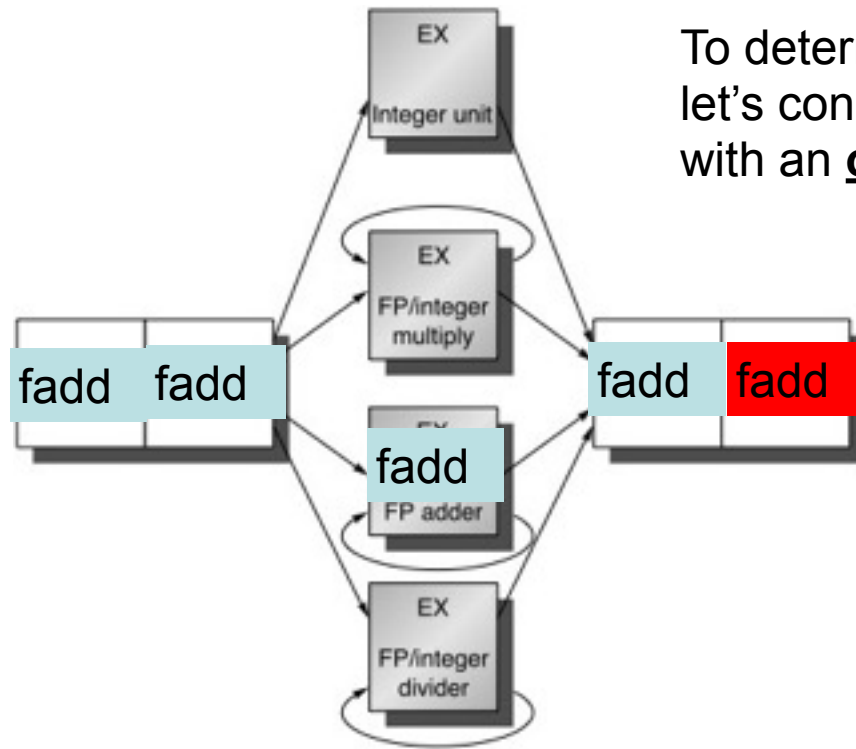


To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

```
i: fdiv F1, ..., ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards?

Multicycle Operations

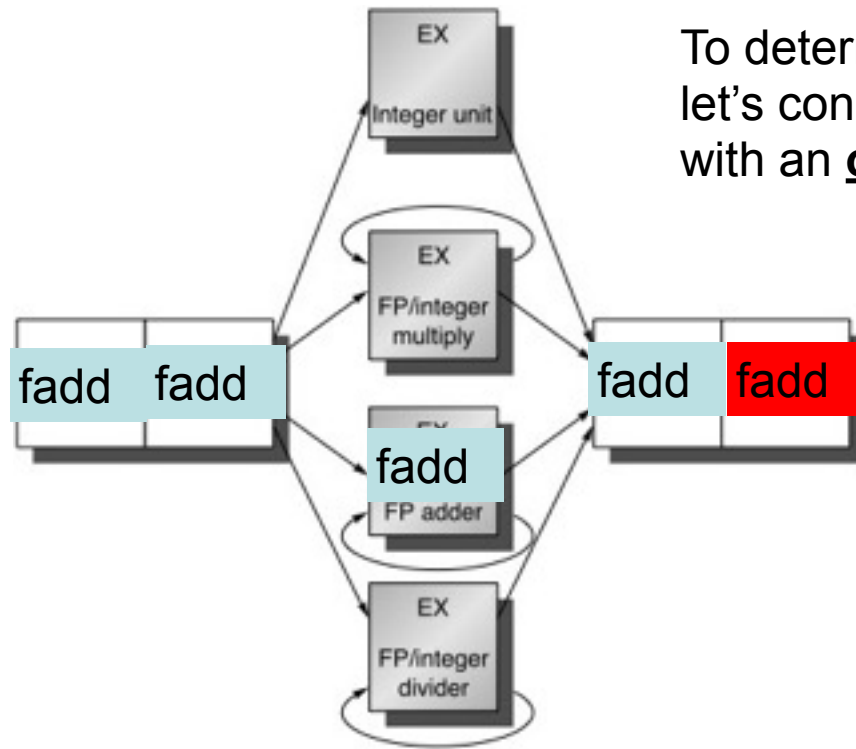


To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

```
i: fdiv F1, ..., ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards?

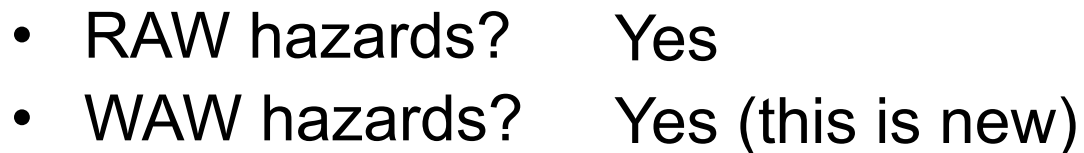
Multicycle Operations



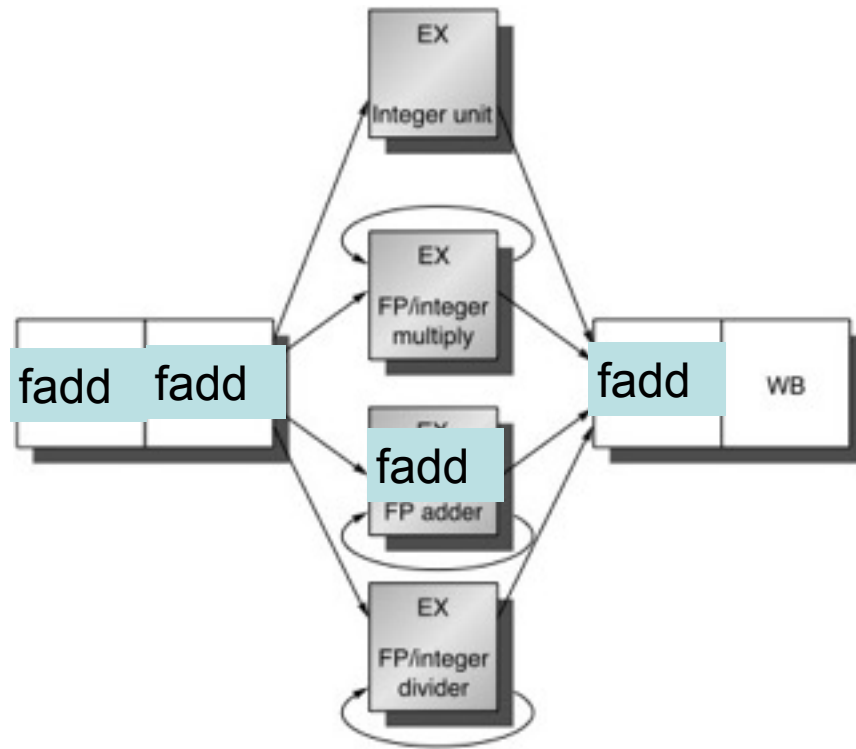
To determine if there can be a WAW hazard let's consider a sequence of two instructions with an **output dependence**:

```
i: fdiv F1, ..., ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards? Yes (this is new)

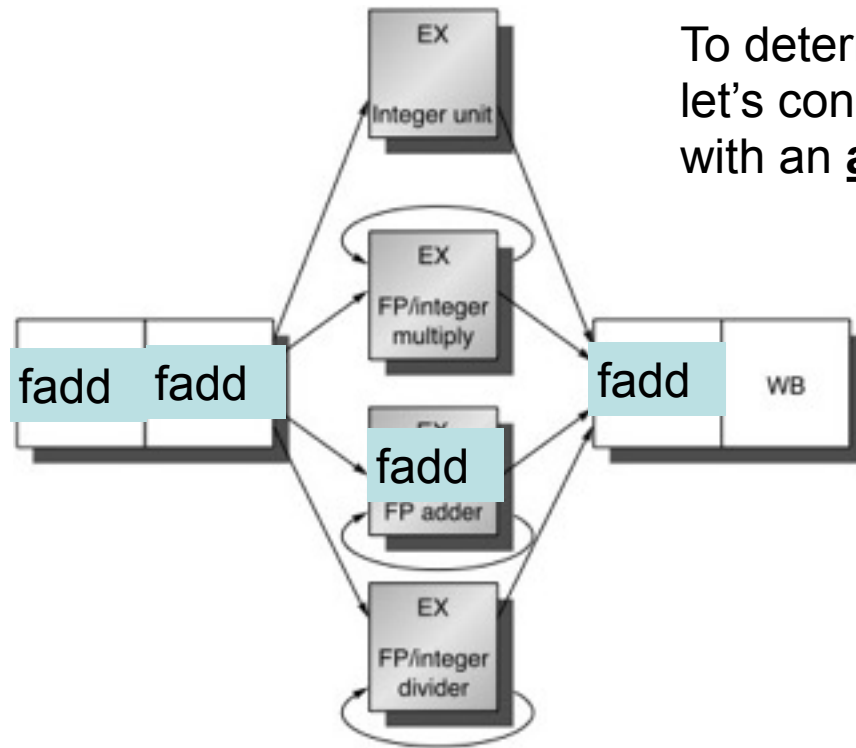


Multicycle Operations



- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards?

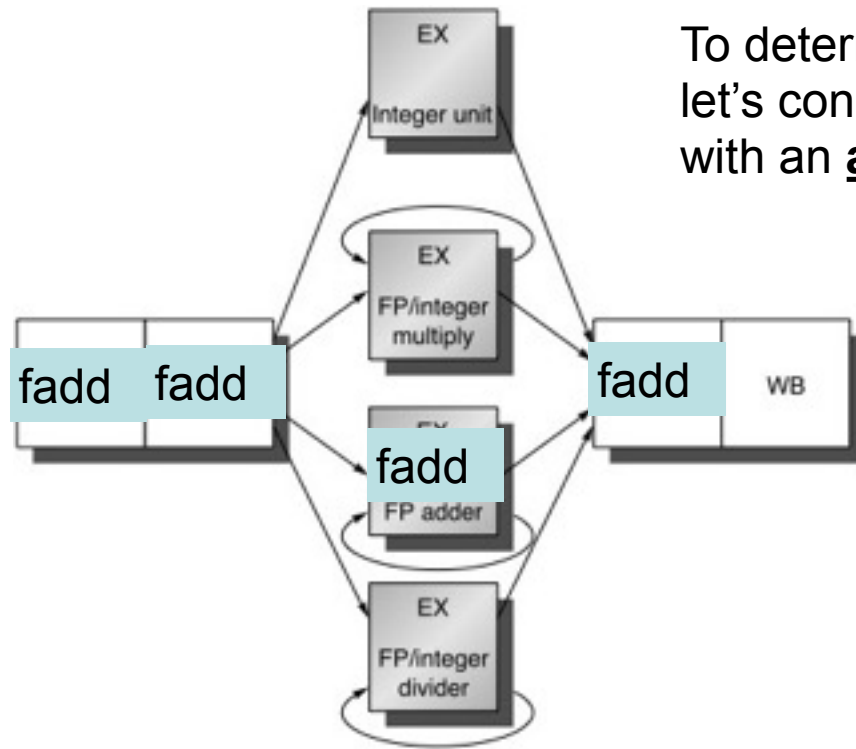
Multicycle Operations



To determine if there can be a WAR hazard let's consider a sequence of two instructions with an **anti dependence**:

- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards?

Multicycle Operations

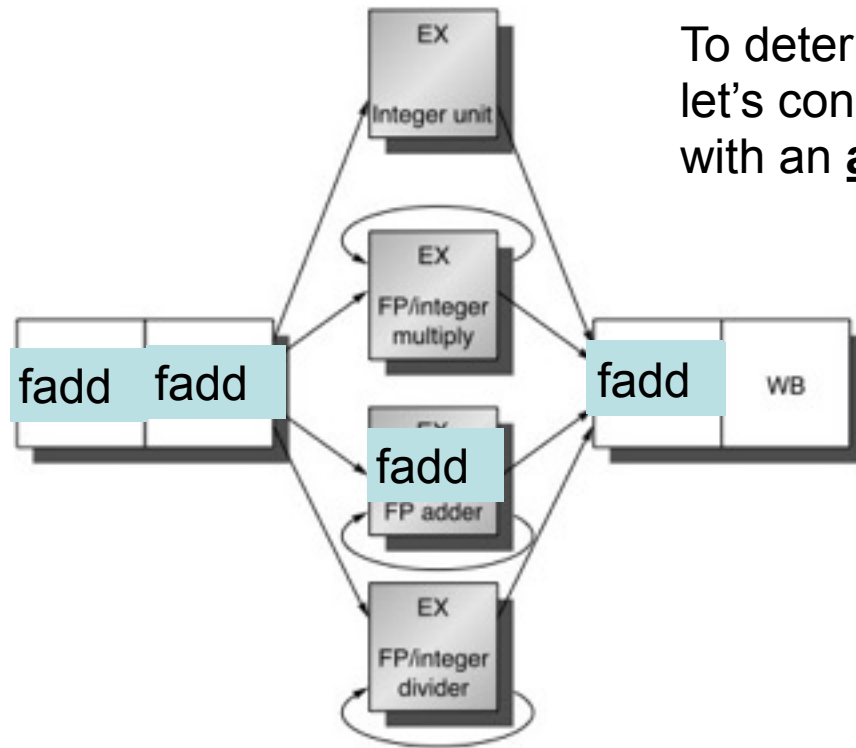


To determine if there can be a WAR hazard let's consider a sequence of two instructions with an **anti dependence**:

```
i: fdiv    ...,F1,...
j: fadd    F1,...,...
```

- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards?

Multicycle Operations

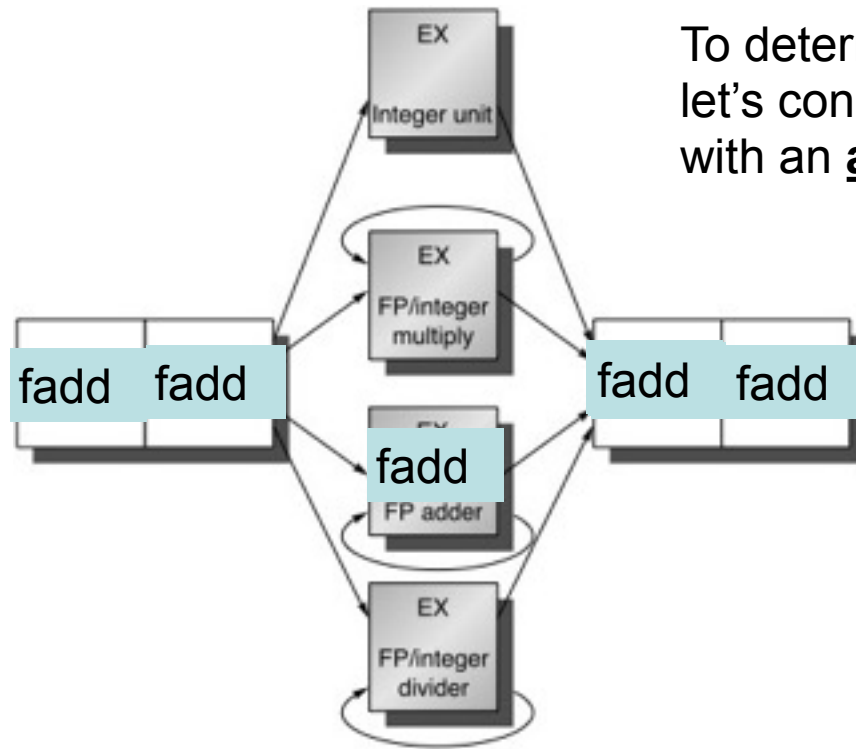


To determine if there can be a WAR hazard let's consider a sequence of two instructions with an **anti dependence**:

```
i: fdiv    ...,F1,...
j: fadd    F1,...,...
```

- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards?

Multicycle Operations

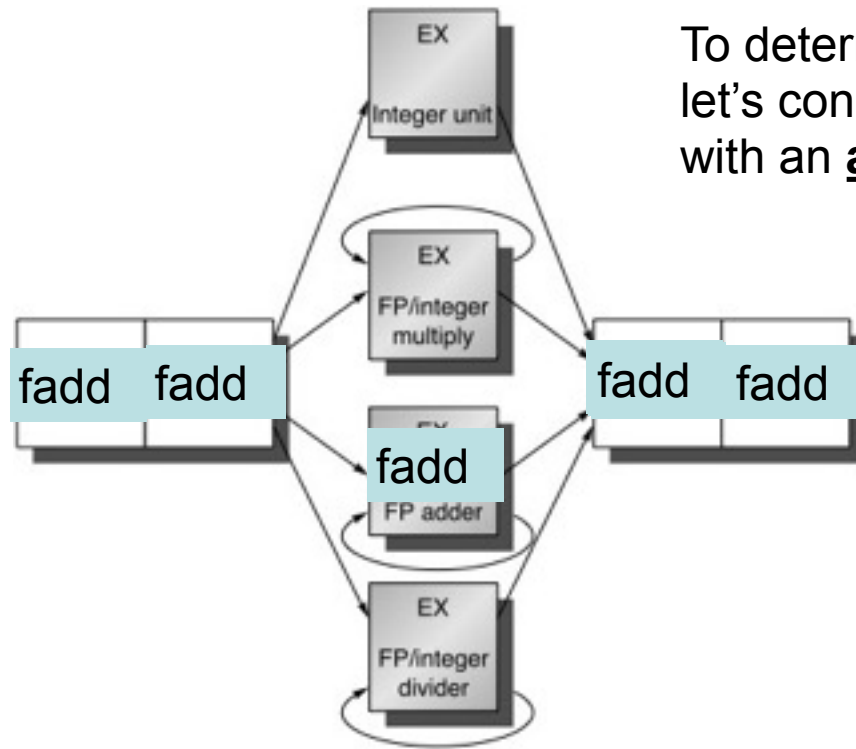


To determine if there can be a WAR hazard let's consider a sequence of two instructions with an **anti dependence**:

```
i: fdiv ..., F1, ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards?

Multicycle Operations



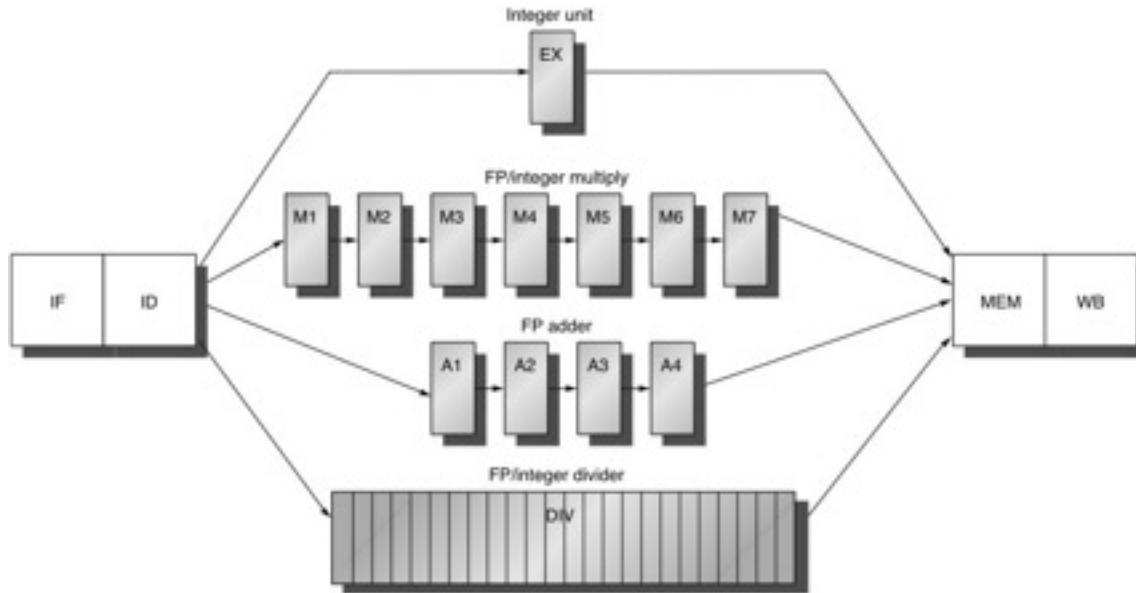
To determine if there can be a WAR hazard let's consider a sequence of two instructions with an **anti dependence**:

```
i: fdiv ..., F1, ...
j: fadd F1, ..., ...
```

- RAW hazards? Yes
- WAW hazards? Yes (this is new)
- WAR hazards? No! (why?)



Pipelined Function Units



To increase throughput, pipeline function units that are used frequently.

Challenges:

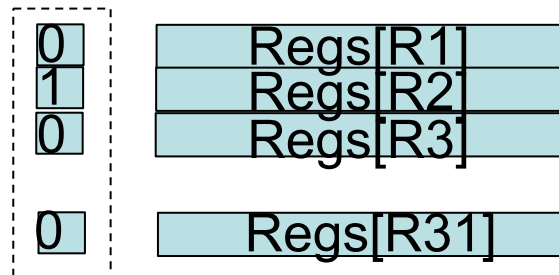
1. Longer RAW-hazards,
2. Structural hazard for MEM, WB.

	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
ADD.D		IF	ID	A1	A2	A3	A4	MEM	WB		
L.D			IF	ID	EX	MEM	WB				
S.D				IF	ID	EX	MEM	WB			



Preventing RAW and WAW hazards with in-order Multicycle Pipelines

- To prevent RAW and WAW hazards, we can add a hardware table (memory) with a single bit for each register in the register file. This bit tracks if an instruction in the pipeline will write to the associated register.
- Stall instruction in decode stage if it reads or will write a register with the bit set.
- Set bit for destination register when complete decode, clear when instruction reaches writeback.





In-order Scoreboard

- Scoreboard: a bit-array, 1-bit for each GPR
 - If the bit is *not* set: the register has valid data
 - If the bit is set: the register has stale data
i.e., some outstanding instruction is going to change it



In-order Scoreboard

- Scoreboard: a bit-array, 1-bit for each GPR
 - If the bit is *not* set: the register has valid data
 - If the bit is set: the register has stale data
i.e., some outstanding instruction is going to change it
- Issue in Order: $RD \leftarrow Fn(RS, RT)$
 - If SB[RS] or SB[RT] is set \rightarrow RAW, stall
 - If SB[RD] is set \rightarrow WAW, stall
 - Else, dispatch to FU (Fn) and set SB[RD]



In-order Scoreboard

- Scoreboard: a bit-array, 1-bit for each GPR
 - If the bit is *not* set: the register has valid data
 - If the bit is set: the register has stale data
i.e., some outstanding instruction is going to change it
- Issue in Order: $RD \leftarrow Fn(RS, RT)$
 - If SB[RS] or SB[RT] is set \rightarrow RAW, stall
 - If SB[RD] is set \rightarrow WAW, stall
 - Else, dispatch to FU (Fn) and set SB[RD]
- Complete out-of-order
 - Update GPR[RD], clear SB[RD]



RAW hazards with pipelined function units

	Clock Number																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
L.D F4,0(R2)	IF	ID	EX	M	W												
MUL.D F0,F4,F6		IF	ID	s	m1	m2	m3	m4	m5	m6	m7	M	W				
ADD.D F2,F0,F8			IF	s	ID	s	s	s	s	s	s	a1	a2	a3	a4	M	W
S.D F2,0(R2)					IF	s	s	s	s	s	s	ID	EX	s	s	s	M

(Figure A.37 in H&P 5th edition)

Above, timing assumes we stall instructions at the latest pipeline stage possible before operand value is used (rather than stalling in IF/ID).



Structural Hazards

	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB



Structural Hazards

	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

- In theory, only L.D uses MEM on cycle 10, but definitely MUL.D, ADD.D and L.D all want to use WB on cycle 11 which causes a structural hazard.



Structural Hazards

	Clock Number										
	1	2	3	4	5	6	7	8	9	10	11
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB
		IF	ID	EX	MEM	WB					
...			IF	ID	EX	MEM	WB				
ADD.D F2,F4,F6				IF	ID	A1	A2	A3	A4	MEM	WB
					IF	ID	EX	MEM	WB		
...						IF	ID	EX	MEM	WB	
L.D F2,0(R2)							IF	ID	EX	MEM	WB

- In theory, only L.D uses MEM on cycle 10, but definitely MUL.D, ADD.D and L.D all want to use WB on cycle 11 which causes a structural hazard.
- Let's say we wish to eliminate this structural hazard by stalling. In the next slide we see a simple hardware mechanism for inserting correct number of stall cycles before letting instructions leave ID stage.



Structural Hazards

	Clock Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB		
		IF	ID	EX	MEM	WB							
...			IF	ID	EX	MEM	WB						
ADD.D F2,F4,F6				IF	s	ID	A1	A2	A3	A4	MEM	WB	
						IF	ID	EX	MEM	WB			
...							IF	ID	EX	MEM	WB		
L.D F2,0(R2)								IF	s	ID	EX	MEM	WB

- In theory, only L.D uses MEM on cycle 10, but definitely MUL.D, ADD.D and L.D all want to use WB on cycle 11 which causes a structural hazard.
- Let's say we wish to eliminate this structural hazard by stalling. In the next slide we see a simple hardware mechanism for inserting correct number of stall cycles before letting instructions leave ID stage.



Structural Hazards

	Clock Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB		
		IF	ID	EX	MEM	WB							
...			IF	ID	EX	MEM	WB						
ADD.D F2,F4,F6				IF	s	ID	A1	A2	A3	A4	MEM	WB	
						IF	ID	EX	MEM	WB			
...							IF	ID	EX	MEM	WB		
L.D F2,0(R2)								IF	s	ID	EX	MEM	WB

- In theory, only L.D uses MEM on cycle 10, but definitely MUL.D, ADD.D and L.D all want to use WB on cycle 11 which causes a structural hazard.
- Let's say we wish to eliminate this structural hazard by stalling. In the next slide we see a simple hardware mechanism for inserting correct number of stall cycles before letting instructions leave ID stage.
- The key idea is to use a “shift register” (just like the shift registers you saw how to write VHDL for in EECE 353). We use this shift register to maintain a “schedule” in hardware for when the WB stage is busy.



Structural Hazards

	Clock Number												
	1	2	3	4	5	6	7	8	9	10	11	12	13
MUL.D F0,F4,F6	IF	ID	M1	M2	M3	M4	M5	M6	M7	MEM	WB		
		IF	ID	EX	MEM	WB							
...			IF	ID	EX	MEM	WB						
ADD.D F2,F4,F6				IF	s	ID	A1	A2	A3	A4	MEM	WB	
						IF	ID	EX	MEM	WB			
...							IF	ID	EX	MEM	WB		
L.D F2,0(R2)								IF	s	ID	EX	MEM	WB

- In theory, only L.D uses MEM on cycle 10, but definitely MUL.D, ADD.D and L.D all want to use WB on cycle 11 which causes a structural hazard.
- Let's say we wish to eliminate this structural hazard by stalling. In the next slide we see a simple hardware mechanism for inserting correct number of stall cycles before letting instructions leave ID stage.
- The key idea is to use a “shift register” (just like the shift registers you saw how to write VHDL for in EECE 353). We use this shift register to maintain a “schedule” in hardware for when the WB stage is busy.
- If bit “N” is ‘1’ then the WB stage will be busy N+M cycles from now (where M is the minimum number of cycles between decode and writeback).



Detection of Structural Hazard at ID Using Shift Register



Detection of Structural Hazard at ID Using Shift Register

An 8-bit shift register initially contains the following contents: “10100000”. Assuming the contents are shifted one bit to the right each cycle, what would the contents be after one clock cycle if the next input bit is ‘0’?

- A: “01000000”
- B: “01000001”
- C: “00100000”
- D: “01010000”
- E: Not sure



Detection of Structural Hazard at ID Using Shift Register

An 8-bit shift register initially contains the following contents: “10100000”. Assuming the contents are shifted one bit to the right each cycle, what would the contents be after one clock cycle if the next input bit is ‘0’?

- A: “01000000”
- B: “01000001”
- C: “00100000”
- D: “01010000” ✓
- E: Not sure



Detection of Structural Hazard at ID Using Shift Register

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

cycle
1

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	1



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB

cycle

1



Detection of Structural Hazard at ID Using Shift Register

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

cycle
2

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
2

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD

R1,R2,R2



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
2

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD R1,R2,R2



Shift Register
Direction of
Movement

Latency from ID:
MUL.D 9 cycles to WB
ADD.D 6 cycles to WB
DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
3

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD

R1,R2,R2



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
3

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
3

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
4

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6 ???



Detection of Structural Hazard Using Shift Register

Does i4 (ADD.D) need to stall?

A: Yes

B: No

C: Not sure

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB

cycle

4

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6 ???



Detection of Structural Hazard Using Shift Register

Does i4 (ADD.D) need to stall?

A: Yes ✓

B: No

C: Not sure

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB

cycle

4

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6 ???



Detection of Structural Hazard at ID Using Shift Register

cycle
4

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6 ???



Detection of Structural Hazard at ID Using Shift Register

cycle
5

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard Using Shift Register

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

cycle
5

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6

How many flip-flops (bits) does this shift register contain?

A: 9

B: 7

C: $9 \times 32 = 288$

D: $7 \times 4 = 28$

E: Not sure

Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard Using Shift Register

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9	0

cycle
5

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6

How many flip-flops (bits) does this shift register contain?

A: 9

B: 7 ✓

C: $9 \times 32 = 288$

D: $7 \times 4 = 28$

E: Not sure

Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

ADD.D 6 cycles to WB

DADD 3 cycles to WB



Detection of Structural Hazard at ID Using Shift Register

cycle
5

Cycles to WB	Used?
3	0
4	0
5	0
6	0
7	0
8	0
9: MUL.D F0, F4, F0	0

i2: DADD R1,R2,R2

i3: DADD R3,R4,R5

i4: ADD.D F2, F4, F6



Shift Register
Direction of
Movement

Latency from ID:

MUL.D 9 cycles to WB

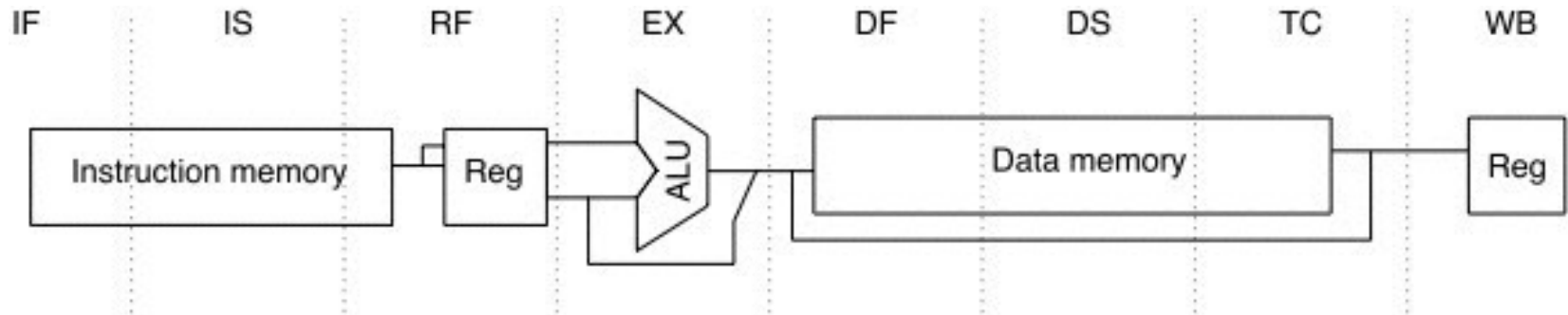
ADD.D 6 cycles to WB

DADD 3 cycles to WB



MIPS R4000

MIPS R4000 was introduced in 1991 (first 64-bit microprocessor).
Used an 8-stage pipeline to achieve higher clock frequency.



IF - first half of instruction fetch

IS - second half of instruction fetch

RF - instruction decode and register fetch

EX - execution

DF - data fetch, first half data cache access

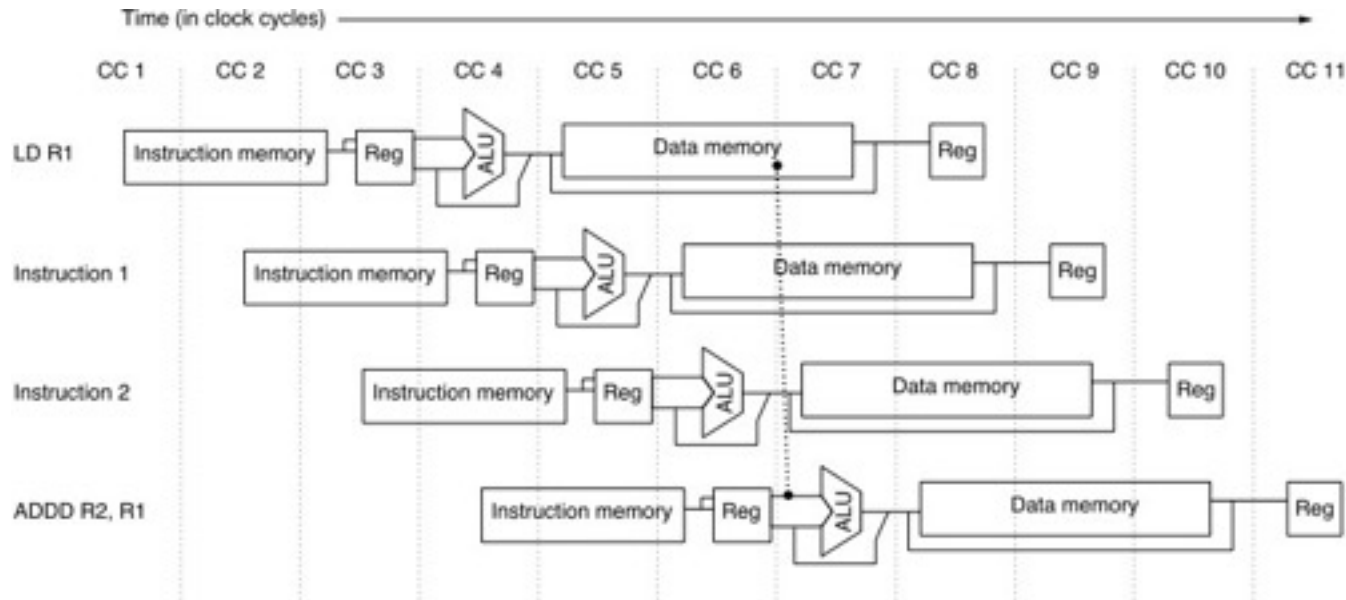
DS - second half of data cache access

TC - tag check, determine whether the data cache access hit.

WB - write back.



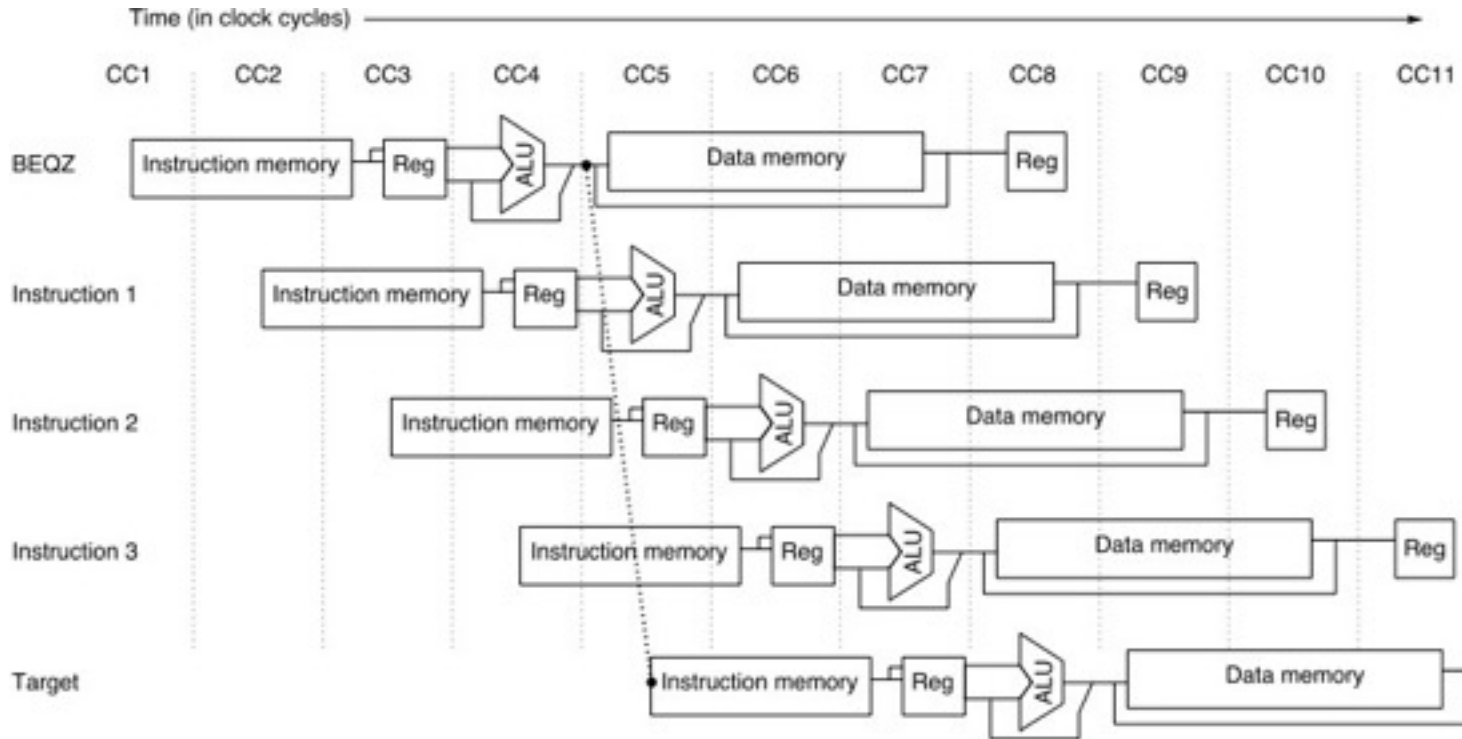
MIPS R4000: 2-cycle load delay



	Clock Number								
	1	2	3	4	5	6	7	8	9
LD R1, ...	IF	IS	RF	EX	DF	DS	TC	WB	
DADD R2, R1, ...		IF	IS	RF	stall	stall	EX	DF	DS
DSUB R3, R1, ...			IF	IS	stall	stall	RF	EX	DF
OR R4, R1, ...				IF	stall	stall	IS	RF	EX



MIPS R4000: 3-cycle branch penalty



Delayed branch can “hide” one cycle out of three (if we fill branch delay slot). We could try to increase number of delay slots, but quickly becomes impossible to fill all of them.



MIPS R4000: Delayed Branch Execution

delay slot
squashed

<u>TAKEN</u> BRANCH	Clock Number								
	1	2	3	4	5	6	7	8	9
Branch	IF	IS	RF	EX	DF	DS	TC	WB	
Branch inst + 1		IF	IS	RF	EX	DF	DS	TC	WB
Branch inst + 2			IF	IS	nop	nop	nop	nop	nop
Branch inst + 3				IF	nop	nop	nop	nop	nop
Branch target					IF	IS	RF	EX	DF

delay slot

<u>UNTAKEN</u> BRANCH	Clock Number								
	1	2	3	4	5	6	7	8	9
Branch	IF	IS	RF	EX	DF	DS	TC	WB	
Branch inst + 1		IF	IS	RF	EX	DF	DS	TC	WB
Branch inst + 2			IF	IS	RF	EX	DF	DS	TC
Branch inst + 3				IF	IS	RF	EX	DF	DS



MIPS R4000: Floating Point Pipeline

Add, multiple, divide floating-point operations share some logic =>
Opportunity to save area.

Terminology:

Latency = Time for single operation to complete

Initiation interval = How often a new operation of same type can start

Stage	Functional unit	Description
A	FP adder	Mantissa ADD stage
D	FP divider	Divide pipeline stage
E	FP multiplier	Exception test stage
M	FP multiplier	First stage of multiplier
N	FP multiplier	Second stage of multiplier
R	FP adder	Rounding stage
S	FP adder	Operand shift stage
U		Unpack FP numbers

Figure A.42 The eight stages used in the R4000 floating-point pipelines.

FP instruction	Latency	Initiation interval	Pipe stages
Add, subtract	4	3	U, S + A, A + R, R + S
Multiply	8	4	U, E + M, M, M, M, N, N + A, R
Divide	36	35	U, A, R, D ²⁷ , D + A, D + R, D + A, D + R, A, R
Square root	112	111	U, E, (A+R) ¹⁰⁸ , A, R
Negate	2	1	U, S
Absolute value	2	1	U, S
FP compare	3	2	U, A, R



Multiply followed by Add

		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							



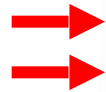
Multiply followed by Add



		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							



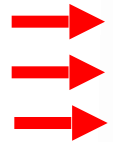
Multiply followed by Add



		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						



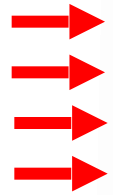
Multiply followed by Add



		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						
	Issue				U	S + A	A + R	R + S					



Multiply followed by Add



		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						
	Issue				U	S + A	A + R	R + S					
	Stall					U	S + A	A + R	R + S				

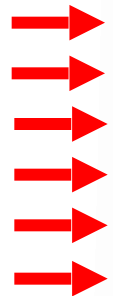


Multiply followed by Add

		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						
	Issue				U	S + A	A + R	R + S					
	Stall					U	S + A	A + R	R + S				
	Stall						U	S + A	A + R	R + S			



Multiply followed by Add



		Clock cycle												
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11	12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R					
Add	Issue		U	S + A	A + S	R + S								
	Issue			U	S + A	A + R	R + S							
	Issue				U	S + A	A + R	R + S						
	Stall					U	S + A	A + R	R + S					
	Stall						U	S + A	A + R	R + S				
	Issue							U	S + A	A + R	R + S			



Multiply followed by Add

		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						
	Issue				U	S + A	A + R	R + S					
	Stall					U	S + A	A + R	R + S				
	Stall						U	S + A	A + R	R + S			
	Issue							U	S + A	A + R	R + S		
	Issue								U	S + A	A + R	R + S	



Multiply followed by Add

		Clock cycle												
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11	12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R					
Add	Issue		U	S + A	A + S	R + S								
	Issue			U	S + A	A + R	R + S							
	Issue				U	S + A	A + R	R + S						
	Stall					U	S + A	A + R	R + S					
	Stall						U	S + A	A + R	R + S				
	Issue							U	S + A	A + R	R + S			
	Issue								U	S + A	A + R	R + S		

- Structural Hazards!



Multiply followed by Add

		Clock cycle												
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11	12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R					
Add	Issue		U	S + A	A + S	R + S								
	Issue			U	S + A	A + R	R + S							
	Issue				U	S + A	A + R	R + S						
	Stall					U	S + A	A + R	R + S					
	Stall						U	S + A	A + R	R + S				
	Issue							U	S + A	A + R	R + S			
	Issue								U	S + A	A + R	R + S		

- Structural Hazards!
- Add issued 4 cycles after Mult => 2 stall cycles



Multiply followed by Add

		Clock cycle											
Operation	Issue/stall	0	1	2	3	4	5	6	7	8	9	10	11 12
Multiply	Issue	U	E + M	M	M	M	N	N + A	R				
Add	Issue		U	S + A	A + S	R + S							
	Issue			U	S + A	A + R	R + S						
	Issue				U	S + A	A + R	R + S					
	Stall					U	S + A	A + R	R + S				
	Stall						U	S + A	A + R	R + S			
	Issue							U	S + A	A + R	R + S		
	Issue								U	S + A	A + R	R + S	

- Structural Hazards!
- Add issued 4 cycles after Mult => 2 stall cycles
- Add issued 5 cycles after Mult => 1 stall cycles



Add followed by Mult

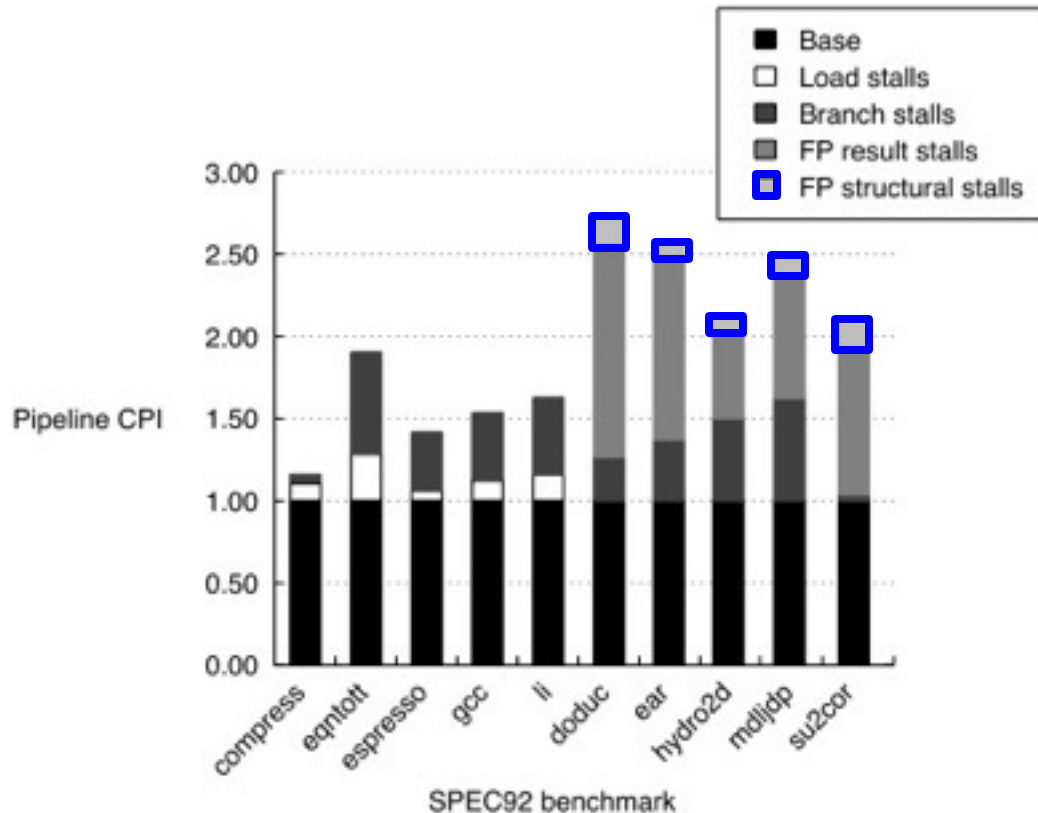
Operation	Issue/stall	Clock cycle												
		0	1	2	3	4	5	6	7	8	9	10	11	12
Add	Issue	U	S + A	A + R	R + S									
Multiply	Issue		U	E + M	M	M	M	N	N + A	R				
	Issue			U	M	M	M	M	N	N + A	R			

- No stall since the shorter operation (Add) clears shared stages before longer operation (Mult) reaches them.



MIPS R4000

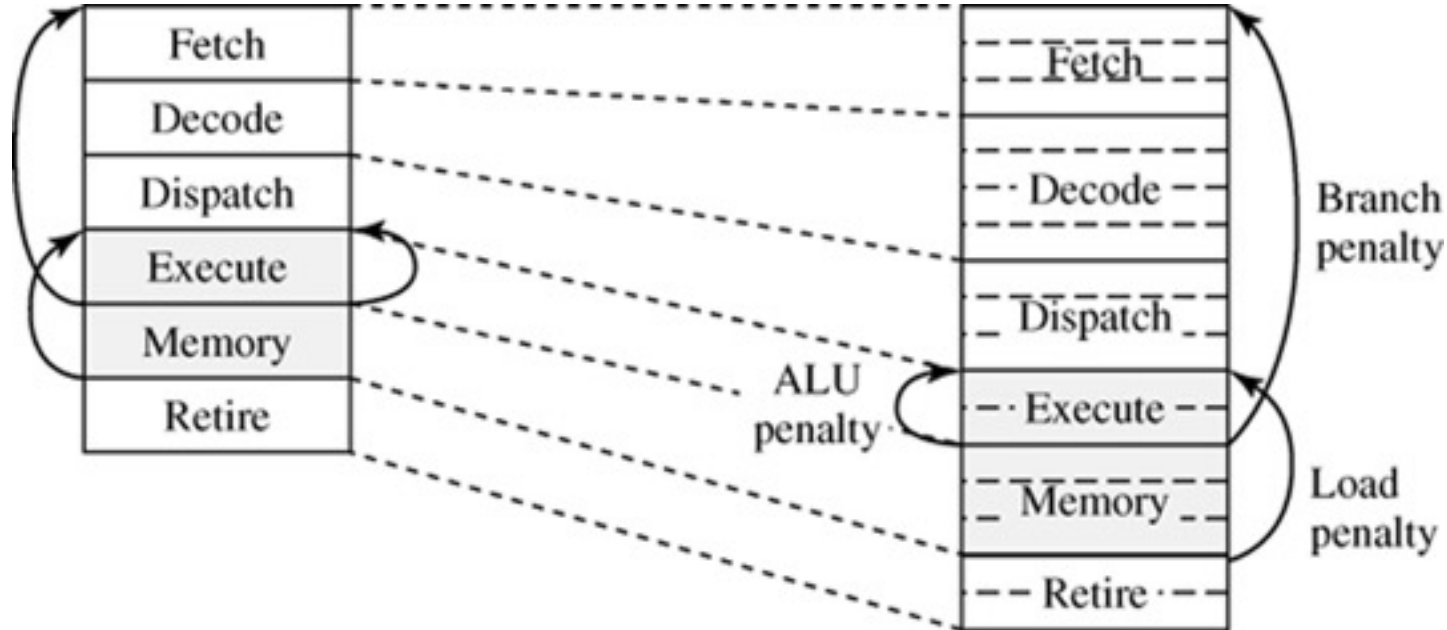
Performance Evaluation



- Five bars on left are integer benchmarks; five on right are floating-point benchmarks
- FP structural hazards due to initiation interval restrictions and shared pipeline stages do not increase average CPI by much. This implies that increased pipelining of execute stage or reducing structural hazards by duplicating shared hardware units will not increase performance.



Superpipelining and Hazard Penalties



- Dividing up logical pipeline stage into multiple stages is sometimes referred to as “superpipelining”
- Deeper pipelines increase delay (in clock cycles) from stage producing result to stage consuming result.
- CPI increases with increasing pipeline depth due to increase in stall cycles
- Increase in stall cycles result from “loose loops” in which forwarding loops have an increased number of pipeline stages between begin and end of forwarding path.
- Aside: Can think of a branch a “forwarding” a value to PC register in fetch stage when predicted path is incorrect.