



**WPI**

# ECE 505 Computer Architecture

Branch Prediction

Exceptions

Floating Point Instructions

Berk Sunar and Thomas Eisenbarth

# Branch Prediction

- **Static:** Always predict as taken or not taken.
  - Examples: First MIPS and SPARC architectures would predict not taken always.
  - Forward branches as taken and backwards as not taken.
- **Dynamic:** Predict branch based on its history.
  - Local: use only history of the analyzed branch.
  - General: use history of all branches.
- Static prediction is still used when the branch is not in the history.

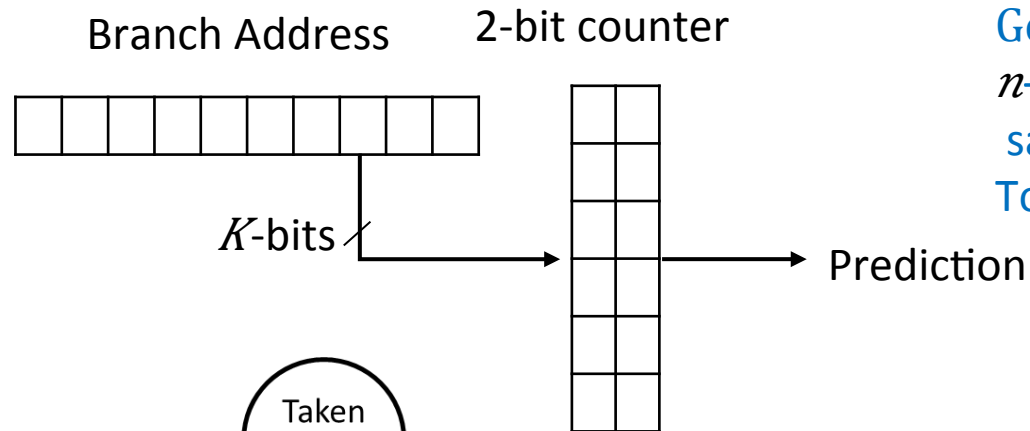
# Advanced Branch Prediction

- We have a long code with many branches. How can we predict the future behavior (T or UT) of each branch?

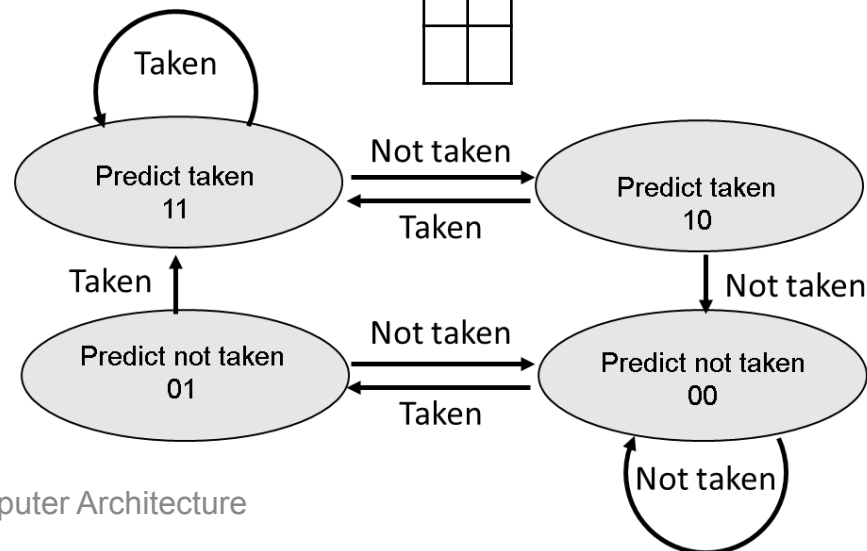
1<sup>st</sup> try: Use the  $k$ -LSB of branch address to select 1-bit predictor.

# Advanced Branch Prediction

- 2<sup>st</sup> try: Use the  $k$ -LSB of branch address to select 2-bit predictor. Total size =  $2*2k$  bits



Generalized to:  
 $n$ -bit predictor using  $n$ -bit  
saturating counter  
Total Size =  $n*2k$  bits



# Advanced Branch Prediction

- 3<sup>rd</sup> try: Correlating Branch Predictor  
Track the last  $m$  branches. For each sequence, use a separate predictor (Global Information).

Why?

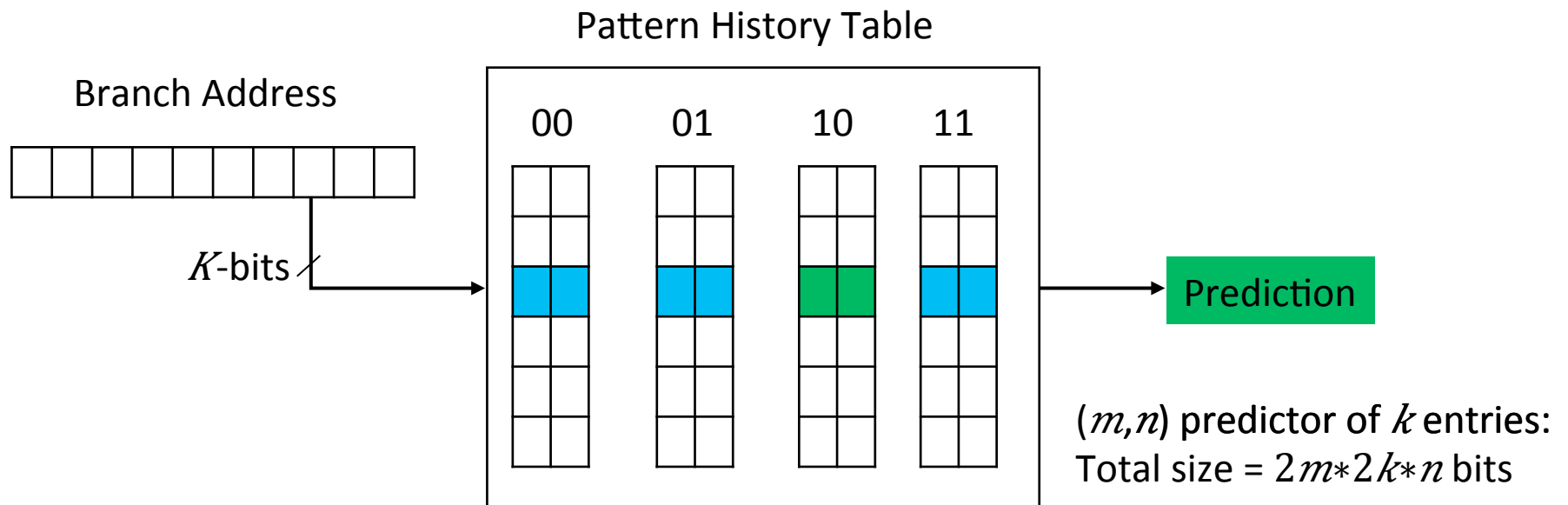
```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) {
```

Branch behavior depends on previous branches

# Advanced Branch Prediction

- 3<sup>rd</sup> try: Correlating Branch Predictor
  - Use the  $k$ -LSB of branch address
  - Also, use the result of  $m$ -last branches
  - Select  $n$ -bit predictor.

```
if (aa==2)
    aa=0;
if (bb==2)
    bb=0;
if (aa!=bb) {
```

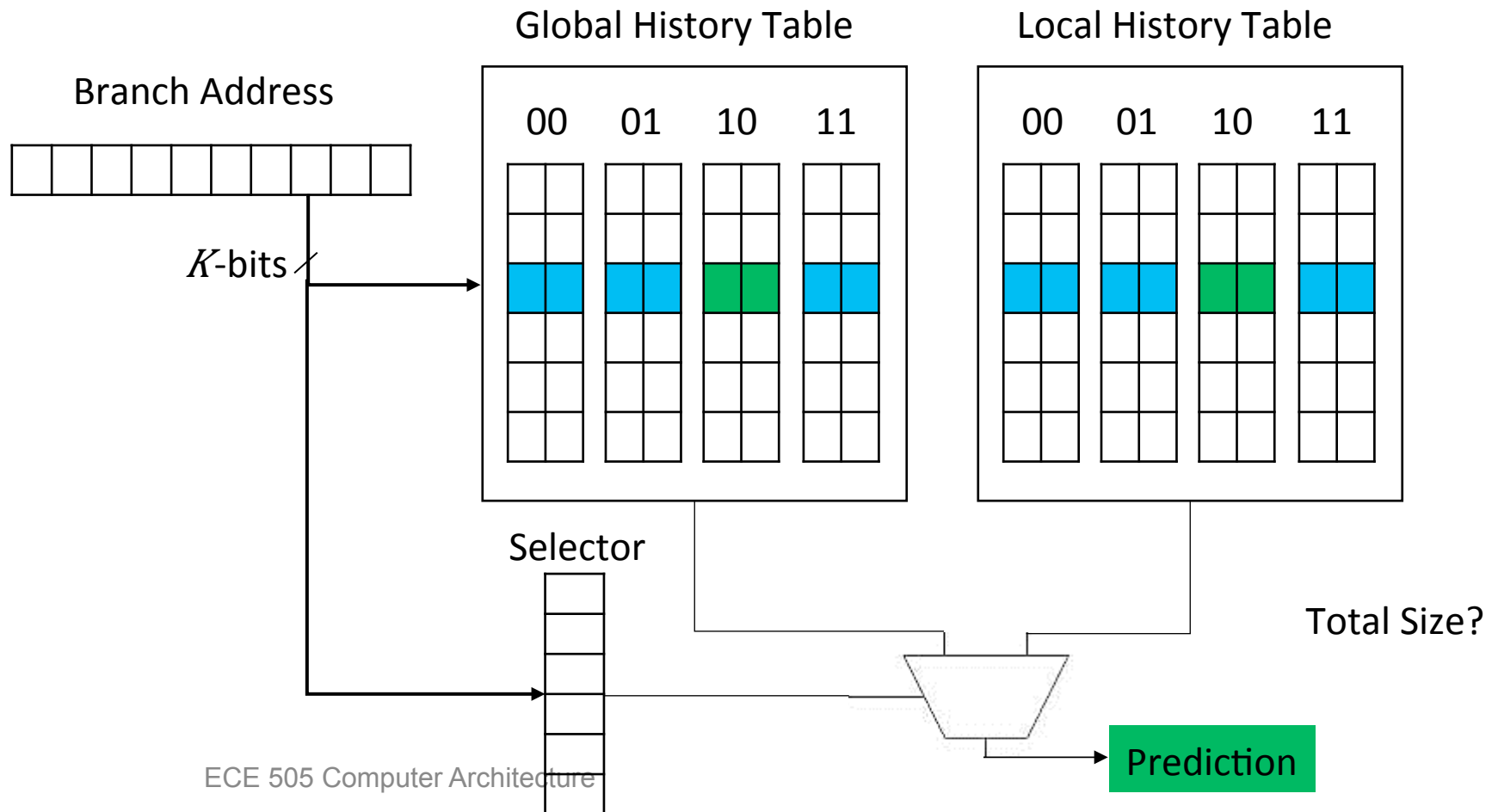


# Advanced Branch Prediction

- But, local information also follow patterns
- 4<sup>th</sup> try: Tournament Predictors
  - Use 2 different predictors:
    - Local, following the pattern of the individual branch
    - Global, following the pattern of previous branches
  - Use a 2-bit saturating counter to select between them.

# Advanced Branch Prediction

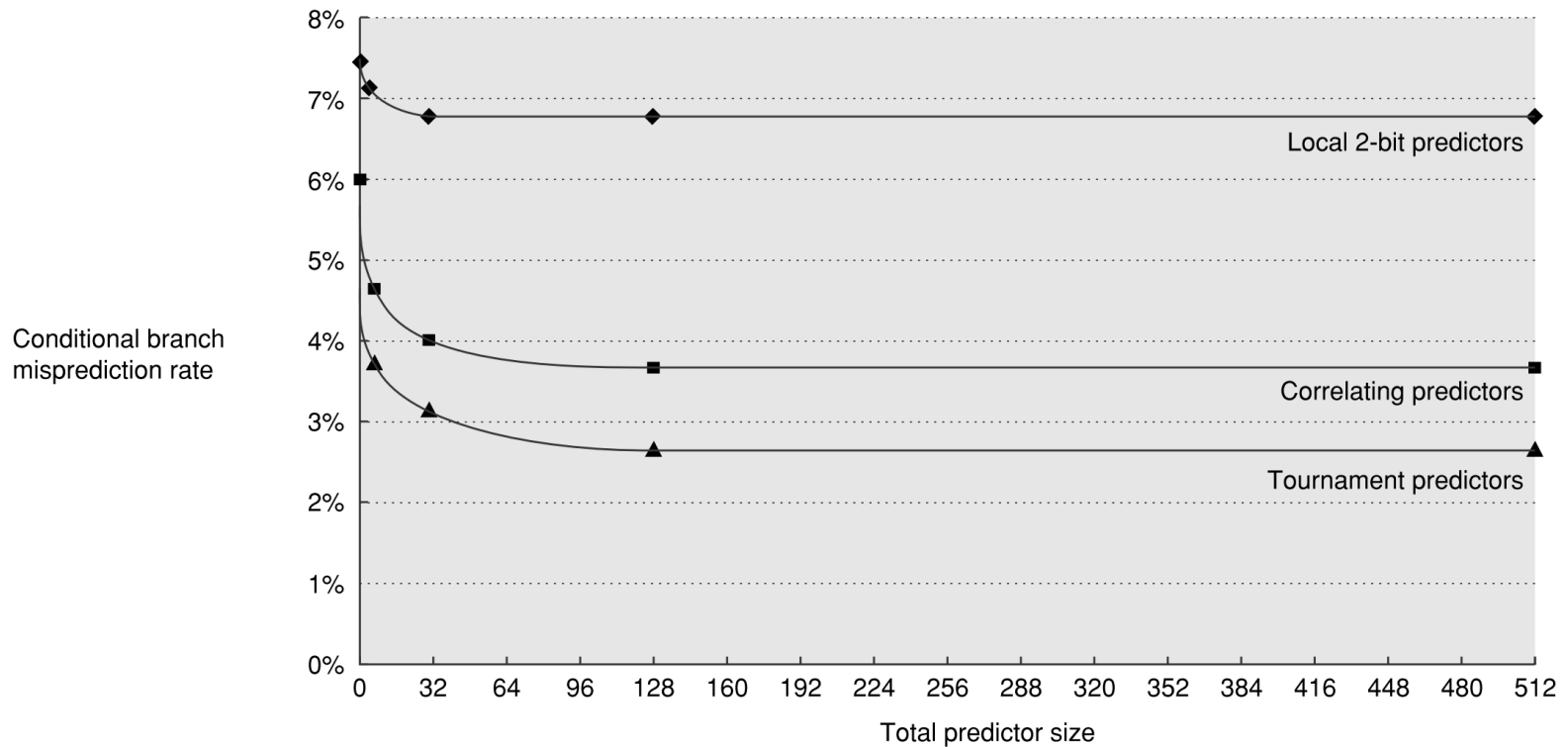
- 4<sup>th</sup> try: Tournament Predictors





# Advanced Branch Prediction

- Performance:



Tournament Predictors can use 3 different predictors: Local, Global and a Special (Loops)

# Further Problems with Pipelining

- Handling Exceptions

```
SW R2, 0(R4)
```

- Handling Floating-Point Operations

```
MUL.D F2, F0, F8
```

- Case Example (MIPS R4000)

# Exceptions

- What is an exception?

```
A problem has been detected and windows has been shut down to prevent damage
to your computer.

If this is the first time you've seen this stop error screen,
restart your computer. If this screen appears again, follow
these steps:

Check to be sure you have adequate disk space. If a driver is
identified in the stop message, disable the driver or check
with the manufacturer for driver updates. Try changing video
adapters.

Check with your hardware vendor for any BIOS updates. Disable
BIOS memory options such as caching or shadowing. If you need
to use Safe Mode to remove or disable components, restart your
computer, press F8 to select Advanced Startup options, and then
select Safe Mode.

Technical information:

*** STOP: 0x0000001E (0xFFFFFFFFC0000094,0xFFFFF8000C074D1E,0x0000000000000000,0
xFFFFFFFFFFFFFFFFFD)
```

# Exceptions

- Characteristics of exceptions:
  - Synchronous Vs Asynchronous  
Cause by the code itself?
  - User Requested Vs Coerced  
Is it predictable?
  - User Maskable VS User Nonmaskable  
The hardware response can be disabled?
  - Within Vs Between Instructions  
During the execution of an instruction?
  - Resume Vs Terminate  
Try to resume or terminate the program?

# Exceptions

Exception type	Synchronous?	User request?	Maskable?	Within?	Resume or Terminate?
I/O device request	Asynchronous	Coerced	Non-maskable	Between	Resume
Invoke operating system	Synchronous	User request	Non-maskable	Between	Resume
Breakpoint	Synchronous	User request	maskable	Between	Resume
arithmetic overflow	Synchronous	Coerced	maskable	Within	Resume
Page fault	Synchronous	Coerced	Non-maskable	Within	Resume
Using undefined instructions	Synchronous	Coerced	Non-maskable	Within	Terminate
Hardware malfunctions / Power failure	Asynchronous	Coerced	Non-maskable	Within	Terminate

# Exceptions

- How can we “resume” after “within” exception?
  - Force a trap instruction into the IF
  - Turn-off all writes into memory or registers  
(Never commit any further instruction)
  - Saves the PC to return later  
If out-of-order execution (e.g. delay branch):  
Save PC of all instructions

# Exceptions

IF	ID	EX	MEM	WB						✓
	IF	ID	EX	MEM	WB					✓
		IF	ID	<del>EX</del>	MEM	WB				✗
			IF	ID	EX	MEM	WB			✗
				IF	ID	EX	MEM	WB		✗

Arithmetic Exception

- Precise Exception

If the pipeline can be stopped so that the instructions just before the faulting instruction are completed and those after it can be restarted from scratch.

- Problems?

Multiple exceptions, out-of-order completion

# Exceptions

- Handling Multiple Exceptions

LD	IF	ID	EX	<del>MEM</del>	WB		Page Fault
DADD		IF	ID	<del>EX</del>	MEM	WB	Arithmetic Exception

Resolve one-by-one

LD	IF	ID	EX	<del>M</del> MEM	WB		Page Fault
DADD		<del>IF</del>	ID	EX	MEM	WB	Page Fault

Exception Status Vector



# Exceptions

## Possible exceptions in each stage

- **IF:** Page fault; misaligned memory access; memory protection violation
- **ID:** Undefined or illegal opcode
- **EX:** Arithmetic exception
- **MEM:** Page fault; misaligned memory access; memory protection violation
- **WB:** None

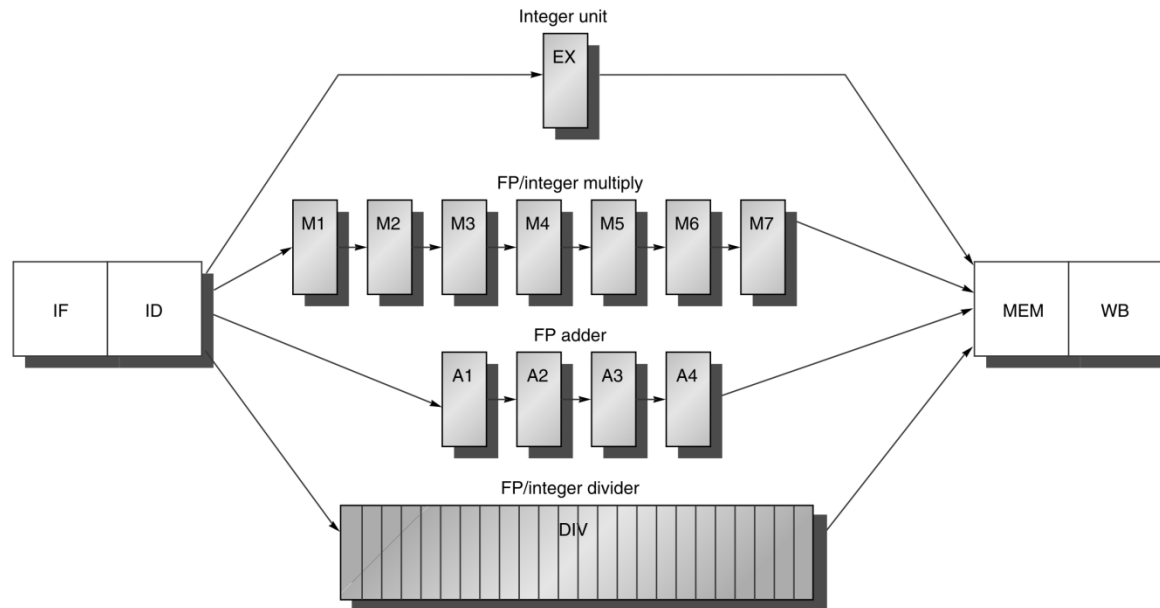
# Floating-Point Operations

```
MUL.D F2, F0, F8
```

- Problems?
  - FP unit require longer latency
  - Do all FP operations require same latency?
  - Complications on Pipelining?
  - Do we have to stall?
  - Can we commit a later but faster instruction?
  - What happen if we have an arithmetic exception?

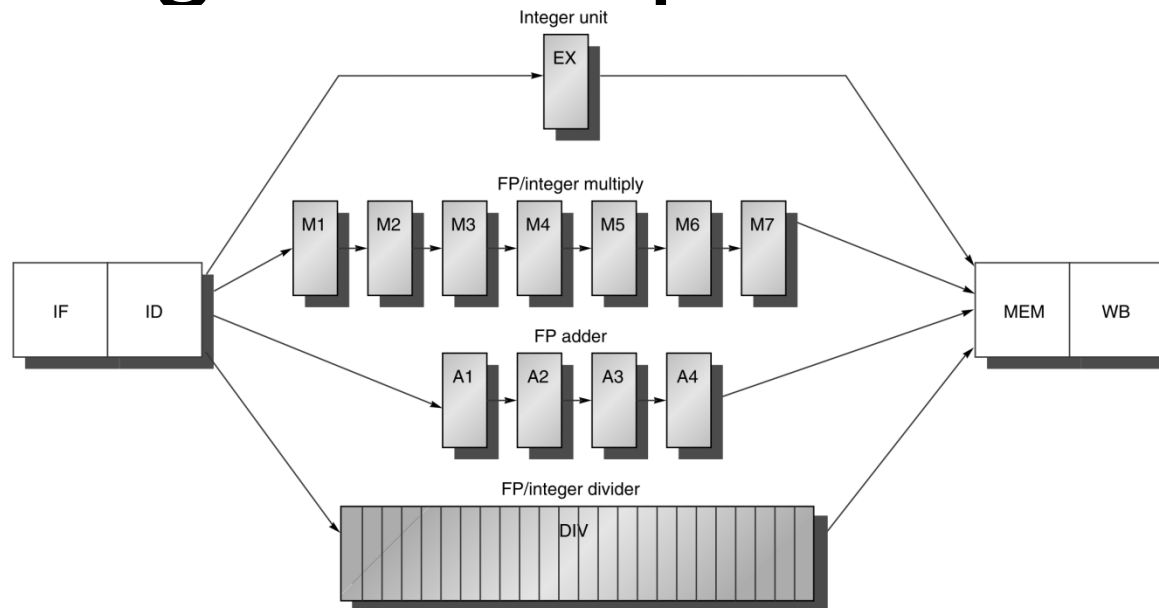
# Floating-Point Operations

- Assume four separate EX units



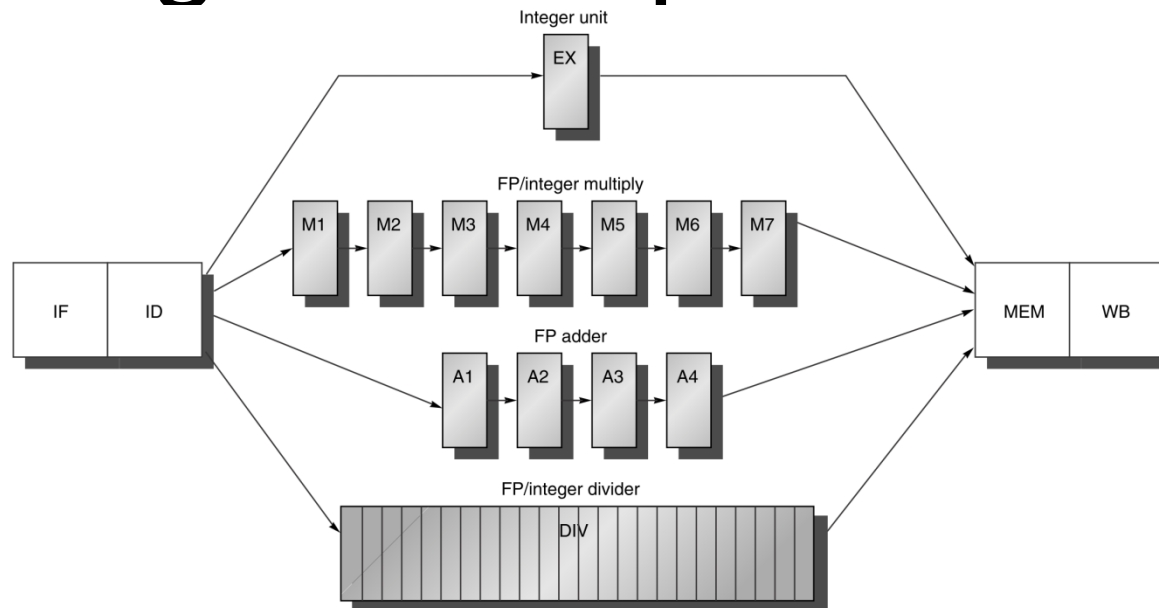
- Latency: The number of cycles between an instruction that produces a result and an instruction that uses the result.
- Initiation Interval: The number of cycles that must elapse between issuing two operations of the same type.

# Floating-Point Operations



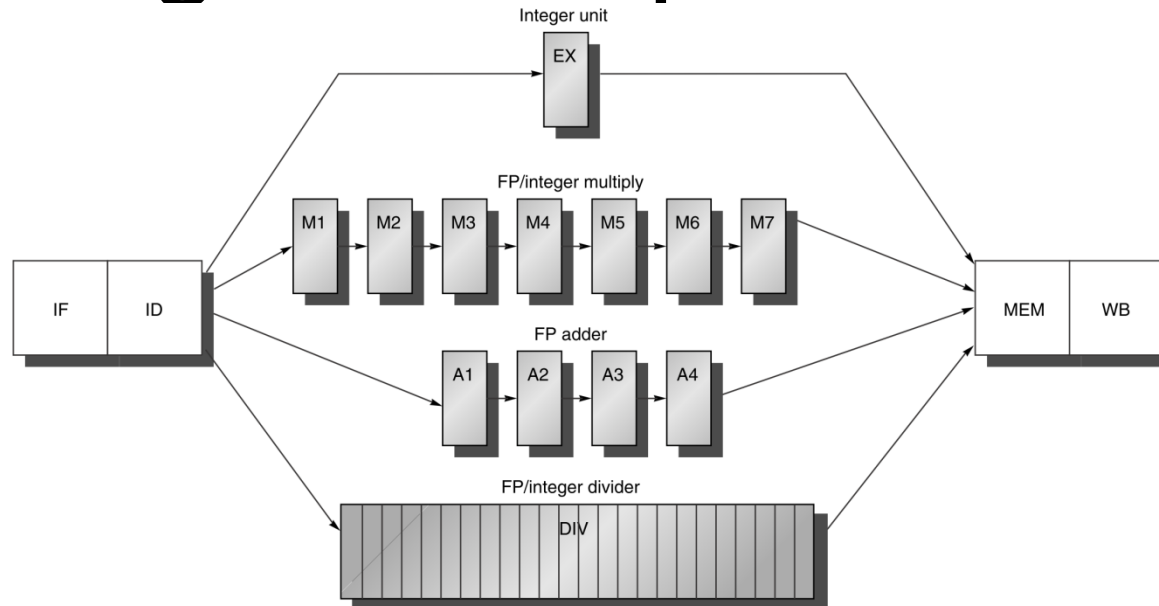
Functional unit	Latency	Initiation interval
Integer ALU	0	1
Data memory (integer and FP loads)	1	1
FP add	3	1
FP multiply (also integer multiply)	6	1
FP divide (also integer divide)	24	25

# Floating-Point Operations



	Clock cycle number																
Instruction	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	MEM	WB												
MUL.D F0,F4,F6																	
ADD.D F2,F0,F8																	
S.D F2,0(R2)																	

# Floating-Point Operations



	Clock cycle number																
Instruction	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	MEM	WB												
MUL.D F0,F4,F6	IF	ID	stall		M1	M2	M3	M4	M5	M6	M7	MEM	WB				
ADD.D F2,F0,F8		IF	stall	ID	stall	stall	stall	stall	stall	stall	stall	A1	A2	A3	A4	MEM	WB
S.D F2,0(R2)				IF	stall	stall	stall	stall	stall	stall	stall	ID	EX	stall	stall	stall	MEM

# Floating-Point Operations

- Precise Exception?

```
DIV.D F0, F2, F4
```

```
ADD.D F10, F10, F8
```

```
SUB.D F12, F12, F14
```

Out-of-order completion. If `DIV` raised an exception, the value of `F10` may be lost.

Solutions?

- Give up (I am imprecise exception)
- Two modes of operation
- Buffer all the results
- Mark committed instructions back to software